Shri Labhubhai Trivedi Institute of Engineering & Technology

Name of the Faculty	:	Ms. Pooja P. Vasani
Department	:	Computer Science & Engineering
Semester	:	7th Semester
Subject	:	Soft computing
Subject Code	:	173101

QUESTION BANK FROM GTU-PAPERS

1. What are Evolutionary Algorithms? Explain Simple Genetic Algorithm with the help of a flowchart. Also explain how GA differs from other traditional algorithms.

(nov 2011, dec 2012)

In artificial intelligence, an evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based meta-heuristic optimization algorithm. An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Evolutionary algorithms often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape; this generality is shown by successes in fields as diverse as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, politics and chemistry

Genetic Algorithms (GAs) can be seen as a software tool that tries to find structure in data that might seem random, or to make a seemingly unsolvable problem more or less 'solvable'. GAs can be applied to domains about which there is insufficient knowledge or the size and/or complexity is too high for analytic solution. However, the previous sentence should not be read as 'areas of which we have no knowledge': such problems are often characterised by multiple and complex, sometimes even contradictory constraints, that must be all satisfied at the same time.

So, what is different in GAs compared to more normal optimization and search procedures that it deserves interest? There are lots of reasons why:

- GAs work with a coding for the parameter set, not the parameters themselves.

-Gas search from a population of points, not a single point.

- GAs use payoff (objective function) information, not derivatives or other auxiliary knowledge.

-GAs use probabilistic transition rules, not deterministic rules.

-The Genetic Algorithms are first suited for the optimization problem.

-GAs are robust with respect to local minima, maxima.

-GAs work well on mixed discrete/continuous problems.

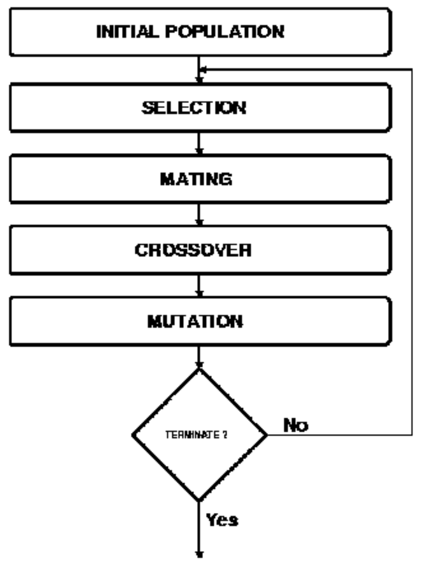
-GAs can operate on various representations.

-GAs are stochastic.

-GAs are easily parallelised.

-GAs are easy to implement and lend themselves well to hybridization.

- Sometimes there is simply no other approach.



The main reasons to use a genetic algorithm are:

- there are multiple local optima
- the objective function is not smooth (so derivative methods can not be applied)
- the number of parameters is very large
- the objective function is noisy or stochastic

A large number of parameters can be a problem for derivative based methods when you don't have the definition of the gradient. In this type of situation, you can find a not-terrible solution via GA and then improve on that with the derivative based method.

2. Write a short note on : GA based Weight optimisation. (dec 2012)

Weight Extraction

Extract weights from each chromosomes, later to determine the fitness values.

Let ***x1 , x2 , ... x d , ... x L*** represent a chromosome and

Let ***xkd+1 , xkd+2 , . . x(k + 1)d*** represent **kth** gene **(k** ≥ **0)** in the chromosomes.

The actual weight ***wk*** is given by

$$w_k = \begin{cases} + \dfrac{x_{kd+2}\,10^{d-2} + x_{kd+3}\,10^{d-3} + \ldots + x_{(k+1)d}}{10^{d-2}}, & \text{if } 5 \le x_{kd+1} \le 9 \\ - \dfrac{x_{kd+2}\,10^{d-2} + x_{kd+3}\,10^{d-3} + \ldots + x_{(k+1)d}}{10^{d-2}}, & \text{if } 0 \le x_{kd+1} < 5 \end{cases}$$

The Chromosomes are stated in the Fig. The weights extracted from all the eight genes are:

- **Gene 0 : 84321** ,

  Here we have, $k = 0, d = 5$, and x_{kd+1} is x_1 such that

  $5 \le x_1 = 8 \le 9$. Hence, the weight extracted is

  $$W_0 = + \frac{4 \times 10^3 + 3 \times 10^2 + 2 \times 10 + 1}{10^3} = +4.321$$

- **Gene 1 : 46234** ,

  Here we have, $k = 1, d = 5$, and x_{kd+1} is x_6 such that

  $0 \le x_6 = 4 \le 5$. Hence, the weight extracted is

  $$W_1 = - \frac{6 \times 10^3 + 2 \times 10^2 + 3 \times 10 + 4}{10^3} = -6.234$$

- Similarly for the remaining genes

Gene 2 : 78901 yields W2 = + 8.901

Gene 3 : 32104 yields W3 = – 2.104

Gene 4 : 42689 yields W4 = – 2.689

Gene 5 : 63421 yields W5 = + 3.421

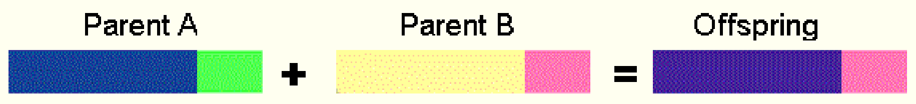
Gene 6 : 46421 yields W6 = – 6.421

Gene 7 : 87640 yields W7 = + 7.640

3. Explain mutation operator and cross-over operator in detail and give its difference with cross-over. (dec 2012, may 2012)

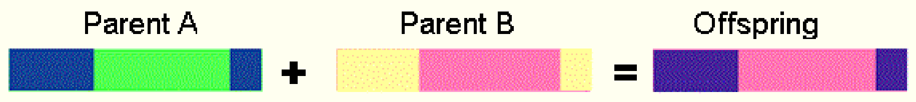
Crossover

Single point crossover - one crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, the rest is copied from the second parent

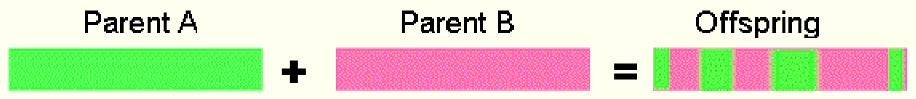


11001011+11011**111** = **11001111**

Two point crossover - two crossover point are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent



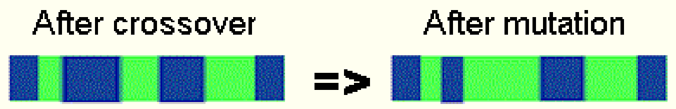
110010**11** + 11**0111**11 = **11011111**
Uniform crossover - bits are randomly copied from the first or from the second parent



1**10**010**11** + **1**10**111**01 = 11011111

Mutation

Bit inversion - selected bits are inverted



11001001 => 1**0**001001

- Crossover is explorative, it makes a *big* jump to an area somewhere "in between" two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent
- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, 50% after performing *n* crossovers)
- To hit the optimum you often need a 'lucky' mutation

4. Describe different selection methods for GA. (nov 2011)

Roulette wheel selection (Fitness-Proportionate Selection)
Roulette-wheel selection, also known as Fitness Proportionate Selection, is a genetic operator, used for selecting potentially useful solutions for recombination.

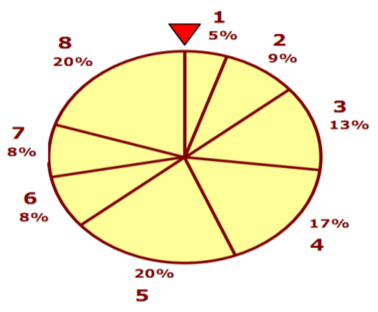


Fig. Roulette-wheel Shows 8 individual with fitness

In fitness-proportionate selection :

− the chance of an individual's being selected is proportional to its fitness, greater or less than its competitors' fitness.
− conceptually, this can be thought as a game of Roulette.

The Roulette-wheel simulates **8** individuals with fitness values **F** marked at its circumference; e.g.,

− the **5th** individual has a higher fitness than others, so the wheel would choose the **5th** individual more than other individuals .

− the fitness of the individuals is calculated as the wheel is spun **n = 8** times, each time selecting an instance, of the string, chosen by the wheel pointer.

Probability of i^{th} string is $p_i = F_i \,/\, (\sum_{j=1}^{n} F_j)$, where

n = no of individuals, called population size; **pi = probability** of i^{th} string being selected; **F_i = fitness** for i^{th} string in the population. Because the circumference of the wheel is marked according to a string's fitness, the Roulette-wheel mechanism is expected to make $\frac{F}{\bar{F}}$ copies of the i^{th} string.

Average fitness = $\bar{F}$ F_j / n ; **Expected count = (n =8) x pi**

Cumulative Probability$_5$ = $\sum_{i=1}^{N=5} p_i$

Tournament

- Binary tournament
- Two individuals are randomly chosen; the fitter of the two is selected as a parent
- Probabilistic binary tournament
- Two individuals are randomly chosen; with a chance *p*, $0.5<p<1$, the fitter of the two is selected as a parent
- Larger tournaments
- *n* individuals are randomly chosen; the fittest one is selected as a parent
- By changing *n* and/or *p*, the GA can be adjusted dynamically

Fitness scaling

- Fitness values are scaled by subtraction and division so that worst value is close to 0 and the best value is close to a certain value, typically 2
- Chance for the most fit individual is 2 times the average
- Chance for the least fit individual is close to 0

- Problems when the original maximum is very extreme (super-fit) or when the original minimum is very extreme (super-unfit)
- Can be solved by defining a minimum and/or a maximum value for the fitness

Fitness ranking

- Individuals are numbered in order of increasing fitness
- The rank in this order is the adjusted fitness
- Starting number and increment can be chosen in several ways and influence the results
- No problems with super-fit or super-unfit
- Often superior to scaling and windowing

5. Explain the following terms related to GA: (dec 2012)

a. Steady-State Replacement

Typically, the run of a genetic algorithm is divided into generations - each generation your selection and reproduction process replaces all (or at least most) of the population. In a steady state genetic algorithm you only replace a few individuals at a time.

b. Offspring Generation

Genetic algorithms are inspired by Darwin's theory about evolution. Solution to a problem solved by genetic algorithms is evolved.

Algorithm is started with a **set of solutions** (represented by **chromosomes**) called **population**. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (**offspring**) are selected according to their fitness - the more suitable they are the more chances they have to reproduce.

c. Generation Gap

The portion of the population that is replaced each generation. A generation gap of 0 means none of the population is replaced, conversely a generation gap of 1 means that the entire population is replaced each generation.

d. Fitness Function

A **fitness function** is a particular type of objective function that is used to summarise, as a single figure of merit, how close a given design solution is to achieving the set aims

e. Encoding

Binary encoding is the most common, mainly because first works about GA used this type of encoding.
In binary encoding every chromosome is a string of bits, 0 or 1.
Permutation encoding can be used in ordering problems, such as travelling salesman problem or task ordering problem.
In permutation encoding, every chromosome is a string of numbers, which represents number in a sequence.
Direct value encoding can be used in problems, where some complicated value, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult.
In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects.

6. Explain Travelling Salesman Problem. Also suggest different operators used to solve it using GA. (nov 2011, dec 2012)

There are cities and given distances between them.Travelling salesman has to visit all of them, but he does not to travel very much. Task is to find a sequence of cities to minimize travelled distance. In other words, find a minimal Hamiltonian tour in a complete graph of ***N*** nodes.

Population of 16 chromosomes is used. For encoding these chromosome permutation encoding is used - in chapter about encoding you can find, how to encode permutation of cities for TSP. TSP is solved on complete graph (i.e. each node is connected to each other) with euclidian distances. Note that after adding and deleting city it is necessary to create new chromosomes and restart whole genetic algorithm.

You can select crossover and mutation type. I will describe what they mean.

Crossover

- One point - part of the first parent is copied and the rest is taken in the same order as in the second parent
- Two point - two parts of the first parent are copied and the rest between is taken in the same order as in the second parent
- None - no crossover, offspring is exact copy of parents

Mutation

- Normal random - a few cities are chosen and exchanged
- Random, only improving - a few cities are randomly chosen and exchanged only if they improve solution (increase fitness)
- Systematic, only improving - cities are systematically chosen and exchanged only if they improve solution (increase fitness)
- Random improving - the same as "random, only improving", but before this is "normal random" mutation performed
- Systematic improving - the same as "systematic, only improving", but before this is "normal random" mutation performed
- None - no mutation

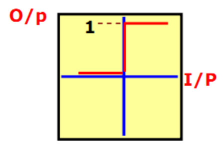
7. Explain the significance of Activation Function in NN? Describe different types of activation function in detail. (may 2012)

An activation function **f** performs a mathematical operation on the signal output. The most common activation functions are:
- Threshold Function,
- Piecewise Linear Function,
- Sigmoidal (S shaped) function,
The activation functions are chosen depending upon the type of problem to be solved by the network.

- **Threshold Function**

A threshold (hard-limiter) activation function is either a binary type or a bipolar type as shown below.

binary threshold



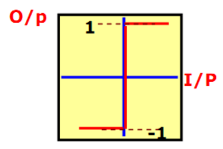
Output of a binary threshold function produces :

1 if the weighted sum of the inputs is +ve,

0 if the weighted sum of the inputs is -ve.

$$Y = f(I) = \begin{cases} 1 & \text{if } I \geq 0 \\ 0 & \text{if } I < 0 \end{cases}$$

bipolar threshold



Output of a bipolar threshold function produces :

1 if the weighted sum of the inputs is +ve,

-1 if the weighted sum of the inputs is -ve.

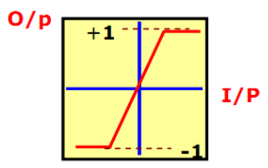
$$Y = f(I) = \begin{cases} 1 & \text{if } I \geq 0 \\ -1 & \text{if } I < 0 \end{cases}$$

• Piecewise Linear Function

This activation function is also called saturating linear function and can have either a binary or bipolar range for the saturation limits of the output.

The mathematical model for a symmetric saturation function is described below.

Piecewise Linear



This is a sloping function that produces :

-1 for a -ve weighted sum of inputs,

1 for a +ve weighted sum of inputs.

$\propto$ I proportional to input for values between **+1** and **-1** weighted sum,

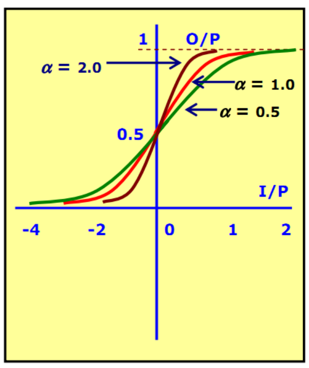
$$Y = f(I) = \begin{cases} 1 & \text{if } I \geq 0 \\ I & \text{if } -1 \geq I \geq 1 \\ -1 & \text{if } I < 0 \end{cases}$$

• Sigmoidal Function (S-shape function)

The nonlinear curved S-shape function is called the sigmoid function.

This is most common type of activation used to construct the neural networks. It is mathematically well behaved, differentiable and strictly increasing function.

Sigmoidal function



A sigmoidal transfer function can be written in the form:

$$Y = f(I) = \frac{1}{1 + e^{-\alpha I}}, \quad 0 \leq f(I) \leq 1$$

$$= 1/(1 + \exp(-\alpha I)), \; 0 \leq f(I) \leq 1$$

This is explained as

$\approx$ **0** for large -ve input values,

1 for large +ve values, with

a smooth transition between the two.

α is slope parameter also called shape parameter; symbol the λ is also used to represented this parameter.

The sigmoidal function is achieved using exponential equation.

By varying α different shapes of the function can be obtained which adjusts the abruptness of the function as it changes between the two asymptotic values.

8. What is artificial neural network? Explain different types of NN architectures. (nov 2011)

Neural Computers mimic certain processing capabilities of the human brain.

- Neural Computing is an information processing paradigm, inspired by biological system, composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems.

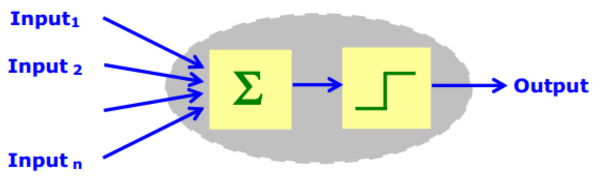
- Artificial Neural Networks (ANNs), like people, learn by example.

- An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

- Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

An artificial neuron is a mathematical function conceived as a simple model of a real (biological) neuron.

This is a simplified model of real neurons, known as a Threshold Logic Unit.



Neuron consists of three basic components - weights, thresholds, and a single activation function.

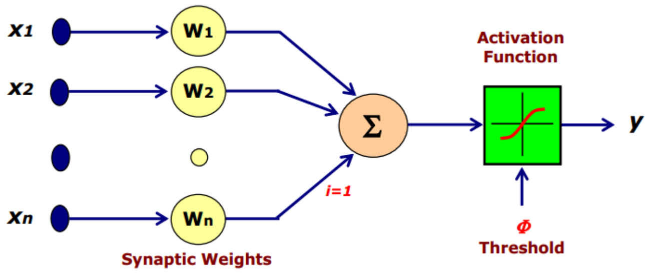


Fig Basic Elements of an Artificial Linear Neuron

Single Layer Feed-forward Network

The Single Layer Feed-forward Network consists of a single layer of weights , where the inputs are directly connected to the outputs, via a series of weights. The synaptic links carrying weights connect every input to every output , but not other way. This way it is considered a network of **feed-forward** type. The sum of the products of the weights and the inputs is calculated in each neuron node, and if the value is above some threshold (typically **0**) the neuron fires and takes the activated value (typically **1**); otherwise it takes the deactivated value (typically **-1**).

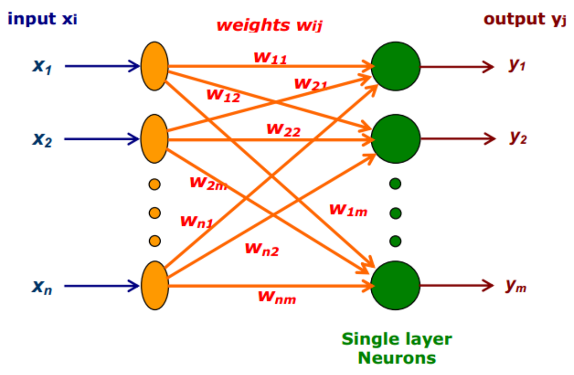


Fig. Single Layer Feed-forward Network

Multi Layer Feed-forward Network

The name suggests, it consists of multiple layers. The architecture of this class of network, besides having the input and the output layers, also have one or more intermediary layers called hidden layers. The computational units of the hidden layer are known as hidden neurons .

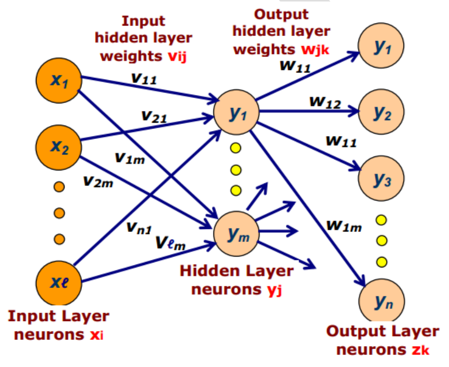


Fig. Multilayer feed-forward network in $(\ell - m - n)$ configuration.

- The hidden layer does intermediate computation before directing the input to output layer.

- The input layer neurons are linked to the hidden layer neurons; the weights on these links are referred to as **input-hidden layer weights** .

- The hidden layer neurons and the corresponding weights are referred to as **output-hidden layer weights** .

- A multi-layer feed-forward network with ***ℓ*** input neurons, **m1** neurons in the first hidden layers, **m2** neurons in the second hidden layers, and n output neurons in the output layers is written as**(*ℓ*-m1-m2–n).**

The Fig. above illustrates a multilayer feed-forward network with a configuration ***(ℓ - m – n).***

Recurrent Networks

The Recurrent Networks differ from feed-forward architecture. A Recurrent network has at least one feed back loop.

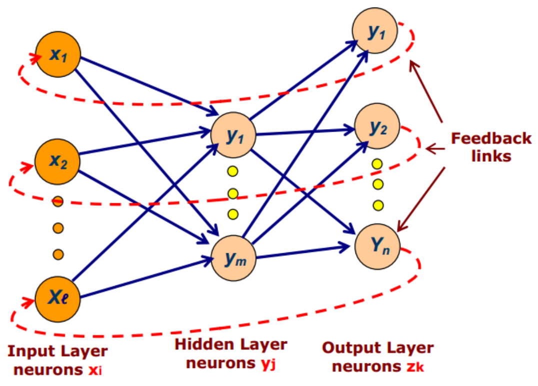


Fig Recurrent Neural Network

There could be neurons with self-feedback links; that is the output of a neuron is fed back into it self as input.

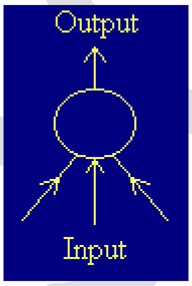
9. Compare: (nov 2011, may 2012)

a. Feed forward Vs. Feedback network

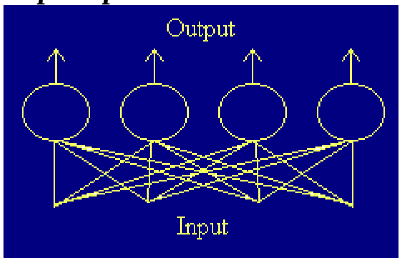
Feed-forward ANNs allow signals to travel one way only: from input to output. There are no feedback (loops); *i.e.*, the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straightforward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organisation is also referred to as bottom-up or top-down
Feedback (or recurrent or interactive) networks can have signals traveling in both directions by introducing loops in the network. Feedback networks are powerful and can get extremely complicated. Computations derived from earlier input are fed back into the network, which gives them a kind of memory. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found

b. Single Layer Vs. Multilayer Perceptron

a neural network consisting of a layer of these things between the input and the output is a ***single layer perceptron*** .



and a network consisting of several layers of these stacked on top of each other between input and output is called a ***multi-layer perceptron***



c. Artifical neuron Vs. Human brain

Brains are analogue; computers are digital
The brain is a massively parallel machine; computers are modular and serial
Processing speed is not fixed in the brain; there is no system clock
Short-term memory is not like RAM
No hardware/software distinction can be made with respect to the brain or mind
Synapses are far more complex than electrical logic gates
Unlike computers, processing and memory are performed by the same components in the brain

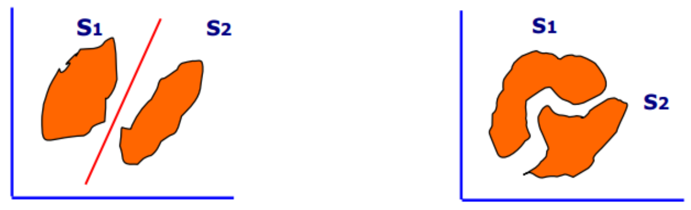
10. What is linear separability? Give example of linear separable and inseparable problem? Why single layer perceptron is not capable of solving linearly inseparable problem? (nov 2011)

Perceptron can not handle tasks which are not separable.

- Definition : Sets of points in 2-D space are linearly separable if the sets can be separated by a straight line.

- Generalizing, a set of points in n-dimensional space are linearly separable if there is a hyper plane of (n-1) dimensions separates the sets.

Example



(a) Linearly separable patterns **(b) Not Linearly separable patterns**

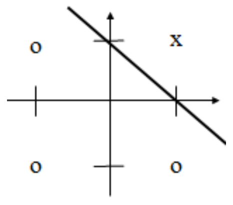
Note : Perceptron cannot find weights for classification problems that are not linearly separable.

- Examples of linearly separable classes
  - Logical **AND** function

    patterns (bipolar) decision boundary

x1	x2	y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

$w1 = 1$
$w2 = 1$
$b = -1$
$q = 0$
$-1 + x1 + x2 = 0$

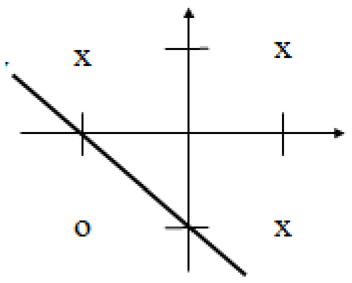


x: class I (y = 1)
o: class II (y = -1)

- Logical **OR** function

patterns (bipolar) decision boundary

x1	x2	y	
-1	-1	-1	w1 = 1
-1	1	1	w2 = 1
1	-1	1	b = 1
1	1	1	q = 0
			1 + x1 + x2 = 0



x: class I (y = 1)
o: class II (y = -1)

- Examples of linearly inseparable classes

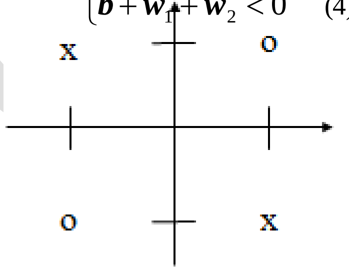
  - Logical **XOR** (exclusive OR) function patterns (bipolar) decision boundary

x1	x2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

No line can separate these two classes, as can be seen from the fact that the following linear inequality system has no solution

$$\begin{cases} b - w_1 - w_2 < 0 & (1) \\ b - w_1 + w_2 \geq 0 & (2) \\ b + w_1 - w_2 \geq 0 & (3) \\ b + w_1 + w_2 < 0 & (4) \end{cases}$$

because we have b < 0 from (1) + (4), and b >= 0 from (2) + (3), which is a contradiction



x: class I (y = 1)
o: class II (y = -1)

11. Write a short note on Competitive learning with its limitations.

(nov 2011, may 2012)

A basic **competitive learning** network has one layer of input neurons and one layer of output neurons. An input pattern **x** is a sample point in the *n*-dimensional real or binary vector space. Binary-valued (1 or 0) *local representations* are more often used for the output nodes. That is, there are as many output neurons as the number of classes and each output node represents a pattern category.

A competitive learning network comprises the feedforward excitatory network(s) and the lateral inhibitory network(s). The feedforward network usually implements an excitatory **Hebbian learning rule**. It consist of when an input cell persistently participates in firing an output cell , the input cell's influence firing that output cell is increased. The lateral competitive network is inhibitory in nature. The network serves the important role of selecting the winner, often via a competitive learning process, highlighting the **"winner-take-all"** schema. Competitive learning is lacking in the capability to add modern clusters whenever deemed necessary.
Competitive learning does not guarantee stability in forming clusters. If the learning rate η is constant, so the winning unit that responds to a pattern may well continue altering during training.
If the learning rate η is minimizing with time, it might become too small to update cluster centers when new data of differentprobability are presented.

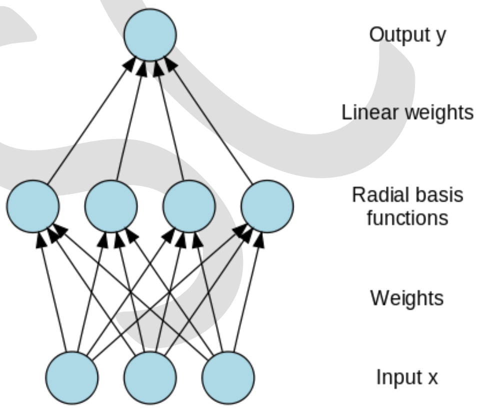
12. Write a short note on Radial Bias Functions. (nov 2011)

In the field of mathematical modeling, a radial basis function network is an artificial neural network that uses radial basis functions as activation functions. The output of the network is a linear combination of radial basis functions of the inputs and neuron parameters. Radial basis function networks have many uses, including function approximation, time series prediction, classification, and system control.

Radial basis function (RBF) networks typically have three layers: an input layer, a hidden layer with a non-linear RBF activation function and a linear output layer. The input can be modeled as a vector of real numbers $\mathbf{x} \in \mathbb{R}^n$. The output of the network is then a scalar function of the input vector, $\varphi : \mathbb{R}^n \to \mathbb{R}$, and is given by

$$\varphi(\mathbf{x}) = \sum_{i=1}^{N} a_i \rho(||\mathbf{x} - \mathbf{c}_i||)$$

where *N* is the number of neurons in the hidden layer, $\mathbf{c}_i$ is the center vector for neuron *i*, and a_i is the weight of neuron *i* in the linear output neuron.



13. Compare: Supervised Vs. Unsupervised Learning (nov 2011, dec 2012)

In **supervised** systems, the data as presented to a machine learning algorithm is fully labelled. That means: all examples are presented with a classification that the machine is meant to reproduce. For this, a classifier is learned from the data, the process of assigning labels to yet unseen instances is called classifi- cation.

Unsupervised systems are not provided any training examples at all and conduct clustering. This is the division of data instances into several groups. The results of clustering algorithms are data driven, hence more 'natural' and better suited to the underlying structure of the data. This advantage is also its major drawback: without a possibility to tell the machine what to do (like in classification), it is difficult to judge the quality of clustering results in a conclusive way. But the absence of training example preparation makes the unsupervised paradigm very appealing.

With unsupervised learning it is possible to learn larger and more complex models than with supervised learning. This is because in supervised learning one is trying to find the connection between two sets of observations. The difficulty of the learning task increases exponentially in the number of steps between the two sets and that is why supervised learning cannot, in practice, learn models with deep hierarchies.

In unsupervised learning, the learning can proceed hierarchically from the observations into ever more abstract levels of representation. Each additional hierarchy needs to learn only one step and therefore the learning time increases (approximately) linearly in the number of levels in the model hierarchy.

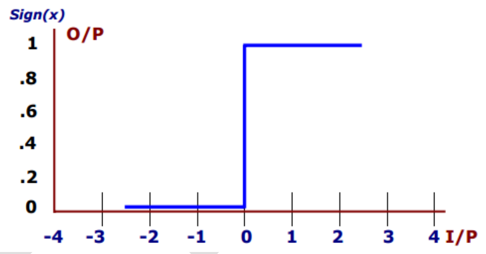
14. Explain the following terms related to NN: (nov 2011, may 2102)

a. Threshold

The Function **y= f(x)** describes a relationship, an input-output mapping, from **x** to **y**.

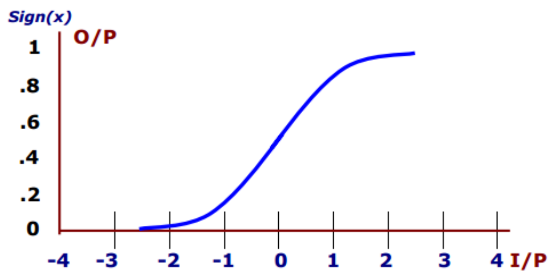
- **Threshold or Sign function :** **sgn(x)** defined as

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$



- **Threshold or Sign function :** **sigmoid(x)** defined as a smoothed (differentiable) form of the threshold function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



b. Learning Rate

Training parameter that controls the size of weight and bias changes during learning.

c. Bias

Weight values associated with individual nodes are also known as biases.

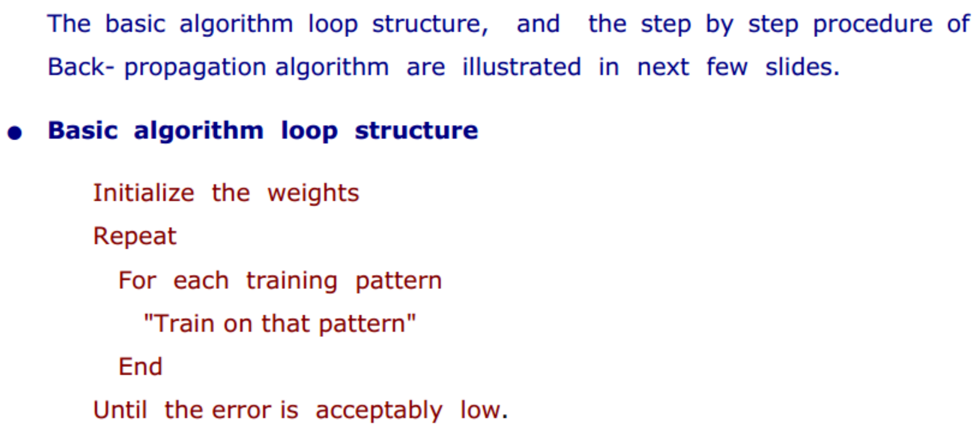
d. Activation function

rule, which governs the manner in which an output node maps input values to output values, is known as an activation function (meaning that this function is used to determine the activation of the output node). Each unit computes a simple function of its input values, which are the weighted outputs from other units. If there are n inputs to a unit, then the unit's output, or **activation** is defined by $a = g((w1 * x1) + (w2 * x2) + ... + (wn * xn))$. Thus each unit computes a (simple) function g of the linear combination of its inputs.

e. Delta Rule

The Delta Rule uses the difference between target activation (i.e., target output values) and obtained activation to drive learning.

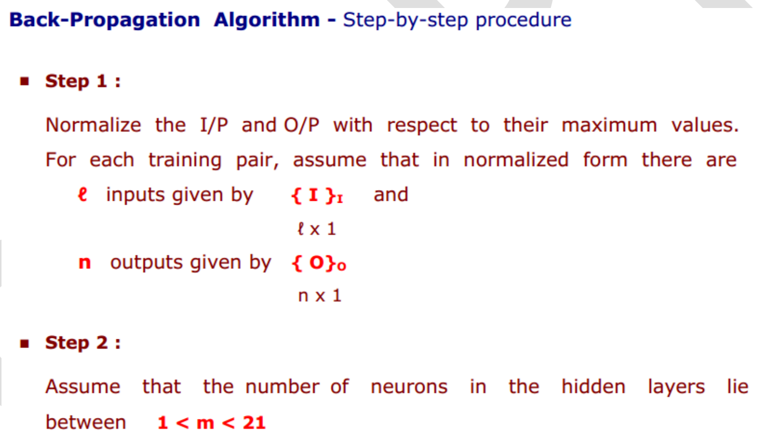
15. One problem of NN (using Backpropagation Algorithm) (nov 2011, may 2012)



The basic algorithm loop structure, and the step by step procedure of Back- propagation algorithm are illustrated in next few slides.

- **Basic algorithm loop structure**

```
Initialize the weights
Repeat
  For each training pattern
    "Train on that pattern"
  End
Until the error is acceptably low.
```



Back-Propagation Algorithm - Step-by-step procedure

- **Step 1 :**

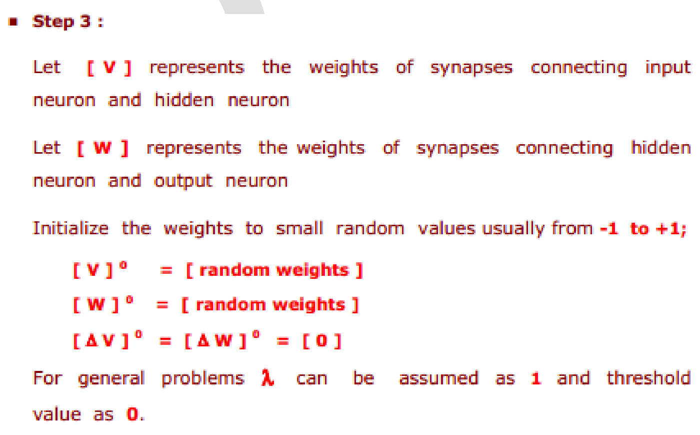
  Normalize the I/P and O/P with respect to their maximum values. For each training pair, assume that in normalized form there are

  ℓ inputs given by $\{ I \}_{I}$ and $\ell \times 1$

  n outputs given by $\{ O \}_{O}$ $n \times 1$

- **Step 2 :**

  Assume that the number of neurons in the hidden layers lie between **$1 < m < 21$**



- **Step 3 :**

  Let **[V]** represents the weights of synapses connecting input neuron and hidden neuron

  Let **[W]** represents the weights of synapses connecting hidden neuron and output neuron

  Initialize the weights to small random values usually from **-1 to +1;**

  $[V]^{0}$ = **[random weights]**

  $[W]^{0}$ = **[random weights]**

  $[\Delta V]^{0} = [\Delta W]^{0} = [0]$

  For general problems **λ** can be assumed as **1** and threshold value as **0**.

- **Step 4 :**

  For training data, we need to present one set of inputs and outputs. Present the pattern as inputs to the input layer $\{ I \}_I$.
  then by using linear activation function, the output of the input layer may be evaluated as

  $$\underset{\ell \times 1}{\{ O \}_I} = \underset{\ell \times 1}{\{ I \}_I}$$

- **Step 5 :**

  Compute the inputs to the hidden layers by multiplying corresponding weights of synapses as

  $$\underset{m \times 1}{\{ I \}_H} = \underset{m \times \ell}{[V]^T} \; \underset{\ell \times 1}{\{ O \}_I}$$

- **Step 6 :**

  Let the hidden layer units, evaluate the output using the sigmoidal function as

  $$\{ O \}_H = \underset{m \times 1}{\begin{Bmatrix} - \\ - \\ \dfrac{1}{(1 + e^{-(I_{Hi})})} \\ - \\ - \end{Bmatrix}}$$

- **Step 7 :**

  Compute the inputs to the output layers by multiplying corresponding weights of synapses as

  $$\underset{n \times 1}{\{ I \}_O} = \underset{n \times m}{[W]^T} \; \underset{m \times 1}{\{ O \}_H}$$

- **Step 8 :**

  Let the output layer units, evaluate the output using sigmoidal function as

  $$\{ O \}_O = \begin{Bmatrix} - \\ - \\ \dfrac{1}{(1 + e^{-(I_{Oj})})} \\ - \\ - \end{Bmatrix}$$

  Note : This output is the network output

- **Step 9 :**

  Calculate the error using the difference between the network output and the desired output as for the **j** th training set as

$$E^P = \frac{\sqrt{\sum (T_j - O_{oj})^2}}{n}$$

- **Step 10 :**

  Find a term **{ d }** as

$$\{ d \} = \left\{ \begin{array}{c} - \\ - \\ (T_k - O_{ok})\, O_{ok}\, (1 - O_{ok}) \\ - \\ - \end{array} \right\}_{n \times 1}$$

- **Step 11 :**

  Find **[Y]** matrix as

$$\underset{m \times n}{[\, Y \,]} = \underset{m \times 1}{\{ O \}_H} \; \underset{1 \times n}{\langle d \rangle}$$

- **Step 12 :**

  Find $$\underset{m \times n}{[\, \Delta W \,]^{t+1}} = \alpha \underset{m \times n}{[\, \Delta W \,]^{t}} + \eta \underset{m \times n}{[\, Y \,]}$$

- **Step 13 :**

  Find $$\underset{m \times 1}{\{ e \}} = \underset{m \times n}{[\, W \,]} \; \underset{n \times 1}{\{ d \}}$$

$$\{ d^* \} = \left\{ \begin{array}{c} - \\ - \\ e_i \; (O_{Hi})\,(1 - O_{Hi}) \\ - \\ - \end{array} \right\}_{m \times 1 \;\; m \times 1}$$

  Find **[X]** matrix as

$$\underset{1 \times m}{[\, X \,]} = \underset{\ell \times 1}{\{ O \}_I} \; \underset{1 \times m}{\langle d^* \rangle} = \underset{\ell \times 1}{\{ I \}_I} \; \underset{1 \times m}{\langle d^* \rangle}$$

- **Step 14 :**

  Find $$\underset{1 \times m}{[\Delta V]^{t+1}} = \alpha \underset{1 \times m}{[\Delta V]^{t}} + \eta \underset{1 \times m}{[X]}$$

- **Step 15 :**

  Find $$[V]^{t+1} = [V]^{t} + [\Delta V]^{t+1}$$
  $$[W]^{t+1} = [W]^{t} + [\Delta W]^{t+1}$$

- **Step 16 :**

  Find error rate as
  $$\text{error rate} = \frac{\sum E_p}{\text{nset}}$$

- **Step 17 :**

  Repeat steps 4 to 16 until the convergence in the error rate is less than the tolerance value

- **End of Algorithm**

  Note : The implementation of this algorithm, step-by-step 1 to 17, assuming one example for training BackProp Network is illustrated in the next section.

- **Problem :**

  Consider a typical problem where there are 5 training sets.

Table : Training sets

S. No.	Input I_1	Input I_2	Output O
1	0.4	-0.7	0.1
2	0.3	-0.5	0.05
3	0.6	0.1	0.3
4	0.2	0.4	0.25
5	0.1	-0.2	0.12

In this problem,

- there are two inputs and one output.
- the values lie between **-1** and **+1** i.e., no need to normalize the values.
- assume two neurons in the hidden layers.
- the NN architecture is shown in the Fig. below.

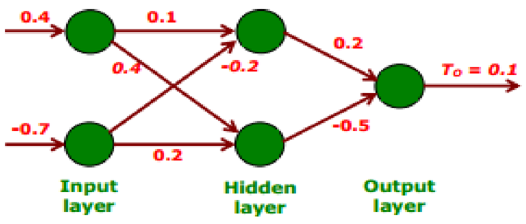


Fig. Multi layer feed forward neural network (MFNN) architecture with data of the first training set

The solution to problem are stated step-by-step in the subsequent slides.

- **Step 1 :** Input the first training set data *(ref eq. of step 1)*

$$\underset{\ell \times 1}{\{ O \}_I} = \underset{\ell \times 1}{\{ I \}_I} = \begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix}_{2 \times 1}$$

from training set s.no 1

- **Step 2 :** Initialize the weights as *(ref eq. of step 3 & Fig)*

$$[V]^0 = \begin{Bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{Bmatrix}_{2x2} \; ; \quad [W]^0 = \begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix}_{2 \times 1}$$

from fig initialization *from fig initialization*

- **Step 3 :** Find $\{ I \}_H = [V]^T \{ O \}_I$ as *(ref eq. of step 5)*

$$\{ I \}_H = \begin{Bmatrix} 0.1 & -0.2 \\ -0.4 & 0.2 \end{Bmatrix} \begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix} = \begin{Bmatrix} 0.18 \\ 0.02 \end{Bmatrix}$$

Values from step 1 & 2

- **Step 4 :** *(ref eq. of step 6)*

$$\{ O \}_H = \begin{Bmatrix} \dfrac{1}{(1 + e^{-(0.18)})} \\ \\ \dfrac{1}{(1 + e^{-(0.02)})} \end{Bmatrix} = \begin{Bmatrix} 0.5448 \\ 0.505 \end{Bmatrix}$$

Values from step 3 values

- **Step 5 :** *(ref eq. of step 7)*

$$\{ I \}_O = [W]^T \{ O \}_H = (0.2 \;\; - 0.5) \begin{Bmatrix} 0.5448 \\ 0.505 \end{Bmatrix} = - 0.14354$$

Values from step 2 , from step 4

- **Step 6 :** *(ref eq. of step 8)*

$$\{ O \}_O = \begin{Bmatrix} \dfrac{1}{(1 + e^{-(0.14354)})} \end{Bmatrix} = 0.4642$$

Values from step 5

- **Step 7 :** *(ref eq. of step 9)*

$$\text{Error} = (T_O - O_{O1})^2 = (0.1 - 0.4642)^2 = 0.13264$$

table first training set o/p *from step 6*

- **Step 8 :** *(ref eq. of step 10)*

$$d = (T_O - O_{O1})(O_{O1})(1 - O_{O1})$$
$$= \underbrace{(0.1}_{\text{Training o/p}} - \underbrace{0.4642)(0.4642)(0.5358)}_{\text{all from step 6}} = -0.09058$$

(ref eq. of step 11)

$$[Y] = \{O\}_H (d) = \underbrace{\begin{Bmatrix} 0.5448 \\ 0.505 \end{Bmatrix}}_{\text{from values at step 4}} \underbrace{(-0.09058)}_{\text{from values at step 8 above}} = \begin{Bmatrix} -0.0493 \\ -0.0457 \end{Bmatrix}$$

- **Step 9 :** *(ref eq. of step 12)*

$$[\Delta W]^1 = \alpha [\Delta W]^0 + \eta [Y] \quad \text{assume } \eta = 0.6$$
$$= \begin{Bmatrix} -0.02958 \\ -0.02742 \end{Bmatrix}$$

from values at step 2 & step 8 above

- **Step 10 :** *(ref eq. of step 13)*

$$\{e\} = [W]\{d\} = \underbrace{\begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix}}_{\text{from values at step 2}} \underbrace{(-0.09058)}_{\text{from values at step 8 above}} = \begin{Bmatrix} -0.018116 \\ -0.04529 \end{Bmatrix}$$

- **Step 11 :** *(ref eq. of step 13)*

$$\{d^*\} = \begin{Bmatrix} (-0.018116)(0.5448)(1 - 0.5448) \\ (0.04529)(0.505)(1 - 0.505) \end{Bmatrix} = \begin{Bmatrix} -0.00449 \\ -0.01132 \end{Bmatrix}$$

from values at step 10 *at step 4* *at step 8*

- **Step 12 :** *(ref eq. of step 13)*

$$[X] = \{O\}_I (d^*) = \underbrace{\begin{Bmatrix} 0.4 \\ -0.7 \end{Bmatrix}}_{\text{from values at step 1}} \underbrace{(-0.00449 \quad 0.01132)}_{\text{from values at step 11 above}}$$

$$= \begin{Bmatrix} -0.001796 & 0.004528 \\ 0.003143 & -0.007924 \end{Bmatrix}$$

- **Step 13 :** *(ref eq. of step 14)*

$$[\Delta V]^1 = \alpha [\Delta V]^0 + \eta [X] = \begin{Bmatrix} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{Bmatrix}$$

from values at step 2 & step 8 above

- **Step 14 :** *(ref eq. of step 15)*

$$[\,V\,]^1 = \underbrace{\begin{Bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{Bmatrix}}_{\text{from values at step 2}} + \underbrace{\begin{Bmatrix} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{Bmatrix}}_{\text{from values at step 13}}$$

$$= \begin{Bmatrix} -0.0989 & 0.04027 \\ 0.1981 & -0.19524 \end{Bmatrix}$$

$$[\,W\,]^1 = \underbrace{\begin{Bmatrix} 0.2 \\ -0.5 \end{Bmatrix}}_{\text{from values at step 2,}} + \underbrace{\begin{Bmatrix} -0.02958 \\ -0.02742 \end{Bmatrix}}_{\text{from values at step 9}} = \begin{Bmatrix} 0.17042 \\ -0.52742 \end{Bmatrix}$$

- **Step 15 :**

  With the updated weights [V] and [W] , error is calculated again and next training set is taken and the error will then get adjusted.

- **Step 16 :**

  Iterations are carried out till we get the error less than the tolerance.

- **Step 17 :**

  Once the weights are adjusted the network is ready for inferencing new objects .

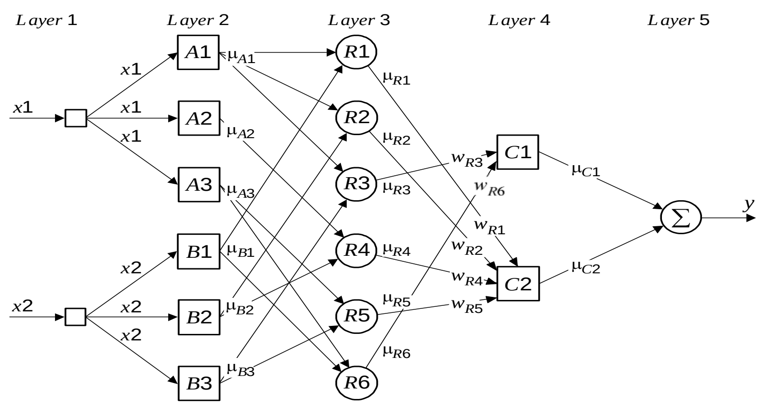
16. Explain ANFIS Architecture in detail. (may 2012, dec 2012)

Adaptive neuro fuzzy inference system (**ANFIS**) is a kind of neural network that is based on Takagi–Sugeno fuzzy inference system. Since it integrates both neural networks and fuzzy logicprinciples, 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.[1] Hence, ANFIS is considered to be a universal estimator.[2]

ANFIS: Artificial Neuro-Fuzzy Inference Systems

- ANFIS are a class of adaptive networks that are funcionally equivalent to fuzzy inference systems.
- ANFIS represent Sugeno e Tsukamoto fuzzy models.

ANFIS Architecture



Each layer in the neuro-fuzzy system is associated with a particular step in the fuzzy inference process.
***Layer* 1** is the **input layer**. Each neuron in this layer transmits external crisp signals directly to the next layer. That is,

$$y_i^{(1)} = x_i^{(1)}$$

***Layer* 2** is the **fuzzification layer**. Neurons in this layer represent fuzzy sets used in the antecedents of fuzzy rules. A fuzzification neuron receives a crisp input and determines the degree to whic this input belongs to the neuron's fuzzy set.
***Layer* 3** is the **fuzzy rule layer**. Each neuron in this layer corresponds to a single fuzzy rule. A fuzzy rule neuron receives inputs from the fuzzification neurons that represent fuzzy sets in the rule antecedents. For instance, neuron *R*1, which corresponds to *Rule* 1, receives inputs from neurons *A*1 and *B*1.
***Layer* 4** is the **output membership layer**. Neurons in this layer represent fuzzy sets used in the consequent of fuzzy rules. An output membership neuron combines all its inputs by using the fuzzy operation **union**. This operation can be implemented by the **probabilistic OR**.
***Layer* 5** is the **defuzzification layer**. Each neuron in this layer represents a single output of the neuro-fuzzy system. It takes the output fuzzy sets clipped by the respective integrated firing strengths and combines them into a single fuzzy set.
Neuro-fuzzy systems can apply standard defuzzification methods
We will use the **sum-product composition** method.
The sum-product composition calculates the crisp output as the weighted average of the centroids of all output membership functions. For example, the weighted average of the centroids of the clipped fuzzy sets $C1$ and $C2$ is calculated as,

$$y = \frac{\mu_{C1} \times a_{C1} \times b_{C1} + \mu_{C2} \times a_{C2} \times b_{C2}}{\mu_{C1} \times b_{C1} + \mu_{C2} \times b_{C2}}$$

17. Explain how GA can be used to determine weights in Neural Network. (may 2012)

Genetic algorithms work with population of individual strings.
The steps involved in GAs are:
− each individual string represent a possible solution of the problem considered,
− each individual string is assigned a fitness value,
− high fit individuals participate in reproduction, yields new strings as offspring and they share some features with each parents,
− low fit individuals are kept out from reproduction and so die,
− a whole new population of possible solutions to the problem is generated by selecting high fit individuals from current generation,
− this new generation contains characteristics which are better than their ancestors,
− processing this way after many generation, the entire population inherits the best and fit solution.
However, before a GA is executed :
− a suitable **coding** for the problem is devised,
− a **fitness function** is formulated,
− parents have to be **selected** for reproduction and crossover to generate offspring.

All these aspects of GAs for determining weights of BPN are illustrated in next few slides.

• Coding

Assume a BPN configuration ***ℓ* - *m* – *n*** where

− ***ℓ*** is input , ***m*** is hidden and ***n*** is output neurons.

– number of weights to be determined are ***(ℓ + n) m.***

– each weight (gene) is a real number.

– assume number of digits (gene length) in weight are ***d*** .

– a string **S** represents weight matrices of input-hidden and the hidden-output layers in a linear form arranged as row-major or column-major selected.

– population size is the randomly generated initial population of ***p***chromosomes.

Example :

Consider a BPN configuration ***ℓ - m – n*** where ***ℓ* = 2** is input , ***m* = 2** is hidden and ***n* = 2** is output neuron.

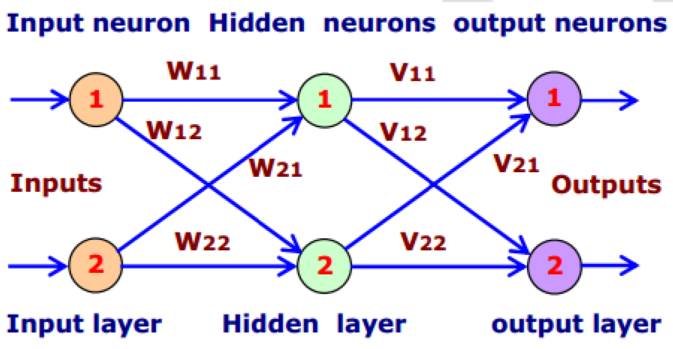


Fig. BPN with 2 – 2 - 2

Input neuron Hidden neurons output neurons

– number of weights is **(ℓ + n) m** = **(2 + 2) . 2 = 8**

– each weight is real number and assume number of digits in weight are **d = 5**

– string **S** representing chromosome of weights is **8 x 5 = 40** in length

– Choose a population size **p = 40** ie choose **40** chromosomes

Gene ← k=0 →	Gene ← k=1 →	Gene ← k=2 →	Gene ← k=3 →	Gene ← k=4 →	Gene ← k=5 →	Gene ← k=6 →	Gene ← k=7 →
84321	46234	78901	32104	42689	63421	46421	87640
←			Chromosome				→
32478	76510	02461	84753	64321	14261	87654	12367
←			Chromosome				→

Fig. Some randomly generated chromosome made of 8 genes representing 8 weights for BPN

• **Weight Extraction**

Extract weights from each chromosomes, later to determine the fitness

values.

Let ***x1* , *x2* , ... *x d* , ... *x L*** represent a chromosome and

Let ***xkd+1* , *xkd+2* , . . *x(k + 1)d*** represent **kth** gene **(k** ≥ **0)** in the chromosomes.

The actual weight ***wk*** is given by

$$w_k = \begin{cases} + \dfrac{x_{kd+2}\,10^{d-2} + x_{kd+3}\,10^{d-3} + \ldots + x_{(k+1)d}}{10^{d-2}}, & \text{if } 5 \le x_{kd+1} \le 9 \\[2ex] - \dfrac{x_{kd+2}\,10^{d-2} + x_{kd+3}\,10^{d-3} + \ldots + x_{(k+1)d}}{10^{d-2}}, & \text{if } 0 \le x_{kd+1} < 5 \end{cases}$$

The Chromosomes are stated in the Fig. The weights extracted from all the eight genes are :

- **Gene 0 : 84321 ,**

  Here we have, $k = 0$, $d = 5$, and x_{kd+1} is x_1 such that $5 \le x_1 = 8 \le 9$. Hence, the weight extracted is

  $$W_0 = + \frac{4 \times 10^3 + 3 \times 10^2 + 2 \times 10 + 1}{10^3} = +4.321$$

- **Gene 1 : 46234 ,**

  Here we have, $k = 1$, $d = 5$, and x_{kd+1} is x_6 such that $0 \le x_6 = 4 \le 5$. Hence, the weight extracted is

  $$W_1 = - \frac{6 \times 10^3 + 2 \times 10^2 + 3 \times 10 + 4}{10^3} = -6.234$$

- Similarly for the remaining genes

Gene 2 : 78901 yields W2 = + 8.901

Gene 3 : 32104 yields W3 = − 2.104

Gene 4 : 42689 yields W4 = − 2.689

Gene 5 : 63421 yields W5 = + 3.421

Gene 6 : 46421 yields W6 = − 6.421

Gene 7 : 87640 yields W7 = + 7.640

18. What are hybrid systems? Discuss advantages, disadvantages and applications of neuro-fuzzy and neuro-genetic hybrid systems. (may 2012)

A **hybrid system** is a dynamic system that exhibits both continuous and discrete dynamic behavior – a system that can both *flow* (described by a differential equation) and *jump* (described by adifference equation or control graph). Often, the term "hybrid dynamic system" is used, to distinguish over hybrid systems such as those that combine neural nets and fuzzy logic, or electrical and mechanical drivelines. A hybrid system has the benefit of encompassing a larger class of systems within its structure, allowing for more flexibility in modelling dynamic phenomena.

In general, the *state* of a hybrid system is defined by the values of the *continuous variables* and a discrete *control mode*. The state changes either continuously, according to a *flow condition*, or discretely according to a *control graph*. Continuous flow is permitted as long as so-called *invariants* hold, while discrete transitions can occur as soon as given *jump conditions* are satisfied. Discrete transition may be associated with *events*.

neuro-fuzzy hybrid systems.

A neuro-fuzzy system is a fuzzy system that uses a learning algorithm derived from or inspired by neural network theory to determine its parameters (fuzzy sets and fuzzy rules) by processing data samples.
The advantages are:

· capacity to represent inherent uncertainties of the human knowledge with linguistic variables;
· simple interaction of the expert of the domain with the engineer designer of the system;
· easy interpretation of the results, because of the natural rules representation;
· easy extension of the base of knowle dge through the addition of new rules;
· robustness in relation of the possible disturbances in the system.

- learning capacity;

· generalization capacity;
· robustness in relation to disturbances

And its disadvantages are:

· incapable to generalize, or either, it only answers to what is written in its rule base;
· not robust in relation the topological changes of the system, such changes would demand alterations in the rule base;
· depends on the existence of a expert to determine the inference logical rules

- impossible interpretation of the functionality;
- difficulty in determining the number of layers and number of neurons.

Integration of GAs and NNs has turned out to be useful.

− Genetically evolved nets have reported comparable results against their conventional counterparts. − The gradient descent learning algorithms have reported difficulties in leaning the topology of the networks whose weights they optimize.

− GA based algorithms have provided encouraging results especially with regard to face recognition, animal control, and others.

− Genetic algorithms encode the parameters of NNs as a string of properties of the network, i.e. chromosomes. A large population of chromosomes representing many possible parameters sets, for the given NN, is generated.

− GA-NN is also known as GANN have the ability to locate the neighborhood of the optimal solution quicker than other conventional search strategies.

− The drawbacks of GANN algorithms are : large amount of memory required to handle and manipulate chromosomes for a given network; the question is whether this problem scales as the size of the networks become large.

19. What is Soft Computing? State difference between Soft computing & Hard computing (Conventional Computing). List Out the soft computing characteristics. (nov 2011)

Soft Computing is term used in computer science to refer the problem in computer science whose solution is not predictable,uncertain and between 0 and 1. **Soft computing** is a term applied to a field within computer science which is characterized by the use of inexact solutions to computationally hard tasks such as the solution of NP-completeproblems, for which there is no known algorithm that can compute an exact solution in polynomial time

soft computing characteristics.

1. Simulation of human expertise
2. Innovative techniques
3. Goal – driven
4. Extensive numerical computations
5. Dealing with partial and incomplete information
6. Fault tolerance

1) **Hard computing**, i.e., conventional computing, requires a precisely stated analytical model and often a lot of computation time. **Soft computing**differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation. In effect, the role model for soft computing is the human mind.
2) **Hard computing** based on binary logic, crisp systems, numerical analysis and crisp software but **soft computing** based on fuzzy logic, neural nets and probabilistic reasoning.
3) **Hard computing** has the characteristics of precision and categoricity and the **soft computing**, approximation and dispositionality. Although in hard computing, imprecision and uncertainty are undesirable properties, in soft computing the tolerance for imprecision and uncertainty is exploited to achieve tractability, lower cost, high Machine Intelligence Quotient (MIQ) and economy of communication
4) **Hard computing** requires programs to be written; **soft computing** can evolve its own programs
5) **Hard computing** uses two-valued logic; **soft computing** can use multivalued or fuzzy logic
6) **Hard computing** is deterministic; **soft computing** incorporates stochasticity
7) **Hard computing** requires exact input data; **soft computing** can deal with ambiguous and noisy data
8) **Hard computing** is strictly sequential; **soft computing** allows parallel computations
9) **Hard computing** produces precise answers; **soft computing** can yield approximate answers

20. Explain the following terms in detail : (dec 2012)

a. Classical Set

In classical set theory, the membership of elements in a set is assessed in binary terms according to a bivalent condition — an element either belongs or does not belong to the set.

b. Fuzzy Sets

Fuzzy sets are sets whose elements have degrees of membership. fuzzy set theory permits the gradual assessment of the membership of elements in a set; this is described with the aid of a membership function valued in the real unit interval [0, 1]. Fuzzy sets generalize classical sets

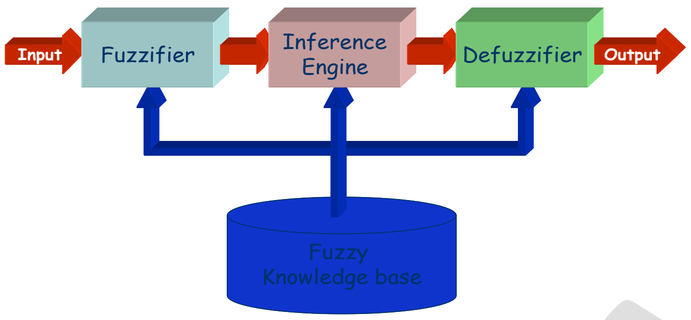
c. Membership Functions

The **membership function** of a fuzzy set is a generalization of the indicator function in classical sets. In fuzzy logic, it represents the degree of truth as an extension of valuation.
For any set X, a membership function on X is any function from X to the real unit interval [0,1].

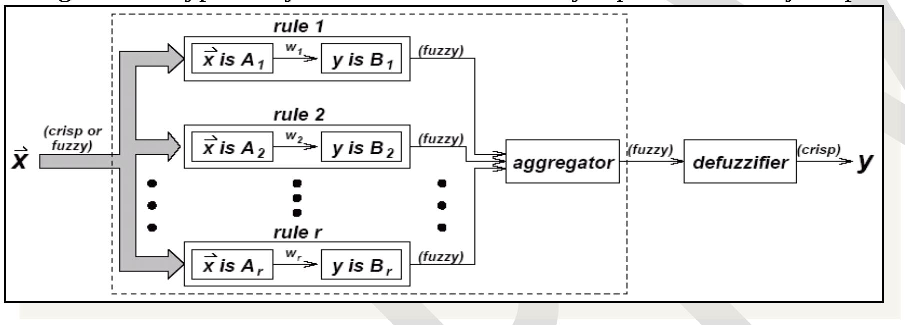
d. Crossover points

The Crossover point of a fuzzy set is the element in U at which its membership function is 0.5.

21. Explain Block diagram of Fuzzy Inference System(FIS). (nov 2011)



Using If-Then type fuzzy rules converts the fuzzy input to the fuzzy output.



Fuzzy inference is the process of formulating the mapping from a given input to an output using fuzzy logic. The mapping then provides a basis from which decisions can be made, or patterns discerned. The process of fuzzy inference involves all of the pieces that are described in Membership Functions, Logical Operations, and If-Then Rules.
This section describes the fuzzy inference process and uses the example of the two-input, one-output, three-rule tipping problem The Basic Tipping Problem that you saw in the introduction in more detail. The basic structure of this example is shown in the following diagram:
Information flows from left to right, from two inputs to a single output. The parallel nature of the rules is one of the more important aspects of fuzzy logic systems. Instead of sharp switching between modes based on breakpoints, logic flows smoothly from regions where the system's behavior is dominated by either one rule or another.

Fuzzy inference process comprises of five parts:

- Fuzzification of the input variables
- Application of the fuzzy operator (AND or OR) in the antecedent
- Implication from the antecedent to the consequent
- Aggregation of the consequents across the rules
- Defuzzification

A fuzzy inference diagram displays all parts of the fuzzy inference process — from fuzzification through defuzzification.

Step 1. Fuzzify Inputs
The first step is to take the inputs and determine the degree to which they belong to each of the appropriate fuzzy sets via membership functions.

Step 2. Apply Fuzzy Operator
After the inputs are fuzzified, you know the degree to which each part of the antecedent is satisfied for each rule. If the antecedent of a given rule has more than one part, the fuzzy operator is applied to obtain one number that represents the result of the antecedent for that rule. This number is then applied to the output function. The input to the fuzzy operator is two or more membership values from fuzzified input variables. The output is a single truth value.

Step 3. Apply Implication Method
Before applying the implication method, you must determine the rule's weight. Every rule has a *weight* (a number between 0 and 1), which is applied to the number given by the antecedent.
Step 4. Aggregate All Outputs
Because decisions are based on the testing of all of the rules in a FIS, the rules must be combined in some manner in order to make a decision. Aggregation is the process by which the fuzzy sets that represent the outputs of each rule are combined into a single fuzzy set. Aggregation only occurs once for each output variable, just prior to the fifth and final step, defuzzification. 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.

Step 5. Defuzzify
The input for the defuzzification process is a fuzzy set (the aggregate output fuzzy set) and the output is a single number. As much as fuzziness helps the rule evaluation during the intermediate steps, the final desired output for each variable is generally a single number. However, the aggregate of a fuzzy set encompasses a range of output values, and so must be defuzzified in order to resolve a single output value from the set.

22. Explain in detail: Mamdani's Fuzzy Model. (nov 2011, may 2012)

Explain in detail: Sugeno's Fuzzy Model and give difference between both of them.

- Mamdani Fuzzy models

Original Goal: Control a steam engine & boiler combination by a set of linguistic control rules obtained from experienced human operators.
The Mamdani-style fuzzy inference process is performed in four steps:
1 fuzzification of the input variables,
2 rule evaluation;
3 aggregation of the rule outputs, and finally
4 defuzzification.

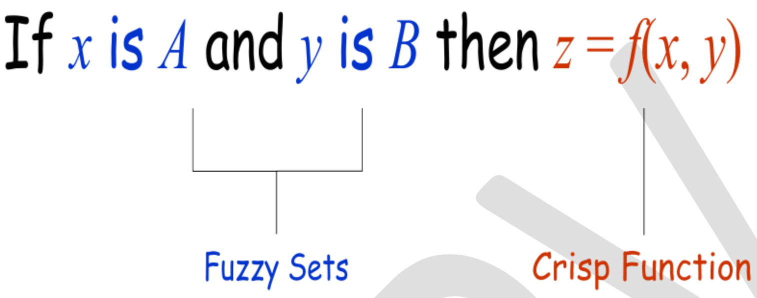
<u>Step 1: Fuzzification</u>
The first step is to take the crisp inputs, *x*1 and *y*1(*project funding* and *project staffing*), and determine the degree to which these inputs belong to each of the appropriate fuzzy sets.
<u>Step 2: Rule Evaluation</u>
The second step is to take the fuzzified inputs, $m_{(x=A1)} = 0.5$, $m_{(x=A2)} = 0.2$, $m_{(y=B1)} = 0.1$ and $m_{(y=B\,2)} = 0.7$, and apply them to the antecedents of the fuzzy rules. If a given fuzzy rule has multiple antecedents, the fuzzy operator (AND or OR) is used to obtain a single number that represents the result of the antecedent evaluation. This number (the truth value) is then applied to the consequent membership function.
<u>Step 3: Aggregation of the rule outputs</u>
Aggregation is the process of unification of the outputs of all rules. We take the membership functions of all rule consequents previously clipped or scaled and combine them into a single fuzzy set.
The input of the aggregation process is the list of clipped or scaled consequent membership functions, and the output is one fuzzy set for each output variable.

Step 4: Defuzzification

The last step in the fuzzy inference process is defuzzification. Fuzziness helps us to evaluate the rules, but the final output of a fuzzy system has to be a crisp number. The input for the defuzzification process is the aggregate output fuzzy set and the output is a single number.

- Sugeno's Fuzzy Model

Goal: Generation of fuzzy rules from a given input-output data set.



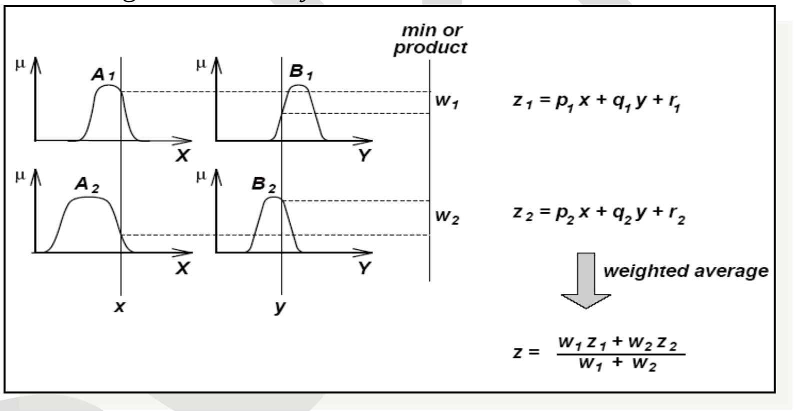
Examples

R1: if X is small and Y is small then $z = -x + y + 1$

R2: if X is small and Y is large then $z = -y + 3$

R3: if X is large and Y is small then $z = -x + 3$

R4: if X is large and Y is large then $z = x + y + 2$

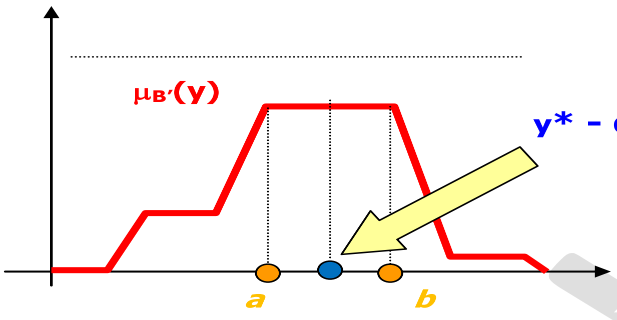


Mamdani method is widely accepted for capturing expert knowledge. It allows us to describe the expertise in more intuitive, more human-like manner. However,
Mamdani-type FIS entails a substantial computational burden. On the other hand, Sugeno method is computationally efficient and works well with optimization
and adaptive techniques, which makes it very attractive in control problems, particularly for dynamic non linear systems. These adaptive techniques can be used to customize the membership functions so that fuzzy system best models the data. The most fundamental difference between Mamdani-type FIS and Sugeno-type FIS is the way the crisp output is generated from the fuzzy inputs. While Mamdani-type FIS
uses the technique of defuzzification of a fuzzy output, Sugeno-type FIS uses weighted average to compute the crisp output.

23. What is defuzzification? Describe different methods of defuzzification.

(may 2012, dec 2012)

Suppose the *inferred fuzzy output* ***y is B'*** is having the aggregated membership function $\mu_{B'}(y)$.**Defuzzification** is **conversion of fuzzy output** (possibility distribution of the output $\mu_{B'}(y)$) **to precise (crisp) value *y****



How to *defuzzify*?

One of criteria is plausibility: *y** should lie approximately in the *middle of the support region of B'(y)* and have a *high degree of membership in B'(y)*

- Among the major defuzzification techniques we can mention:
  a) Mean of Maximum (Middle of Maxima) method (MoM)
  b) Center of Gravity (Centroid, Center of Area) method (CoG)
- **MoM** calculates the average of all variable values having maximum membership degree.
- **CoG** defuzzified output is calculated as **weighted average** over the whole universe Y of output: where $\mu_{B'}(y)$ is aggregated membership function of the *inferred fuzzy output.*

24. Explain the following terms in detail with example: (may 2012, dec 2012)

a. Rough set

The rough set methodology is based on the premise that **lowering the degree of precision in the data** makes the data pattern more visible.

b. Upper approximation

The elements that doubtlessly belong to the set comes under upper approximation

$$\overline{\mathrm{BX}} = \{x_i \in U \mid [x_i]_{\mathrm{Ind}(B)} \cap X \neq 0\}.$$

c. Lower approximation

The elements that possibly belong to the set comes under lower approximation

$$\underline{\mathrm{BX}} = \{x_i \in U \mid [x_i]_{\mathrm{Ind}(B)} \subset X\}.$$

25. What is Machine Learning? Explain Inductive Concept learning with suitable example. (dec 2012)

Machine learning, a branch of artificial intelligence, concerns the construction and study of systems that can learn from data. For example, a machine learning system could be trained on email messages to learn to distinguish between spam and non-spam messages. After learning, it can then be used to classify new email messages into spam and non-spam folders.

The core of machine learning deals with representation and generalization. Representation of data instances and functions evaluated on these instances are part of all machine learning systems. Generalization is the property that the system will perform well on unseen data instances; the conditions under which this can be guaranteed are a key object of study in the subfield of computational learning theory.

In contrast with the deductive method, inductive instruction makes use of student “noticing”. Instead of explaining a given concept and following this explanation with examples, the teacher presents students with many examples showing how the concept is used. The intent is for students to “notice”, by way of the examples, how the concept works.

Using the grammar situation from above, the teacher would present the students with a variety of examples for a given concept without giving any preamble about how the concept is used. As students see how the concept is used, it is hoped that they will notice how the concept is to be used and determine the grammar rule. As a conclusion to the activity, the teacher can ask the students to explain the grammar rule as a final check that they understand the concept.

26. Explain concept formation theory in machine learning. (dec 2012)

Concept learning also refers to a learning task in which a human or machine learner is trained to classify objects by being shown a set of example objects along with their class labels. The learner simplifies what has been observed by condensing it in the form of an example. This simplified version of what has been learned is then applied to future examples. Concept learning may be simple or complex because learning takes place over many areas. When a concept is difficult, it is less likely that the learner will be able to simplify, and therefore will be less likely to learn. Colloquially, the task is known as *learning from examples.* Most theories of concept learning are based on the storage of exemplars and avoid summarization or overt abstraction of any kind.

Concept learning must be distinguished from learning by reciting something from memory (recall) or discriminating between two things that differ (discrimination). However, these issues are closely related, since memory recall of facts could be considered a "trivial" conceptual process where prior exemplars representing the concept are invariant

Discovery - Every baby discovers concepts for itself, such as discovering that each of its fingers can be individually controlled or that care givers are individuals. Although this is perception driven, formation of the concept is more than memorizing perceptions.

Examples - Supervised or unsupervised generalizing from examples may lead to learning a new concept, but concept formation is more than generalizing from examples.

Words - Hearing or reading new words leads to learning new concepts, but forming a new concept is more than learning a dictionary definition. A person may have previously formed a new concept before encountering the word or phrase for it.

Exemplars comparison - Another efficient way to learn new categories and induce new categorization rules is to compare a few objects when their categorical relation is known. For example, comparing two exemplars while being informed that the two are from the same category allows the attributes shared by the category members to be identified, and illustrates the variability permitted within this category. On the other hand, comparing two exemplars while being informed that the two are from different categories may allow an identification of attributes which has diagnostic value. Interestingly, within a category and between categories comparisons are not always similarly useful for category learning, and the capacity to use either one of these two forms of learning by comparison is subject to change during early childhood (Hammer et al., 2009).

Invention - When prehistoric people who lacked tools used their fingernails to scrape food from killed animals or smashed melons, they noticed that a broken stone sometimes had a sharp edge like a fingernail and was therefore suitable for scraping food. Inventing a stone tool to avoid broken fingernails was a new concept.

27. Explain sequence prediction in machine learning. (dec 2012)

Most passive Machine Learning tasks can be (re)stated as sequence prediction problems. This includes pattern recognition, classification, time-series forecasting, and others. Moreover, the understanding of passive intelligence also serves as a basis for active learning and decision making.

28. Discuss learning by observation and learning by analogy with respect to machine learning. (dec 2012)

Learning by ANALOGY
this invovles a benevolent teacher who gives classified examples to the leaner. The learner performs some generalisation the examples to infer new knowledge. Previous knowledge maybe used to steer the generalisations. In analogy the learner performs the generalisation based on some previously learnt situation.

Learning by OBSERVATION (self-taught)
this is similar to the category to Learning from Examples but without classification by teacher - the learner uses pre-learned information to help classify observation (eg “conceptual clustering”)

- Conceptual clustering is a machine learning paradigm for unsupervised classification.
- It is distinguished from ordinary data clustering by generating a concept description for each generated class.
- Most conceptual clustering methods are capable of generating hierarchical category structures.
- Conceptual clustering is closely related to formal concept analysis, decision tree learning.

29. Elaborate printed character recognition as an application of computational intelligence. (dec 2012)

The recognition system consists of two main processing units — a character separator and an isolated character classifier.
Character separation (frequently called segmentation) can work in two modes:

- fixed (constrained) spacing mode (where character size is known in advance and therefore segmentation can be very robust)
- variable (arbitrary) spacing (where no a priori information can be assumed)

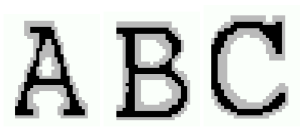
Hence, our demo consists of three parts: recognition of isolated characters, work with constrained segmentation, and work with unconstrained segmentation.

Isolated Character Classifier

The recognition module gets on input an extracted and size-normalized image representing a character to be recognized.
The module produces on output an ordered list of a few of the most probable classification candidates, together with their confidence values.

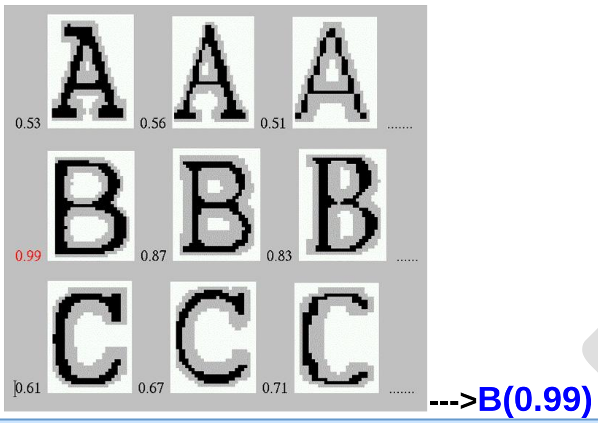
The task is perfomed by matching the raster sample with template masks representing different characters. The masks are prepared by an off-line training phase. A mask can be considered as a raster image containing three types of pixels: black, white, and undefined (gray).

Initially, template masks are built per font. In a single font-set of masks, every character is represented by exactly one mask. Images representing template masks built for Courier font are presented below.



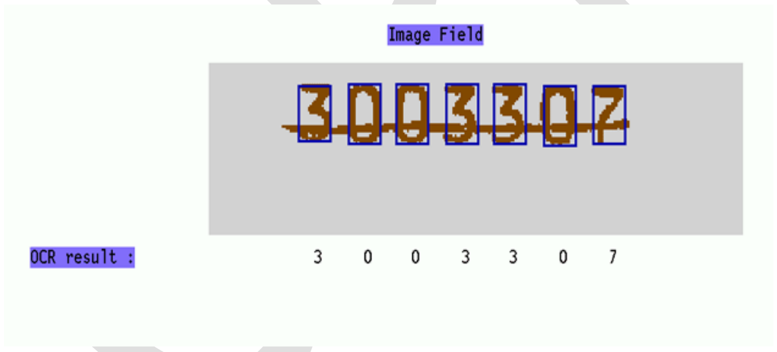
In practice, a font character to be recognized is often unknown a priori. Hence, templates representing the most prevalent fonts are prepared and combined together.

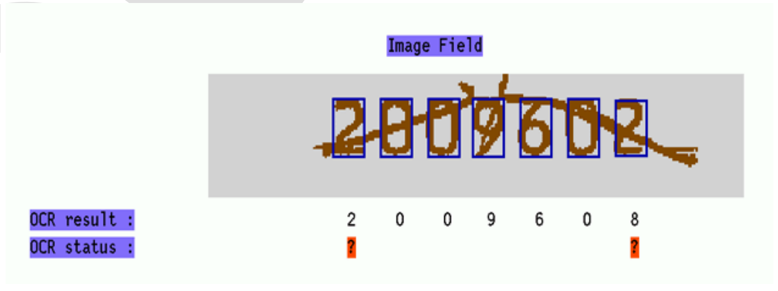
The Omnifont recognizer, containing a number of masks per character, is shown below. An input image is correlated with all the masks stored in the recognizer. The mask which has the highest correlation score is taken to be the primary result of the recognition.



Constrained Printing Recognition

In this case, character spacing is fixed. Hence, segmentation is possible even when fields are distorted, as illustrated below.





Field extracted from medical insurance forms (USA), recognized as Omnifont.

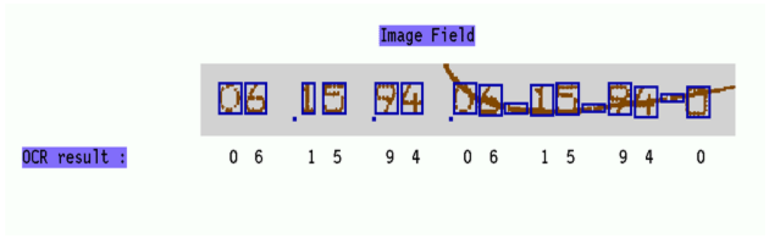


Image Field

OCR result : 0 6 1 5 9 4 0 6 1 5 9 4 0

Credit card numbers impressed through a copy paper on transaction vouchers.

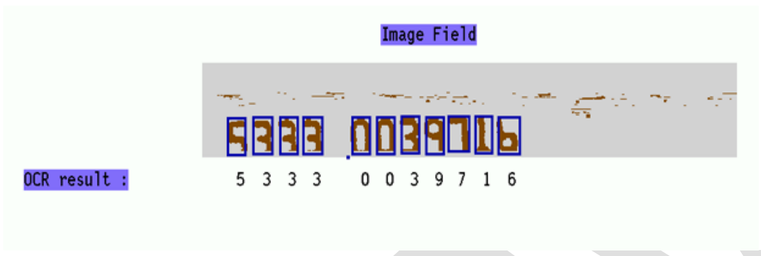


Image Field

OCR result : 5 3 3 3 0 0 3 9 7 1 6

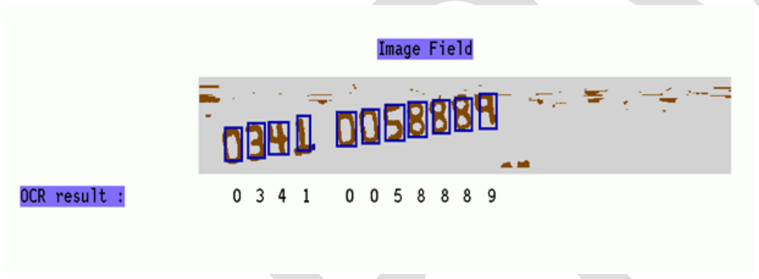


Image Field

OCR result : 0 3 4 1 0 0 5 8 8 8 9