Soft Computing

Prof. Dr. Ahmed T. Sadiq

Computer Science Department

University of Technology

Baghdad, Iraq

Artificial Neural Networks (ANN)

Artificial neural networks are one of the most important techniques of artificial intelligence in the field of machine learning, and perhaps occupies center stage in this field among the different technologies in the field of machine learning, given the size of the areas and applications where this technique was used, the longer the machine learning using this technique the kind of clear (Explicit Learning).

Historical Background

In 1943 two scientists (Warren McCulloch) and (Walter Pitts) invent the first computational model for artificial neural networks relying on math. and algorithms, called (Threshold Logic Model). This model has paved the way to the artificial neural network research, and included two parts, the first focusing on ecological processes in the brain, and the second search in the artificial neural network applications in the field of artificial intelligence.

At the end of the forties of the twentieth century put the world (Donald Hebb) hypotheses based learning mechanical neural plasticity known as (Hebbian Learning). It is a perfect model for learning the rules without supervisor (Unsupervised Learning). These ideas and applied to Computational models of machine (Turing) of the type (B) in 1948.

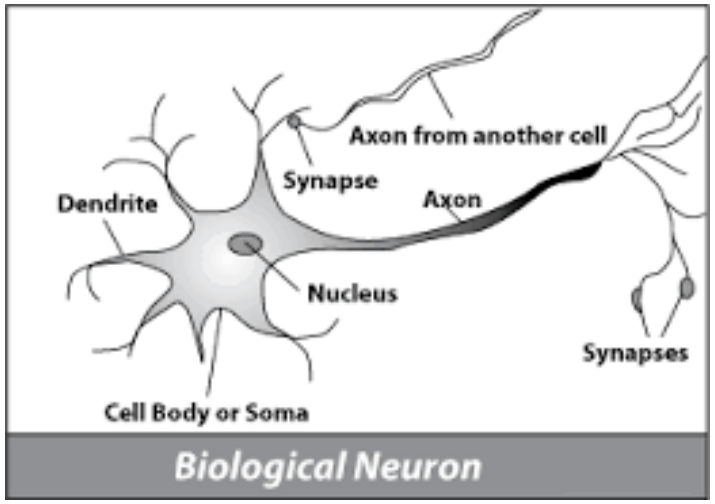
In 1954 for the first time use (Farley) and (Clark) computation machine (ie digital calculator) and through the simulation model (Hebb) at the Massachusetts Institute of Technology (MIT), and there are several scientists worked computation machines neural networks in 1956, such as (Rochester , Holland, Habit, Duda).

Historical Background

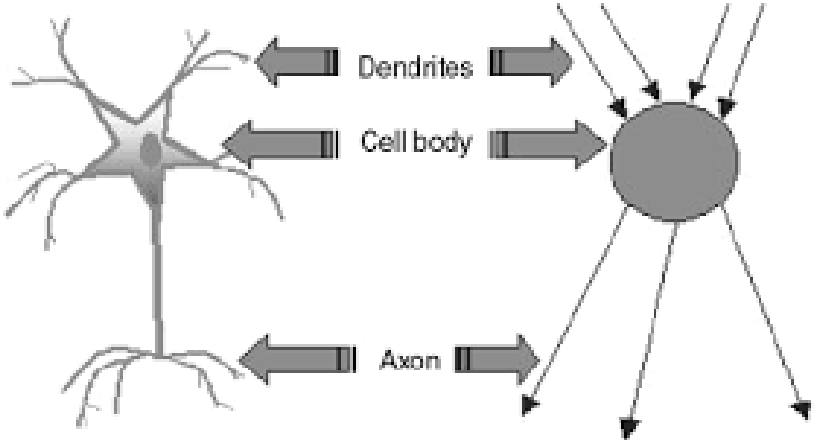
In 1958 the world (Rosenblatt) invented the (Perceptron), a pattern recognition algorithm through a network calculator to learn two-level (Two-Layer) using addition and subtraction. Subsequently developed to include more complicated than the first operations. Then developed the world thereafter (Paul Werbos) to become an algorithm called back propagation algorithm in 1975.

This is worth noting that research on artificial neural networks have experienced a recession between 1960 and 1975 for several reasons, including the models failed to address the issues are not complex algorithms proposed Another reason is the lack of fast computers at that time to deal with such advanced technology needed to execution speed to get the desired learning and these reasons outlined in the report to the worlds (Marvin Minsky) and (Seymor Papert). The period saw the eighties and nineties of the twentieth century, the emergence of many artificial neural network algorithms and helped to steady growth this rapid evolution of computing and increase the memories used, and especially the development that took place in the field of parallel computing processors (Parallel Computers).

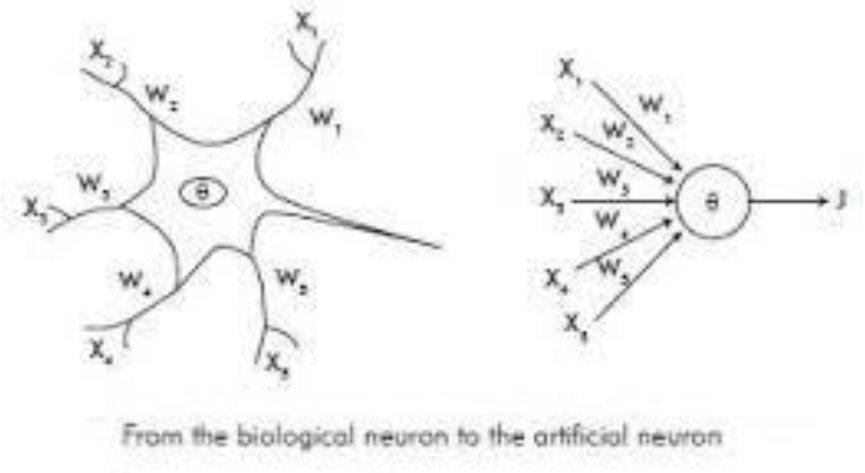
Biological Neuron



Biological to Artificial Neuron



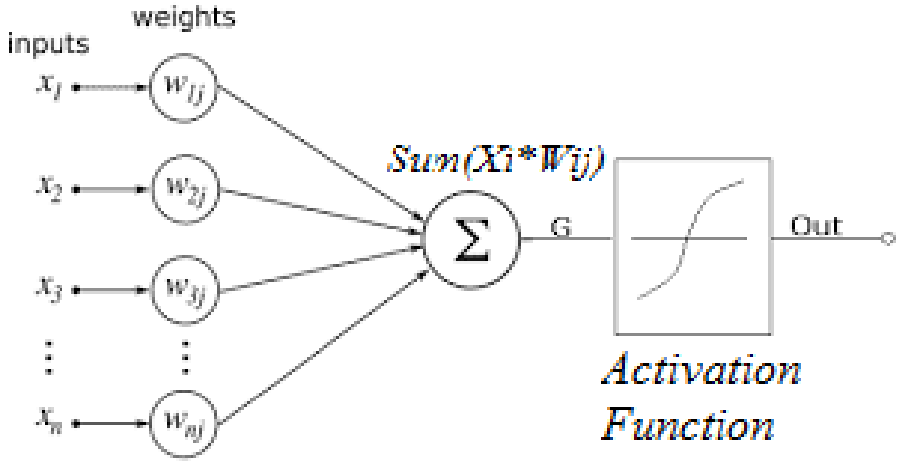
Biological neuron



Artificial neuron

From the biological neuron to the artificial neuron

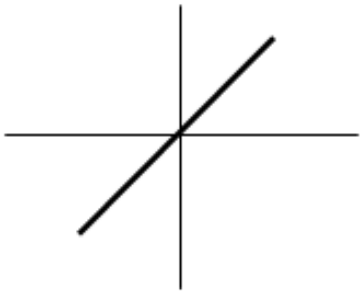
Artificial Neuron



ANN Components:

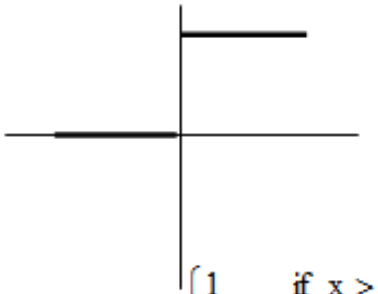
- Inputs (Int., real, binary).
- Output (Int., real, binary).
- Activation function (linear, non-linear).
- Weights (fixed, variable).
- ANN topology.

Activation Function Samples



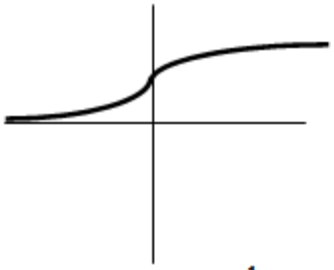
$$f(x) = x$$

Simitric fun.



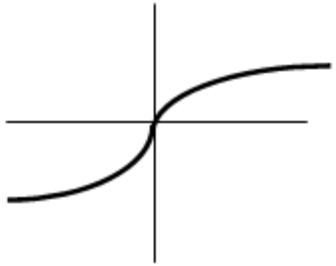
$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \le 0 \end{cases}$$

Binary fun.



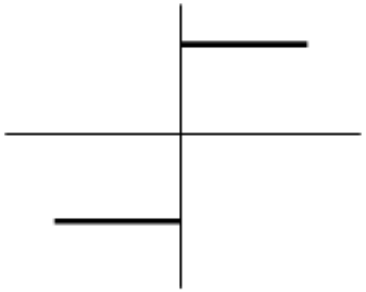
$$f(x) = \frac{1}{1+e^{-x}}$$

Sigmoid fun.



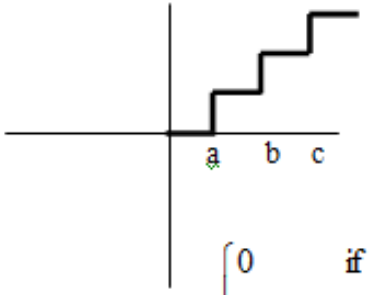
$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Tanh fun.



$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \le 0 \end{cases}$$

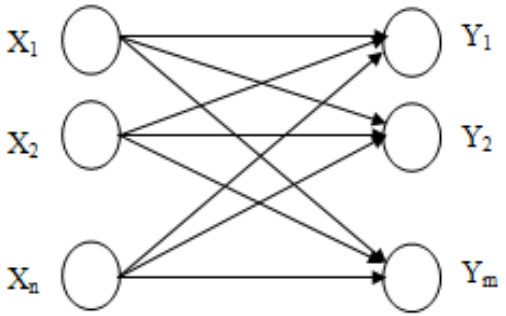
Bipolar Hard Limit fun.



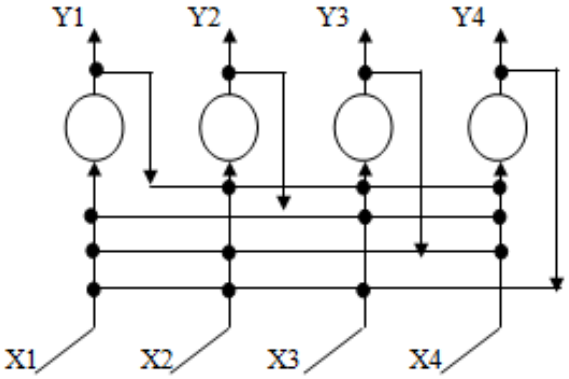
$$f(x) = \begin{cases} 0 & \text{if } 0 \le x < a \\ 1 & \text{if } a \le x < b \\ 2 & \text{if } b \le x < c \\ 3 & \text{if } x \ge c \end{cases}$$

Steps fun.

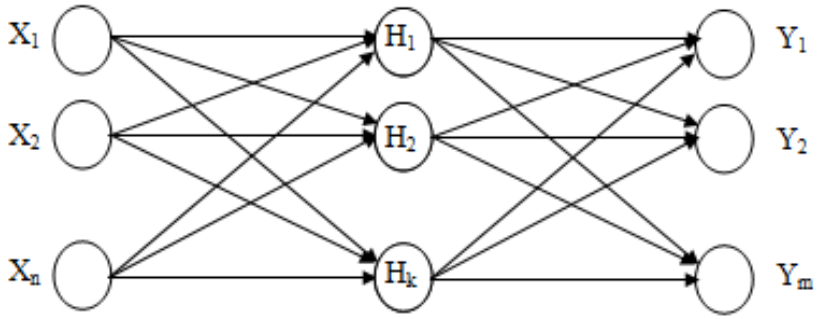
ANN Topology Samples



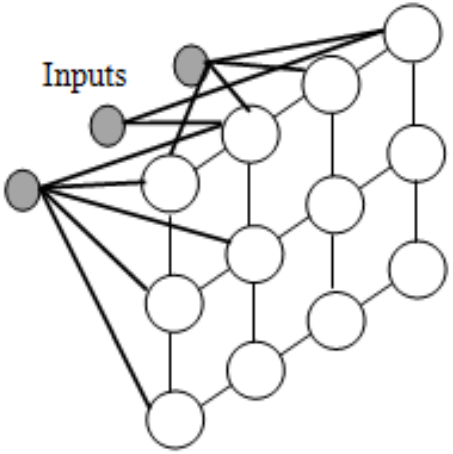
Single Layer NN



Feedback Single Layer NN



Multi-Layer NN



Self-Organization NN

ANN Properties

1. Parallelism or synchronization at work: a fundamental feature of the advantages of artificial neural networks, ie if promised each cell represents a processing unit (Processor Element) itself linked to turn a large number of units similar treatment, they will we increase the processing calculator for orders speed tremendously and this is what you did mainframes, which produced tremendous speed using artificial neural network technology to computers able to record time to solve very complex problems companies.
2. The ability to adapt (Adaptation): It is a very important characteristic of artificial neural networks properties, as they are solving the problem through a particular algorithm rather than through the programming problem is, they adapted themselves to solve the problem through the private data of the problem. In other words, there is no programming to solve the problem but the fact that no application for a specific learning algorithm and its function to cope with the requirements of the problem and adapt.
3. Distributed Memory : When the artificial neural network learns the data problem, it certainly will be storing data or keep it in its own way as any learning base, that is, they become a real problem memory for data broken down by type of artificial neural network. There are types of artificial neural networks are used for storage only serves as a working memory.

4. The ability to generalize (Generalization): If artificial neural network trained on a particular set of models in a particular application, and then were tested this network over other models is that may I trained them and were successful test results of any that artificial neural network introduced new models this means it has circulated the base that came out through her training on the given models, the new models and this is very important characteristic of the properties of artificial neural networks. In other words, it scrapped the concept of rule: "If ... Then ..." through a generalization feature.
5. fault tolerance (Fault Tolerance): As known, the architectural Von Neumann rely on serial execution of operations (Sequential Processing) meaning that errors that occur in the suggestions (steps) program affect what follows from the meta, while in architectural artificial neural networks, that bears the errors that occur in the inputs and correct them or something like that so it will not affect the work of the artificial neural network as well as for software solutions to problems using artificial neural network technology.
6. Definition of the problem: As we know, the basis of the work of artificial neural networks is learning a set of input models or problem situations through certain algorithms, and thus was a paragraph defining the problem exceeds what has to do so in the traditional solve the problem of programming cases, this is very important in many applications because it is difficult to understand what the problem is a mechanism.

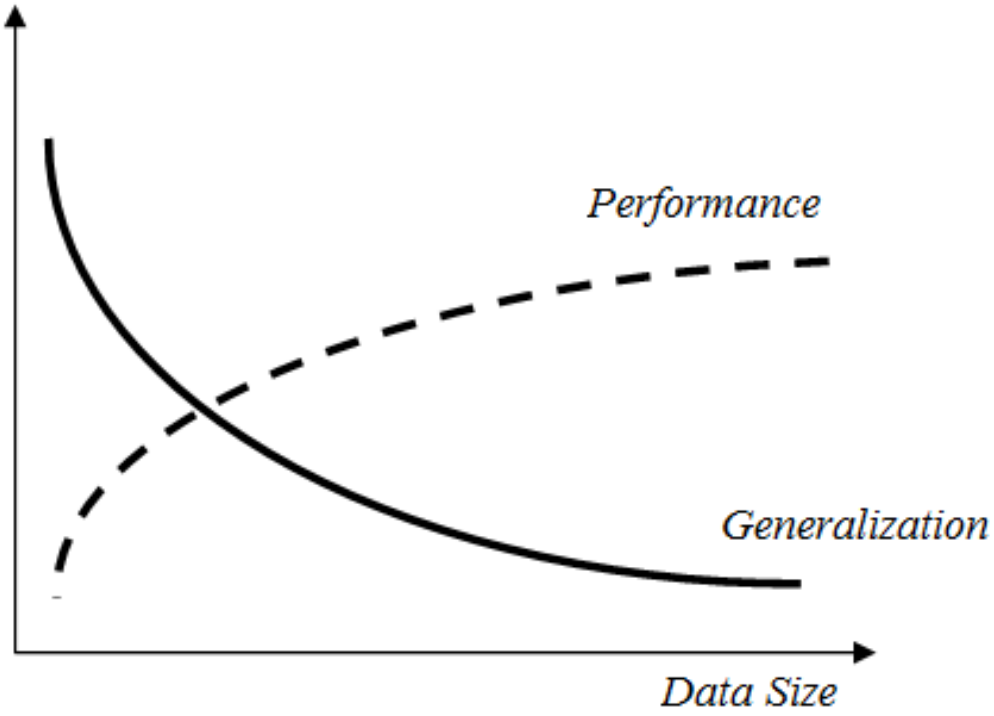
7. Solving confusing data problems (Noisy Data): Using artificial neural network technology (can deal with confusing data, for example, data that are some of the values are not true (the result of a particular fault or negligence is) and incomplete data might be missing value) and this property is the result of carrying the aforementioned error property.

8. Ease of construction and learning: as we shall see later, the building, programming and learning artificial neural networks are an uncomplicated often because of the record (Standard) algorithms and rules to learn and that's what made this technique is desirable in solving a lot of minor ones and complex problems.

Generalization in ANN

Generalization means (learning within minimum learning data size).

Performance of ANN increase when learning data size increase.



Learning Types in ANN

1- Supervised Learning: Based on the data of the problem and included inputs and output target, learning process stops when the neural network learns on the inputs and outputs of the problem.

2- Unsupervised Learning: Based on the Input data (no output), learning process stops when the neural network weights become stable.

3- Self-Organization : It is similar to unsupervised learning with additional processing such as clustering or same as it.

Some Learning Rules

Hebb Rule, $\Delta W(i,j) = \eta\, X(i)\, O(i)$

Widrow-Hoff Rule, $\Delta W(i,j) = \eta\, [Y(i)-O(i)]\, X(i)$

Competitive Rule, $\Delta W(i,j) = \eta\, [X(i)-W(i,j)]\, O(i)$

Generalized Delta Rule (GDR),

$$\Delta W(i,j) = \eta\, \delta(j)\, X(i) - W(i,j)$$

Generalized Delta Rule (GDR) with momentum,

$$\Delta W(i,j) = \eta\, \delta(j)\, X(i) - \alpha\, W(i,j)$$

Some Parameters

η : learning rate ($0< \eta <1$).

Bias or Threshold (θ): it is an added value to the weights or a constant node to increase the convergence or decrease the time learning.

α : momentum term ($0< \alpha <1$), it is also use to increase the convergence or decrease the time learning during the weights updates.

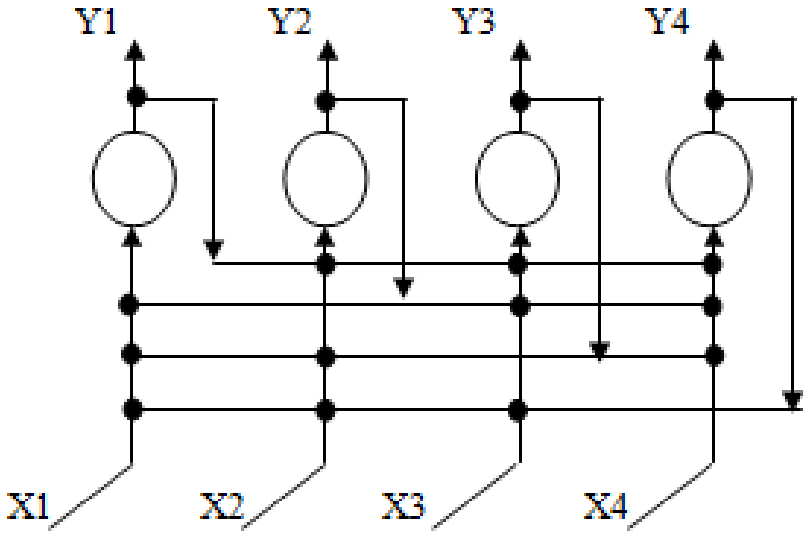
Hopfield NN

Invented by the physician (John Hopfield) in 1982.

Features:

- **Unsupervised learning.**
- **Associative memory.**
- **Full connection.**
- **Single layer.**
- **Feedback.**
- **Fixed weight.**
- **Bipolar.**
- **Linear activation function.**
- **Input Nodes equal to Output Nodes.**

Hopfield NN Topology



Hopfield NN Learning Algorithm

-Initialize, (N) no. of input node, (P) no. of samples, fun. $f(x) = \begin{cases} 1 & if\ x > 0 \\ -1 & if\ x <= 0 \end{cases}$

- Convert 0 to -1,

- Compute weight matrix, $W_{i,j} = \begin{cases} 0 & if\ i = j \\ \sum_{1}^{P} X_i X_j & otherwise \end{cases}$ $where\ i,j = 1,2,.....N$

-

Example

Assume N=4, P=3 as below

1	1	1	1		1	1	1	-1
1	1	1	1	→	-1	-1	-1	-1
1	1	0	0		1	1	-1	-1

Compute the weight matrix

$W_{1,2}$=1*1 + -1*-1 + 1*1 =3 = $W_{2,1}$

$W_{1,3}$=1*1 + -1*-1 + 1*-1 =1 = $W_{3,1}$

$W_{1,4}$=1*-1 + -1*-1 + 1*-1 =-1 = $W_{4,1}$

$W_{2,3}$=1*1 + -1*-1 + 1*-1 =1 = $W_{3,2}$

$W_{2,4}$=1*-1 + -1*-1 + 1*-1 =-1 = $W_{4,2}$

$W_{3,4}$=1*-1 + -1*-1 + -1*-1 =1 = $W_{4,3}$

$$W = \begin{pmatrix} 0 & 3 & 1 & -1 \\ 3 & 0 & 1 & -1 \\ 1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 0 \end{pmatrix}$$

Testing:

Assume we have the sample [1 1 1 1], apply the fun.

$$O_i = f(Z_i + \sum_1^j Z * W_{i,j})$$

$$where\ i,j = 1,2,.....N$$

O1= f(1+(1*0+1*3+1*1+1*-1)) = f(4) =1

O2= f(1+(1*3+1*0+1*1+1*-1)) = f(4)=1

O3= f(1+(1*1+1*1+1*0+1*1)) = f(4) =1

O4= f(1+(1*-1+1*-1+1*1+1*0)) = f(0) = -1

output = [1 1 1 0]

Again,

O1= f(1+(1*0+1*3+1*1+ -1*-1)) = f(6) =1

O2= f(1+(1*3+1*0+1*1+ -1*-1)) = f(6)=1

O3= f(1+(1*1+1*1+1*0+ -1*1)) = f(6) =1

O4= f(0+(1*-1+1*-1+1*1+ -1*0)) = f(-1) = -1

Output = [1 1 1 0], it is same as previous, therefore stop, this final output.

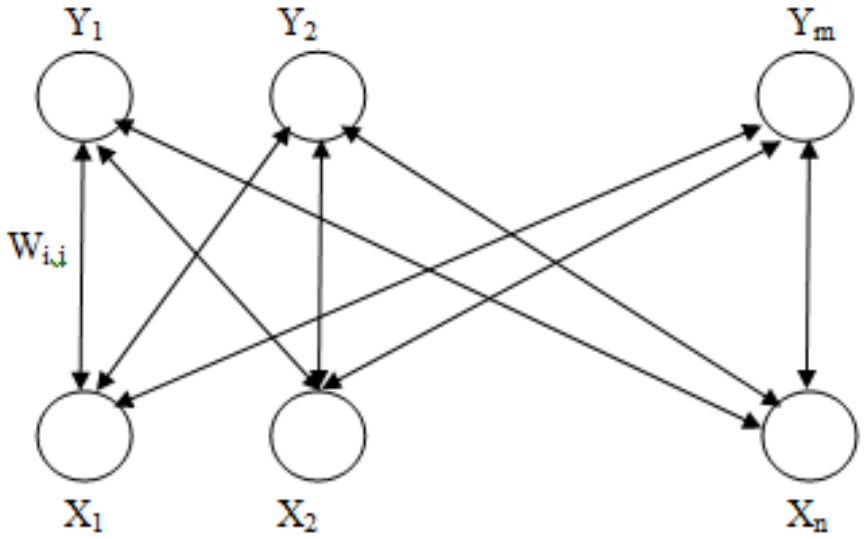
[1 1 1 0]

Relatively it is good.

Binary Associated Memory (BAM)

(Bart Kosko) propose this type of artificial neural networks in 1988, which is very similar to a Hopfield NN, but in BAM there are input layer & output layer.

BAM-NN Topology



BAM features:

- **Unsupervised learning.**
- **Associative memory.**
- **Full connection.**
- **I/O layer.**
- **Feedback.**
- **Fixed weight.**
- **Bipolar.**
- **Linear activation function.**
- **Input Nodes not necessary equal to Output Nodes.**

BAM Algorithm is same as Hopfield, except the weight computation as below equation:

$$W_i = A_i^T * B_i$$

Example

Input	Output
1 0 0	0 0 1
0 1 0	0 1 0
0 0 1	1 1 0

Convert 0 to -1, then

$W_1=[A_1^T][B_1]$

$$W_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}$$

$W_2=[A_2^T][B_2]$

$$W_2 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

$W_3=[A_3^T][B_3]$

$$W_3=\begin{bmatrix}-1\\-1\\1\end{bmatrix}\begin{bmatrix}1 & -1 & -1\end{bmatrix}=\begin{bmatrix}-1 & 1 & 1\\-1 & 1 & 1\\1 & -1 & -1\end{bmatrix}$$

$W = W_1 + W_2 + W_3$

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

Testing : 1 0 0

*B = f(A*W)*

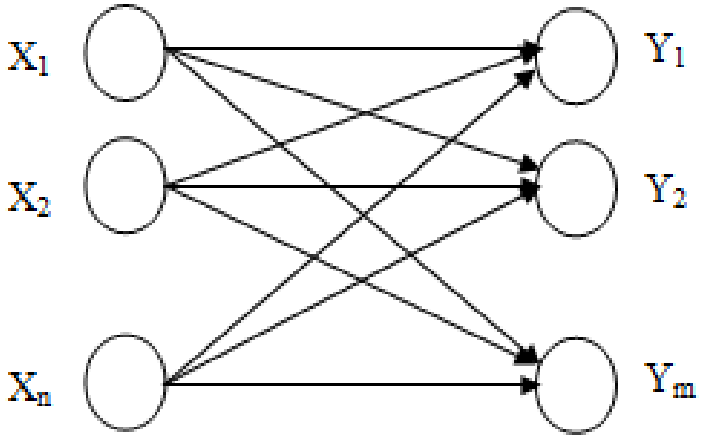
$$\begin{bmatrix}1 & -1 & -1\end{bmatrix}\begin{bmatrix}-1 & -1 & 3\\-1 & 3 & -1\\3 & -1 & -1\end{bmatrix}=\begin{bmatrix}-3 & -3 & 5\end{bmatrix}=\begin{pmatrix}0 & 0 & 1\end{pmatrix}$$

Single Layer NN

Features:

- Supervised learning.
- Single layer (I/O).
- Linear Activation Function.
- Full connection.
- Based on Delta learning rule.
- Variable weights.

Single layer NN topology



Single Layer NN Algorithm

- Initialize X input nodes (N nodes), Y output nodes (M nodes), No. of samples (P), η learning rate and activation fun. *f(x)*.
- Generate weight matrix with size (NxM).
- While error ratio not acceptance Do

  for each sample compute the below equation:

$$O_{k,j} = f(\sum_{i=1}^{N} X_{k,i} * W_{i,j}) \quad \text{where } j = 1,2,....,M \ \& \ k = 1,2,....,P$$

  compute the error ratio as below:

$$\delta = (Y - O)$$

  if error not equal zero then adjust the weight matrix as:

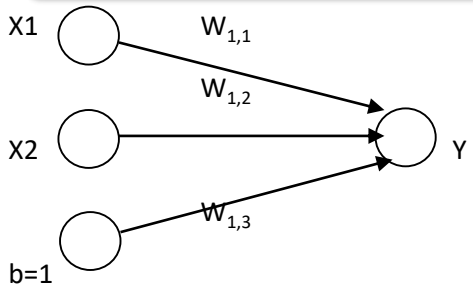
$$W_{i,j} = W_{i,j} \pm \eta * \delta * X_i \quad \text{where } i = 1,2,...., N \ \& \ j = 1,2,...., M$$

  end while

End.

Example: Logical AND Gate

X1	X2	B	Y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



Where X1, X2 are input nodes, B is bias node & Y is output node.

Initialize W = [1 -1 1], activation function is

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \le 0 \end{cases}$$

First sample [1 1 1]

$O_1 = f(1 * 1 + 1 * -1 + 1 * 1) = f(1) = 1$

$\delta = Y_1 - O_1 = 1 - 1 = 0$ O.K.

Second sample [1 -1 1]

$O_2 = f(1 * 1 + -1 * -1 + 1 * 1) = f(3) = 1$

$\delta = Y_2 - O_2 = -1 - 1 = -2$

Adjust W

$W_{1,1} = 1 + 0.7 * (-2) * 1 = -0.4$

$W_{1,2} = -1 + 0.7 * (-2) * (-1) = 0.4$

$W_{1,3} = 1 + 0.7 * (-2) * 1 = -0.4$

W = [-0.4 0.4 -0.4]

Then $O_2 = f(1 * -0.4 + -1 * 0.4 + 1 * -0.4) = f(-4.2) = -1$

Third sample [-1 1 1]

$O_3 = f(-1 * -0.4 + 1 * 0.4 + 1 * -0.4) = f(0.4) = 1$

$\delta = -1 - 1 = -2$

Adjust W

$W_{1,1} = -0.4 + 0.7 * (-2) * (-1) = 1$

$W_{1,2} = 0.4 + 0.7 * (-2) * 1 = -1$

$W_{1,3} = -0.4 + 0.7 * (-2) * 1 = -1.8$

$W = [\ 1 \quad -1 \quad -1.8]$

Then $O_3 = f(-1 * 1 + 1 * -1 + 1 * -1.8) = f(-3.8) = -1$

Fourth sample [-1 -1 1]

$O_٤ = f(-1 * 1 + -1 * 1 + 1 * -1.8) = f(-3.8) = -1$

Now again from the begin

$O_١ = f(1 * 1 + 1 * -1 + 1 * -1.8) = f(-1.8) = -1$

Adjust W

$W_{1,1} = 1 + 0.7 * (2) * 1 = 2.4$

$W_{1,2} = 1 + 0.7 * (2) * 1 = 0.4$

$W_{1,3} = -1.8 + 0.7 * (2) * 1 = -0.4$

$W = [\ 2.4 \quad 0.4 \quad -0.4]$

Continue with weight adjustment until the weight become :

$W = [\ 0.4 \quad 0.4 \quad -0.6]$

Which accept all I/O samples.

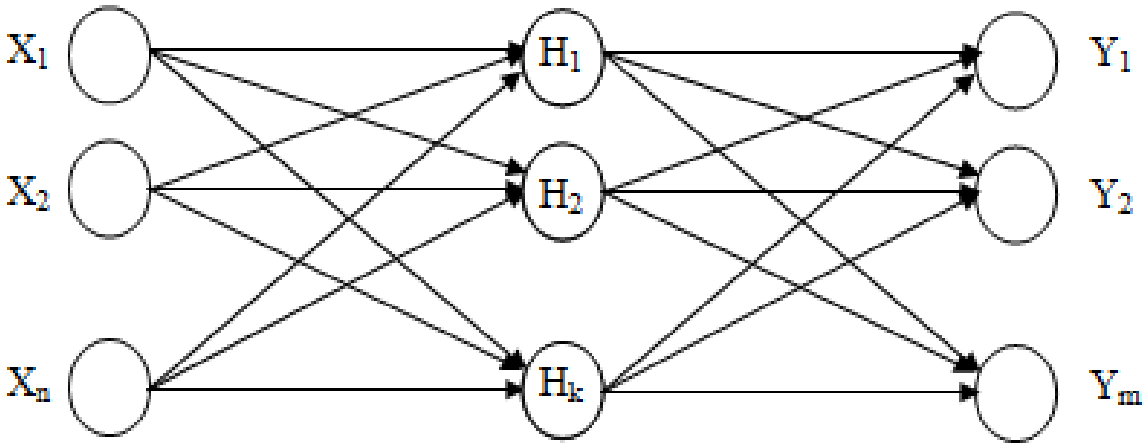
Back-Propagation NN

(Paul Werbos) was developed back propagation algorithm in 1975. It is one of the most artificial neural network algorithms in applications

Features:

- Supervised learning.
- Multi Layers.
- Non-Linear Activation Function.
- Full connection.
- Based on GBDR.
- Variable weights.
- Process Forward-Backward.

BP-NN topology



BP-NN Algorithm

- Initialize the parameters (A: no. of input nodes), (B: no. of hidden nodes), (C: no. of output nodes), (P: no. of samples), (η: learning rate) and activation function.
- Generate the weight matrices (W1[AxB], W2[BxC]) randomly.
- While error is acceptance Do
  - For each sample Do
  - Compute the hidden nodes as below

$$H_j = \frac{1}{1+e^{-V}} \quad \text{where } j = 1,2,.....B \text{ , } V = \sum_{i=1}^{A} W1_{i,j} X_i$$

  - Compute the output nodes as below

$$O_j = \frac{1}{1+e^{-V}} \quad \text{where } j = 1,2,.....C \text{ , } V = \sum_{i=1}^{B} W2_{i,j} H_i$$

  - End for

- Compute error as $$error = \sqrt{\sum_{k=1}^{C} (Y_k - O_k)^2}$$

- If error not acceptance Then
  - Adjust the weight matrices as below

$$\delta 2_k = O_k * (1 - O_k) * (Y_k - O_k) \qquad \text{where } k = 1,2,\ldots\ldots, C$$

$$\delta 1_k = Z_k * (1 - Z_k) * \sum_{j=1}^{C} \delta 2_j W 2_{k,j} \qquad \text{where } k = 1,2,\ldots\ldots, B$$

$$W 2_{i,j} = W 2_{i,j} + \eta * \delta 2_j * H_i \qquad \text{where } i = 1,2,\ldots., B \ \&\ j = 1,2,\ldots., C$$

$$W 1_{i,j} = W 1_{i,j} + \eta * \delta 1_j * X_i \qquad \text{where } i = 1,2,\ldots., A \ \&\ j = 1,2,\ldots., B$$

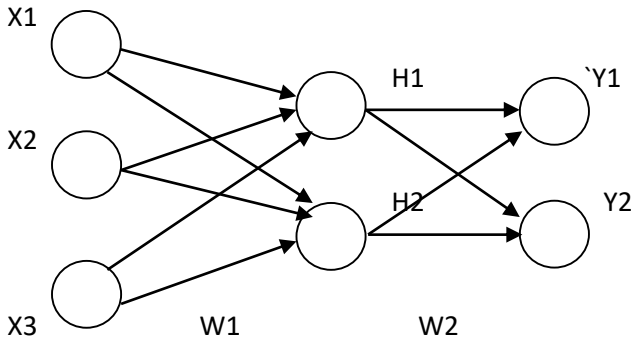
- End If;
- End While;
- Save the last weight values;
- End Algorithm.

Example: Assume we have 3 input nodes, 2 output nodes and 3 samples as below:

X1	X2	X3	Y1	Y2
1	0	0	1	0
0	0	1	0	1
1	1	1	1	1

Suppose we have 2 hidden nodes with the following weight matrices & BP-NN topology:

$$W1 = \begin{bmatrix} 0.2 & 0.1 \\ 0.3 & 0.7 \\ 0.8 & 0.5 \end{bmatrix} \qquad W2 = \begin{bmatrix} 0.1 & 0.6 \\ 0.7 & 0.8 \end{bmatrix}$$



Where η=0.6 and error acceptance ratio = 0.1

Solution:

First sample [1 0 0 1 0]

Sum = $\sum$ W1* X

Sum1 = 1*0.2 + 0*0.3 + 0*0.8 = 0.2

$$h_1 = \frac{1}{1+e^{-0.2}} = 0.5498$$

Sum2 = 1*0.1 + 0*0.7 + 0*0.5 = 0.1

$$h_2 = \frac{1}{1+e^{-0.1}} = 0.525$$

Sum = $\sum$ W2* H

Sum1 = 0.5498*0.1 + 0.525*0.6 = 0.37 $o_1 = \frac{1}{1+e^{-0.37}} = 0.5915$

Sum2 = 0.5498*0.7 + 0.525*0.8 = 0.805 $o_2 = \frac{1}{1+e^{-0.805}} = 0.691$

The error is not acceptance (Desired [1 0], Actual [0.5915 0.691])

Error in output layer

$\delta 2_1$=0.5915*(1-0.5915)*(1-0.5915)=0.0987

$\delta 2_2$=0.691*(1-0.691)*(0-0.691)=-0.1475

Error in hidden layer

$\delta 1_1$=0.5498*(1-0.5498)*(0.1*0.0987+0.6*-0.1475)=-0.0195

$\delta 1_2$=0.525*(1-0.525)*(0.7*0.0987+0.8*-0.1475)=-0.0196

Adjust the weights

W2

$W2_{1,1}$=0.1+0.6*0.0987*0.5498=0.1326

$W2_{1,2}$=0.6+0.6*-0.1475*0.525=0.551

$W2_{2,1}$=0.7+0.6*0.0987*0.5498=0.7311

$W2_{2,2}$=0.8+0.6*-0.1475*0.525=0.7535

W1

$W1_{1,1}$=0.2+0.6*-0.0195*1=0.1883, $W1_{1,2}$=0.1+0.6*-0.196*1=0.0882

$W1_{2,1}$=0.3+0.6*-0.0195*0=0.3, $W1_{2,2}$=0.7+0.6*-0.196*0=0.7

$W1_{3,1}$=0.2+0.6*-0.0195*0=0.8, $W1_{3,2}$=0.1+0.6*-0.196*0=0.5

Some drawbacks in BP-NN

- **Parameter values** : like η, α values, there are no canonical method to determine the reasonable values for these parameters, therefore the experiments is the way to choose the reasonable values.
- **No. of Hidden Nodes** : hidden layer play a big role in the convergence of weights and results, also there is no way to determine the reasonable no. of nodes in this layer, the solution is the experiments. Also sometimes there are more than hidden layer in the NN.
- **Choose the activation function** : there are several non-linear activation function, we must select one or more be carefully.
- **Weights update** : there are few weight update formals and parameters, we must select one from these be carefully.
- **No. of Bias nodes** : How many Bias nodes in the input & hidden layers? By experiments !!
- **Error ratio computation** : sample-by-sample or accumulatively.

Kohonen - NN

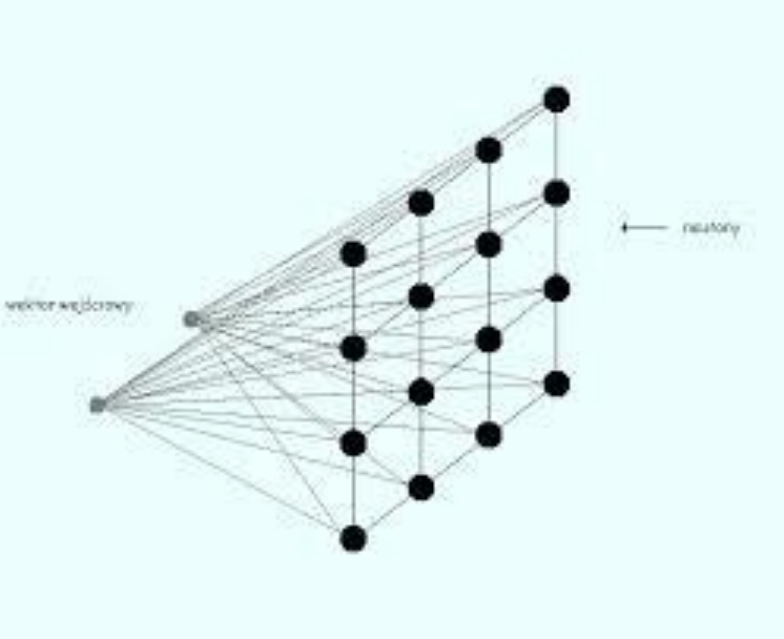
(Teuvo Kohonen) developed this NN in 1982, and adopted this algorithm as a kind of algorithms that does not need to supervisor and the competitive type are also used for the purposes of clustering. The structure of this network includes two layers, the first concerning the inputs (features) and the second is the output (clusters or groups), the weights determining which outputs a matrix fits the input. Given the values of the vectors weights randomly and then examine the inputs start and the extent of bias to the output through the equation of Euclidean dimension, takes less output value in terms of the Euclidean dimension siding designated entrance into this category, and so the second and third inputs to the end, that is, network self-reorganizing through mathematical equations that each input to the exit is biased or a particular class. In other words, that there is competition between them (input) on the bias to the output. Certainly the inputs would side similar to the same items in other words, it will be based on the principle of neighboring (Neighborhood).

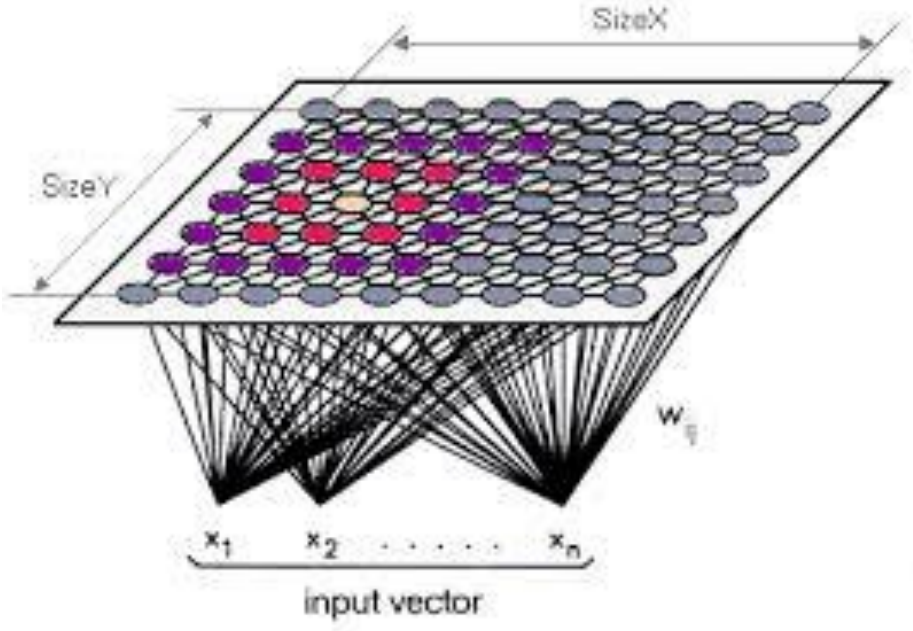
Kohonen – NN Features & Topologies

Features:

- Self-Organization (Unsupervised).
- Depends on competitive learning.
- Used for complex problems.

There are several topologies for Kohonen-NN as:





Kohonen-NN Algorithm

- Initialize the parameters (N: no. of input nodes X), (M: no. of clusters C), (P: no. of samples), (α: learning rate), (min- α).
- Generate the weight matrix W[NxM] randomly.
- While α < min- α Do
  - For each sample Do
  - Compute D for each cluster (j) $D(j) = \sum \left(x_i - w_{ij} \right)^2$
  - Select Minimum D(j)
  - Update column j in the weight matrix $w_{i,j} = w_{i,j} + \alpha(X_i - w_{i,j})$
  - End For
  - Update α value
  - End While

End Algorithm.

Update of α

- Linear : $\alpha = \alpha - \beta$, $(\beta>0)$

- Non-Linear : $\alpha = \alpha\ \beta$, $(0<\beta<1)$

$\alpha = \alpha / \beta$, $(\beta>1)$

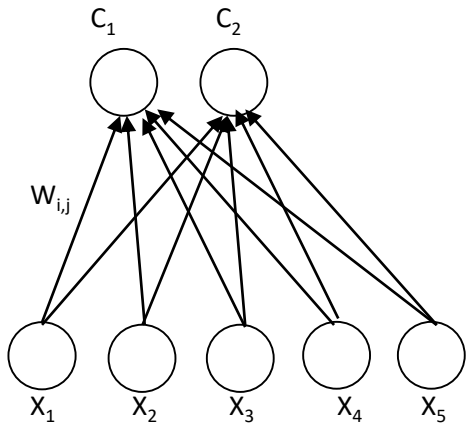
$\alpha = \log(\alpha)$

Example: Assume we have the following samples

1 1 0 0 0
0 0 0 1 1
1 0 1 0 0
0 0 0 0 1

Suppose there are 2 clusters, with initial α =0.6, min- α = 0.1, β=0.1 and initial W matrix as below:

$$w = \begin{bmatrix} 0.3 & 0.7 \\ 0.4 & 0.2 \\ 0.3 & 0.6 \\ 0.1 & 0.5 \\ 0.5 & 0.4 \end{bmatrix}$$



For the first sample 1 1 0 0 0 , compute D(j) as below

$D(1)= (1-0.3)^2+(1-0.4)^2+(0-0.3)^2+(0-0.1)^2+(0-0.5)^2=1.2$

$D(2)= (1-0.7)^2+(1-0.2)^2+(0-0.6)^2+(0-0.5)^2+(0-0.4)^2=1.5$

The min j=1 so,

$W_{1,1}$=0.3+0.6*(1-0.3)= 0.72
$W_{1,2}$=0.4+0.6*(1-0.4)= 0.76
$W_{1,3}$=0.3+0.6*(0-0.3)= 0.12
$W_{1,4}$=0.1+0.6*(0-0.1)= 0.04
$W_{1,5}$=0.5+0.6*(0-0.5)= 0.2

W =
0.72 0.7
0.76 0.2
0.12 0.6
0.04 0.5
0.2 0.4

For the first sample 0 0 0 1 1, compute D(j) as below

$D(1)= (٠-0.٧٢)^2+(٠-0.٧٦)^2+(0-0.١٢)^2+(١-0.٠٤)^2+(١-0.٢)^2=2.672$

$D(2)= (٠-0.7)^2+(٠-0.2)^2+(0-0.6)^2+(١-0.5)^2+(١-0.4)^2=1.5$

The Min j= 2

$W_{٢,1}=0.7+0.6*(0-0.7)= 0.28$

$W_{٢,2}=0.2+0.6*(0-0.2)= 0.08$

$W_{٢,3}=0.6+0.6*(0-0.6)= 0.24$

$W_{٢,4}=0.5+0.6*(1-0.5)= 0.8$

$W_{٢,5}=0.4+0.6*(1-0.4)= 0.76$

$$W = \begin{matrix} 0.72 & 0.28 \\ 0.76 & 0.08 \\ 0.12 & 0.24 \\ 0.04 & 0.8 \\ 0.2 & 0.76 \end{matrix}$$

The Third sample 1 0 1 0 0 , d1 = 1.472 , d2 = 2.32

$$W = \begin{matrix} 0.888 & 0.28 \\ 0.304 & 0.08 \\ 0.648 & 0.24 \\ 0.016 & 0.8 \\ 0.08 & 0.76 \end{matrix}$$

The fourth sample 0 0 0 0 1, d1 = 2.1475, d2 = 0.84

$$W = \begin{matrix} 0.888 & 0.112 \\ 0.304 & 0.032 \\ 0.648 & 0.096 \\ 0.016 & 0.32 \\ 0.08 & 0.904 \end{matrix}$$

Update the learning rate (α), as : = α - β = 0.6 - 0.1= 0.5

Then again from the first sample, d1 = 0.9235, d2 = 2.6544

0.944 0.112
0.652 0.032
W= 0.324 0.096
0.008 0.32
0.04 0.904

The second sample, d1 = 3.3269, d2 = 0.4944

0.944 0.056
0.652 0.016
W = 0.324 0.048
0.008 0.66
0.04 0.952

The Third sample, d1 = 0.8869, d2 = 3.1396

0.972 0.056
0.326 0.016
W = 0.662 0.048
0.004 0.66
0.02 0.952

The fourth sample, d1 = 2.4497, d2 = 0.4436

0.972 0.028
0.326 0.008
W = 0.662 0.024
0.004 0.33
0.02 0.976

Genetic Algorithms

"Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime."- Salvatore Mangano, *Computer Design*, May 1995.

- Originally developed by John Holland (1975)
- The genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution
- Uses concepts of "Natural Selection" and "Genetic Inheritance" (Darwin 1859)

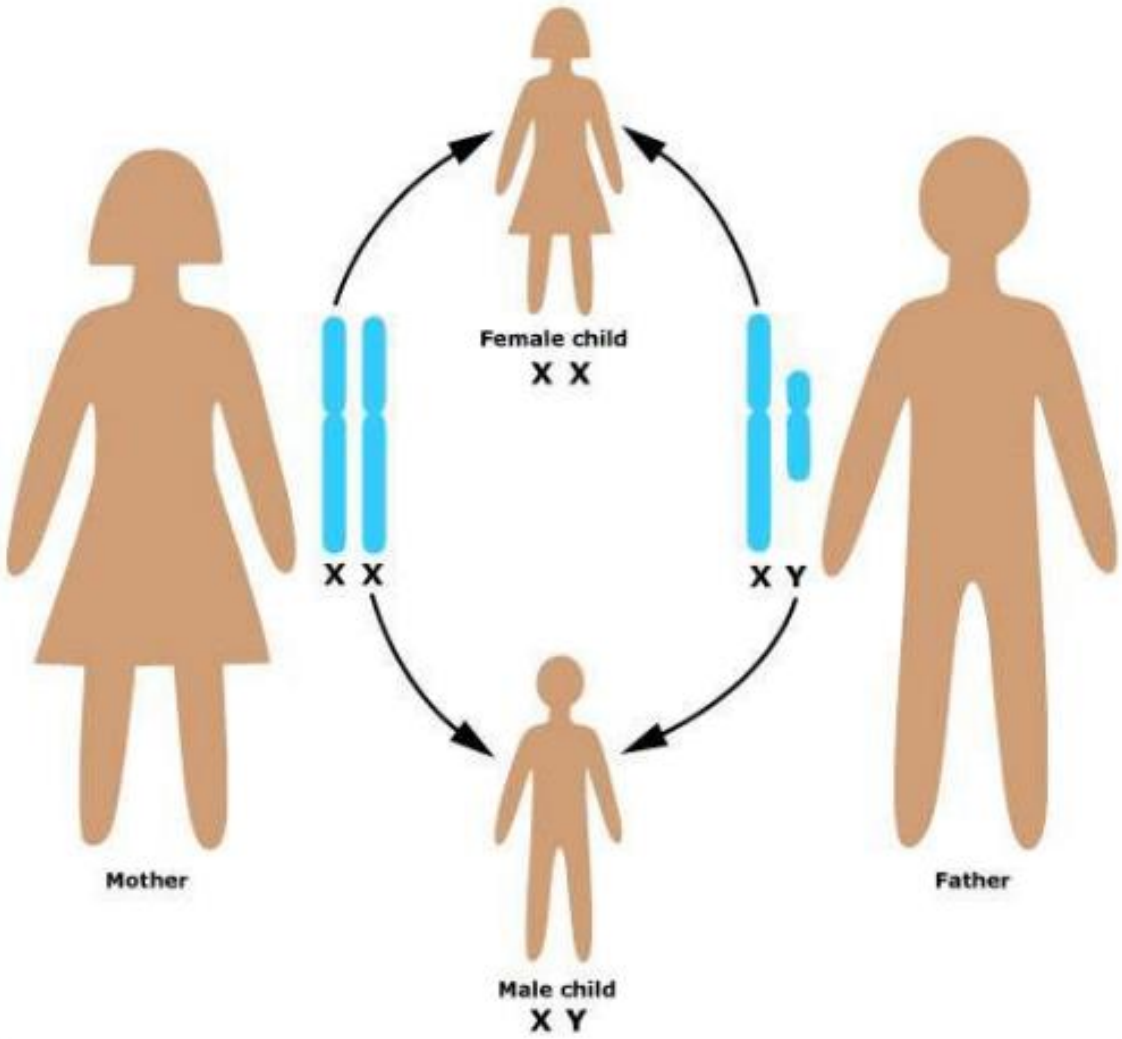
A genetic algorithm maintains a population of candidate solutions for the problem at hand, and makes it evolve by iteratively applying a set of stochastic operators

Biologically - GA

Our body is made up of trillions of **cells**. Each cell has a core structure (**nucleus**) that contains your **chromosomes**.

Each **chromosome** is made up of tightly coiled strands of deoxyribonucleic acid (**DNA**). **Genes** are segments of DNA that determine **specific traits**, such as eye or hair color. You have more than 20,000 genes.

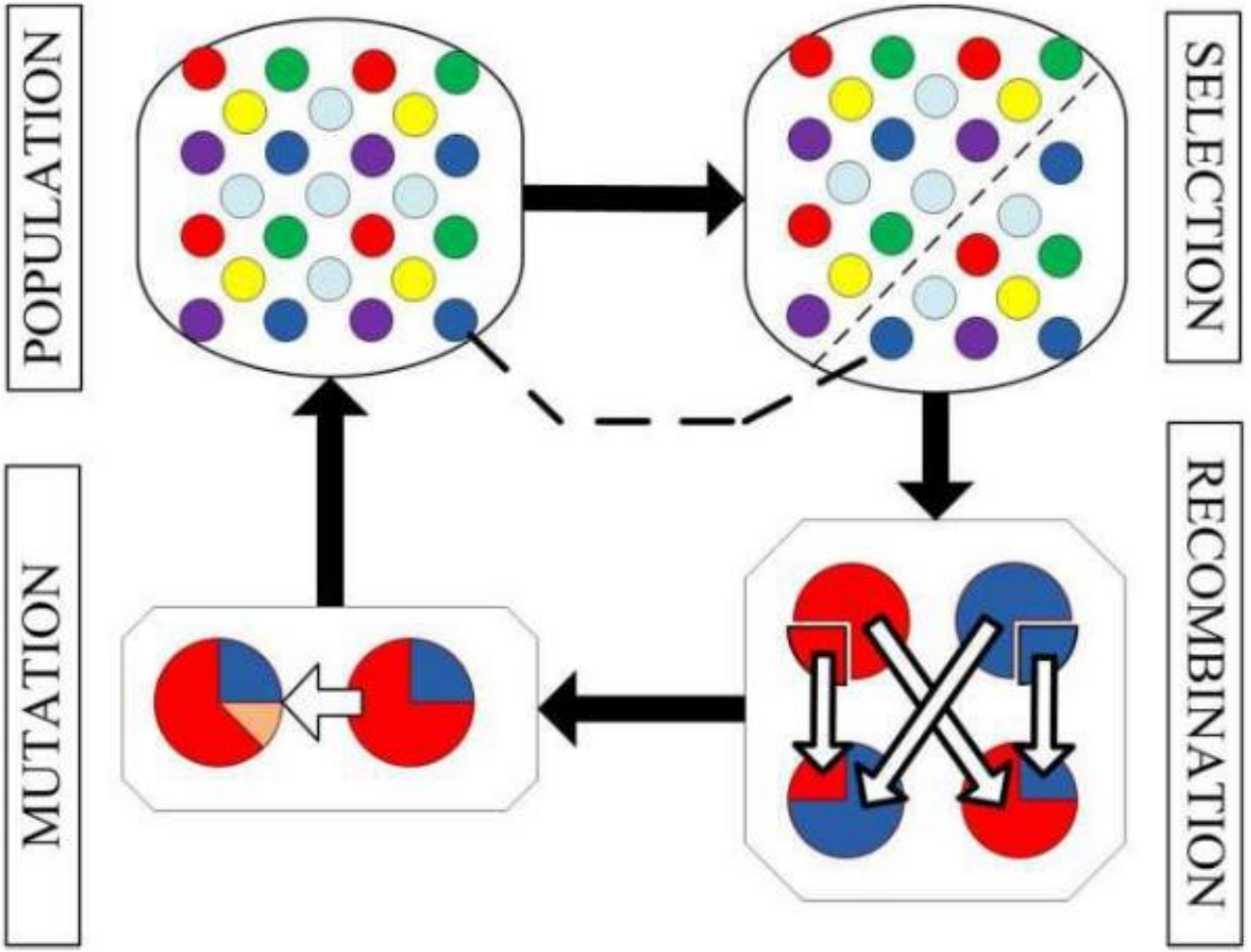
A gene **mutation** is an alteration in your DNA. It can be inherited or acquired during your lifetime, as cells age or are exposed to certain chemicals. Some changes in your genes result in genetic disorders.


Female child
X X
X X
X Y
Mother
Father
Male child
X Y

Bio-Genetic Vs. Genetic Algorithm

Biological Genetic	Genetic Algorithms
Chromosomes	Strings or Symbols
Genes	Strings or Symbols (May Be One)
Allele	Feature
Position	Position of String or Symbol
Genotype	Representation or Structure
Phenotype	Coding of Representation or Structure

Components of GA



Classical Form of GA

Begin
Encode the problem;
Generate initial population of individuals;
Evaluate the fitness of all individuals;
While not (Termination Conditions) **Do**
    Begin
        Select pair of individuals;
        Crossover between the individuals (recombine);
        Mutate individuals;
        Produce a new population;
        Evaluate the fitness of the modified individuals;
    End While
End Algorithm

Encoding

- Binary : 01001111
- Integer : 12 5 38 11 09
- Real : 1.2 0.3 11.4 0.11 1.0
- Char : A C V F
- String : abc cvc123

Initial Population

We start with a population of n random strings. Suppose that $l = 10$ and $n = 6$

We toss a fair coin 60 times and get the following initial population:

$s_1 = 1111010101$

$s_2 = 0111000101$

$s_3 = 1110110101$

$s_4 = 0100010011$

$s_5 = 1110111101$

$s_6 = 0100110000$

Fitness Function : *F(X)*

$s_1 = 1111010101 \quad f(s_1) = 7$

$s_2 = 0111000101 \quad f(s_2) = 5$

$s_3 = 1110110101 \quad f(s_3) = 7$

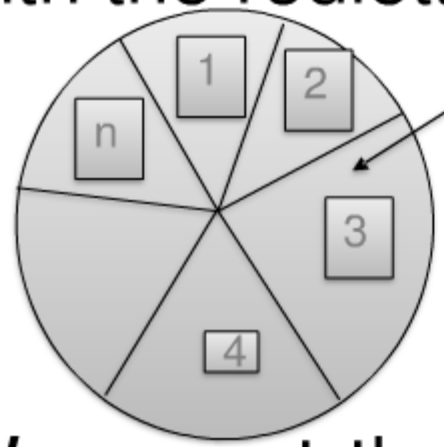
$s_4 = 0100010011 \quad f(s_4) = 4$

$s_5 = 1110111101 \quad f(s_5) = 8$

$s_6 = 0100110000 \quad f(s_6) = 3$

Selection Operator

- Next we apply fitness proportionate selection with the roulette wheel method:



Area is Proportional to fitness value

Individual *i* will have a probability to be chosen

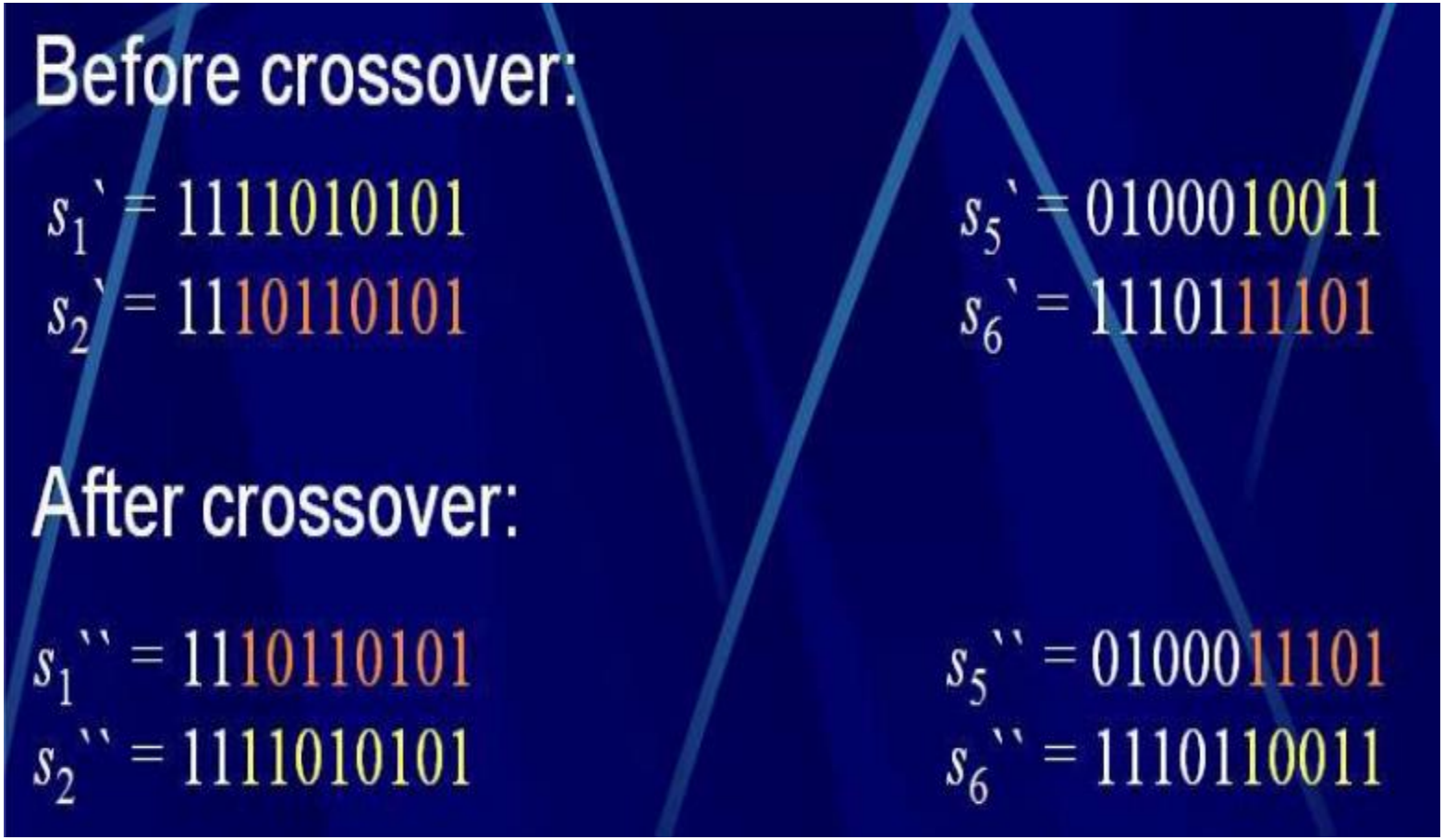
$$\frac{f(i)}{\sum_{i} f(i)}$$

- We repeat the extraction as many times as the number of individuals
- we need to have the same parent population size (6 in our case)

Other Selection Operator

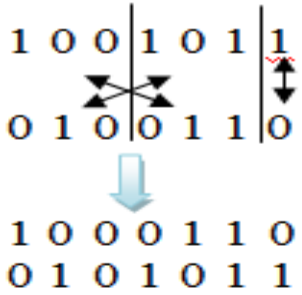
- Elitism.
- Tournament.
- Rank.
- Boltezman.
- Sigma Scaling.
- Steady State.

Crossover (Recombination)

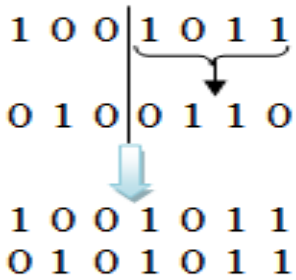


Other Crossover Operator

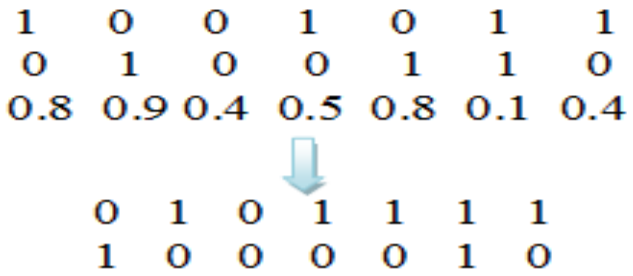
- Multi-Points Crossover



- Bacteria Conjugation.



- Uniform Crossover



Other Crossover Operator

- Crowding.
- Fitness Sharing Function.
- Inversion.
- Segregation.
- Migration.
- Translocation.
- Flat.
- Partially Matched.

Mutation

- Before applying mutation:

$s_1'' = 1110110101$

$s_2'' = 1111010101$

$s_3'' = 1110111101$

$s_4'' = 0111000101$

$s_5'' = 0100011101$

$s_6'' = 1110110011$

- After applying mutation:

$s_1''' = 1110100101$

$s_2''' = 1111110100$

$s_3''' = 1110101111$

$s_4''' = 0111000101$

$s_5''' = 0100011101$

$s_6''' = 1110110001$

GA Parameters

- Population Size (Pop_Size): Its play a major role in providing appropriate diversity in the nature of the solutions generated by increasing the size of the community, but the greater the size of the community has increased processing time and this is contrary to what is required of a genetic algorithm to find a solution as soon as possible.
- Chromosome Length (Chrom_Len): This parameter depends mainly on the size of the problem addressed by the genetic algorithm it represents the length of the solution and not the number of solutions that we have dealt with in the size of the community coefficient. Here, the length of the solution is the chromosome length, for example, it is required to find the shortest path to map where 7 cities surely that the length of the chromosome must not exceed the number 7, and if the number of cities 100 the length of the chromosome does not exceed the number 100. It is possible to make this a coefficient variable in the event that the problem is dealing with the subject so or are variable by itself or can be doing this if it leads to the production of high quality solutions for high-appropriate time.

GA Parameters

Prof. Dr. Ahmed Tariq Sadiq

- Maximum Generation (Max_Gen): It represents a number of loops implementing the genetic algorithm operations, any number of generations generated starting from the first community generated randomly. Certainly it is the greater problem is the data to be solved size preferably increase the number of generations generated on the grounds that the problem cannot find a solution it quickly. Number of generations has generated up to several hundred volumes when they are required to solve large problems, and sometimes the number of generations could be up to four decimal places (thousands) in a very large problems can not be sufficient to prepare the hundreds of generations, where due to the large size of the problem.
- Crossover Probability (Pc): There is no doubt that the crossover of genetic algorithm process is one of the key processes in this algorithm because it is responsible for the changes in the next generation changes, so there is a possibility that the Pc process with a very big impact on future generations and therefore the speed of finding a solution (or solutions) is required. As is known, the likelihood of something between zero and 1, it has zero indicates not happen and the one referring to the constant occurrence, as we explained earlier, the chromosomes of the fetus born consists of a combination of the chromosomes of the mother and the chromosomes of the father, and sometimes they tend chromosomes of the fetus to the mother and sometimes those tend chromosomes to strongly Father, this means that the likelihood of the exchange between chromosomes means that the fetus takes on both sides, so the likelihood of boom should not be a small value (for example, less than 0.2) so that they reduce the mutation process and should not be so big that strongly occur (for example, greater than 0.8).

GA Parameters

Prof. Dr. Ahmed Tariq Sadiq

- Mutation Probability (Pm): Unlike the exchange process, the probability of occurrence of a genetic mutation (Pm) in the fetus are certainly a few, but they occur out of the question can not be neglected in the next generation as a result of several medical and genetic and social reasons are not in the process of addressing them. It is the probability of an adverse genetic mutation (Pm) is definitely going to be a few as it is in the best cases, does not exceed 0.2 according to most research and sources, and sometimes much less may be the value of the boom (Pm) is equal to 0.003, for example. In fact, to the benefit of genetic mutation in the genetic algorithm has many people think is useless and the opposite is true.
- Fitness Function: A mathematical function reflects how close the current solution of the desired solution (or current solutions required) solutions, which play a very important role in obtaining the required solutions and whenever selected successful was getting faster solution, but sometimes represent the difficulty of obtaining congruence resolve function a specific problem by using a genetic algorithm is a problem in itself. There are those who assumed certain mathematical function and try them in terms of execution speed and accuracy solutions and after experiments to be agreed on a specific function.

GA Parameters

Prof. Dr. Ahmed Tariq Sadiq

- Stop Conditions: When we stop work the genetic algorithm running in search of a solution (or solutions)? We can be summarized in terms of stopping three (of course the occurrence of one of those conditions leads to stop the execution of a genetic algorithm and not combined) conditions. The first condition is that finding the solution according to the criteria and parameters given. The second condition is the arrival of the number of generations generated to the limit (Max-Generation) given in execution. The third condition is the occurrence of the case of no return and are intended to here the case of (Local-Minima), in other words, that the generations that breed in turn where there is no significant diversity or change in her condition, and is therefore in a state of inertia leading to inaccessibility of the solution It is required in view of the lack of diversity property (diversity) which plays an important role in reaching a solution. The availability of any of the above conditions will result in an end to the implementation of a genetic algorithm and reboot again, or return to the first step of generating a primary community and call the main loop implementation of genetic algorithm may be that this treatment if there is no solution to the result.

Samples of Fitness Function

$$f(x) = x^2 + 2x + 1$$

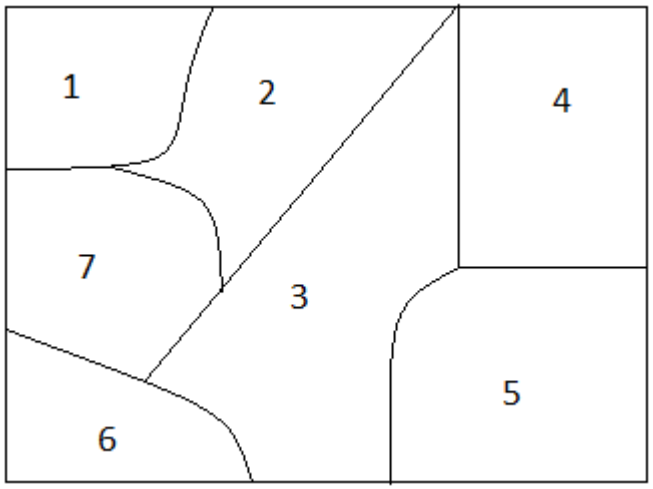
$$f(x) = x^2 + 1$$

$$f(x) = \frac{x}{\sum x^2}$$

Problem	Fitness Function
4-Color Mapping	A number of the same color neighboring cities
Shortest Path	Total cost of the path from the starting point to the end point
Traveling Salesman	Total cost of the first track of the city and return to it passing through all the cities at once
Graph Planarization	The number of intersections in the diagram
Best Path Robot	The remaining distance between the current situation and the state of the target

Application: 4-Color Mapping Problem

There are 7 cities as in the map, the adjacency matrix as below:



$$Adjcency = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Assume the GA generate 4 solutions (pop_size) randomly as :

G R R Y B G B

R B Y Y G R B

B G G Y R R Y

Y G R R B G B

Apply the fitness function on the population :

	f(x)
G R R Y B G B	1
R B Y Y G R B	2
B G G Y R R Y	1
Y G R R B G B	1

Assume the selection operation choice chromosome 1 & 2, to perform crossover operation as below:

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

crossover in the 2nd position

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

The mutate chromosome 1 (last gene) and chromosome 2 (2th gene) as below:

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

Compute the fitness function for the 2 new chromosome :

	f(x)
R B R Y B G Y	0
G G Y Y G R B	2

That means there is no 2 adjacency cities has same color as in the below map:

R B R Y B G Y