الحقيبة التعليمية : نمذجة الانظمة الذكية / المرحلة الرابعة فرع الكترونيات الحاسوب

Week	Syllabus
1st, 2nd	Introduction and role of ANNs, fundamentals of biological Neural Network, basic principles of ANNs and their early structures
3rd	Properties of ANN, advantage, and disadvantage
,4th, 5th , 6th ,7th	network architectures, logic gates
8th , 9th , 10th	Types of learning rules, learning algorithms, training styles
11th ,12th , 13th ,14th	Hebb , Adaline, Madalines, delta rule
15th,16th	Important perception function, neuron model, perception architecture, learning rules, training (train)
17th ,18th	The back propagation learning procedure, derivation of the BP algorithm, Back propagation training algorithm
19th , 20th	Search algorithm, Genetic algorithm
21th,22th,23rd ,24th	Type of operators, population, selection, crossover, crossover rate, mutation, mutation rate
25th ,26th, 27th	Population, selection, crossover, and mutation algorithms
28th	Application of genetic algorithms
29th , 30th	Advantage and disadvantage of Genetic algorithms

What Is A Neural Net?

- **Artificial Neural Networks:**
- An artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

l. Information processing occurs at many simple elements called neurons.

2. Signals are passed between neurons over connection links.

3. Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.

4. Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A neural network is characterized by (1) its pattern of connections between the neurons (called its architecture), (2) its method of determining the weights on the connections (called its training, or learning, algorithm), and (3) its activation function.

What Is A Neural Net?

Why neural Networks And Why now

Biological Neural Networks

Brains versus Computers : Some numbers

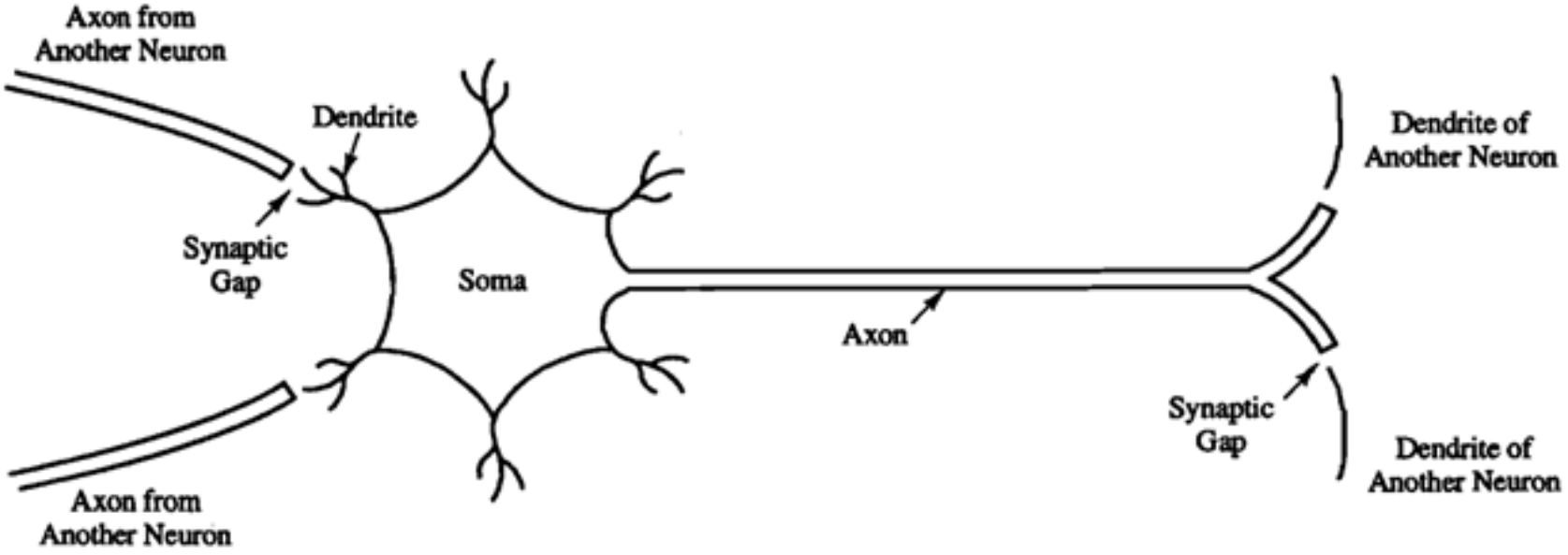
Where Are Neural Nets Being used?

Why neural Networks And Why now

- As modern computers become ever more powerful, scientists continue to be challenged to use machines effectively for tasks that are relatively simple for humans.
- We learn easily to recognize the letter A or distinguish a cat from a bird.
- More experience allows us to refine our responses and improve our performance.
- Even without a teacher, we can group similar patterns together.

Biological Neural Networks

- A biological neuron has three types of components that are of particular interest in understanding an artificial neuron: its dendrites, soma, and axon.
- The many dendrites receive signals from other neurons. The signals are electric impulses that are transmitted across a synaptic gap by means of a chemical process.
- The action of the chemical transmitter modifies the incoming signal(typically, by scaling the frequency of the signals that are received) in a manner similar to the action of the weights in an artificial neural network.



Key Features

- Several key features of the processing elements of artificial neural networks are:

1. The processing element receives many signals.
2. Signals may be modified by a weight at the receiving synapse.
3. The processing element sums the weighted inputs.
4. Under appropriate circumstances (sufficient input), the neuron transmits a single output.
5. The output from a particular neuron may go to many other neurons (the axon branches).

Where Are Neural Nets Being used?

- 1- Signal Processing.
- 2- Control.
- 3- Pattern Recognition.
- 4- Medicine.
- 5- Speech Production.
- 6- Speech Recognition.

Brains versus Computers : Some numbers

1. There are approximately 10 billion neurons in the human cortex, compared with 10 of thousands of processors in the most powerful parallel computers.
2. Each biological neuron is connected to several thousands of other neurons, similar to the connectivity in powerful parallel computers.
3. Lack of processing units can be compensated by speed. The typical operating speeds of biological neurons is measured in milliseconds (10^{-3} s), while a silicon chip can operate in nanoseconds (10^{-9} s).
4. The human brain is extremely energy efficient, using approximately 10^{-16} joules per operation per second, whereas the best computers today use around 10^{-6} joules per operation per second.
5. Brains have been evolving for tens of millions of years, computers have been evolving for tens of decades.

Properties of ANN, advantage, and disadvantage

Typical Architectures

Setting the Weights

Common Activation Functions

THE McCULLOCH-PITTS NEURON

Logic Functions

Advantages

1- ANN with multiple hidden layers, and it is responsible for the rapid development that's going on in the Machine Learning industry right now.
2- ANNs provided us the first step towards AI by generating a model based on how our own human body learns.

Disadvantages

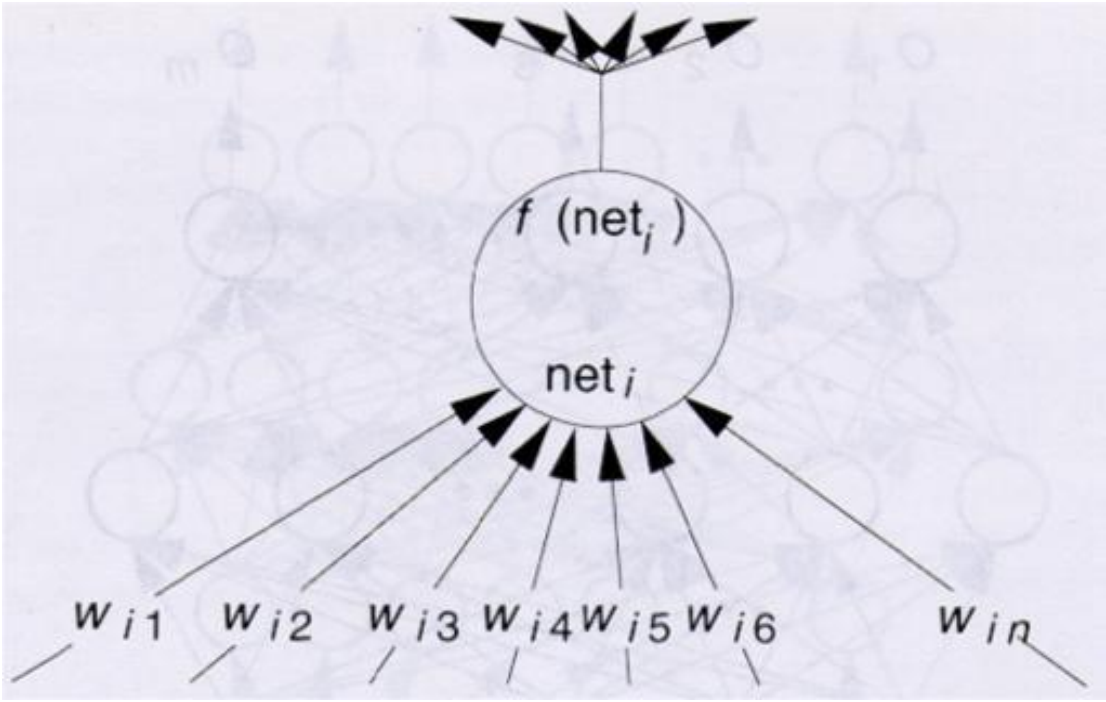
1- Pretty much just overuse of it. Everyone's trying to apply deep learning to everything now, even things that don't require it.
2- It's leading to a huge misunderstanding of the whole field, Neural Networks, and Machine Learning but have no clue what they actually mean.

- **Elements of Artificial Neural Networks**
  - Processing Units
  - Topology
  - Learning Algorithm

- **Processing Units**

Node input: $net_i = \sum_j w_{ij} I_i$

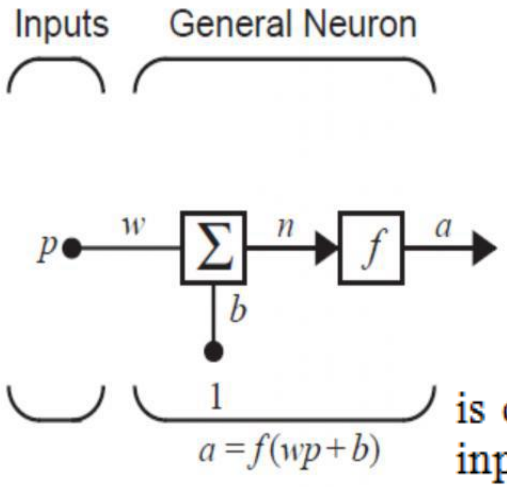
Node Output: $O_i = f(net_i)$



Neuron Model

Single-Input Neuron

A single-input neuron is shown in Figure below. The scalar input is P multiplied by the scalar *weight* w to form WP, one of the terms that is sent to the summer. The other input, 1, is multiplied by a *bias* b and then passed to the summer. The summer output n, often referred to as the *net input*, goes into a *transfer function* , which produces the scalar neuron output a . If we relate this simple model back to the biological neuron that we discussed in Lecture 1, the weight W corresponds to the strength of a synapse, the cell body is represented by the summation and the transfer function, and the neuron output a represents the signal on the axon.



The neuron output is calculated as

$$a = f(wp + b)$$

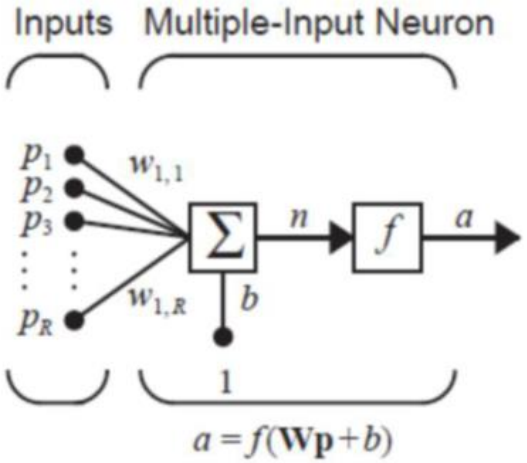
If, for instance $w = 3$ $p = 2$ and $b = -1.5$ then

$$a = f(3\ (2) - 1.5) = f(4.5)$$

The actual output a depends on the particular transfer function that is chosen. The bias is much like a weight, except that it has a constant input of 1 . Note that W and P are both *adjustable* scalar parameters of the neuron.

Multiple-Input Neuron

Typically, a neuron has more than one input. A neuron with R inputs is shown in Figure below. The individual inputs $P_1, P_2, \ldots, P_R$ are each weighted $w_{1\,1}, w_{1,2} \ldots\ldots w_{1\,R}$ by corresponding elements of the *weight matrix* W.

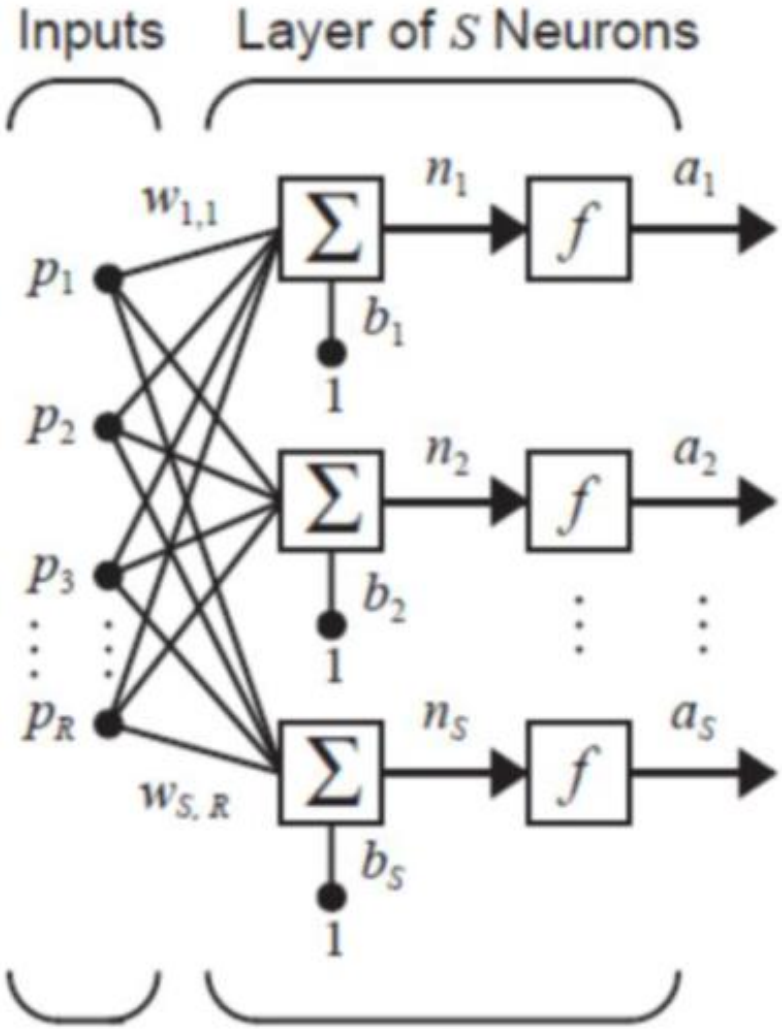


Network Architectures

Commonly one neuron, even with many inputs, may not be sufficient. We might need five or ten, operating in parallel, in what we will call a layer." This concept of a layer is discussed below.

A Layer of Neurons

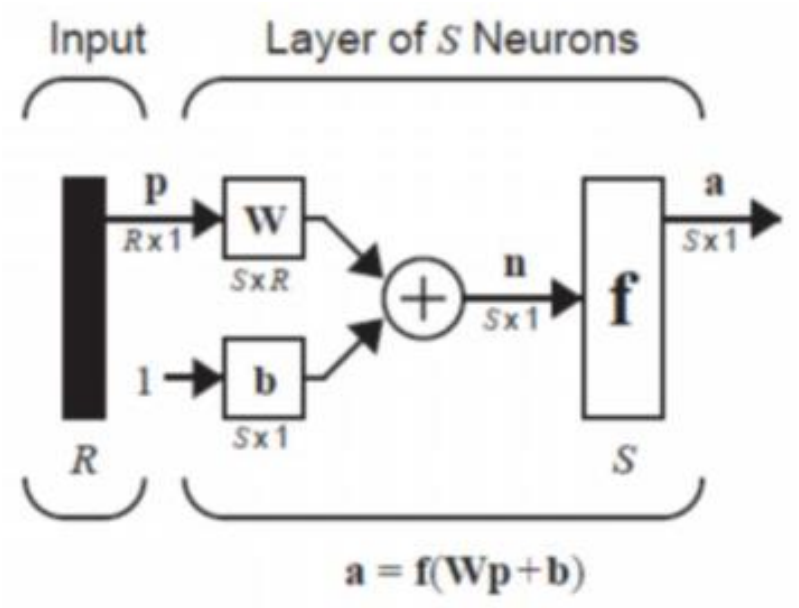
A single-*layer* network of S neurons is shown. Note that each of the inputs is connected to each of the neurons and that the weight matrix now has rows.



$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

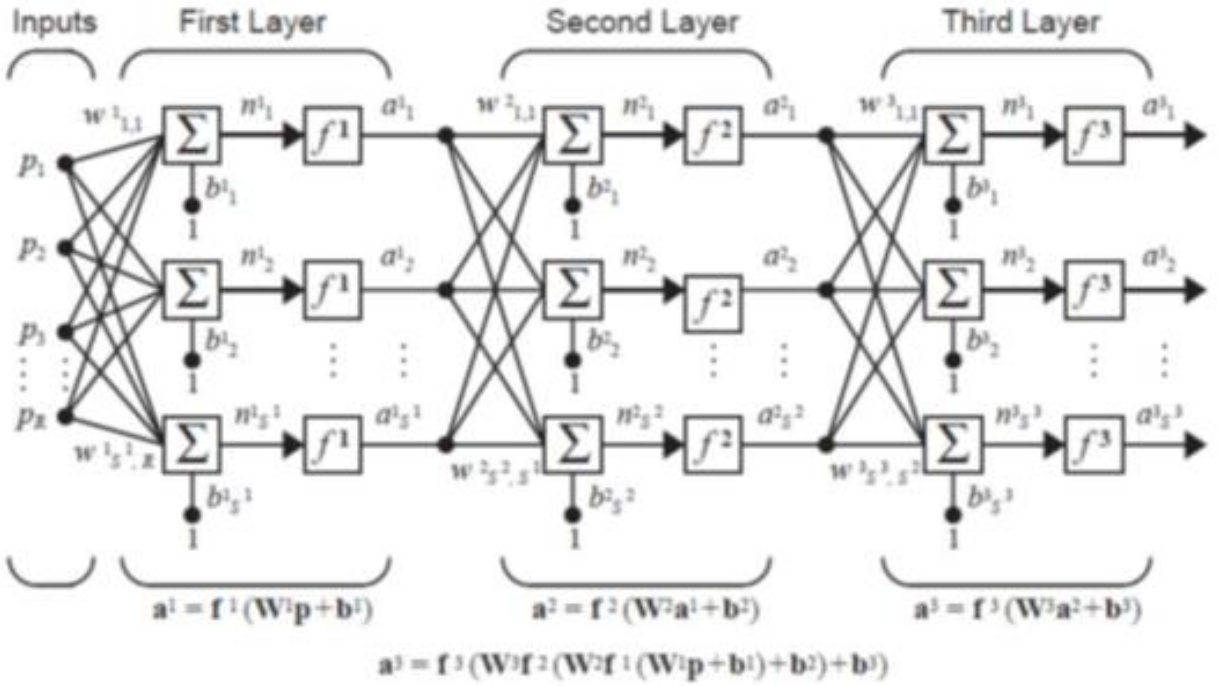
one-layer network also can be drawn in abbreviated notation,

as shown in Figure

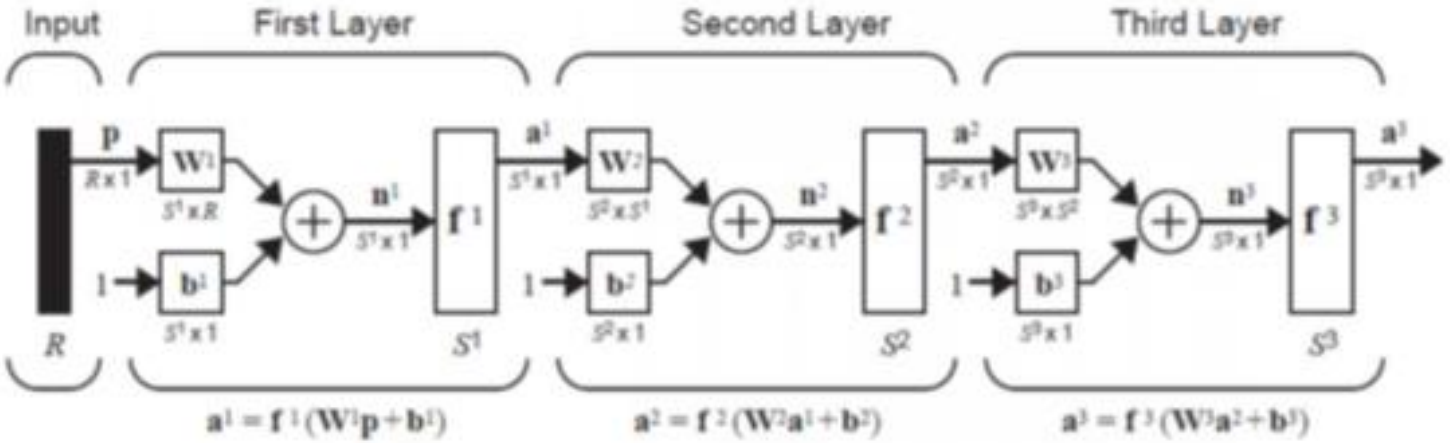


Multiple Layers of Neurons

Now consider a network with several layers. Each layer has its own weight matrix **W** , its own bias vector **b** , a net input vector **n** and an output vector . We need to introduce some additional notation to distinguish between these layers. We will use superscripts to identify the layers. Specifically, we append the number of the layer as a *superscript* to the names for each of these variables. Thus, the weight matrix for the first layer is written as $\mathbf{W}^1$, and the weight matrix for the second layer is written as $\mathbf{W}^{21}$. This notation is used in the three-layer network shown in Figure



The same three-layer network discussed previously also can be drawn using our abbreviated notation, as shown in Figure.



Setting the Weights

- In addition to the architecture, the method of setting the values of the weights **(training)** is an important distinguishing characteristic of different neural nets.
- The weights are then adjusted according to a learning algorithm. This process is known as **supervised training**.
- **Unsupervised training** Self-organizing neural nets group similar input vectors together with out the use of training data to specify what a typical member of each group looks like or to which group each vector belongs.

Unsupervised learning: A means of modifying the weights of a neural net without specifying the desired output for any input patterns. Used in self-organizing neural nets for clustering data, extracting principal components, or curve fitting.

Supervised training: Process of adjusting the weights in a neural net using a learning algorithm; the desired output for each of a set of training input vectors is presented to the net. Many iterations through the training data may be required.

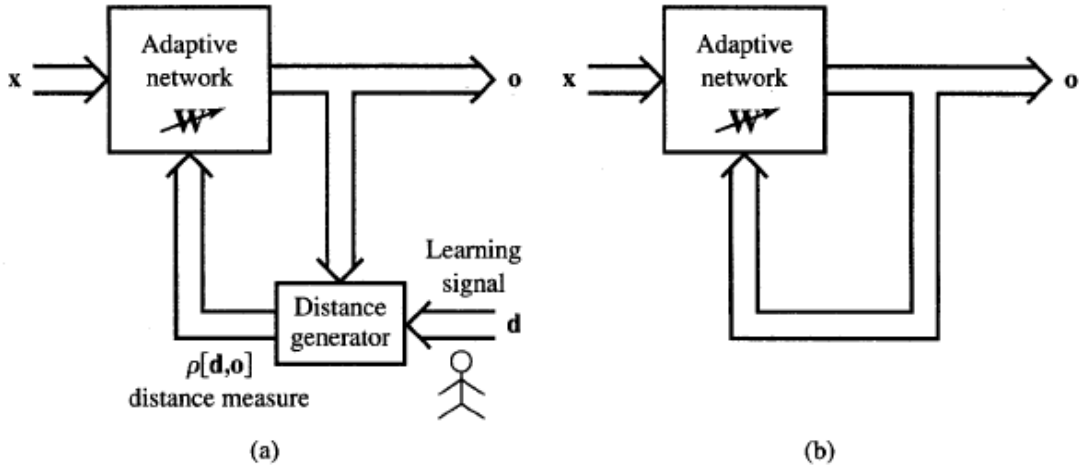


Figure 2.19 Block diagram for explanation of basic learning modes: (a) supervised learning and (b) unsupervised learning.

Applications

- Pattern Classification

- Clustering/Categorization

- Function approximation

- Prediction/Forecasting

- Optimization

- Content-addressable Memory

- Control

Transfer function (type of activate functions)

Common Activation Functions

(i) Identity function:

$$f(x) = x \quad \text{for all } x.$$

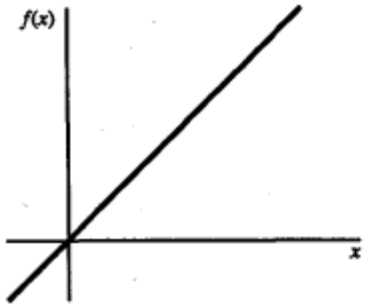


Figure 1.7 Identity function.

(ii) Binary step function (with threshold θ):

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

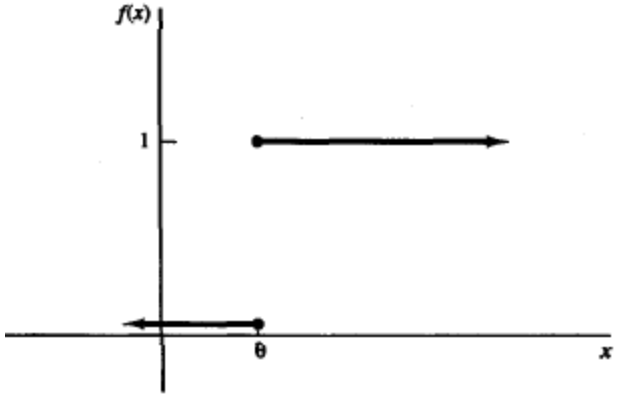


Figure 1.8 Binary step function.

(iii) Binary sigmoid:

$$f(x) = \frac{1}{1 + \exp(-\sigma x)}.$$

$$f'(x) = \sigma f(x)\,[1 - f(x)].$$

$$a = \frac{1}{1 + e^{-n}}.$$

As is shown in Section 6.2.3, the logistic sigmoid function can be scaled to have any range of values that is appropriate for a given problem. The most common range is from -1 to 1; we call this sigmoid the *bipolar sigmoid*. It is illustrated in Figure 1.10 for $\sigma = 1$.

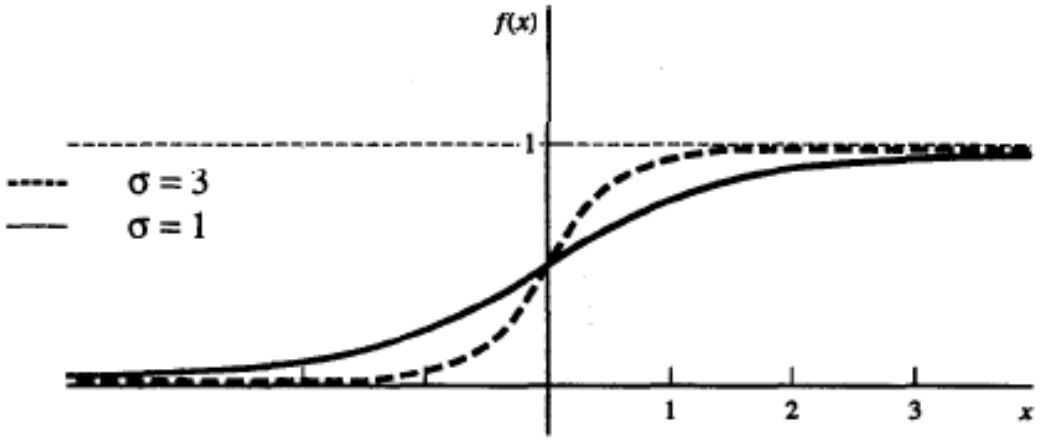


Figure 1.9 Binary sigmoid. Steepness parameters $a = 1$ and $a = 3$.

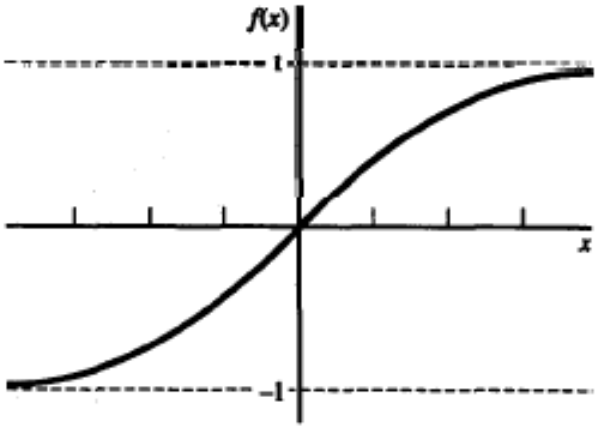


Figure 1.10 Bipolar sigmoid.

(iv) Bipolar sigmoid:

$$g(x) = 2f(x) - 1 = \frac{2}{1 + \exp(-ax)} - 1$$

$$= \frac{1 - \exp(-\sigma x)}{1 + \exp(-\sigma x)}.$$

$$g'(x) = \frac{\sigma}{2}[1 + g(x)][1 - g(x)].$$

The bipolar sigmoid is closely related to the hyperbolic tangent function, which is also often used as the activation function when the desired range of output values is between -1 and 1. We illustrate the correspondence between the two for $a = 1$. We have

$$g'(x) = \frac{\sigma}{2}[1 + g(x)][1 - g(x)].$$

The bipolar sigmoid is closely related to the hyperbolic tangent function, which is also often used as the activation function when the desired range of output values is between -1 and 1. We illustrate the correspondence between the two for $a = 1$. We have

$$g(x) = \frac{1 - \exp(-x)}{1 + \exp(-x)}.$$

The hyperbolic tangent is

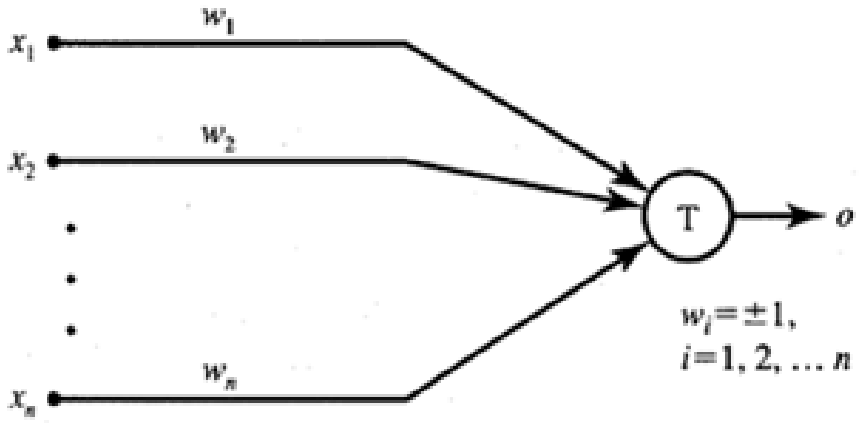
$$h(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

$$= \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

THE McCULLOCH-PITTS NEURON

- The first formal definition of a synthetic neuron model based on the highly simplified considerations of the biological model described in the preceding section was formulated by McCulloch and Pitts (1943). The inputs xi, for i = 1, 2, . . . , n, are 0 or 1, depending on the absence or presence of the input impulse at instant k. The neuron's output signal is denoted as o. The firing rule for this model is defined as follows

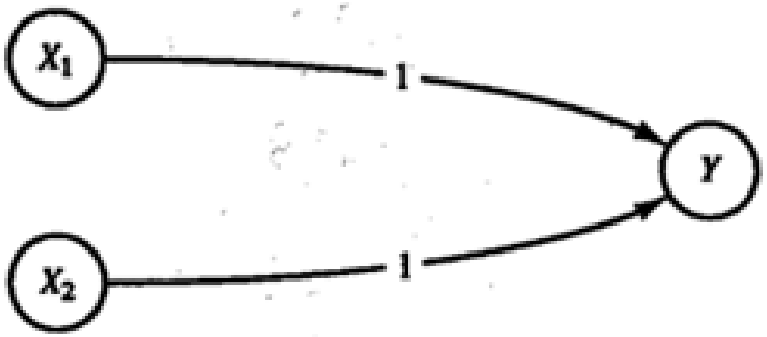
$$o^{k+1} = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i^k < T \end{cases}$$



Logic Functions

- AND
- The AND function gives the response "true" if both input values are "true"; otherwise the response is "false." If we represent "true" by the value I, and "false" by 0, this gives the following four training input, target output pairs:

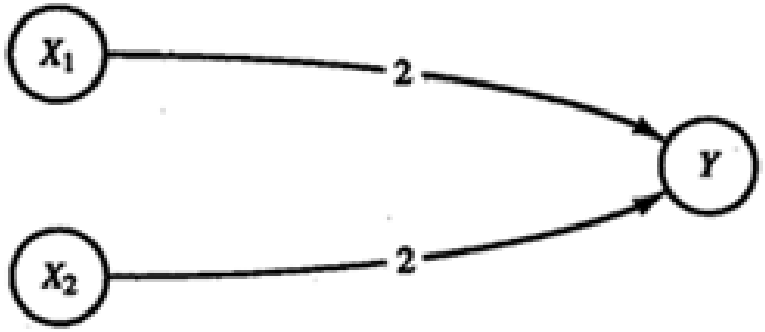
x_1	x_2	→	y
1	1		1
1	0		0
0	1		0
0	0		0



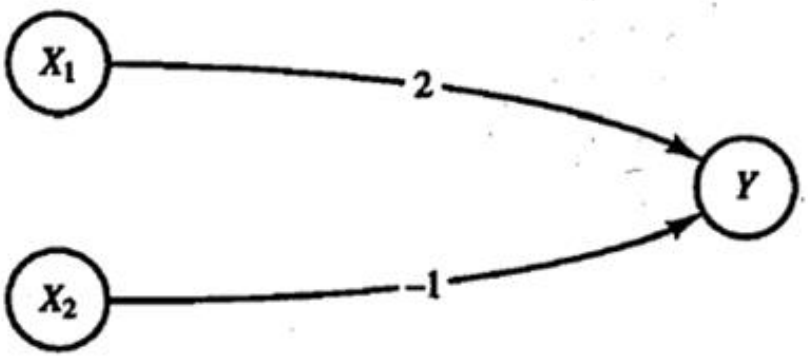
OR gate

The threshold on unit Y is 2.

x_1	x_2	→	y
1	1		1
1	0		1
0	1		1
0	0		0



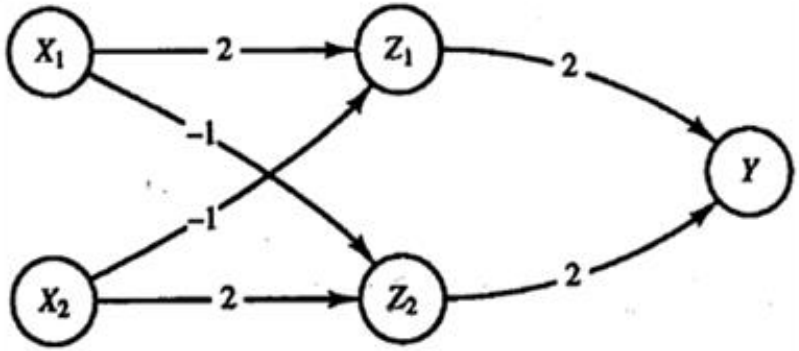
AND NOT



x_1	x_2	→	y
1	1		0
1	0		1
0	1		0
0	0		0

XOR

x_1 XOR $x_2 \leftrightarrow (x_1$ AND NOT $x_2)$ OR $(x_2$ AND NOT $x_1)$.



x_1	x_2	→	y
1	1		0
1	0		1
0	1		1
0	0		0

$z_1 = x_1$ AND NOT x_2

$z_2 = x_2$ AND NOT x_1.

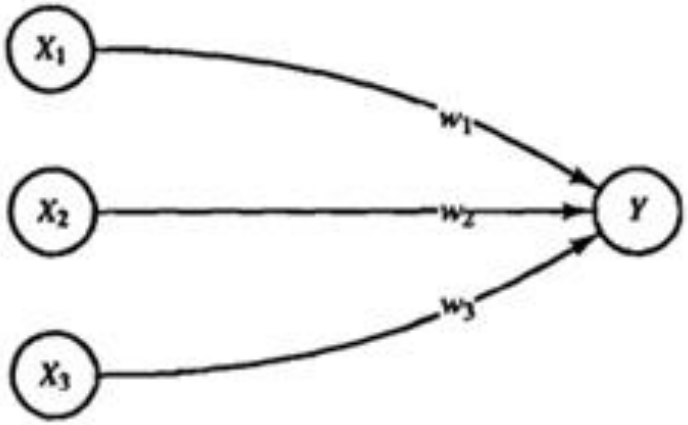
Activation or Activity Level

- For example, consider a neuron Y, that receives inputs from neurons X_1, X_2, and X_3 The weights on the connections from X_1, X_2, and X_3 to neuron Y are w_1 to W_2, and W_3, respectively. The net input, y-in, to neuron Y is the sum of the weighted signals from neurons X_1, X_2, and X_3,i.e.,

$$y\_in = w_1x_1 + w_2x_2 + w_3x_3.$$

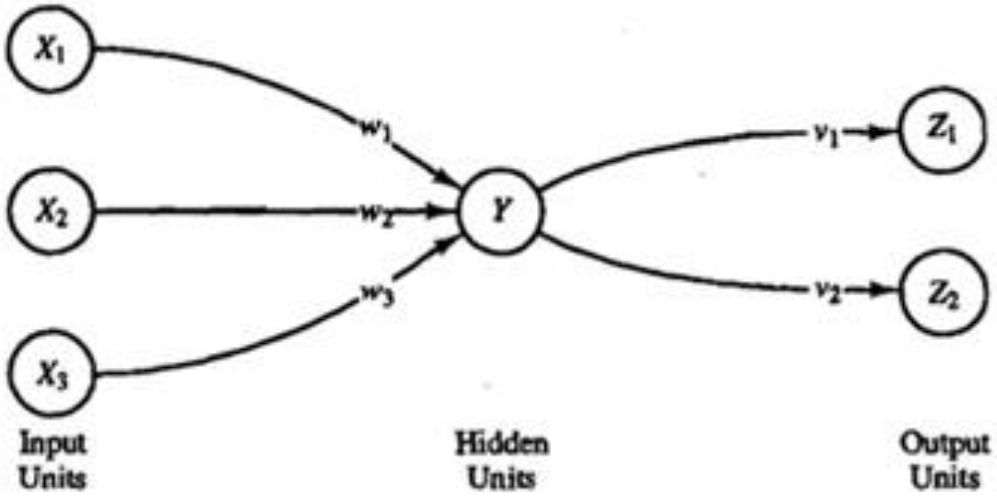
The activation y of neuron Y is given by some function of its net input, y = f(y-in), e.g., the logistic sigmoid function (an S-shaped curve)

$$f(x) = \frac{1}{1 + \exp(-x)},$$



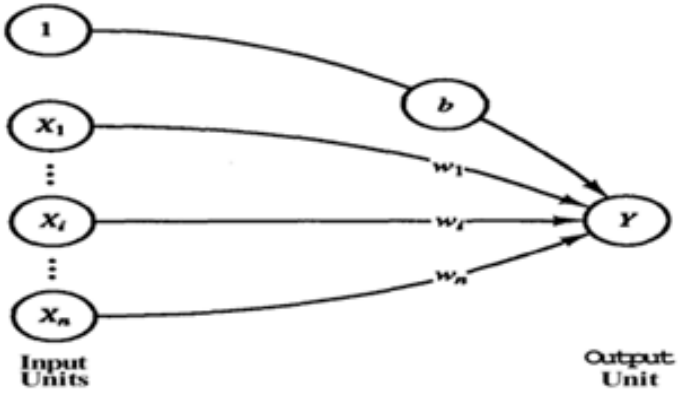
Activation or Activity Level Cont.

- Now suppose further that neuron Y is connected to neurons Z_1 and Z_2,with weights v_1 and v_2,respectively. Neuron Y sends its signal y to each of these units. However, in general, the values received by neurons Z_1 andZ_2 will be different, because each signal is scaled by the appropriate weight, v_1 or v_2 .In a typical net, the activations Z_1 and Z_2 of neurons Z_1 and Z_2 would depend on inputs from several or even many neurons, not just one, as shown in this simple example.



Architecture

- The basic architecture of the simplest possible neural networks that perform pattern classification consists of a layer of input units (as many units as the patterns to be classified have components) and a single output unit.



Hebb learning

Donald Hebb, a psychologist at McGill University, designed the first learning law for artificial neural networks [Hebb, 1949]. His premise was that if two neurons were active simultaneously, then the strength of the connection between them should be increased. Refinements were subsequently made to this rather general statement to allow computer simulations [Rochester, Holland, Haibt & Duda, 1956]. The idea is closely related to the correlation matrix learning developed by Kohonen (1972) and Anderson (1972) among others. An expanded form of Hebb learning [McClelland & Rumelhart, 1988] in which units that are simultaneously off also reinforce the weight on the connection between them will be presented in Chapters 2 and 3.

2.2.1 Algorithm

Step 0. Initialize all weights:

$w_i = 0$ (i = 1 to n).

Step 1. For each input training vector and target output pair, s : *t*, do steps 2–4.

Step 2. Set activations for input units:

$x_i = s_i$ (i = 1 to n).

Step 3. Set activation for output unit:

$y = t$.

Step 4. Adjust the weights for

$$w_i(\text{new}) = w_i(\text{old}) + x_i y \qquad (i = 1 \text{ to } n).$$

Adjust the bias:

$$b(\text{new}) = b(\text{old}) + y.$$

Note that the bias is adjusted exactly like a weight from a "unit" whose output signal is always 1. The weight update can also be expressed in vector **form** as

$$\mathbf{w}(\text{new}) = \mathbf{w}(\text{old}) + \mathbf{x}y.$$

This is often written in terms of the weight change, $\Delta\mathbf{w}$, as

$$\Delta\mathbf{w} = \mathbf{x}y$$

and

$$\mathbf{w}(\text{new}) = \mathbf{w}(\text{old}) + \Delta\mathbf{w}.$$

There are several methods of implementing the Hebb rule for learning. The foregoing algorithm requires only one pass through the training set; other equivalent methods of finding the weights are described in Section 3.1.1, where the Hebb rule for pattern association (in which the target is a vector) is presented.

Application

Bias types of inputs are not explicitly used in the original formulation of Hebb learning. However, they are included in the examples in this section (shown as a third input component that is always 1) because without them, the problems discussed cannot be solved.

Logic functions

Example 2.5 A Hebb net for the AND function: binary inputs and targets

INPUT			TARGET
(x_1	x_2	1)	
(1	1	1)	1
(1	0	1)	0
(0	1	1)	0
(0	0	1)	0

For each training input: target, the weight change is the product of the input vector and the target value, i.e.,

For each training input: target, the weight change is the product of the input vector and the target value, i.e.,

$$\Delta w_1 = x_1 t, \qquad \Delta w_2 = x_2 t, \qquad \Delta b = t.$$

Application

Logic functions

Example A Hebb net for the AND function: binary inputs and targets

INPUT			TARGET
(x_1	x_2	1)	
(1	1	1)	1
(1	0	1)	0
(0	1	1)	0
(0	0	1)	0

$$\Delta w_1 = x_1 t, \qquad \Delta w_2 = x_2 t, \qquad \Delta b = t.$$

The new weights are the sum of the previous weights and the' weight change. Only one iteration through the training vectors is required. The weight updates for the first input are as follows:

INPUT	TARGET	WEIGHT CHANGES	WEIGHTS
$(x_1 \quad x_2 \quad 1)$		$(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	$(w_1 \quad w_2 \quad b)$
			(0 0 0)
(1 1 1)	1	(1 1 1)	(1 1 1)

INPUT	TARGET	WEIGHT CHANGES	WEIGHTS
$(x_1 \quad x_2 \quad 1)$		$(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	$(w_1 \quad w_2 \quad b)$
(1 0 1)	0	(0 0 0)	(1 1 1)
(0 1 1)	0	(0 0 0)	(1 1 1)
(0 0 1)	0	(0 0 0)	(1 1 1)

Example 2 A Hebb net for the AND function: binary inputs bipolar targets

INPUT			TARGET
$(x_1$	x_2	1)	
(1	1	1)	1
(1	0	1)	−1
(0	1	1)	−1
(0	0	1)	−1

INPUT			TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	1)		$(\Delta w_1$	Δw_2	$\Delta b)$	$(w_1$	w_2	$b)$
							(0	0	0)
(1	1	1)	1	(1	1	1)	(1	1	1)

INPUT			TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	1)		$(\Delta w1$	Δw_2	$\Delta b)$	$(w_1$	w_2	$b)$
(1	0	1)	-1	$(-1$	0	$-1)$	(0	1	0)
(0	1	1)	-1	$(0$	-1	$-1)$	(0	0	$-1)$
(0	0	1)	-1	(0	0	$-1)$	$(0$	0	$-2)$

For (binary input and binary output) and (binary input and bipolar output) the Final weight Does not achieve all the Targets

Example A Hebb net for the AND function: bipolar inputs and targets

INPUT			TARGET
$(x_1$	x_2	1)	
(1	1	1)	1
(1	−1	1)	−1
(−1	1	1)	−1
(−1	−1	1)	−1

INPUT			TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	1)		$(\Delta w_1$	Δw_2	$\Delta b)$	$(w_1$	w_2	$b)$
							(0	0	0)
(1	1	1)	1	(1	1	1)	(1	1	1)
(1	−1	1)	−1	(−1	1	−1)	(0	2	0)
(−1	1	1)	−1	(1	−1	−1)	(1	1	−1)
(−1	−1	1)	−1	(1	1	−1)	(2	2	−2)

Final Weights

Character Recognition

Ex / classify the two dimensional input pattern (representing letters) **using Hebb. Rule** (T , C)

Solution

Training Pattern

Pattern	Input										Target
	X1	X2	X3	X4	X5	X6	X7	X8	X9	b	Y
T	1	1	1	-1	1	-1	-1	1	-1	1	1
C	1	1	1	1	-1	-1	1	1	1	1	-1

For the training pair

T: $X_i = [1\ 1\ 1, -1\ 1\ -1, -1\ 1\ -1, 1]^T$ and $Y_1 = [1]$ Target

$W_{new} = W_{old} + X_1^T\ Y_1$

$= [0\ 0\ 0, 0\ 0\ 0, 0\ 0\ 0, 0]^T + [1\ 1\ 1, -1\ 1\ -1, -1\ 1\ -1, 1]^T * [1]$

$= [1\ 1\ 1, -1\ 1\ -1, -1\ 1\ -1, 1]^T$

Case 2:

C: $X_2 = [1\ 1\ 1, 1\ -1\ -1, 1\ 1\ 1, -1]^T$ and $Y_1 = [-1]$ Target

$W_{new} = W_{old} + X_2^T\ Y_2$

$= [1\ 1\ 1, -1\ 1\ -1, -1\ 1\ -1, 1]^T + [1\ 1\ 1, 1\ -1\ -1, 1\ 1\ 1, -1]^T * [-1]$

$= [1\ 1\ 1, -1\ 1\ -1, -1\ 1\ -1, 1]^T + [-1\ -1\ -1, -1\ 1\ 1, -1\ -1\ -1, -1]^T$

$= [0\ 0\ 0, -2\ 2\ 0, -2\ 0\ -2, 0]$

$Y = b + \sum X_i W_j$

$= 0 + [1\ 1\ 1, -1\ 1\ -1, -1\ 1\ -1] * [0\ 0\ 0, -2\ 2\ 0, -2\ 0\ -2]$

$= 0 + (0 + 0 + 0 + 2 + 2 + 0 + 2 + 0 + 2) = 8 > 0$

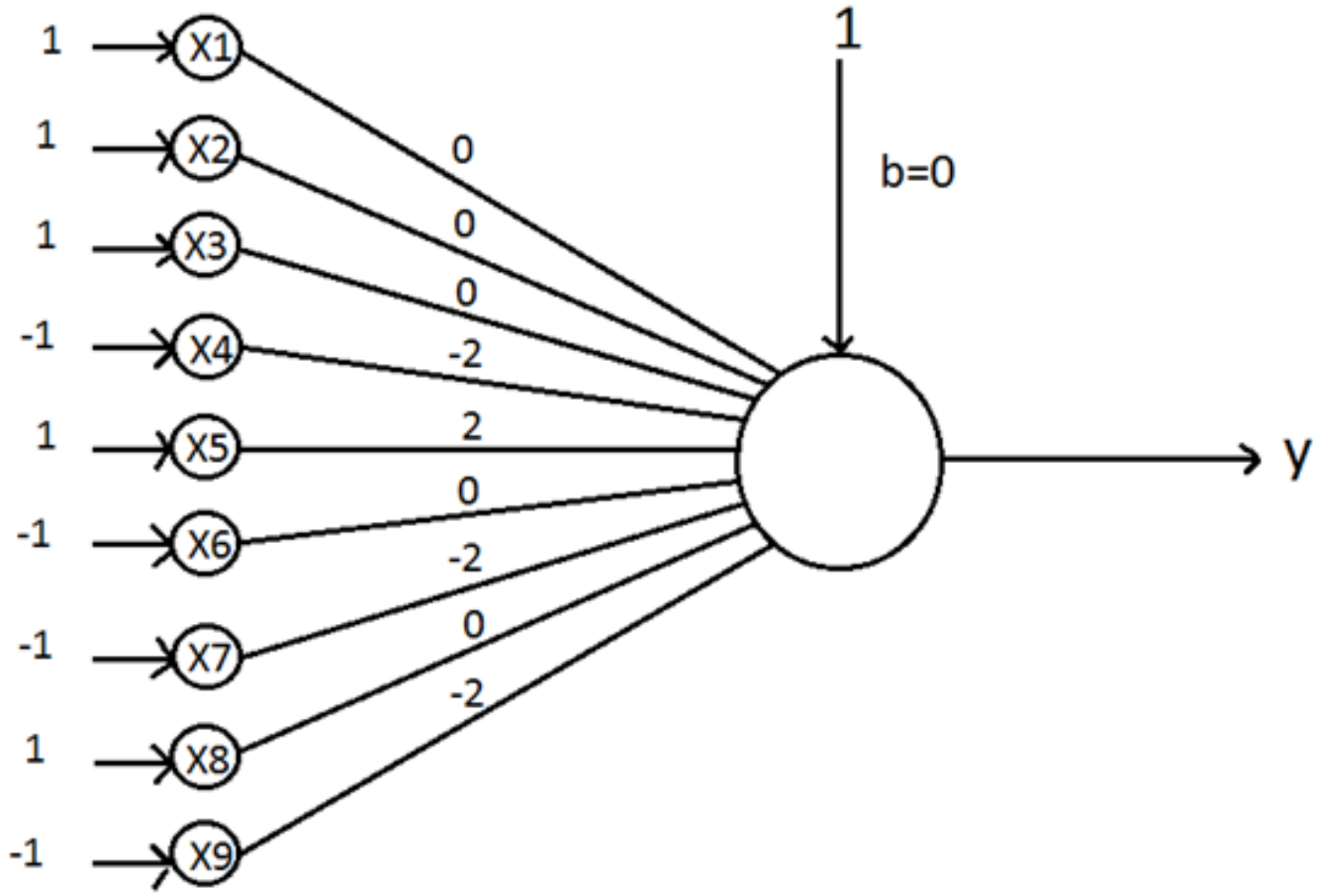
Then the letter is T

$Y = b + \sum X_i W_j$

$= 0 + [1\ 1\ 1, 1\ -1\ -1, 1\ 1\ 1] * [0\ 0\ 0, -2\ 2\ 0, -2\ 0\ -2]$

$= 0 + (0 + 0 + 0 - 2 - 2 + 0 - 2 + 0 - 2 = -8 < 0$

Then the letter is C


1
X1
1
X2
1
X3
-1
X4
1
X5
-1
X6
-1
X7
1
X8
-1
X9
0
0
0
-2
2
0
-2
0
-2
1
b=0
y

EXAMPLE 2.4

This example illustrates Hebbian learning with binary and continuous activation functions of a very simple network. Assume the network shown in Figure 2.22 with the initial weight vector

$$\mathbf{w}^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

needs to be trained using the set of three input vectors as below

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix}, \quad \mathbf{x}_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

for an arbitrary choice of learning constant $c = 1$. Since the initial weights are of nonzero value, the network has apparently been trained beforehand. Assume first that bipolar binary neurons are used, and thus $f(net) = \text{sgn}(net)$.

Step 1 Input $\mathbf{x}_1$ applied to the network results in activation net^1 as below:

$$net^1 = \mathbf{w}^{1t}\mathbf{x}_1 = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = 3$$

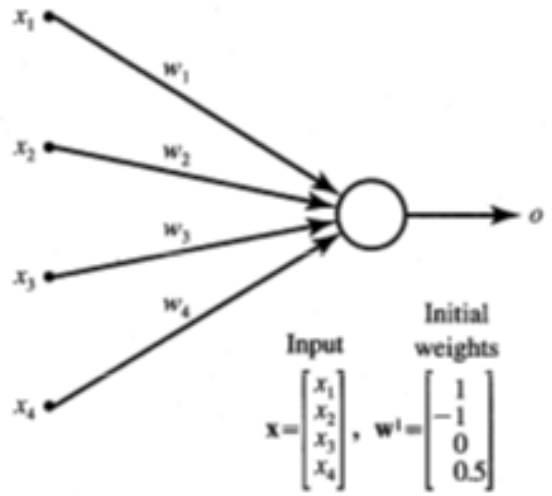


Figure 2.22 Network for training in Examples 2.4 through 2.6.

The updated weights are

$$\mathbf{w}^2 = \mathbf{w}^1 + \text{sgn}(net^1)\mathbf{x}_1 = \mathbf{w}^1 + \mathbf{x}_1$$

and plugging numerical values we obtain

$$\mathbf{w}^2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 1.5 \\ 0.5 \end{bmatrix}$$

where the superscript on the right side of the expression denotes the number of the current adjustment step.

Step 2 This learning step is with $\mathbf{x}_2$ as input:

$$net^2 = \mathbf{w}^{2t}\mathbf{x}_2 = \begin{bmatrix} 2 & -3 & 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = -0.25$$

The updated weights are

$$\mathbf{w}^4 = \mathbf{w}^3 + \operatorname{sgn}(net^3)\mathbf{x}_3 = \mathbf{w}^3 - \mathbf{x}_3 = \begin{bmatrix} 1 \\ -3.5 \\ 4.5 \\ 0.5 \end{bmatrix}$$

It can be seen that learning with discrete $f(net)$ and $c = 1$ results in adding or subtracting the entire input pattern vectors to and from the weight vector, respectively. In the case of a continuous $f(net)$, the weight incrementing/decrementing vector is scaled down to a fractional value of the input pattern.

Revisiting the Hebbian learning example, with continuous bipolar activation function $f(net)$, using input $\mathbf{x}_1$ and initial weights $\mathbf{w}^1$, we obtain neuron output values and the updated weights for $\lambda = 1$ as summarized in Step 1. The only difference compared with the previous case is that instead of $f(net) = \operatorname{sgn}(net)$, now the neuron's response is computed from (2.3a).

Step 1

$$f(net^1) = 0.905$$

$$\mathbf{w}^2 = \begin{bmatrix} 1.905 \\ -2.81 \\ 1.357 \\ 0.5 \end{bmatrix}$$

Subsequent training steps result in weight vector adjustment as below:

Step 2

$$f(net^2) = -0.077$$

$$\mathbf{w}^3 = \begin{bmatrix} 1.828 \\ -2.772 \\ 1.512 \\ 0.616 \end{bmatrix}$$

Step 3

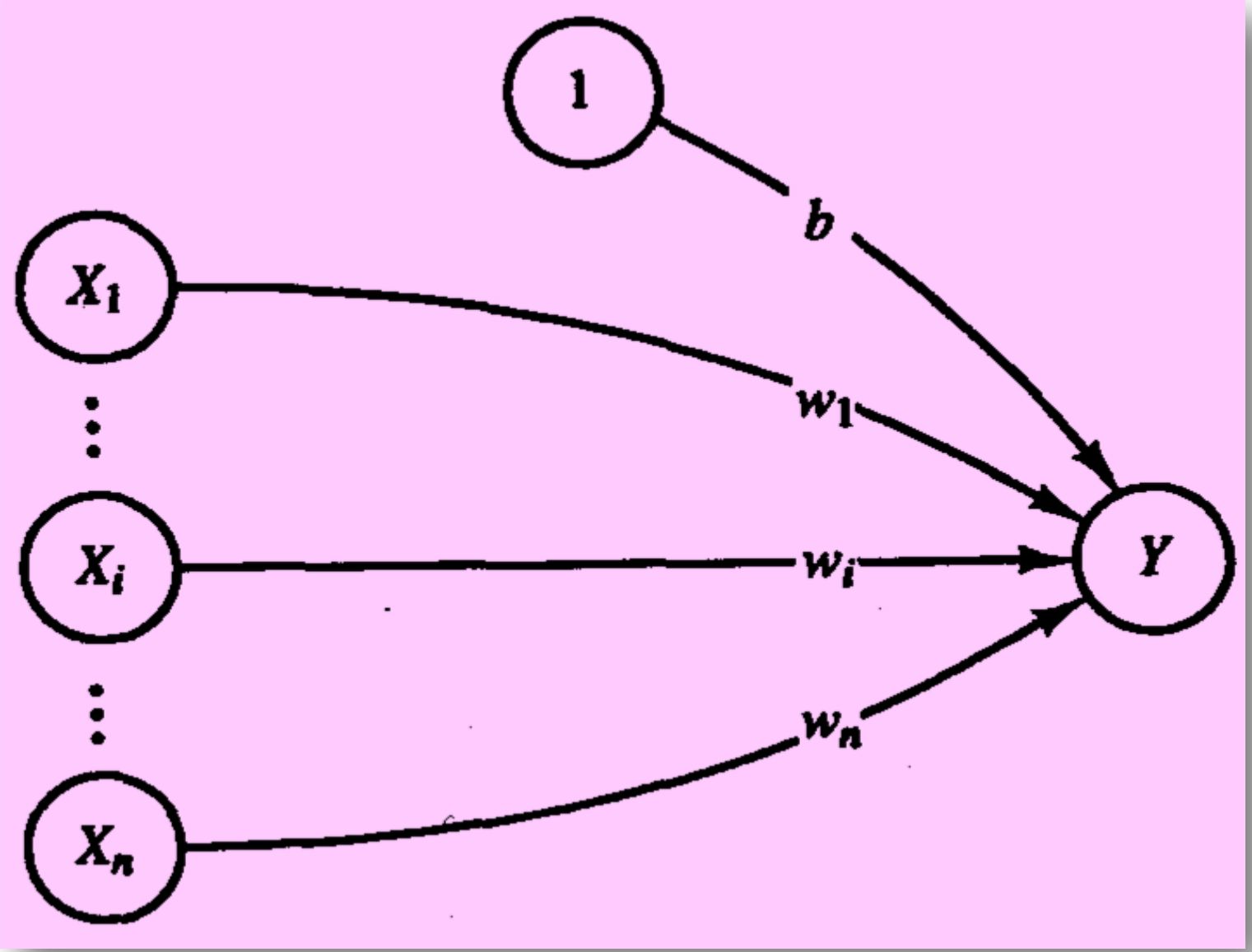
$$f(net^3) = -0.932$$

$$\mathbf{w}^4 = \begin{bmatrix} 1.828 \\ -3.70 \\ 2.44 \\ -0.783 \end{bmatrix}$$

Comparison of learning using discrete and continuous activation functions indicates that the weight adjustments are tapered for continuous $f(net)$ but are generally in the same direction. ■

PERCEPTRON

Architecture



Algorithm

Step 0: ***Initialize weights and bias .***

(For simplicity, set weights and bias to zero.)

Set learning rate a **(0 < a ≤ 1).** (For simplicity, **a** can be set to **1**.)

Step 1: ***While stopping condition is false, do Steps 2-6 .***

Step 2: ***For each training pair s:t, do Steps 3-5 .***

Step 3: ***Set activations of input units .***

$$X_i = S_i$$

Step 4: ***Compute response of output unit .***

$$Y\_in = b + \sum_i X_i * W_i$$

$$Y = \begin{cases} 1 & \text{If } Y\_in > \theta \\ 0 & \text{If } -\theta \le Y\_in \le \theta \\ -1 & \text{If } Y\_in < -\theta \end{cases}$$

Step 5: ***Update weights and bias if an error occurred .***

If $(Y \neq t)$ ➢

$$W_i(new) = W_i(old) + \alpha t X_i$$
$$b(new) = b(old) + \alpha t$$

else ➢

$$W_i(new) = W_i(old)$$
$$b(new) = b(old)$$

Step 6: ***Test stopping .***

If no weights changed in Step 2, stop; else: continue.

Application

Example 2.11 :

A Perceptron for the AND function: binary inputs , bipolar targets
take (a = 1) and (θ = 0.2) .

Solution :-

- **The first epoch**

INPUT			NET	OUT	TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	$1)$							$(w_1$	w_2	$b)$
									(0	0	0)
(1	1	1)	0	0	1	(1	1	1)	(1	1	1)
(1	0	1)	2	1	−1	(−1	0	−1)	(0	1	0)
(0	1	1)	1	1	−1	(0	−1	−1)	(0	0	−1)
(0	0	1)	−1	−1	−1	(0	0	0)	(0	0	−1)

❖ The second epoch

INPUT			NET	OUT	TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	1)							$(w_1$	w_2	$b)$
									(0	0	$-1)$
(1	1	1)	-1	-1	1	(1	1	1)	(1	1	0)

INPUT			NET	OUT	TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	1)							$(w_1$	w_2	$b)$
									(1	1	0)
(1	0	1)	1	1	-1	$(-1$	0	$-1)$	(0	1	$-1)$

INPUT			NET	OUT	TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	1)							$(w_1$	w_2	$b)$
									(0	1	$-1)$
(0	1	1)	0	0	-1	(0	-1	$-1)$	(0	0	$-2)$

INPUT			NET	OUT	TARGET	WEIGHTS CHANGE			WEIGHTS		
$(x_1$	x_2	1)							$(w_1$	w_2	$b)$
									(0	0	$-2)$
(0	0	1)	-2	-1	-1	(0	0	0)	(0	0	$-2)$

The eighth epoch yields

(1	1	1)	−1	−1	1	(1	1	1)	(2	3	−3)
(1	0	1)	−1	−1	−1	(0	0	0)	(2	3	−3)
(0	1	1)	0	0	−1	(0	−1	−1)	(2	2	−4)
(0	0	1)	−4	−1	−1	(0	0	0)	(2	2	−4)

and the ninth

(1	1	1)	0	0	1	(1	1	1)	(3	3	−3)
(1	0	1)	0	0	−1	(−1	0	−1)	(2	3	−4)
(0	1	1)	−1	−1	−1	(0	0	0)	(2	3	−4)
(0	0	1)	−4	−1	−1	(0	0	0)	(2	3	−4)

Finally, the results for the tenth epoch are:

(1	1	1)	1	1	1	(0	0	0)	(2	3	−4)
(1	0	1)	−2	−1	−1	(0	0	0)	(2	3	−4)
(0	1	1)	−1	−1	−1	(0	0	0)	(2	3	−4)
(0	0	1)	−4	−1	−1	(0	0	0)	(2	3	−4)

ADALINE

- ADALINE (Adaptive Linear Neuron) typically uses bipolar (1 or -1) activations for its input signals and its target output (although it is not restricted to such values). The ADALINE also has a bias, which acts like an adjustable weight on a connection from a unit whose activation is always 1.

algorithm

Step *0.* Initialize weights.
(Small random values are usually used.)
Set learning rate **a.**
(See comments following algorithm.)

Step 1. While stopping condition is false, do Steps 2–6.

Step 2. For each bipolar training pair s:t, do Steps 3–5.

Step 3. Set activations of input units, i = 1, . . . , n:

$$x_i = s_i.$$

Step 4. Compute net input to output unit:

$$y\_in = b + \sum_i x_i w_i.$$

Step 5. Update bias and weights, i = 1, . . . , n:

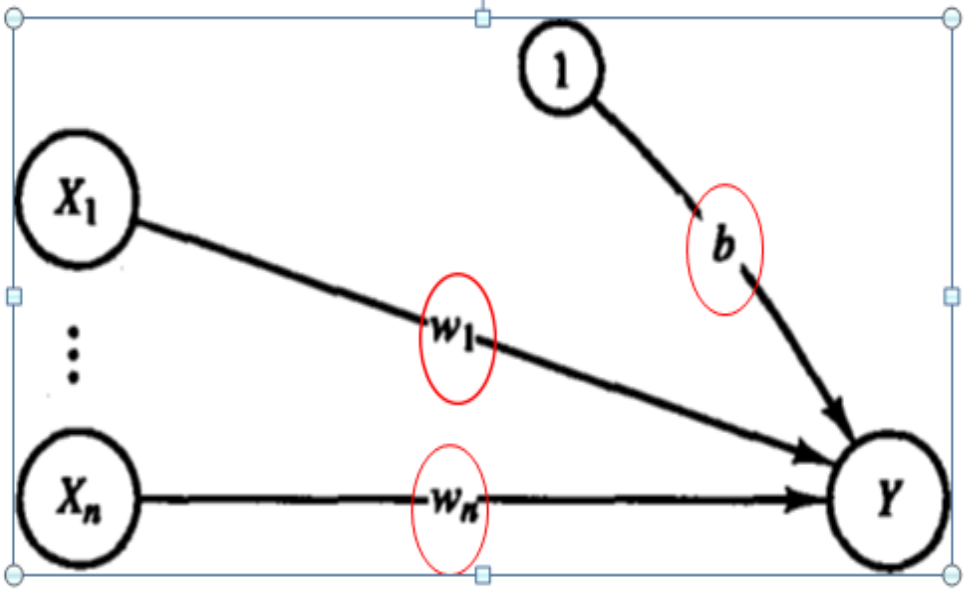
$$b(\text{new}) = b(\text{old}) + a(t - yin).$$

$$w_i(\text{new}) = w_i(\text{old}) + \alpha(t - y\_in)x_i.$$

Step 6. Test for stopping condition:
If the largest weight change that occurred in Step **2** is smaller than a specified tolerance, then stop; otherwise continue.

ARCHITECTURE

An Adaline is a single unit (neuron) that receives input from several units. It also receives input from a "unit" whose signal is always + 1, in order for the bias weight to be trained by the same process (the delta rule) as is used to train the other weights.



APPLICATIONS

After training, an Adaline unit can be used to classify input patterns. If the target values are bivalent (binary or bipolar), a step function can be applied as the activation function for the output unit. The following procedure shows the step function for bipolar targets, the most common case:

***Step* 0.** Initialize weights
(from Adaline training algorithm given in Section 2.4.2).

Step 1. For each bipolar input vector **x**, do Steps 2–4.

***Step* 2.** Set activations of the input units to **x**.

Step 3. Compute net input to output unit:

$$y\_in = b + \sum_i x_i w_i.$$

Step 4. Apply the activation function:

$$y = \begin{cases} 1 & \text{if } y\_in \geq 0; \\ -1 & \text{if } y\_in < 0. \end{cases}$$

EXAMPLE

- An adaline for the OR_GATE bipolar input and target with the initial weight (0.1)and the learning rate (0.1)

X1	X2	B	T	YIN	T-YIN	ΔW1	ΔW2	ΔB	W1 (0.1)	W2 (0.1)	B (0.1)	ERROR
1	1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	1	0.17	0.83	0.083	-0.083	0.083	0.253	0.087	0.253	0.6889
-1	1	1	1	0.087	0.913	-0.09113	0.09113	0.09113	0.16117	0.17883	0.34343	0.83356
-1	-1	1	-1	0.00043	-1.00043	0.10043	0.10043	-0.10043	0.26213	0.27887	0.24239	1.00861849 E=3.02107849

X1	X2	B	T	YIN	T-YIN	ΔW1	ΔW2	ΔB	W1 (0.1)	W2 (0.1)	B (0.1)	ERROR
1	1	1	1	0.78473	0.21557	0.021557	0.021557	0.021557	0.28368	0.30003	0.2654	0.046470
1	-1	1	1	0.2488	0.7512	0.07 51	-0.07 51	0.07 51	0.3588	0.2251	0.3405	0.5643014
-1	1	1	1	0.2069	0.7931	-0.07 93	0.07 93	0.07 93	0.2795	0.3044	0.4198	0.6290076 1
-1	-1	1	-1	-0.1641	-0.8359	0.0836	0.0836	-0.0836	0.3631	0.388	0.3362	0.6987281 E=1.984977

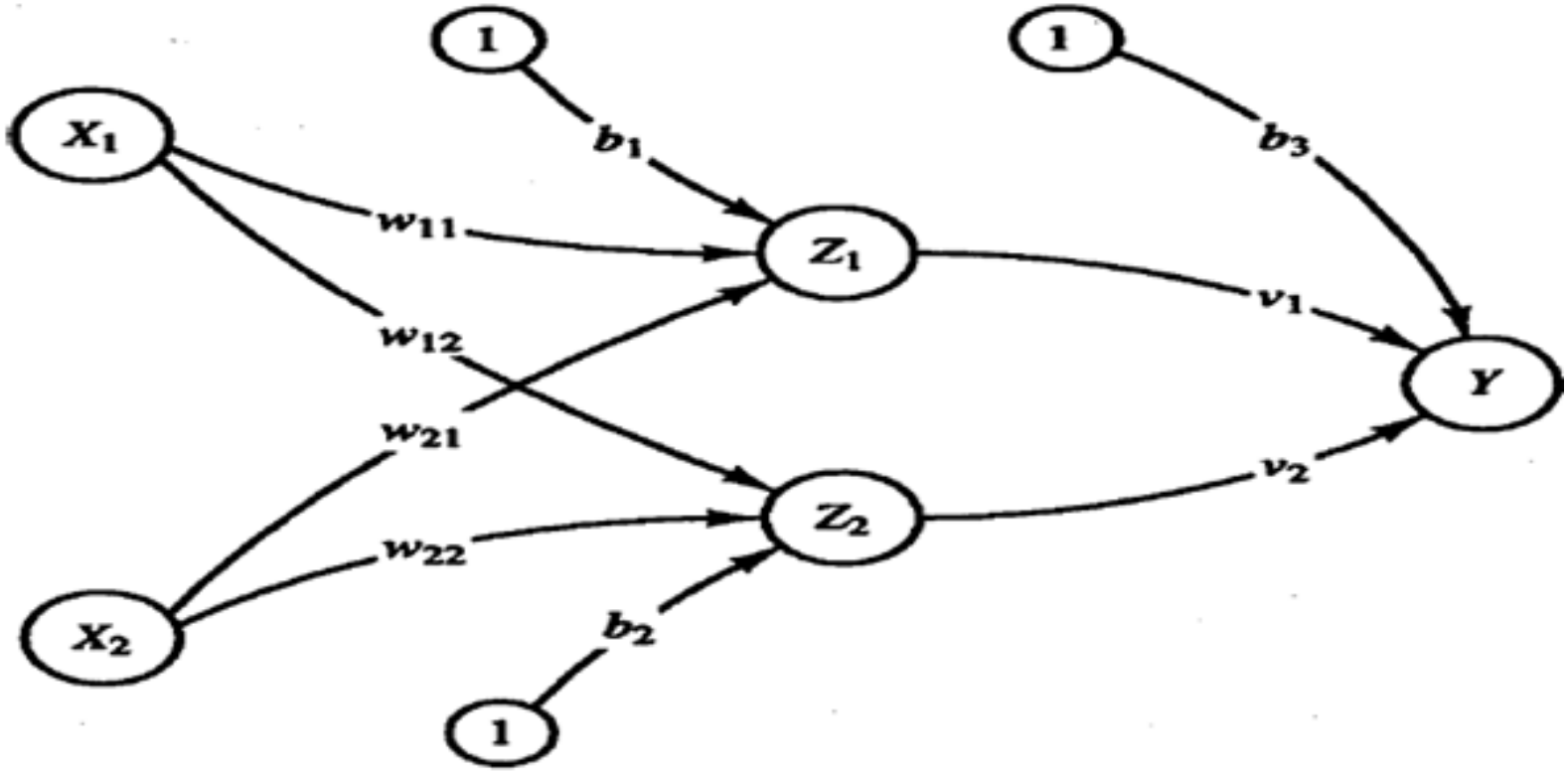
X1	X2	B	T	YIN	T-YIN	ΔW1	ΔW2	ΔB	W1 (0.1)	W2 (0.1)	B (0.1)	ERROR
1	1	1	1	1.0873	0.0873	-0.0087	0.0087	-0.0087	0.3543	0.3793	0.327	0.00762129
1	-1	1	1	0.3025	0.6975	0.0697	-0.0697	0.0697	0.4241	0.3096	0.3973	0.48650625
-1	1	1	1	0.2827	0.7173	-0.0717	0.0717	0.0717	0.3523	0.3813	0.4690	0.51451929
-1	-1	1	-1	-0.2647	-0.7353	0.0735	0.0735	-0.0735	0.4259	0.4548	0.3954	0.54125999 E=1.54990132

X1	X2	B	T	YIN	T-YIN	ΔW1	ΔW2	ΔB	W1 (0.1)	W2 (0.1)	B (0.1)	ERROR
1	1	1	1	1.2761	0.2761	-0.0276	-0.0276	-0.0276	0.17	0.3983	0.3678	0.0762312 1
1	-1	1	1	0.3389	0.6611	0.0661	-0.0661	0.0661	0.253	0.4644	0.4339	0.4370532 1
-1	1	1	1	0.3307	0.6693	-0.0669	0.0669	0.0669	0.1617	0.3974	0.5009	0.4479624 9
-1	-1	1	-1	-0.3246	-0.6754	0.0675	0.0675	-0.0675	0.26213	0.4650	0.4333	0.4561651 6 E=1.41741207

X1	X2	B	T	YIN	T-YIN	ΔW1	ΔW2	ΔB	W1 (0.1)	W2 (0.1)	B (0.1)	ERROR
1	1	1	1	1.3939	-0.3939	-0.03939	-0.03939	-0.03939	0.4256	0.4562	0.3939	0.1551571
1	-1	1	1	0.3634	0.6366	0.06366	-0.06366	0.06366	0.4893	0.392	0.4576	0.4052595
-1	1	1	1	0.3609	0.6391	-0.06391	0.06391	0.06391	0.4253	0.456	0.5215	0.4084481
-1	-1	1	-1	-0.3609	-0.6397	0.06397	0.06397	-0.06397	0.4893	0.520	0.4575	0.4092160 9 E=1.37808167

E=1.37808167

MADALINE



Algorithm

- MRI algorithm only the weights for the hidden Adalines are adjusted; the weights for the output unit are fixed.
- The MRII algorithm provides a method for adjusting all weights in the net.

Training Algorithm for Madaline (MRI). The activation function for units Z_1, Z_2, and Y is

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ -1 & \text{if } x < 0. \end{cases}$$

Step 1. While stopping condition is false, do Steps 2–8.

Step 2. For each bipolar training pair, s:t, do Steps 3–7.

Step 3. Set activations of input units:

$$x_i = s_i.$$

Step 4. Compute net input to each hidden Adaline unit:

$$z\_in_1 = b_1 + x_1 w_{11} + x_2 w_{21},$$

$$z\_in_2 = b_2 + x_1 w_{12} + x_2 w_{22}.$$

Step 5. Determine output of each hidden Adaline unit:

$$z_1 = f(z\_in_1),$$

$$z_2 = f(z\_in_2).$$

Step 6. Determine output of net:

$$y\_in = b_3 + z_1 v_1 + z_2 v_2;$$

$$y = f(y\_in).$$

Step 7. Determine error and update weights:
If $t = y$, no weight updates are performed.
Otherwise:
If $t = 1$, then update weights on Z_J, the unit whose net input is closest to **0**,

$$b_J(\text{new}) = b_J(\text{old}) + \alpha(1 - z\_in_J),$$

$$w_{iJ}(\text{new}) = w_{iJ}(\text{old}) + \alpha(1 - z\_in_J)x_i;$$

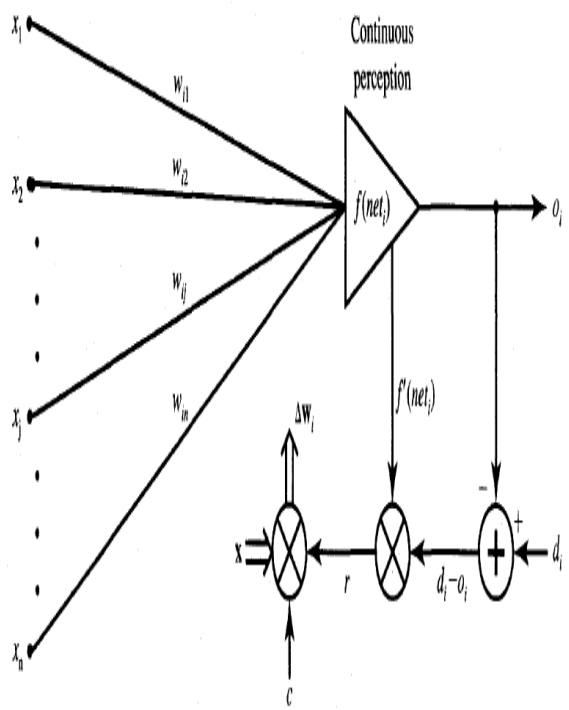
If $t = -1$, then update weights on all units Z_k that have positive net input,

$$b_k(\text{new}) = b_k(\text{old}) + \alpha(-1 - z\_in_k),$$

$$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha(-1 - z\_in_k)x_i.$$

Step 8. Test stopping condition.
If weight changes have stopped (or reached an acceptable level), or if a specified maximum number of weight update iterations (Step 2) have been performed, then stop; otherwise continue.

Delta Learning Rule



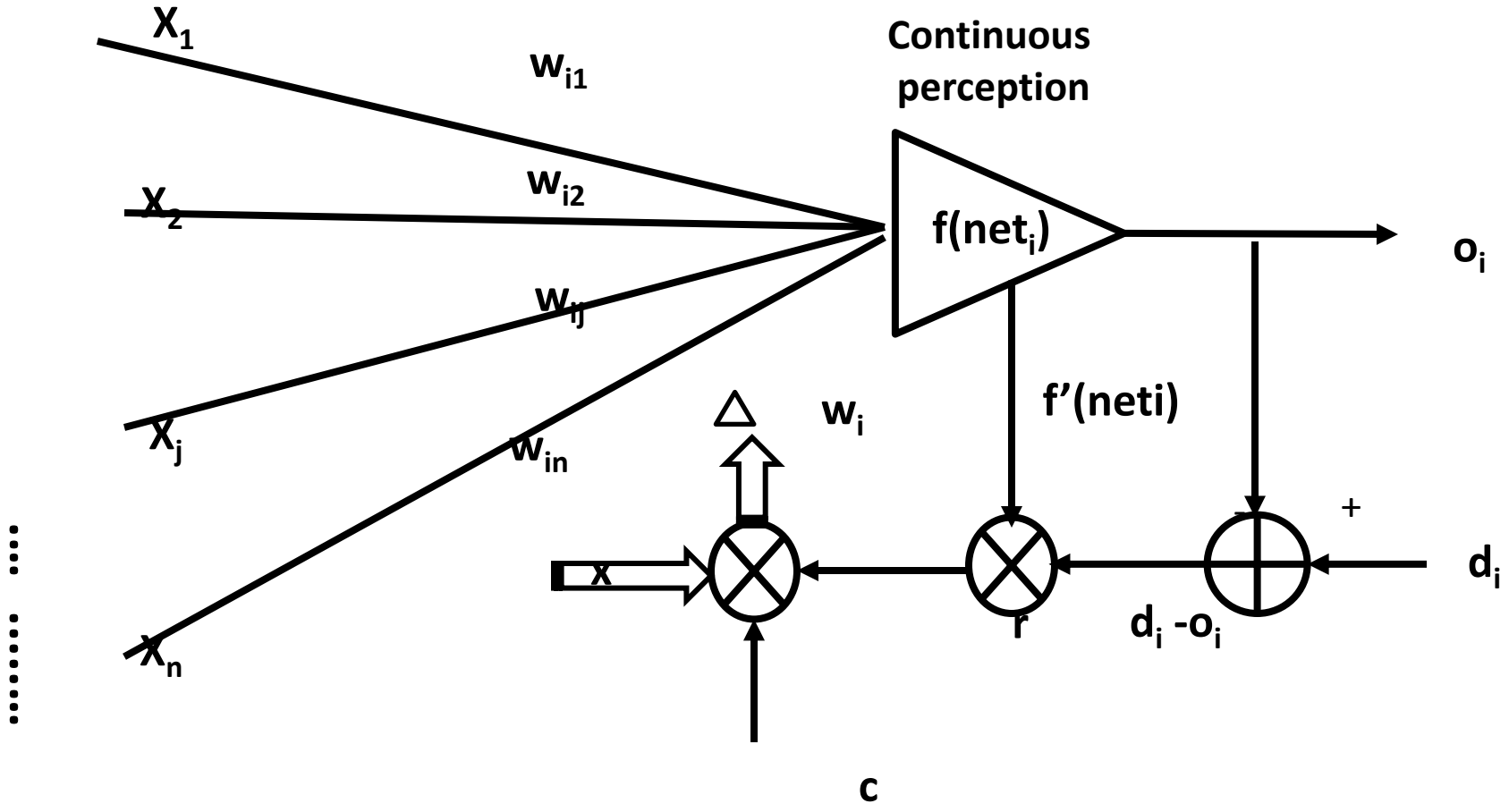
Delta Learning Rule

- The delta learning rule is only valid for continuous activation functions.
- It is a supervised training mode.
- The single weight w is updated using the following increment:

$$\Delta \mathbf{w}_i = c(d_i - o_i) f'(\text{net}_i)\mathbf{x}$$

- Where f'(neti) is the derivate of the activation function f(neti).

Delta Learning Rule



Example

$$\left[x1 = \begin{pmatrix} 1 \\ -2 \\ 0 \\ -1 \end{pmatrix} \quad x2 = \begin{pmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{pmatrix} \quad x3 = \begin{pmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{pmatrix} \quad w1 = \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{pmatrix} \right]$$

c = 0.1 and ƛ = 1. The desired responses for x_1, x_2, and x_3 are d_1= -1, d_2= -1, d_3= 1. Use continues bipolar activation function $f(net) = \frac{2}{1+\exp(-\lambda net)} - 1$

$$F'(net) = \frac{1}{2}(1 - Oi^2)$$

Step 1 :-

Input is xl, desired output is dl :

$$Net^1 = w^{1t} * X^1 = [1\ -1\ 0\ 0.5] \begin{pmatrix} 1 \\ -2 \\ 0 \\ -1 \end{pmatrix} = 2.5$$

$$O^1 = f(net^1) = \frac{2}{1+e\ ^{\wedge}-2.5} - 1 = 0.848 \text{ di} \neq Oi$$

We thus obtain updated weight vector

$$f'(net^1) = \frac{1}{2}[\ 1-(O^1)^2] = = \frac{1}{2}[\ 1-(0.848)^2] = 0.140$$

$$\Delta w^1 = c\ (\ d^1 - O^1\ ) * f'(net^1) * X^1$$

$$= 0.1 * (-1 - 0.848)\ (\ 0{,}140\ ) * \begin{pmatrix} 1 \\ -2 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} -0.0258 \\ 0.0517 \\ 0 \\ 0.0258 \end{pmatrix}$$

$$W^{i+1} = \Delta w^i + W^i$$

$$W^2 = \Delta w^1 + W^1 = \begin{pmatrix} -0.0258 \\ 0.0517 \\ 0 \\ 0.0258 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.526 \end{pmatrix}$$

Step 2 :-

Input is x2, desired output is d2 :

$$F(Net^2) = w^{2t} * X^2 = \begin{pmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.526 \end{pmatrix}^T * \begin{pmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{pmatrix} = -1.948$$

$$O^2 = f(net^2) = \frac{2}{1+e\ ^{\wedge}1.948} - 1 = -0.750 \quad di \neq Oi$$

We thus obtain updated weight vector

$$f'(net^2) = \frac{1}{2}[1-(O^2)^2] = = \frac{1}{2}[1-(-0.750)^2] = 0.218$$

$$\Delta w^2 = c\,(d^2 - O^2) * f'(net^2) * X^2$$

$$= 0.1 * (-1 + 0.750)\ 0.218 * \begin{pmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ -8.175 * 10^{-3} \\ 2.725 * 10^{-3} \\ 5.45 * 10^{-3} \end{pmatrix}$$

$$W^{i+1} = \Delta w^i + W^i$$

$$W^3 = \Delta w^2 + W^2 = \begin{pmatrix} 0 \\ -8.175 * 10^{-3} \\ 2.725 * 10^{-3} \\ 5.45 * 10^{-3} \end{pmatrix} + \begin{pmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.526 \end{pmatrix} = \begin{pmatrix} 0.974 \\ -0.956 \\ 0.002 \\ 0.531 \end{pmatrix}$$

Step 3 :-

Input is x3, desired output is d3 :

$$F(Net^3)_= w^{3t} * X^3 = \begin{pmatrix} 0.974 \\ -0.956 \\ 0.002 \\ 0.531 \end{pmatrix}^T * \begin{pmatrix} -1 \\ 1 \\ 0.5 \\ -0.5 \end{pmatrix} = -2.46$$

$$O^3 = f(net^3) = \frac{2}{1+e\ ^{\wedge}2.46} -1 = -0.842 \quad di \neq Oi$$

We thus obtain updated weight vector

$$f'(net^3) = \frac{1}{2}[1-(O^3)^2] = = \frac{1}{2}[1-(-0.842)^2] = 0.145$$

$$\Delta w^3 = c\,(d^3 - O^3) * f'(net^3) * X^3$$

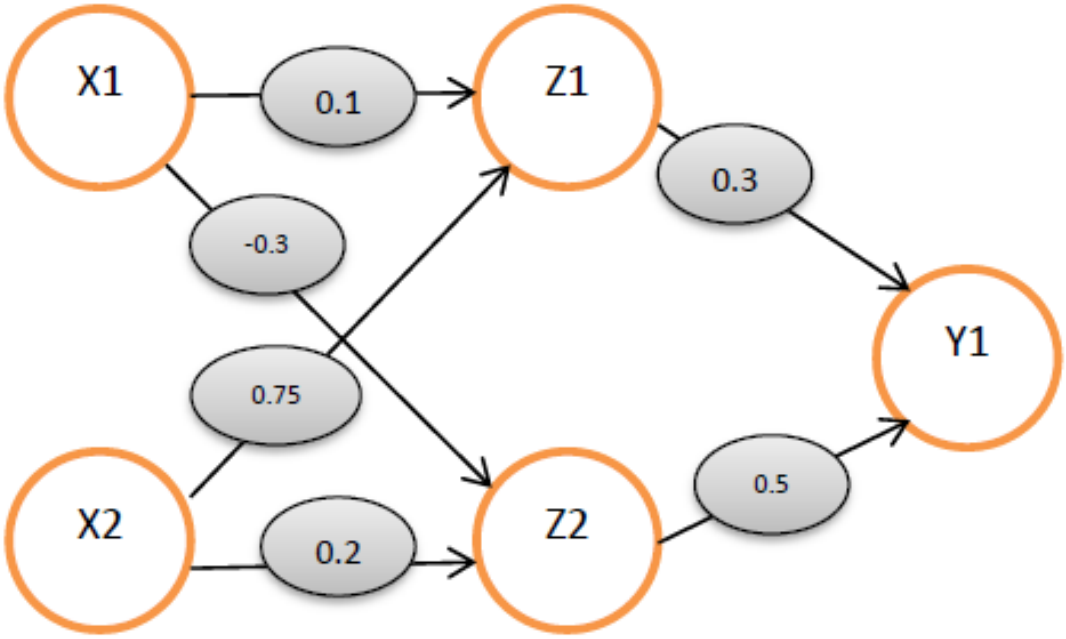
$$= 0.1 * (1 + 0.842) * 0.145 * \begin{pmatrix} -1 \\ 1 \\ 0.5 \\ -0.5 \end{pmatrix} = \begin{pmatrix} -0.026 \\ 0.026 \\ 0.0133 \\ -0.0133 \end{pmatrix}$$

$$W^{i+1} = \Delta w^i + W^i$$

$$W^4 = \Delta w^3 + W^3 = \begin{pmatrix} -0.026 \\ 0.026 \\ 0.0133 \\ -0.0133 \end{pmatrix} + \begin{pmatrix} 0.974 \\ -0.956 \\ 0.002 \\ 0.531 \end{pmatrix} = \begin{pmatrix} 0.974 \\ -0.929 \\ 0.016 \\ 0.505 \end{pmatrix}$$

Backpropagation

Example 1: Suppose you have Bp-ANN with 2-inputs, 2-hidden, 1-output with sigmoid function and the following weights, trace with 1-iteration, where $\alpha = 0.9$, $\eta = 0.45$, $x = (1, 0)$, and $T_k = 1$



1. Forward phase:

Zin1=x1 V11+x2 V21=1*0.1+0*0.75=0.1

$$Z1=f(Zin1)=\frac{1}{1+e^{-0.1}}=0.524$$

Zin2=x1 V12+x2 V22=1*-0.3+0*0.2= -0.3

$$Z2=f(Zin2)=\frac{1}{1+e^{-(-0.3)}}=0.426$$

Yin1=Z1 w11+Z2 w21=0.524*0.3+0.426*0.5= 0.37

$$Y1=f(Y1)=\frac{1}{1+e^{-(0.37)}}=0.59$$

2. Backward phase

$\delta_{yk}=yk(1-yk)*(Tk-yk)$

$\delta_{21}=0.59*(1-0.59)*(1-0.59)=0.\ 099$

$\delta_{1j}=Zj*(1-zj)*\sum \delta_{2k}\ wjk$

$\delta_{11}= Z1*(1-z1)*\sum \delta_{21}\ w11= 0.524*(1-0.524)*(0.099*0.3)=0.0074$

$\delta_{12}=Z2(1-Z2)*\ \delta_{21}w21=0.426*(1-0.426)*(0.099*0.5)=\ 0.012$

3. Update weights phase

$W_{jk}(new) = \eta * \delta\ 2k * zj * + \alpha[wjk\ (old)]$

$W11 = \eta * \delta\ 21 * z1 + \alpha * [w11]$

$= 0.45 * 0.099 * 0.524 + 0.9 * 0.3 = 0.293$

$W21 = \eta * \delta\ 21 * z2 + \alpha * [w12]$

$= 0.45 * 0.099 * 0.426 + 0.9 * 0.5 = 0.468$

$Vij(new) = \eta * \delta\ 1j * xi + \alpha * [vij(old)]$

$V11 = \eta * \delta\ 11 * x1 + \alpha\ [v11]$

$= 0.45 * 0.0074 * 1 + 0.9 * 0.1 = 0.0933$

$V12 = \eta * \delta\ 12 * x1 + \alpha * [v12]$

$$V12 = \eta * \delta\ 12 * x1 + \alpha * [v12]$$

$$= 0.45 * 0.012 * 1 + 0.9 * -0.3 = -0.2646$$

$$V21 = \eta * \delta\ 11 * x2 + \alpha * [v21]$$

$$= 0.45 * 0.0074 * 0 + 0.9 * 0.75 = 0.675$$

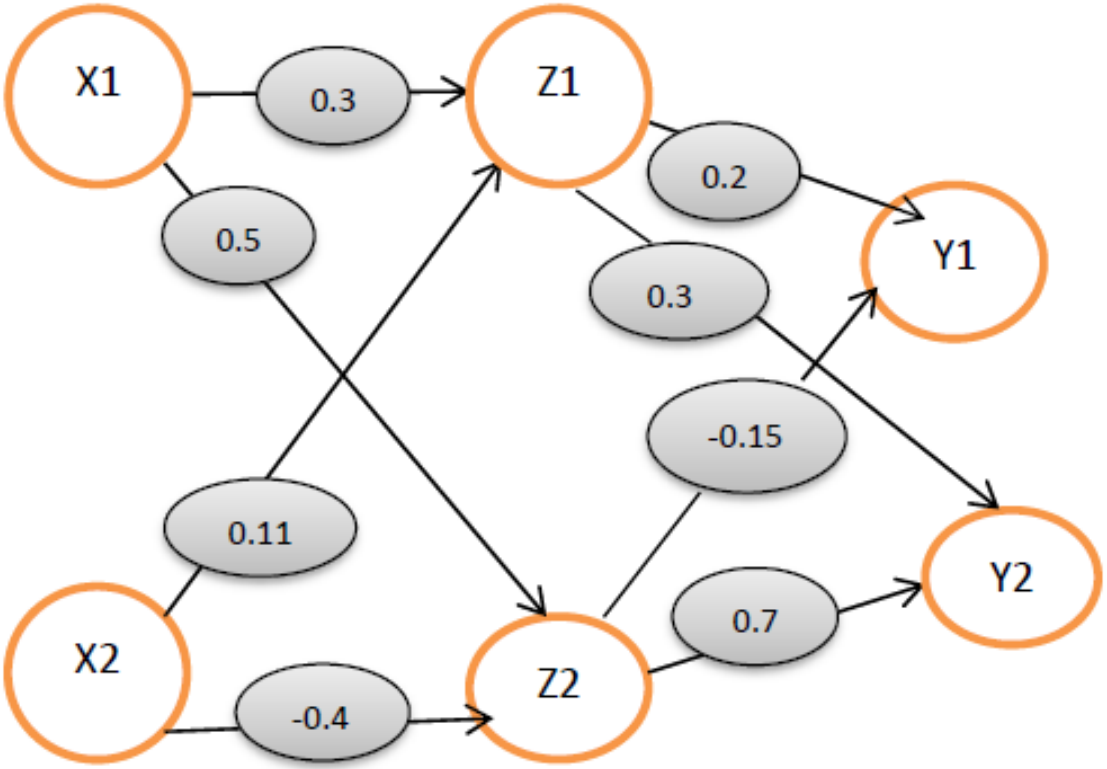
$$V22 = \eta * \delta\ 12 * x2 + \alpha [v22]$$

$$= 0.45 * 0.012 * 0 + 0.9 * 0.2 = 0.18$$

$$V = \begin{bmatrix} 0.0933 & -0.2646 \\ 0.675 & 0.18 \end{bmatrix} \qquad w = [0.293 \quad 0.468]$$

Example 2: Suppose you have Bp-ANN with 2-inputs, 2-hidden, 2-output with Hyperbolic function and the following weights, trace with 1-iteration, where $\alpha = 0.82$, $\eta = 0.60$, x= (1, 0), and $T_k = 1$

$$V = \begin{pmatrix} 0.3 & 0.5 \\ 0.11 & -0.4 \end{pmatrix} \qquad f(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}} \qquad W = \begin{pmatrix} 0.2 & 0.3 \\ -0.15 & 0.76 \end{pmatrix}$$



1. Forward phase:

Z1-in=x1* v11 + x2* v21

Z1-in= 1*0.3 + 0* 0.11 =0.3

Z1= $\dfrac{e^{0.3} - e^{-0.3}}{e^{0.3} + e^{-0.3}}$ **z1= 0.29**

Z2-in=x1*v12 + x2*v22

Z2-in= 1* 0.5 +0* -0.4= 0.5

Z2= $\dfrac{e^{0.5} - e^{-0.5}}{e^{0.5} + e^{-0.5}}$ **z2= 0.46**

Y1-in= z1*w11 + z2* w21

Y1-in= 0.29* 0.2 + 0.46* -0.15= -0.01

Y1= $\dfrac{e^{-0.01} - e^{0.01}}{e^{-0.01} + e^{0.01}}$ **Y1= -0.009**

Y2-in= z1*w12 + z2*w22

Y2-in= 0.29* 0.3 + 0.46* 0.76 =0.43

Y2= $\dfrac{e^{0.43} - e^{-0.43}}{e^{0.43} + e^{-0.43}}$ **Y2= 0.40**

2. Backward phase:

$\delta y1 = y1(1-y1)^* (T_k-y1)$

$= 0.009(1-0.009)^*(1-0.009) =$ **-0. 009**

$\delta y2 = y2(1-y2)^* (T_k-y2)$

$= 0.40(1-0.40)^*(1-0.40) =$ **0.144**

$\delta z1 = z1(1-z1)^*[(\delta y1^*w11)+ (\delta y2^*w12)]$

$= 0.29(1-0.29)^*[(-0.009^*0.2)+(0.144^* 0.3)]=$ **0.006**

$\delta z2= z2(1-z2)^*[(\delta y1^* w21)^* (\delta y2^* w22)]$

$= 0.46(1-0.46)^*[(-0.009^* -0.15)+(0.144^* 0.76)]=$ **0.027**

3. Update weights phase:

$W11 = \eta * \delta y1 * z1 + \alpha * w11_{old}$

$= (0.60 * -0.009 * 0.29) + (0.82 * 0.2) =$ **0.16**

$W12 = \eta * \delta y2 * z1 + \alpha * w12_{old}$

$= (0.60 * 0.144 * 0.29) + (0.82 * 0.3) =$ **0.271**

$W21 = \eta * \delta y1 * z2 + \alpha * w21_{old}$

$= (0.60 * -0.009 * 0.46) + (0.82 * -0.15) =$ **-0.125**

$W22 = \eta * \delta y2 * z2 + \alpha * w22_{old}$

$= (0.60 * 0.144 * 0.46) + (0.82 * 0.76) =$ **0.662**

$$w_{new} = \begin{pmatrix} 0.16 & 0.271 \\ -0.125 & 0.662 \end{pmatrix}$$

$V11 = \eta * \delta z1 * x1 + \alpha * v11_{old}$

$= (0.60 * 0.006 * 1) + (0.82 * 0.3) =$ **0.249**

$V12 = \eta * \delta z2 * x1 + \alpha * v12_{old}$

$= (0.60 * 0.027 * 1) + (0.82 * 0.5) =$ **0.426**

$V21 = \eta * \delta z1 * x2 + \alpha * v21_{old}$

$= (0.60 * 0.006 * 0) + (0.82 * 0.11) =$ **0.09**

$V22 = \eta * \delta z2 * x2 + \alpha * v22_{old}$

$= (0.60 * 0.027 * 0) + (0.82 * -0.4) =$ **-0.32**

$$v_{new} = \begin{pmatrix} 0.249 & 0.426 \\ 0.09 & -0.32 \end{pmatrix}$$

Genetic Algorithms

Genetic algorithms work on two types of spaces alternatively:

1- coding space, in other word (genotype space)
2- solution space, in other word(phenotype space)

Genetic operators are: (crossover and mutation)
(Evolution and selection)

- Crossover and mutation work on genotype space.
  While evolution and selection work on phenotype space.

- The selection is the link between chromosomes and the performance of decoded solutions.
- The mapping from genotype space to phenotype space has a considerable influence on the performance of genetic algorithms.
- The genetic algorithms provide a directed random search in complex landscapes.

There are two important issues with respect to search strategies:

1- Exploration (investigate new and unknown areas in search space)
2- Exploitation (make use of knowledge of solutions previously found in search space to help in find better solutions).

Genetic Algorithms

Genetic Algorithms (or simply GAs) are powerful and widely applicable stochastic search and optimization methods based on the concepts of natural selection and natural evaluation.

GAs work on a population of individuals represents candidate solutions to the optimization problem. These individual are consists of a strings (called chromosomes) of genes. The genes are a practical allele (gene could be a bit, an integer number, a real value or an alphabet character,…,etc depending on the nature of the problem). GAs applying the principles of survival of the fittest , selection , reproduction , crossover (recombining) , and mutation on these individuals to get , hopefully , a new butter individuals (new solutions) .

GAs are applied for those problems which either can not be formulated in exact and accurate mathematical forms and may contain noisy or irregular data or it take so much time to solve or it is simply impossible to solve by the traditional computational methods.

How Genetic Algorithms Work

Genetic algorithm maintains a population of individuals, say **P(t)**, for generation **t**. Each individual represents a potential solution to the problem at hand. Each individual is evaluated to give some measure of its fitness. Some individuals undergo stochastic transformations by means of genetic operations to form new individuals. There are two type of transformation:-

1) Mutation, which creates new individuals by making changes in a single individual.

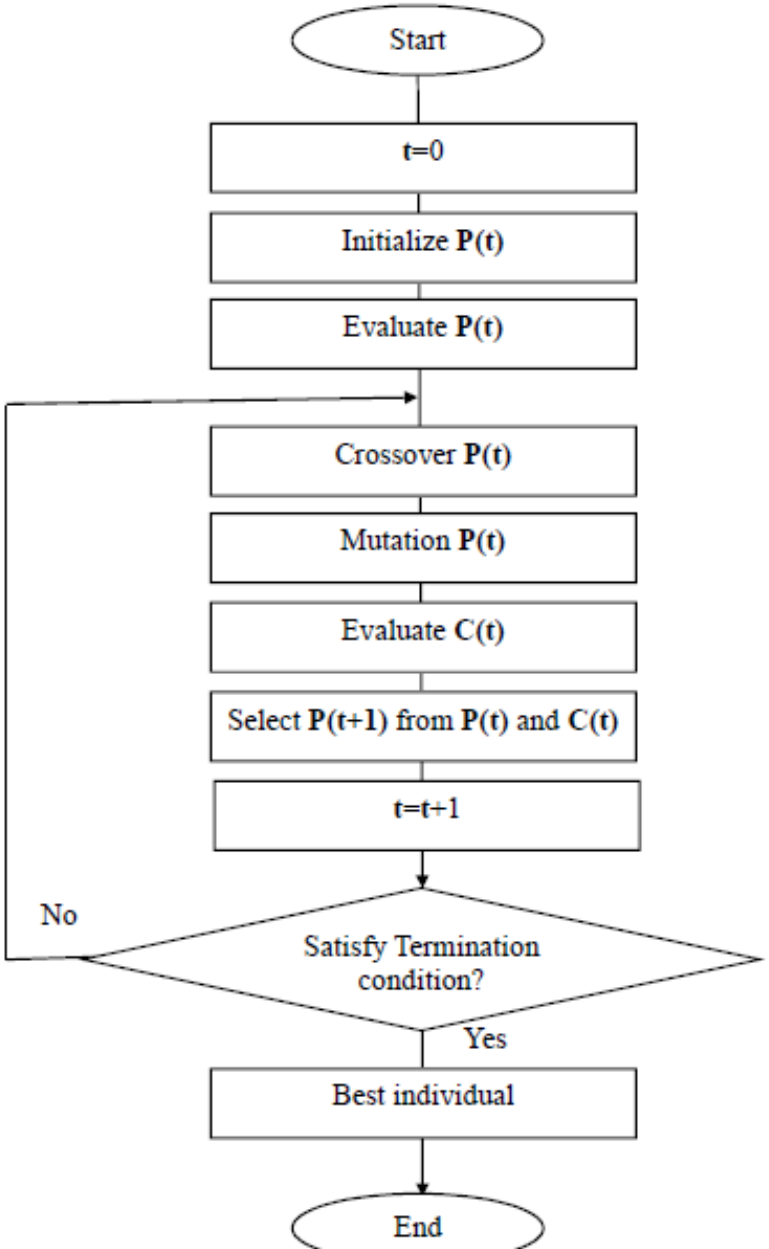
2) Crossover, which creates new individuals by combining parts from two individuals.

The new individuals, called offspring **C(t)**, are then evaluated. A new population is formed by selecting the more fit individuals from the parent population and offspring population.

The general structure of the Genetic algorithms is

```
Begin
  {
  t=0;
  Initialize P(t);
  Evaluate P(t);
  While (not termination condition) do
  Begin
      {
      Apply crossover and mutation to P(t) to yield C(t);
      Evaluate C(t);
      Select P(t+1) from P(t) and C(t);
       t=t+1;
       }
 End
}
End
```

The flowchart explains how genetic algorithms work is showing in the figure



Encoding

How to encode the solutions of the problem into chromosomes is a key issue when using genetic algorithms.

One outstanding problem associated with encoding is that some individuals correspond to infeasible or illegal solutions to a given problem. This may become very severe for constrained optimization problems and combinatorial optimization problems.

It must be distinguished between two concepts: infeasibility and illegality, as shown in figure (2).

Infeasibility refers to the phenomenon that a solution decoded from chromosome lies outside the feasible region of given problem. Penalty methods can be used handle infeasible chromosomes [19]. One of these methods is by force genetic algorithms to approach optimal form both sides of feasible and infeasible regions.

Illegality refers to the phenomenon that a chromosome does not represent a solution to a given problem. Repair techniques are usually adopted to convert an illegal chromosome to legal one.

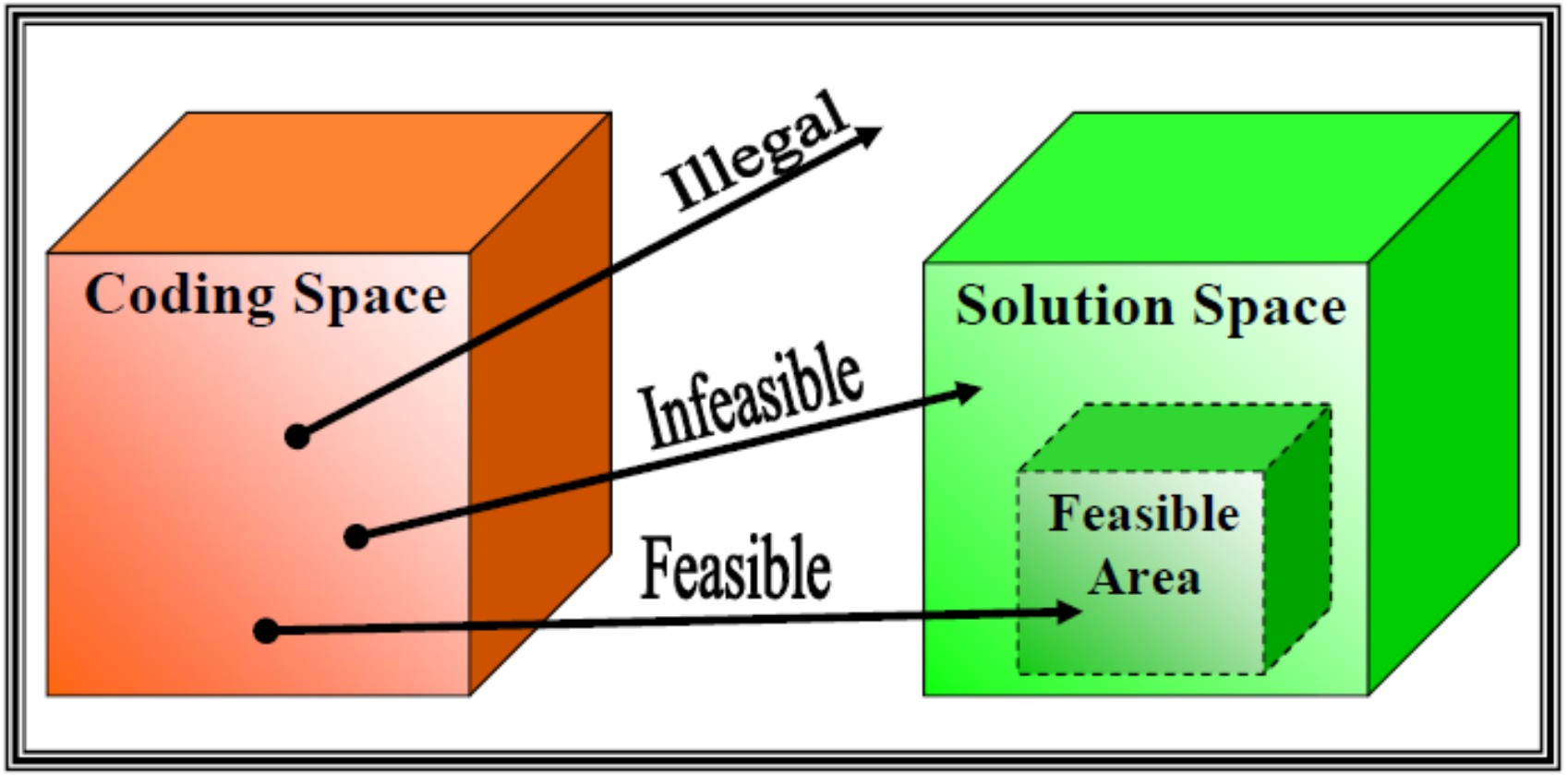


Figure (2) Infeasibility and illegality.

Various encoding methods have been created for particular problems to provide effective implementation of genetic algorithms. According to what kind of symbol is used as the alleles of a gene, the encoding methods can be classified as follows:

1) Binary encoding
2) Real-number encoding
3) Integer or literal permutation encoding

Binary Encoding

Binary encoding (i.e., the bit strings) are the most common encoding used for several of reasons. One is historical: in their earlier work, Holland and his students concentrated on such encodings and genetic algorithms practices have tended to follow this lead. Another reason for that was because much of existing GAs theories is based on the assumption of using binary encoding.

In spite of all that, binary encoding for function optimization problems is know to severe drawbacks due to the existence of Hamming cliffs, pairs of encoding having a large Hamming distance (The Hamming distance between two bit strings is defined as the number of corresponding positions in these bit strings where the bits have a different value) while belonging to points of minimal distance in phenotype space. For example, the pair 01111111111 and 10000000000 belongs to neighboring points in phenotype space but have maximum Hamming distance in genotype space. To cross the Hamming cliff, all bits have to be changed simultaneously. The probability that crossover and mutation will occur can be very small. In this sense, the binary code doses not preserve the locality of points in the phenotype space.

Real Number Encoding

Real number encoding is best used for function optimization problems. It has been widely confirmed that real number encoding perform better than binary encoding for function optimization and constrained optimizations problems. In real number encoding, the structure of genotype space is identical to that of the phenotype. Therefore, it is easy to form effective genetic operators by borrowing useful techniques from conventional methods.

Integer or Literal Permutation Encoding

Integer or literal permutation encoding is best used for combinational optimization problems because the essence of this kind of problems is to search for the best permutation or combination of items subject to constrains.

Genetic Algorithms Operators

There are two basic genetic algorithms operators which are crossover and mutation. These two operators are work together to explore and exploit the search space by creating new variants in the chromosomes. There are many empirical studies on a comparison between crossover and mutation. It is confirmed that mutation operator play the same important role as that of the crossover.

1 Crossover

One of the unique aspects of the work involving genetic algorithms (GAs) is the important role that Crossover (recombination) plays in the design and implementation of robust evolutionary systems. In most GAs, individuals are represented by fixed-length strings and crossover operates on pairs of individuals (parents) to produce new strings (offspring) by exchanging segments from the parents' strings.

1 Single Point Crossover

A commonly used method for crossover is called single point crossover. In this method, a single point crossover position (called cut-point) is chosen at random (e.g., between the 4th and 5th variables) and the parts of two parents after the crossover position are exchanged to form two offspring [1, 24], as shown in figure (3).

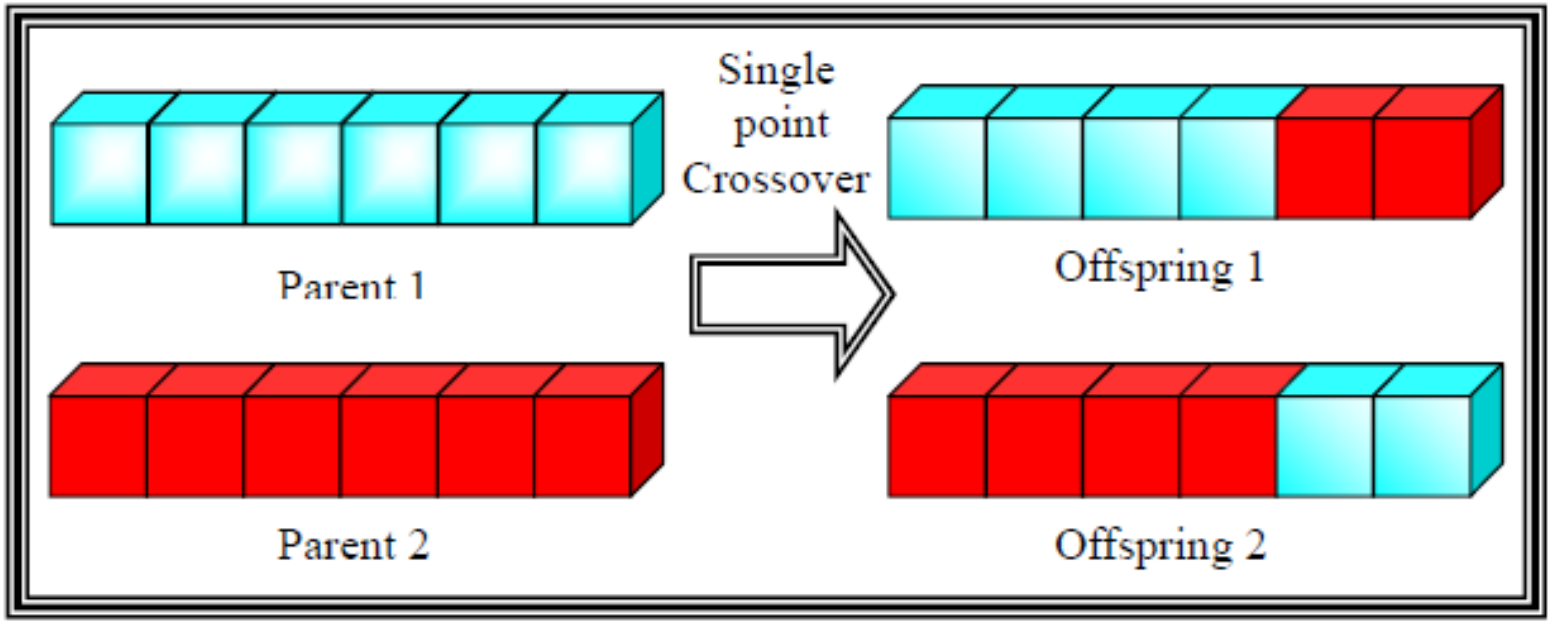


Figure (3) Single point crossover.

2 Multi Point Crossover

Multi-point crossover is a generalization of single point crossover, introducing a higher number of cut-points. In this case multi positions are chosen at random and the segments between them are exchanged [1, 24], as shown in figure (4).

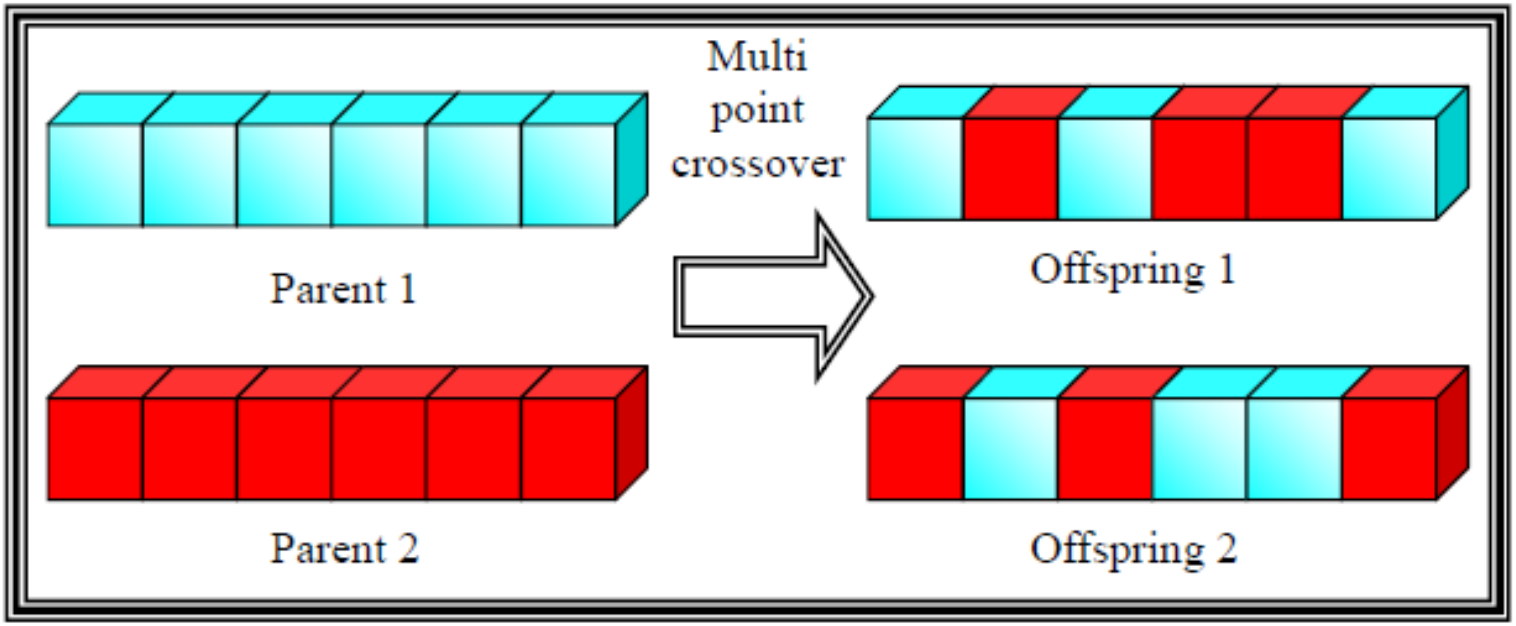


Figure (4) Multi point crossover.

3 Uniform Crossover

Uniform crossover does not use cut-points, but simply uses a global parameter to indicate the likelihood that each variable should be exchanged between two parents , as shown in figure (5).

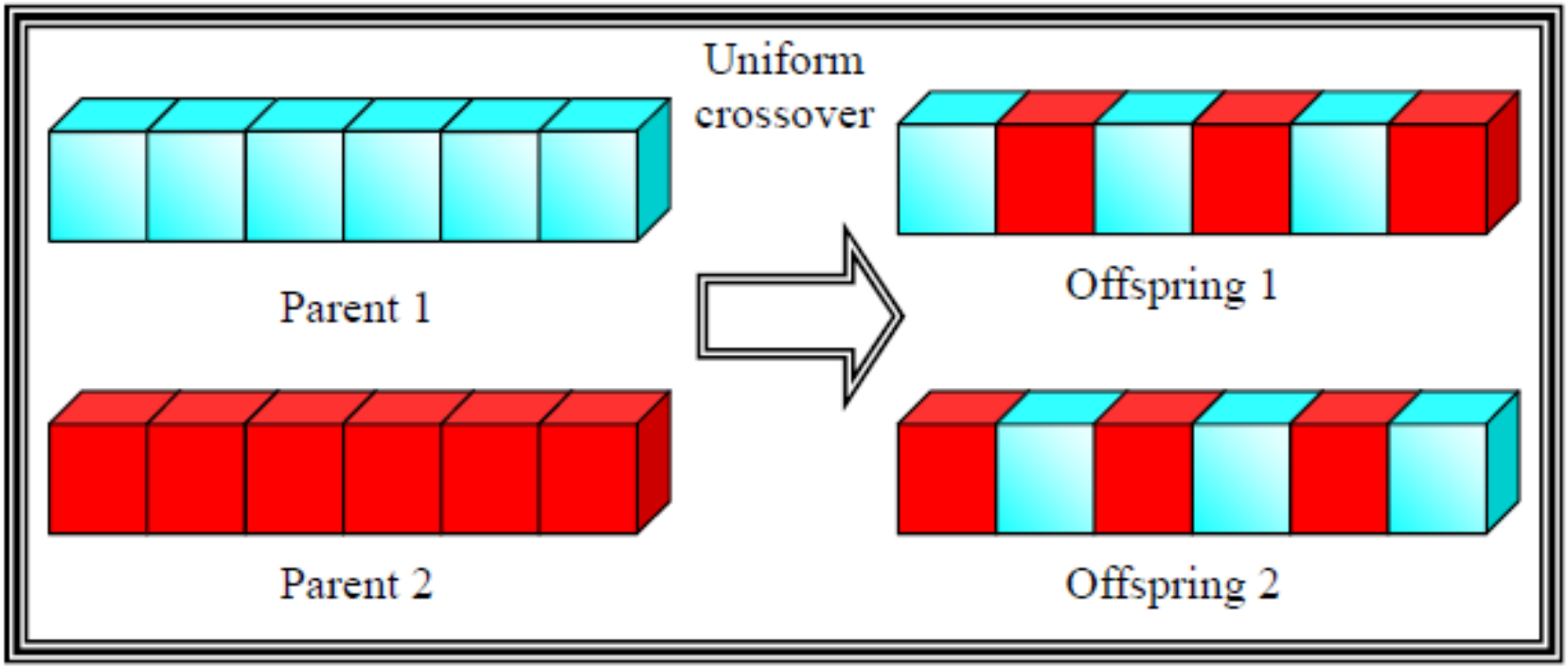


Figure (5) Uniform crossover.

2 Mutation

Mutation is a common operator used to help preserve diversity in the population by finding new points in the search pace to evaluate. When a chromosome is chosen for mutation, a random change is made to the values of some locations in the chromosome.

A commonly used method for mutation is called single point mutation. Though, a special mutation types used for varies problem kinds and encoding methods.

1 Single Point Mutation

Single gene (chromosome or even individual) is randomly selected to be mutated and its value is changed depending on the encoding type used, as shown in figure (6).

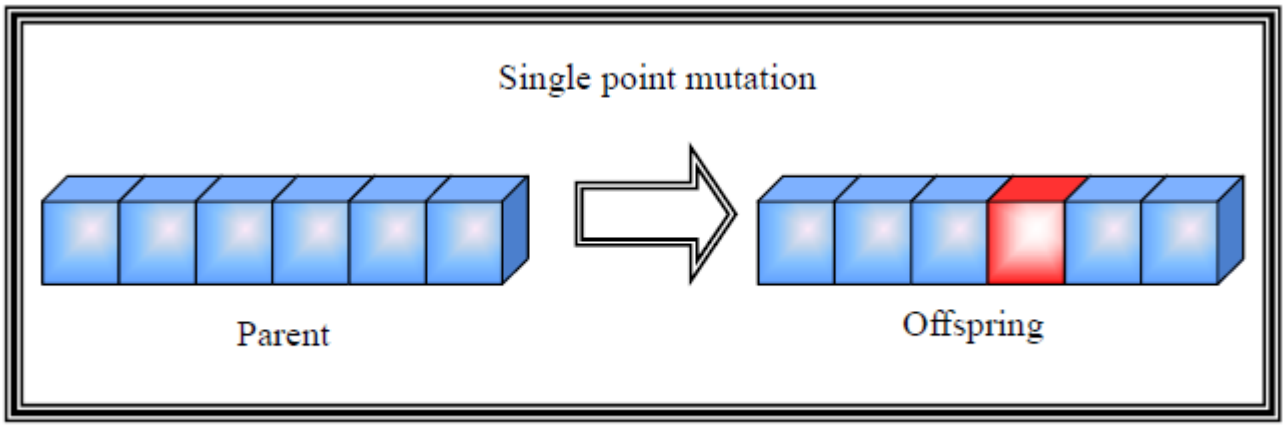


Figure (6) Single point mutation.

2 Multi Point Mutation

Multi genes (chromosomes or even individuals) are randomly selected to be mutated and there values are changed depending on the encoding type used, as shown in figure (7).

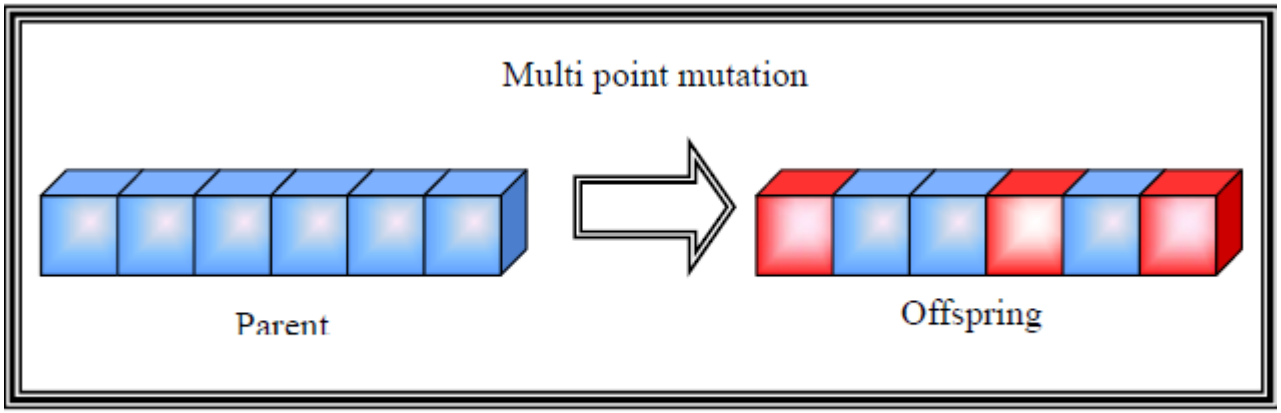


Figure (7) Multi point mutation.

Selection

Selection is the process of determining the number of times a particular individual is chosen for reproduction and, thus, the number of offspring that an individual will produce. The principle behind genetic algorithms is essentially Darwinian natural selection.

Selection provides the driving force in genetic algorithms. With too much force, genetic search will terminate prematurely. While with too little force, evolutionary progress will be slower than necessary.

Typically, a lower selection pressure is indicated at the start of genetic search in favor of a wide exploration of the search space, while a higher selection pressure is recommended at the end to narrow the search space. In this way, the selection directs the genetic search toward promising regions in the search space and that will improve the performance of genetic algorithms. Many selection methods have been proposed, examined and compared. The most common types are :

Type of selection

1) Roulette wheel selection
2) Rank selection
3) Tournament selection
4) Steady state selection
5) Elitism

1 Roulette Wheel Selection

Roulette wheel selection is most common selection method used in genetic algorithms for selecting potentially useful individuals (solutions) for crossover and mutation.

In roulette wheel selection, as in all selection methods, possible solutions are assigned a fitness by the fitness function. This fitness level is used to associate a probability of selection with each individual. While candidate solutions with a higher fitness will be less likely to be eliminated, there is still a chance that they may be. With roulette wheel selection there is a chance some weaker solutions may survive the selection process; this is an advantage, as though a solution may be weak, it may include some component which could prove useful following the recombination process.

The analogy to a roulette wheel can be envisaged by imagining a roulette wheel in which each candidate solution represents a pocket on the wheel; the size of the pockets are proportionate to the probability of selection of the solution. Selecting N individual from the population is equivalent to playing N games on the roulette wheel, as each candidate is drawn independently, as shown in figure (8).

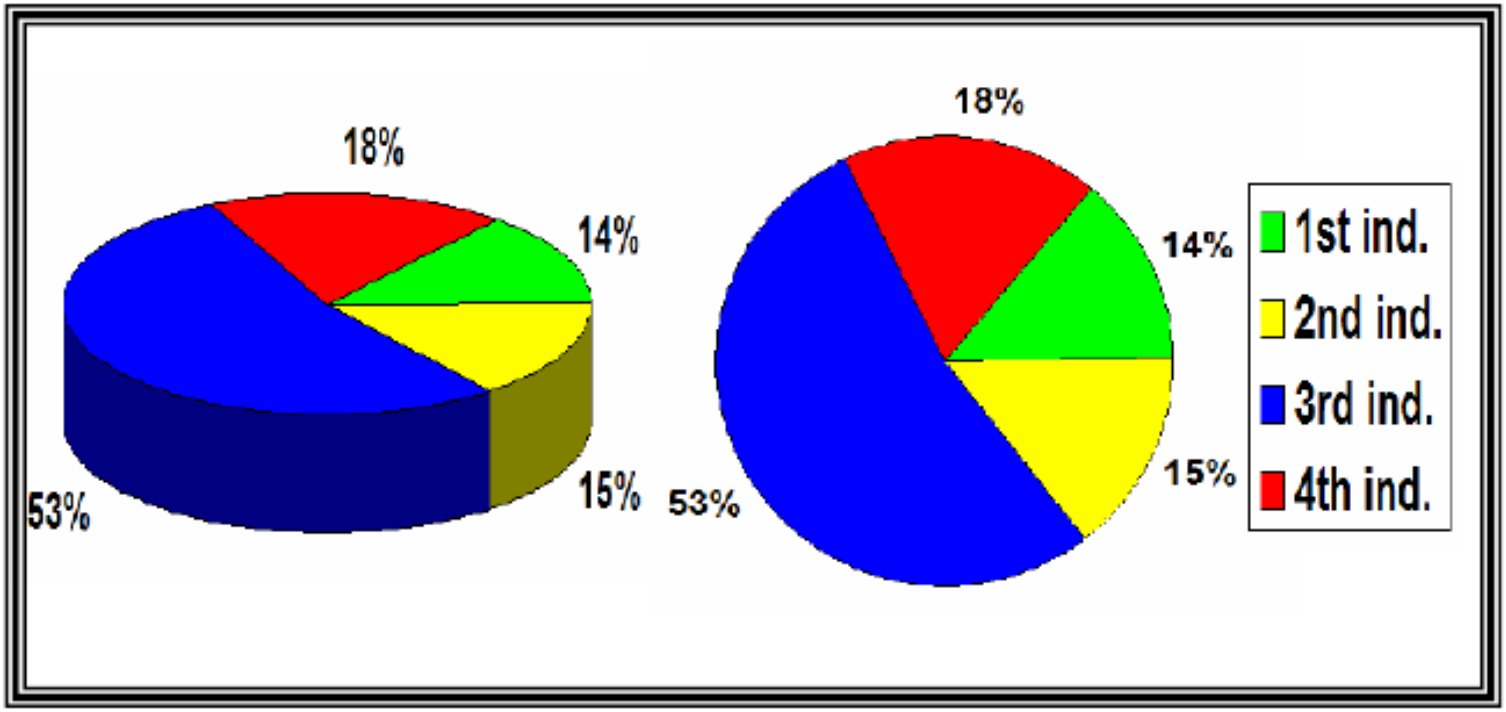


Figure (8) Roulette wheel selection.

2 Rank Selection

In ranking selection, the individuals in the population are sorted from best to worst according to their fitness values. Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking rather than differences in fitness. The advantage of this method is that it can prevent very fit individuals from gaining dominance early at the expense of less fit ones, which would reduce the population's genetic diversity and might hinder attempts to find an acceptable solution. The disadvantage of this method is that it required sorting the entire population by rank which is a potentially time consuming procedure. Rank selection effect is shown in figure (9) (a and b).

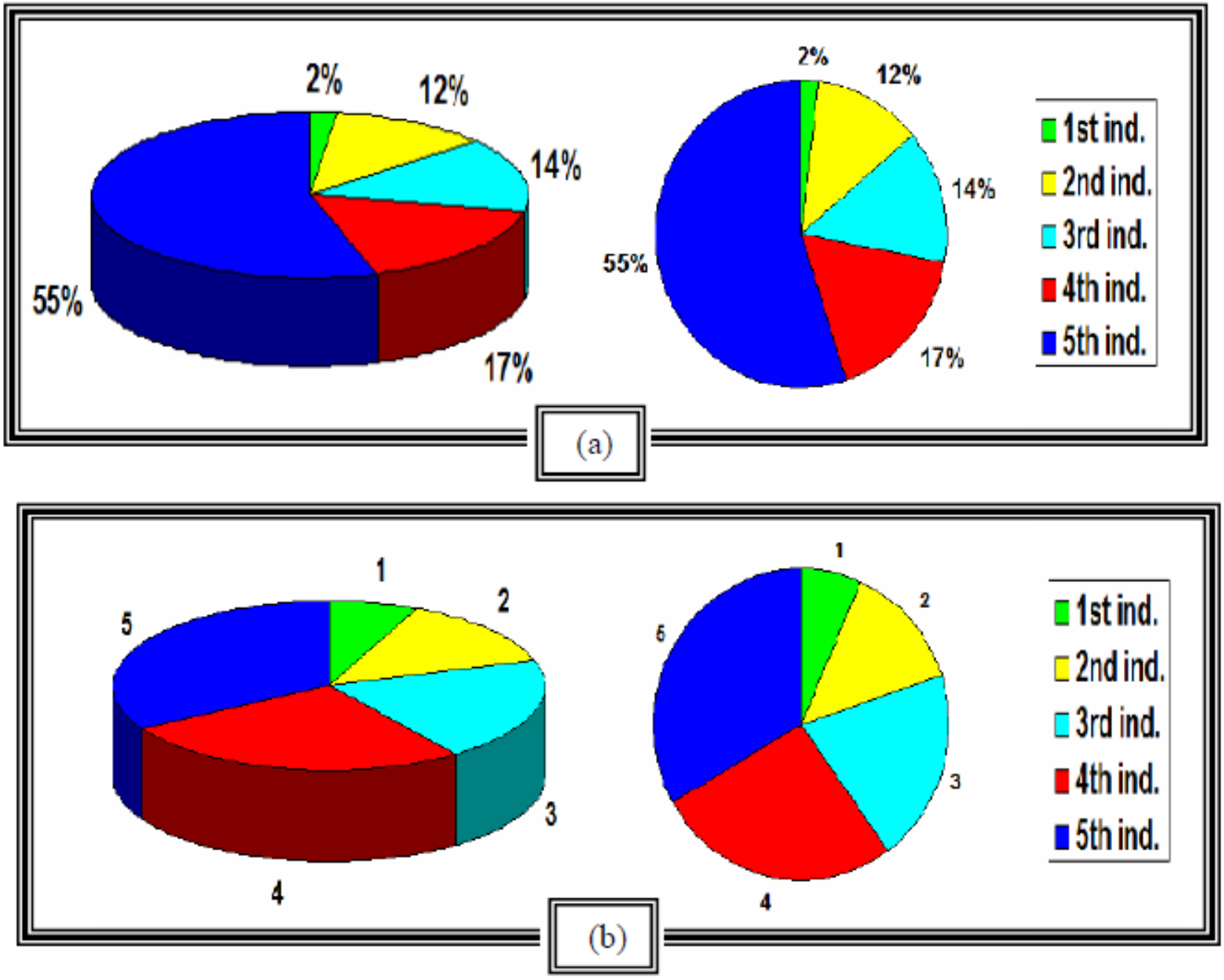


Figure (9) Rank selection effect. (a) Before ranking. (b) After ranking.

3 Tournament Selection

This method randomly chooses a set of individual and picks out the best individual for reproduction. The number of individual in the set is called the tournament size. A common tournament size is 2, this is called binary tournament.

By adjusting tournament size, the selection pressure can be made arbitrarily large or small. For example, using large Tournament size has the effect of increasing the selection pressure, since below average individuals are less likely to win a tournament while above average individuals are more likely to win it.

4 Steady State Selection

The steady state selection will eliminate the worst of individuals in each generation. It work as follow; the offspring of the individuals selected from each generation go back into the pre-existing population, replacing some of the less fit members of the previous generation .

Genetic Algorithms Parameters

1-population size 2-crossover rate 3-mutation rate

1-population size

population size dictates the number of individuals in the population. Larger population sizes increase the amount of variation present in the initial population at the expense of requiring more fitness evaluations. It is found that the best population size is both application dependent and related to the individual size (number of chromosomes within). A good population of individuals contains a diverse selection of potential building blocks resulting in better exploration .

If the population loses diversity the population is said to have "premature convergence" and little exploration is being done. For larger individuals and challenging optimization problems, larger population sizes are needed to maintain diversity (higher diversity can also be achieved through higher mutation rates and uniform crossover) and hence better exploration. Many researchers suggest population sizes between 25 and 100 individual, while others suggest that it must be very much larger (1000 individual or more).

2 Crossover Rate

Crossover rate determines the probability that crossover will occur. The crossover will generate new individuals in the population by combining parts of existing individuals. The crossover rate is usually high and 'application dependent'. Many researchers suggest crossover rate to be between 0.6 and 0.95 .

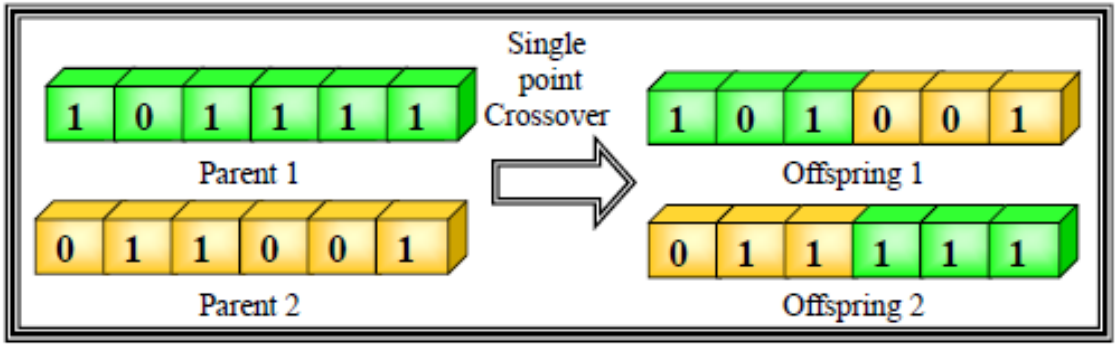
3 Mutation Rate

Mutation rate determines the probability that a mutation will occur. Mutation is employed to give new information to the population (uncover new chromosomes) and also prevents the population from becoming saturated with similar chromosomes, simply said to avoid premature convergence. Large mutation rates increase the probability that good schemata will be destroyed, but increase population diversity. The best mutation rate is 'application dependent'. For most applications, mutation rate is between 0.001 and 0.1 [16,36], while for automated circuit design problems, it is usually between 0.3 and 0.8.

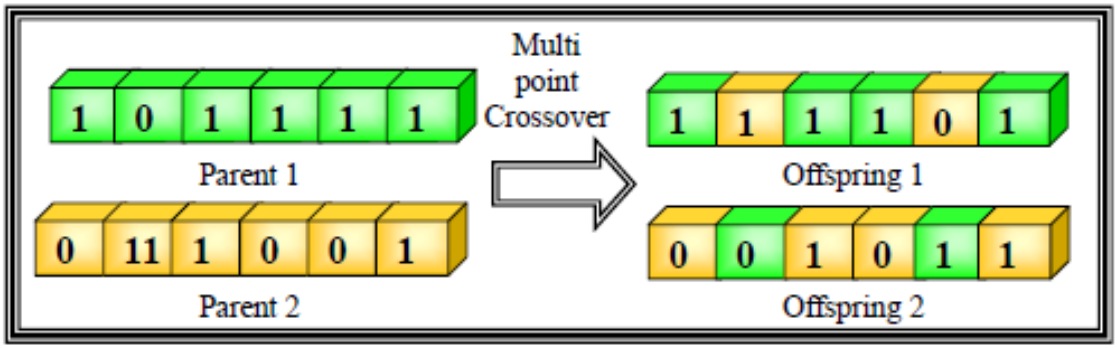
How Crossover and Mutation Work

1 Binary Encoding

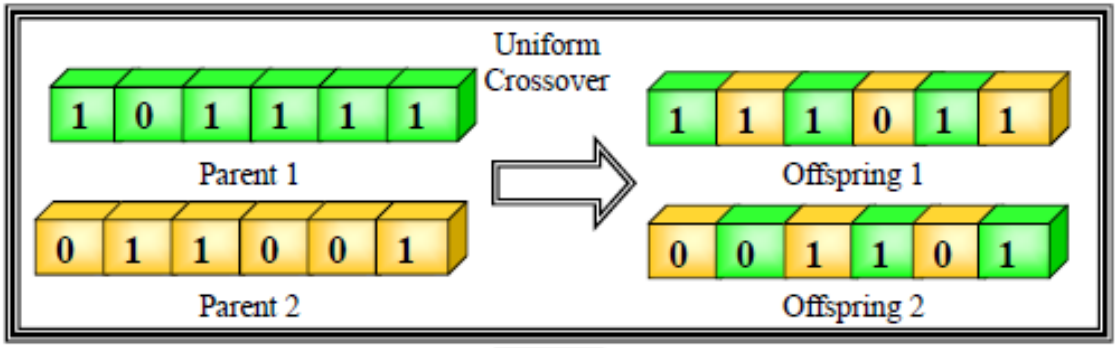
1) Crossover



(a)



(b)



(c)

2) Mutation

The single point and multi point can implement in binary encoding as explained in section (5.2) before, the effect of mutation is shown in the figure (11) (a and b).

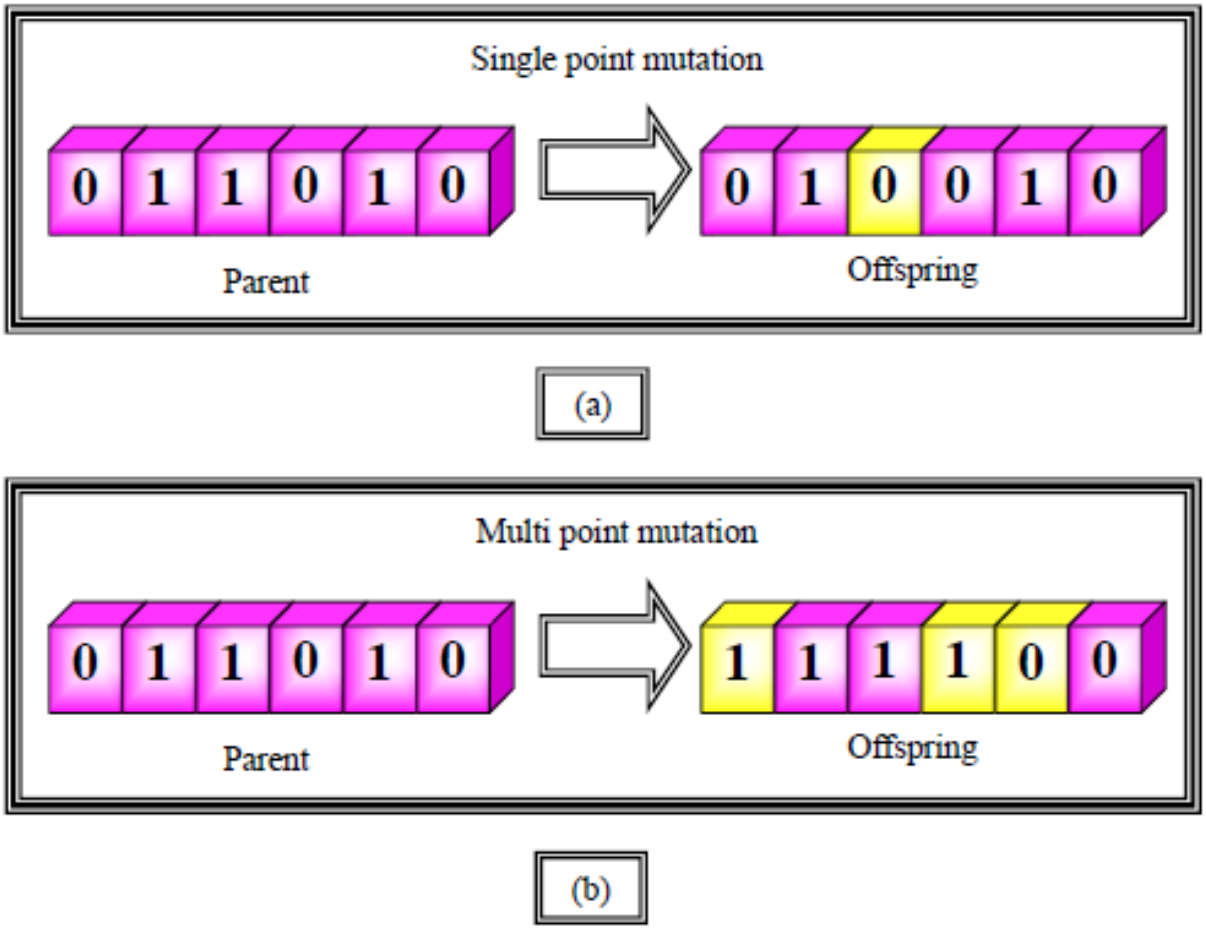
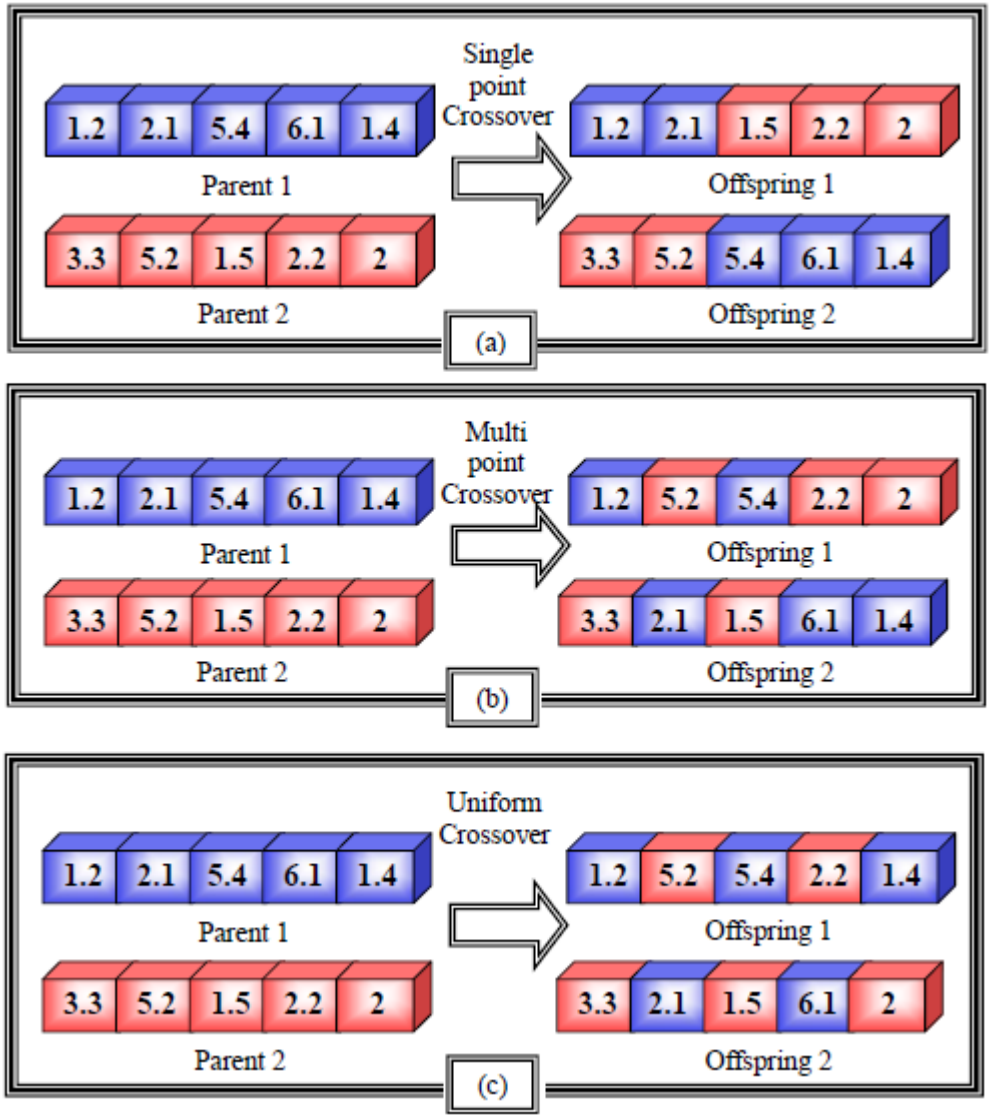


Figure (11) Explanation of mutation effect on binary string. (a) Single point mutation. (b) Multi point mutation.

2 Real Number Encoding

1) Crossover

Crossover in real number encoding has the same effect as in binary encoding, the single point, multi point, and uniform crossover in real number encoding is shown in figure (12)(a,b and c).



2) Mutation

In this type of encoding, there is several ways to implement mutation. This done, usually, by adding (or subtract) a random number to (or from) the mutated gene, but in another cases gene might by replaced by a new value generated randomly within the used real number limitation, as shown in figure (13) (a and b).

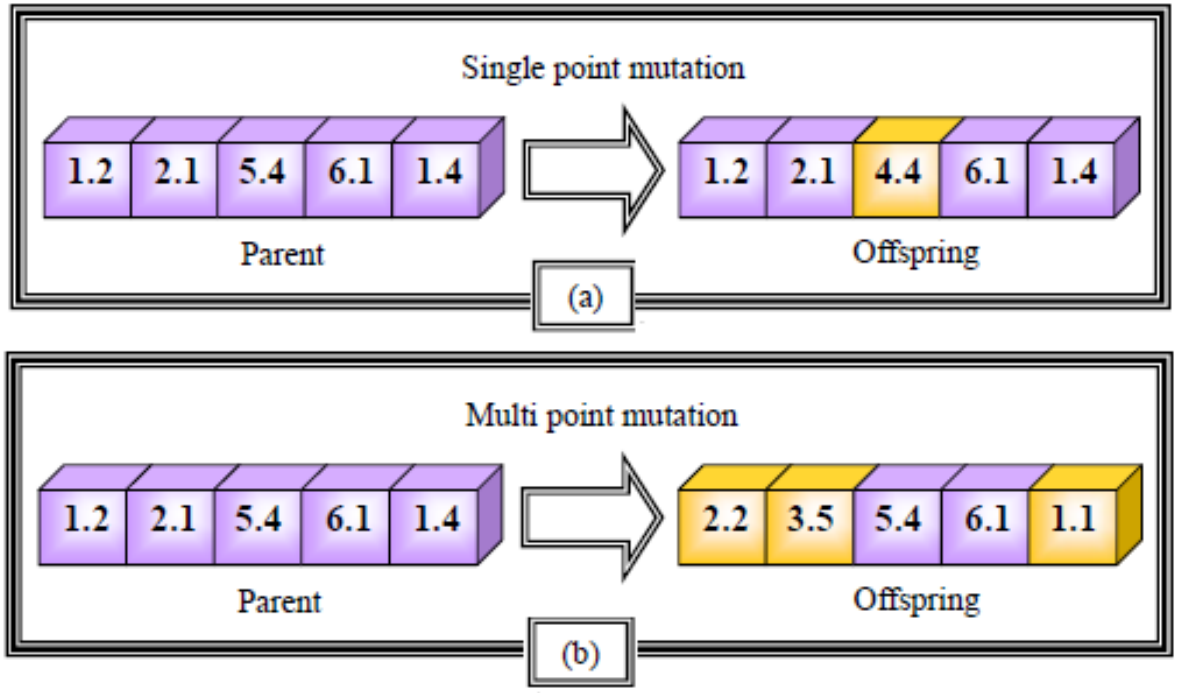


Figure (13) Explanation of the mutation effect on the real number string.

(a) Single point mutation. (b) Multi point mutation.

Integer or Literal Permutation Encoding

1) Crossover

Unlike the binary and real value encoding, this type has a special crossover rule, for single point crossover the crossover is done as follow: first copy a substring from first parent till crossover point to the first offspring, then copy all the symbols that not contents in that substring from the second parent, do the same to second parent to produce the second offspring. The same procedure can be done in multi point

crossover while uniform crossover is not usually used because it gives illegal offspring . The single point crossover is shown in figure (14).

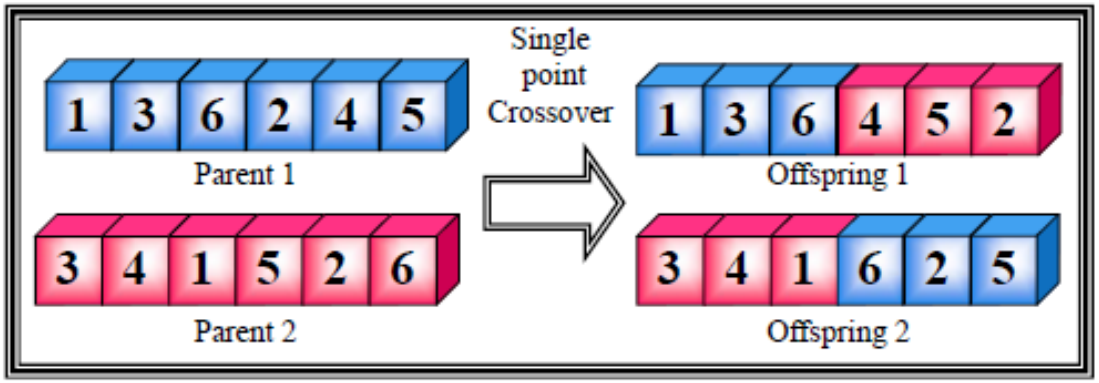


Figure (14) Explanation of single point crossover effect on Integer permutation encoding string.

2) Mutation

A special type of mutation is used, called reorder mutation, in which a pair of genes will select randomly and swap there contents. The mutation effect is shown in figure (15) (a and b).

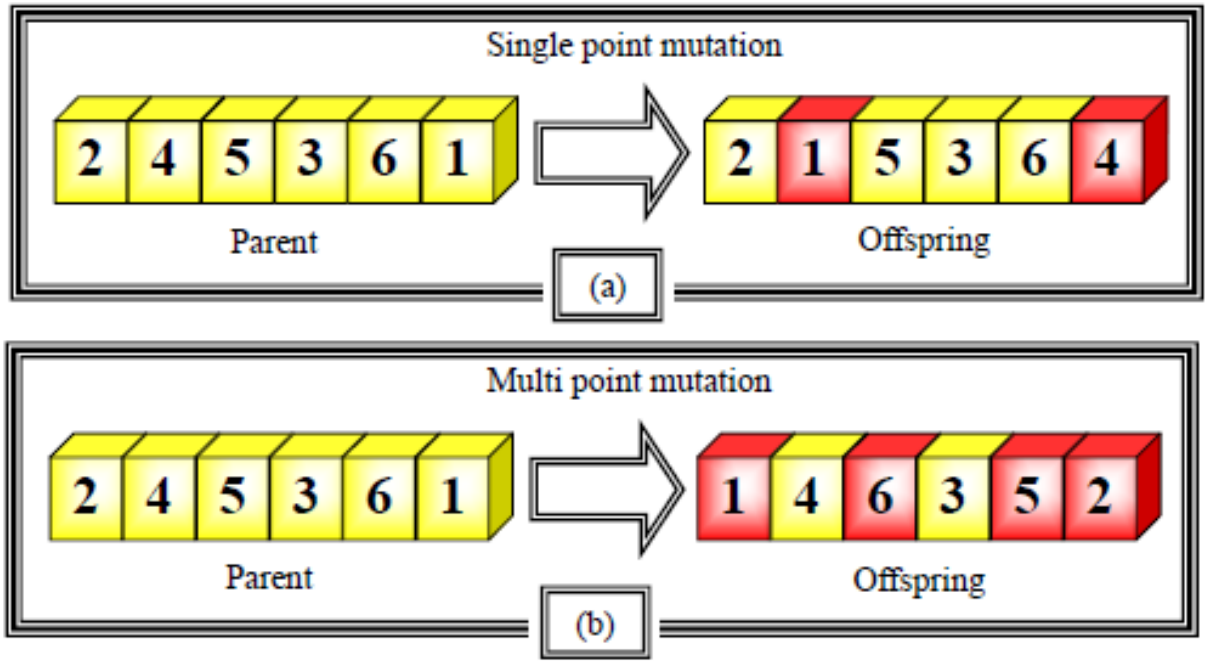


Figure (15) Explanation of the mutation effect on the Integer permutation encoding string. (a) Single point mutation. (b) Multi point mutation.

Application of Genetic Algorithms

1) Optimization.
2) Automatic Programming.
3) Machine and robot learning.
4) Economic models.
5) Immune system models.
6) Ecological models.
7) Population genetics models.
8) Interactions between evolution and learning

Questions and Answers

Q1: Explain type of Activation Function?

Answer:

(i) Identity function:

$$f(x) = x \quad \text{for all } x.$$

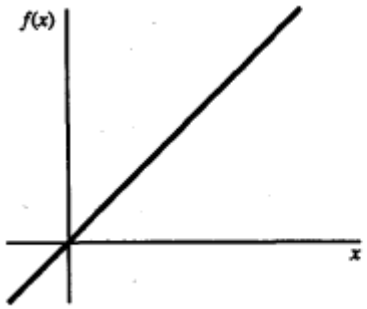


Figure 1.7 Identity function.

(ii) Binary step function (with threshold θ):

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

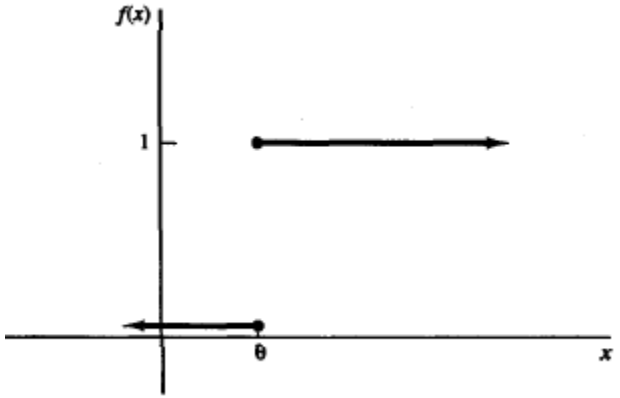


Figure 1.8 Binary step function.

(iii) Binary sigmoid:

$$f(x) = \frac{1}{1 + \exp(-\sigma x)}.$$

$$f'(x) = \sigma f(x)\,[1 - f(x)].$$

$$a = \frac{1}{1 + e^{-n}}.$$

As is shown in Section 6.2.3, the logistic sigmoid function can be scaled to have any range of values that is appropriate for a given problem. The most common range is from -1 to 1; we call this sigmoid the *bipolar sigmoid*. It is illustrated in Figure 1.10 for $\sigma = 1$.

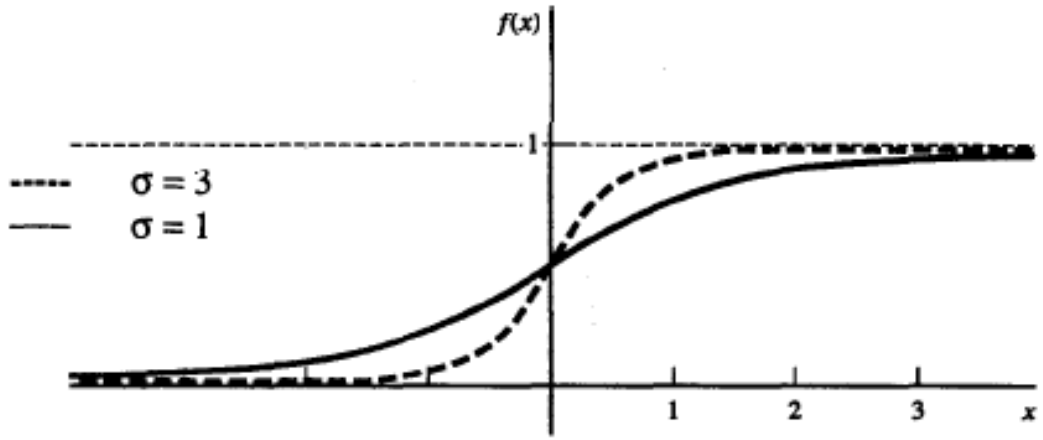


Figure 1.9 Binary sigmoid. Steepness parameters $\mathbf{a} = 1$ and $\mathbf{a} = 3$.

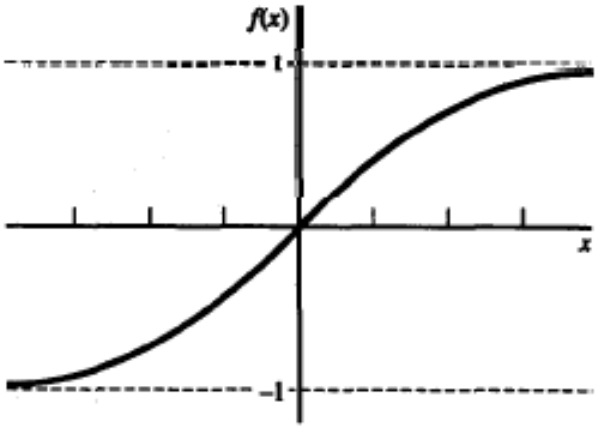


Figure 1.10 Bipolar sigmoid.

(iv) Bipolar sigmoid:

$$g(x) = 2f(x) - 1 = \frac{2}{1 + \exp(-ax)} - 1$$

$$= \frac{1 - \exp(-\sigma x)}{1 + \exp(-\sigma x)}.$$

$$g'(x) = \frac{\sigma}{2}[1 + g(x)][1 - g(x)].$$

The bipolar sigmoid is closely related to the hyperbolic tangent function, which is also often used as the activation function when the desired range of output values is between -1 and 1. We illustrate the correspondence between the two for $a = 1$. We have

$$g'(x) = \frac{\sigma}{2}[1 + g(x)][1 - g(x)].$$

The bipolar sigmoid is closely related to the hyperbolic tangent function, which is also often used as the activation function when the desired range of output values is between -1 and 1. We illustrate the correspondence between the two for $a = 1$. We have

$$g(x) = \frac{1 - \exp(-x)}{1 + \exp(-x)}.$$

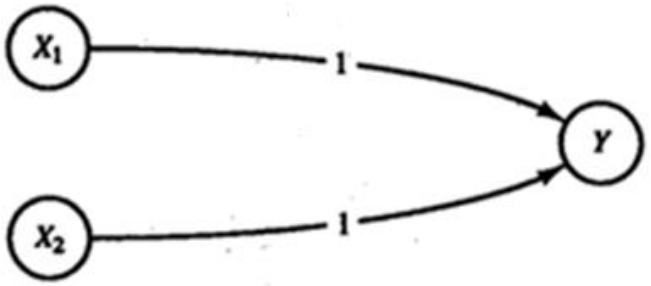
The hyperbolic tangent is

$$h(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

$$= \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

Q2: Design AND function using ANN

Answer:



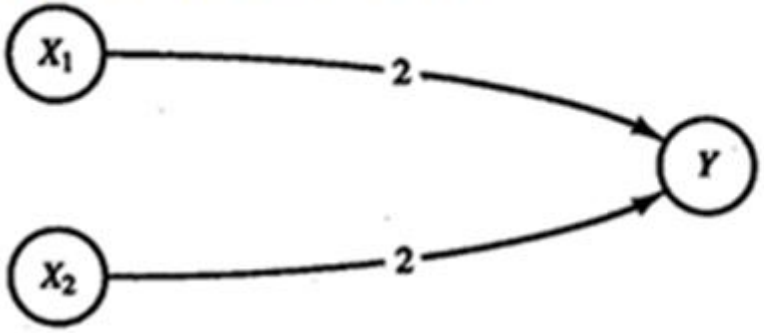
x_1	x_2	→	y
1	1		1
1	0		0
0	1		0
0	0		0

Q3: Design OR function using ANN

Answer:

OR gate

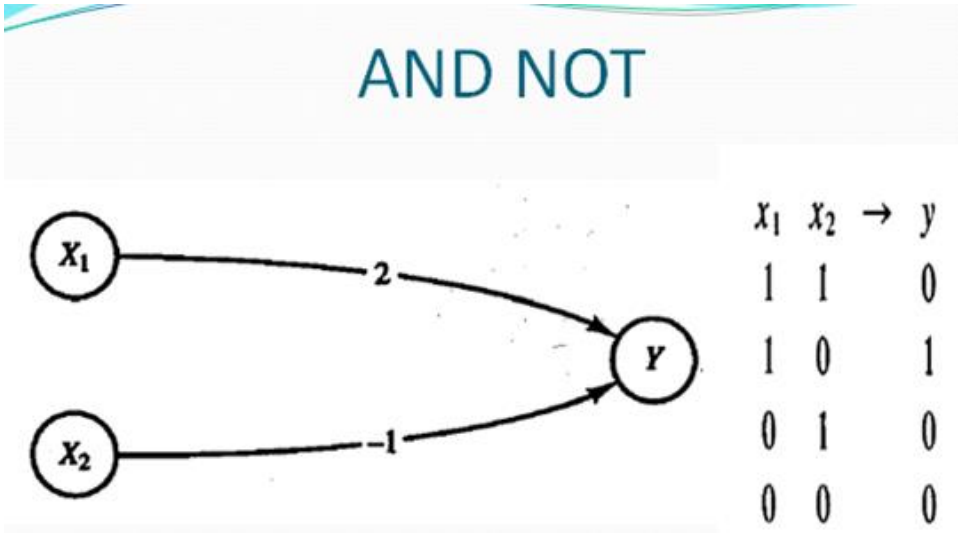
The threshold on unit Y is 2.



x_1	x_2	→	y
1	1		1
1	0		1
0	1		1
0	0		0

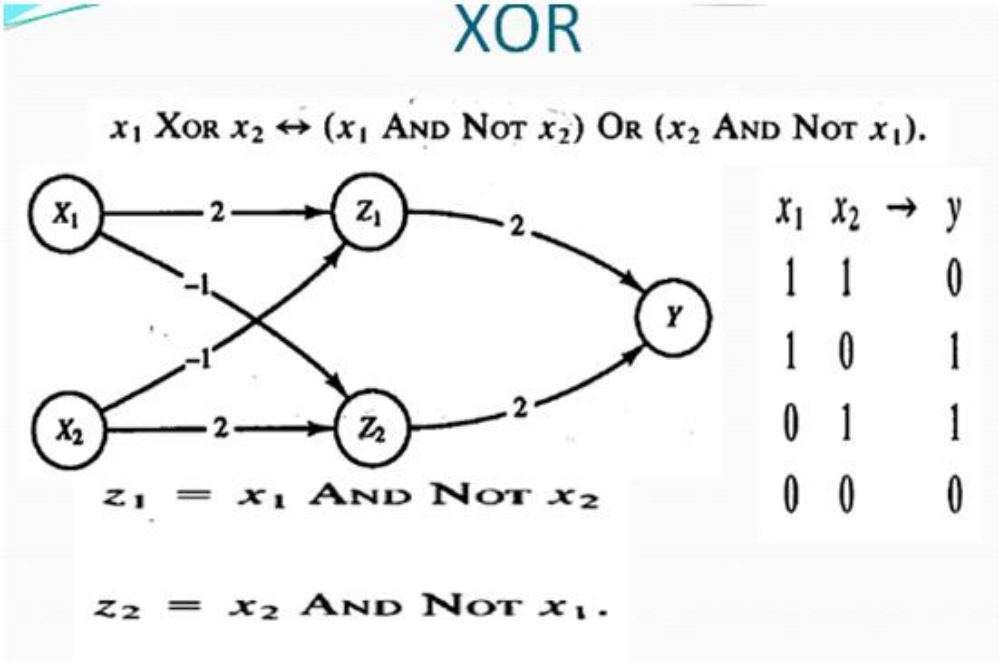
Q4: Design AND NOT function using ANN

Answer:



Q5: Design XOR function using ANN

Answer:



Q6: Hebb rule AND function binary input and target

Q7: Hebb rule AND function binary input and Bipolar target

Q8: Hebb rule AND function bipolar input and target

Q9: Hebb rule OR function binary input and target

Q10: Hebb rule OR function binary input and Bipolar target

Q11: Hebb rule OR function bipolar input and target

Q12: training Hebb rule AND function to find truth table

Q13: Explain the algorithm of Hebb rule

Q13: Explain the algorithm of perceptron rule

Answer Q

Example 2.5 A Hebb net for the AND function: binary inputs and targets

INPUT	TARGET
$(x_1 \quad x_2 \quad 1)$	
(1 1 1)	1
(1 0 1)	0
(0 1 1)	0
(0 0 1)	0

INPUT	TARGET	WEIGHT CHANGES	WEIGHTS
$(x_1 \quad x_2 \quad 1)$		$(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	$(w_1 \quad w_2 \quad b)$
			(0 0 0)
(1 1 1)	1	(1 1 1)	(1 1 1)
(1 0 1)	0	(0 0 0)	(1 1 1)
(0 1 1)	0	(0 0 0)	(1 1 1)
(0 0 1)	0	(0 0 0)	(1 1 1)

Answer Q

Example 2 A Hebb net for the AND function: binary inputs bipolar targets

INPUT	TARGET
$(x_1 \quad x_2 \quad 1)$	
(1 1 1)	1
(1 0 1)	−1
(0 1 1)	−1
(0 0 1)	−1

INPUT	TARGET	WEIGHT CHANGES	WEIGHTS
$(x_1 \quad x_2 \quad 1)$		$(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	$(w_1 \quad w_2 \quad b)$
			(0 0 0)
(1 1 1)	1	(1 1 1)	(1 1 1)

INPUT			TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	1)		$(\Delta w1$	Δw_2	$\Delta b)$	$(w_1$	w_2	$b)$
(1	0	1)	-1	$(-1$	0	$-1)$	(0	1	0)
(0	1	1)	-1	$(0$	-1	$-1)$	(0	0	$-1)$
(0	0	1)	-1	(0	0	$-1)$	$(0$	0	$-2)$

For (binary input and binary output) and (binary input and bipolar output) the Final weight Does not achieve all the Targets

Answer Q

Example A Hebb net for the AND function: bipolar inputs and targets

INPUT			TARGET
$(x_1$	x_2	1)	
(1	1	1)	1
(1	−1	1)	−1
(−1	1	1)	−1
(−1	−1	1)	−1

INPUT			TARGET	WEIGHT CHANGES			WEIGHTS		
$(x_1$	x_2	1)		$(\Delta w_1$	Δw_2	$\Delta b)$	$(w_1$	w_2	$b)$
							(0	0	0)
(1	1	1)	1	(1	1	1)	(1	1	1)
(1	−1	1)	−1	(−1	1	−1)	(0	2	0)
(−1	1	1)	−1	(1	−1	−1)	(1	1	−1)
(−1	−1	1)	−1	(1	1	−1)	(2	2	−2)

Final Weights

Answer Q

INPUT $(x_1 \quad x_2 \quad 1)$	TARGET	WEIGHT CHANGES $(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	WEIGHTS $(w_1 \quad w_2 \quad b)$
			(0 0 0)
(1 1 1)	1	(1 1 1)	(1 1. 1)
(1 0 1)	0	(0 0 0)	(1 1 1)
(0 1 1)	0	(0 0 0)	(1 1 1)
(0 0 1)	0	(0 0 0)	(1 1 1)

INPUT $(x_1 \quad x_2 \quad 1)$	TARGET	WEIGHT CHANGES $(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	WEIGHTS $(w_1 \quad w_2 \quad b)$
			(0 0 0)
(1 1 1)	1	(1 1 1)	(1 1 1)
(1 0 1)	-1	$(-1 \quad 0 \quad -1)$	(0 1 0)
(0 1 1)	-1	$(0 \quad -1 \quad -1)$	$(0 \quad 0 \quad -1)$
(0 0 1)	-1	$(0 \quad 0 \quad -1)$	$(0 \quad 0 \quad -2)$

INPUT	TARGET	WEIGHT CHANGES	WEIGHTS
$(x_1 \quad x_2 \quad 1)$		$(\Delta w_1 \quad \Delta w_2 \quad \Delta b)$	$(w_1 \quad w_2 \quad b)$
			(0 0 0)
(1 1 1)	1	(1 1 1)	(1 1 1)
(1 −1 1)	−1	(−1 1 −1)	(0 2 0)
(−1 1 1)	−1	(1 −1 −1)	(1 1 −1)
(−1 −1 1)	−1	(1 1 −1)	(2 2 −2)

Final Weights

Choose the suitable choice for the following statements

1. The areas in which neural networks are currently being applied are: ----------

a. speech Recognition b. searching c. control problems d. both a&c

2 . In neuron input paths called ----------

a. dendrites b. synapses c. weights d. axon

3. --------- is the junction between the (axon) of the neuron and the dendrites of the other neuron

a. synapse b. dendrites c. weight d. activation function

4. Information processing occurs at many simple elements called ----------

a. weight b. path c. neuron d. axon

5. Each neuron applies an action ----------

6. A Neural network is characterized by: ----------

a. Training Learning Algorithm b. Activation function c. architecture d. all a,b,c

7. One of ANN properties is

a. serial b. Fault tolerance c. activation function d. supervision

8. There are ----- types of learning in which the weights organize themselves according to the task to be learnt, these types are----------, ----------, ----------

a. 3,supervised, unsupervised, activated b.3, self learned ,activated, supervised c. 3,unsupervised, self learned, activated d. 3, supervised, unsupervised, self learned

9. In the unsupervised learning, the correct final vector is not ---------, but instead the -------- are changed through random numbers

a. learned, weight b. specified, weights c. activated, weights d. changed, weights

10. In the supervised learning, at every step the system is informed about the ------ output vector. The ------ are changed according to a formula

a. exact, weights b. next, weights c. only, weights d. nearly, weights

11. Neural nets are often classified as

a. single layer, double layer b. single layer, triple layer c. single layer, multi layer d. single layer, all layer

12. In determining the number of layers, the input units are ---------- as a layer

a. not specified b. not counted c. not determined d. not countered

13. ---------- units are between the input units and the output units in the ---------- net.

a. hidden, multi layer b. hidden, double layer c. hidden, triple layer, d. hidden, single layer

14. the hard limiter activation function is ----------& ----------

a. binary, tri-polar bi-polar, tri-polar c. binary, bi-polar d. binary, single polar

15. the input to binary hard limits is ------& -------- while in the bipolar is -----& -------

a. 1, 0, 1,-1 b. 0, 1, 1/2,-1/2 c. 0,1,2,-2 d. 1,-1,0,1

Example2 – Delta learning rule

Perform two training steps of neural network, using the delta learning rule for λ = 1 and c = 0.25 . Train the network using the following data pairs

$$\left(\mathbf{x}_1 = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}, \quad d_1 = -1\right), \quad \left(\mathbf{x}_2 = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}, \quad d_2 = 1\right)$$

The initial weights are $W_1 = [1\ 0\ 1]^t$ and f(net) is bipolar continuous activation function.

Solution

F(net) = bipolar activation function = $\dfrac{2}{1 + \exp(-\lambda x)} - 1$

F'(net) = ½[1-(Oi)^2)]

Net1= X1*W1 = [2 0 -1] * [1 0 1] = 1

O1 = f(net1) = 0.47

Since Sgn(O1) not equal d_1 (-1) the correction is necessary

$F'(net_1)=1/2(1-O^2) = \frac{1}{2}(1- (0.47)^2) = 0.39$

$W_2=W_1 + c(di-Oi)\ F'(net1)^*X1$

$W2=[1\ 0\ 1] + 0.25(-1-0.47)^*0.39\ ^*\ [2\ 0\ -1] = [1\ 0\ 1] + (-\ 0.14)\ ^*\ [2\ 0\ -1]$

$=[1\ 0\ 1] +\ [-0.28\ \ 0\ \ 0.14\ ]$ $\quad W2=[0.72\ 0\ \ 1.14]$

$Net_2 = X_2{}^*W_2 = [1\ -2\ -1]\ ^*\ [0.72\ \ 0\ \ 1.14]=\ -0.42$

O2= f(net2) = -0.2

Since Sgn(O2) not equal d_2 (1) the correction is necessary

$F'(net_2)=1/2(1-O^2) = \frac{1}{2}\ (1- (-0.2)^2\ ) = 0.48$

$W_3=W_2 + c(d2-O2)\ F'(net2)^*X2$

$W_3 = [0.72\ \ \ 0\ \ \ 1.14] + 0.25\ (1+0.2)\ ^*\ 0.48\ ^*\ [1\ -2\ -1]$

$W_3 = [0.72\ \ \ 0\ \ \ 1.14] + [1.44\ \ \ -2.88\ \ -1.44]$

$W_3 = [2.16\ \ -2.88\ \ \ -0.3]$

اسئلة تخص الخوارزميات الجينية

1. GA works on two types of spaces --------& ----------

Genotype& phenotype

2. The--------- is the link between chromosomes and the performance decoded solutions

Selection

3. GA provides -------- search in complex landscapes

Random

4. There are two important issues with respect to search strategies----- & --------

Exploration& Exploitation

5. Infeasibility refers to the phenomenon that a solution decoded from chromosome lies---------- the feasible region of given problem

outside

6. Genetic operators --------& ---------- work on genotype space

Crossover& Mutation

7. Genetic operators --------& ---------- work on phenotype space

Evolution& Selection

8. To cross the Hamming cliffs in --------- all bits have to be ----------

Binary encoding, changed simultaneously

9. In --------- the structure of genotype space is -------- to that in phenotype.

Real number encoding, Identical

10. Integer or literal permutation encoding is best for --------

Combinational optimization

What are the important issues with respect to search strategies in Genetic Algorithms?

- There are two important issues with respect to search strategies:: exploration (investigate new and unknown areas in search space) and exploitation (make use of knowledge of solutions previously found in search space to help in find better solutions).

What are Genetic Algorithms? Where GAs work?

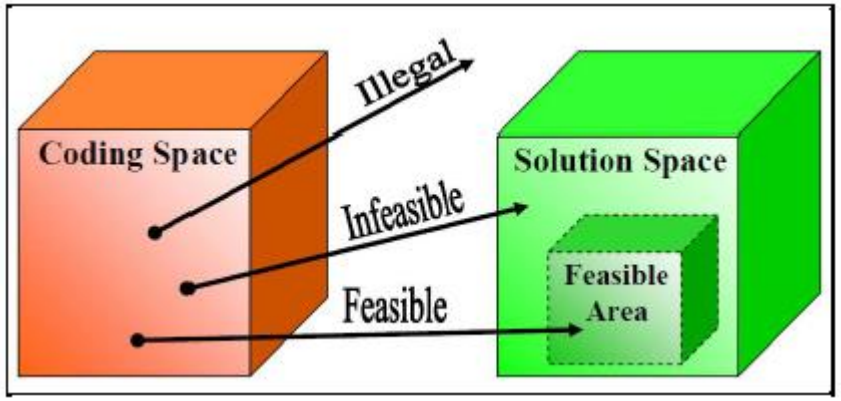
- Genetic Algorithms (or simply GAs) are powerful and widely applicable stochastic search and optimization methods based on the concepts of natural selection and natural evaluation.

GAs work on a population of individuals represents candidate solutions to the optimization problem. These individual are consists of a strings (called chromosomes) of genes.

Explain with diagram infeasibility and illegality?

- **Infeasibility** refers to the phenomenon that a solution decoded from chromosome lies outside the feasible region of given problem. Penalty methods can be used handle infeasible chromosomes. One of these methods is by force genetic algorithms to approach optimal form both sides of feasible and infeasible regions.

- **Illegality** refers to the phenomenon that a chromosome does not represent a solution to a given problem. Repair techniques are usually adopted to convert an illegal chromosome to legal one.



Tutorial Sheet of Genetic Algorithm

Part A: Crossover and Mutation

Q_1: Apply single crossover of 6^{th} point and mutation of 3^{rd} point on the following population :

Parent	Pattern	X	2X-1
P1	00000110		
P2	00000010		
P3	00001000		
P4	00001001		
P5	00000110		
P6	00001111		

Solution:

1. **Calculate the both value of X and 2X-1 from the above Pattern**

Parent	Pattern	X	2X-1
P1	00000110	6	11
P2	00000010	2	3
P3	00001000	8	15
P4	00001001	9	17
P5	00000110	6	11
P6	00001111	15	29

2. **After crossover 6th point**

Children	Pattern	X	2X-1
Child 1	00000010	2	3
Child 2	00000110	6	11
Child 3	00001001	9	17
Child 4	00001000	8	15
Child 5	00000111	7	13
Child 6	00001110	14	27

3. Apply Mutation 3rd point

Children	Pattern	X	2X-1
Child 1	00000010	2	3
Child 2	00000110	6	11
Child 3	00001001	9	17
Child 4	00001000	8	15
Child 5	00000111	7	13
Child 6	00001110	14	27

4. After Mutation 3rd point

Children	Pattern	X	2X-1
Child 1	00100010	34	67
Child 2	00100110	38	75
Child 3	00101001	41	81
Child 4	00101000	40	79
Child 5	00100111	39	77
Child 6	00101110	46	91

Q_2: For the population below with the fitness function **Y= A+B+C+D-30**, apply the following:

1. Single crossover of 4^{th} point
2. Mutation of 4^{th} and 1^{st} points
3. Repeat **No 2** with Mutation of 3^{rd} point

Parent	Pattern		
P1	[12;05;07,15]		
P2	[01;08;20,26]		
P3	[05;10;13,20]		
P4	[07;13;09,11]		

Solution:

Calculate the value of Y(Evaluation) from the above Pattern

Parent	Pattern	Evaluation **Y= A+B+C+D-30**
P1	[12;05;07,15]	12+5+7+15-30=9
P2	[01;08;20,26]	25
P3	[05;10;13,20]	18
P4	[07;13;09,11]	10

1. **After crossover 4th point**

Children	Pattern	Evaluation **Y= A+B+C+D-30**
Child 1	[12;05;07,26]	20
Child 2	[01;08;20,15]	14
Child 3	[05;10;13,11]	9
Child 4	[07;13;09,20]	19

2. After Mutation of 4^{th} and 1^{st} points

4^{th} in Ch_2
1^{st} in Ch_4
Randomly Ch_2, Ch_4

Children	Pattern	Evaluation **Y= A+B+C+D-30**
Child 1	[12;05;07,26]	20
Child 2	[01;08;20,07]	6
Child 3	[05;10;13,11]	9
Child 4	[15;13;09,20]	27

3. H.W

Part B: Selection: Roulette Wheel Selection

Q_3: Apply roulette wheel selection for the following population:

Parent	Pattern	X	2X-1
P1	00000110		
P2	00000010		
P3	00001000		
P4	00001001		
P5	00000110		
P6	00001111		

Randomly values

1. 0.2
2. 0.01
3. 0.8
4. 0.21
5. 0.11
6. 0.65

Solution:

Calculate the both value of X and 2X-1 from the Pattern

Parent	Pattern	X	2X-1
P1	00000110	6	11
P2	00000010	2	3
P3	00001000	8	15
P4	00001001	9	17
P5	00000110	6	11
P6	00001111	15	29

1. Roulette Selection

Parent	Pattern	X	F_{obj}= 2X-1	Fitness Evaluation $[]=\frac{1}{1+f_{obj}}$	Fitness Probability $P=\frac{Part}{All}=\frac{Fitness}{Total\ Fitness}$		Randomly values	New Population
P1	00000110	6	11	1/1+11=0.083	0.083/0.5544=0.15	0.15	0.2	P2
P2	00000010	2	3	1/1+3=0.25	0.25/0.5544=0.45	0.15+0.45 = 0.6	0.01	P1
P3	00001000	8	15	0.0625	0.11	0.71	0.8	P4
P4	00001001	9	17	0.055	0.09	0.8	0.21	P2
P5	00000110	6	11	0.0714	0.13	0.93	0.11	P1
P6	00001111	15	29	0.0333	0.06	0.98=1	0.65	P3
				Total Fitness =0.5552				

Hint:

1. Objective function= f(x) = F_obj
2. Fitness of each individual = Fitness[] = (1 / (1+F_obj[])
3. Fitness probability P[i] = Fitness[] / Total

Q_4: For the following population below, apply the following:

1. Roulette wheel selection
2. Single Crossover over 5^{th} point
3. Mutation of 6^{th} point
4. Repeat **No(3)** to make **Mutations** of 1^{st} point on **Fourth** pattern and 3^{rd} point on **Five** pattern

Randomly values

1. 0.2
2. 0.01
3. 0.8
4. 0.21
5. 0.11
6. 0.65

Solution:

Calculate the both value of X and 2X-1 from the above Pattern

Parent	Pattern	X	2X-1
P1	00000110	6	11
P2	00000010	2	3
P3	00001000	8	15
P4	00001001	9	17
P5	00000110	6	11
P6	00001111	15	29

1. Roulette Selection

Parent	Pattern	X	F-obj= 2X-1	Fitness[]= $\frac{1}{1+f_{-obj}}$	Fitness Probability p[ ]= $\frac{Part}{All}$		Randomly values	New Population
P1	00000110	6	11	1/1+11=0.083	0.083/0.5544=0.15	0.15	0.2	P2
P2	00000010	2	3	1/1+3=0.25	0.25/0.5544=0.45	0.15+0.45= 0.6	0.01	P1
P3	00001000	8	15	0.0625	0.11	0.71	0.8	P4
P4	00001001	9	17	0.055	0.09	0.8	0.21	P2
P5	00000110	6	11	0.0714	0.13	0.93	0.11	P1
P6	00001111	15	29	0.0333	0.06	0.98=1	0.65	P3
				Total=0.5552				

Hint:

1. Objective function= f(x) = F_obj
2. Fitness of each individual = Fitness[] = (1 / (1+F_obj[])
3. Fitness probability P[i] = Fitness[] / Total

2. Single Crossover over 5th point

New Population

Parent	Pattern	X	2X-1
P1	00000010	2	3
P2	00000110	6	11
P3	00001001	9	17
P4	00000010	2	3
P5	00000110	6	11
P6	00001000	8	15

After Crossover over 5th point

Children	Pattern	X	2X-1
Child 1	00000110	6	11
Child 2	00000010	2	3
Child 3	00000010	2	3
Child 4	00001010	10	19
Child 5	00001000	8	15
Child 6	00000110	6	11

3. After Mutation of 6th point

Children	Pattern	X	2X-1
Child 1	00000010	2	11
Child 2	00000110	6	3
Child 3	00000110	6	3
Child 4	00001110	14	19
Child 5	00001100	12	15
Child 6	00000010	2	11

4. H.W

Q_5: For the following population below with the function **Y= A+B+C+D-30**, apply the following:

1. Roulette wheel selection
2. Single Crossover over 4^{th} point
3. Mutation for 4^{th} and 1^{st} point

Randomly values

1. **0.1**
2. **0.7**
3. **0.3**
4. **0.5**

Parent	Pattern		
P1	[12;05;07,15]		
P2	[01;08;20,26]		
P3	[05;10;13,20]		
P4	[07;13;09,11]		

Solution: Calculate the value of Y(Evaluation) from the above Pattern

Parent	Pattern	Evaluation **Y= A+B+C+D-30**
P1	[12;05;07,15]	12+5+7+15-30=9
P2	[01;08;20,26]	25
P3	[05;10;13,20]	18
P4	[07;13;09,11]	10

1. Roulette Selection

Parent	Pattern	F_{obj}= Y= A+B+C+D -30	Fitness[]= $\frac{1}{1+f_{obj}}$	Fitness Probability p[ ]= $\frac{Part}{All}$		Randomly values	New population
P1	**[12;05;07,15]**	**9**	**0.1**	**0.355**	**0.355**	**0.1**	**P1**
P2	**[01;08;20,26]**	**25**	**0.038**	**0.135**	**0.49**	**0.7**	**P4**
P3	**[05;10;13,20]**	**18**	**0.053**	**0.188**	**0.678**	**0.3**	**P1**
P4	**[07;13;09,11]**	**10**	**0.09**	**0.32**	**1**	**0.5**	**P3**
			Total Fitness = 0.2819				

The new population will be:

Parent	Pattern
P1	**[12;05;07,15]**
P2	**[07;13;09,11]**
P3	**[12;05;07,15]**
P4	**[05;10;13,20]**

Hint:

1. Objective function= f(x) = F_obj
2. Fitness of each individual = Fitness[] = (1 / (1+F_obj[])
3. Fitness probability P[i] = Fitness[] / Total

2. **After Single Crossover 4th point**

Children	Pattern
Child 1	[12;05;07,11]
Child 2	[07;13;09,15]
Child 3	[12;05;07, 20]
Child 4	[05;10;13,15]

3. **Mutation for 4th and1st point**

By assuming the Mutation Rate= Population size × rate; rate=0.4 (Given)

= 4×0.4=2

Choose randomly c[2], c[4]

Point 4,1

Children	Pattern
Child 1	[12;05;07,11]
Child 2	[07;13;09,15]
Child 3	[12;05;07, 20]
Child 4	[05;10;13,15]

After Mutation

Children	Pattern
Child 1	[12;05;07,11]
Child 2	[07;13;09,05]
Child 3	[12;05;07, 20]
Child 4	[15;10;13,15]

The new population can be represented as Roulette Selection (H.W)

Parent	Pattern	F_{obj}= Y= A+B+C+D-30	Fitness[] $= \frac{1}{1+f_{obj}}$	Fitness Probability $p[\,] = \frac{Part}{All}$		Randomly	New population
P1	[12;05;07,11]					0.1	P1
P2	[07;13;09,05]					0.7	P4
P3	[12;05;07,20]					0.3	P1
P4	[15;10;03,15]					0.5	P3
			Total=				

Hint:

1. Objective function= f(x) = F_obj
2. Fitness of each individual = Fitness[] = (1 / (1+F_obj[])
3. Fitness probability P[i] = Fitness[] / Total

Q_6:H.W: Suppose a genetic algorithm uses populations as in pattern below with fitness function of individual x: **f(x) = a +2 b +3 c +4 d**

x_1 =[a;b;c;d] = [12;05;23;08]
x_2 = [a;b;c;d] = [02;21;18;03]
x_3 = [a;b;c;d] = [10;04;13;14]
x_4 = [a;b;c;d] = [20;01;10;06]

1. Calculate the objective function of each individual
2. Compute the fitness of each individual
3. Calculate the fitness probability of each individual
4. Apply Single Crossover of 2nd point
5. Apply Mutation for 1st and3rd point

Hint:

1. Objective function= f(x) = F_obj
2. Fitness of each individual = Fitness[] = (1 / (1+F_obj[])
3. Fitness probability P[i] = Fitness[] / Total

Part C: Questions and Answers

Q_1: What are the important issues with respect to search strategies in Genetic Algorithms?

Answer:

There are two important issues with respect to search strategies:

1. Exploration (investigate new and unknown areas in search space)
2. Exploitation (make use of knowledge of solutions previously found in search space).

Q_2: What are Genetic Algorithms? Where GAs work?

Answer:

Genetic Algorithms (or simply GAs) are powerful and widely applicable stochastic search and optimization methods based on the concepts of natural selection and natural evaluation.

GAs work on a population of individuals represents candidate solutions to the optimization problem.

Q_3: **S**tate the Encoding methods? Explain each.

Answer:

Encoding methods can be classified as follows:

1) Binary encoding
2) Real-number encoding
3) Integer or literal permutation encoding

Q_4: What are the two requirements should a problem satisfy in order to be suitable for solving it by a GA?

Answer:

1. The fitness function can be well–defined.
2. Solutions should be decomposable into steps (building blocks) which could be then encoded as chromosomes.

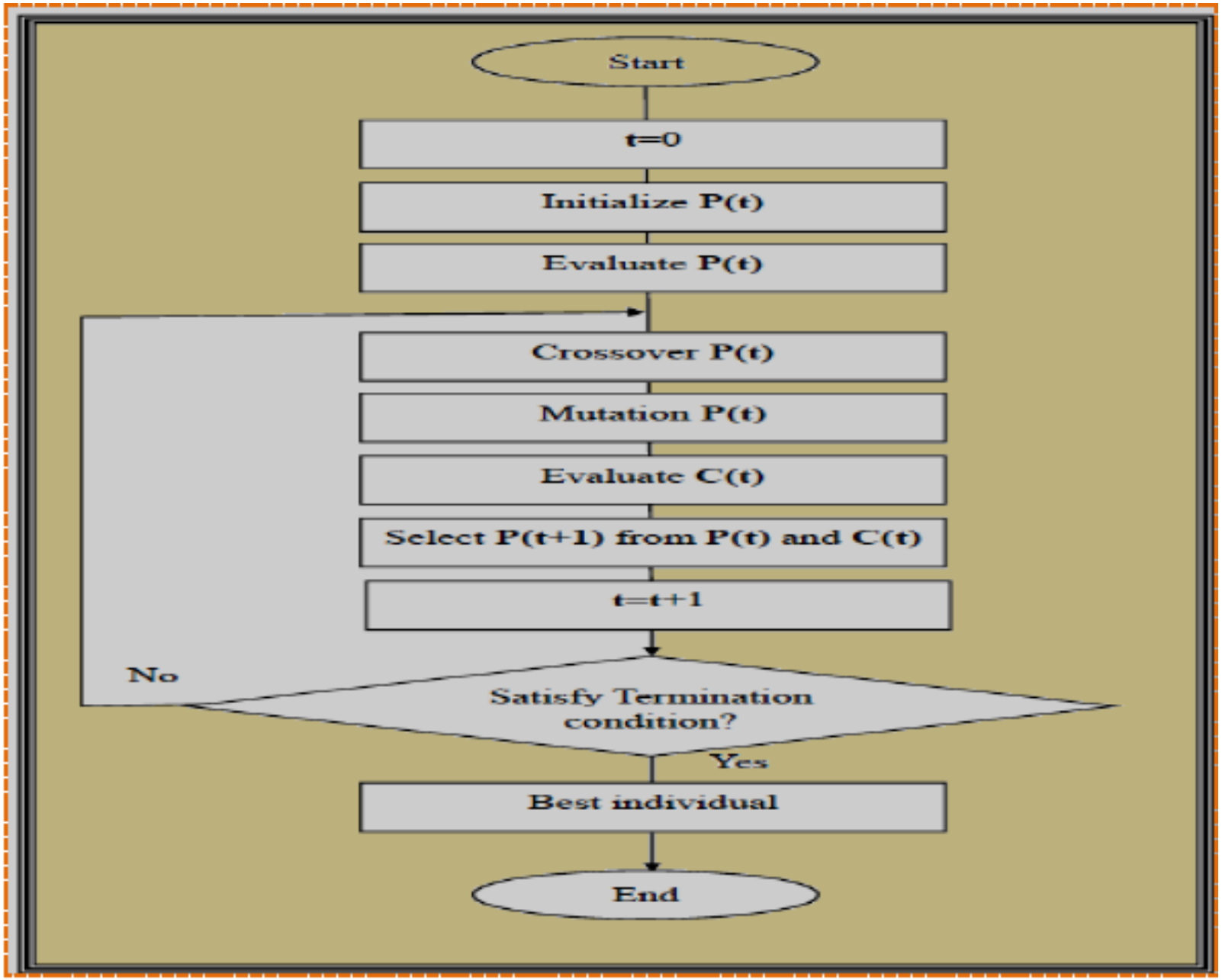
Q_6: What is the general structure of the Genetic Algorithms?

Answer:

The general structure of the Genetic algorithms is as follow:

```
Begin
{
t=0;
Initialize P(t);
Evaluate P(t);
While (not termination condition) do
Begin
{
Apply crossover and mutation to P(t) to yield C(t);
Evaluate C(t);
Select P(t+1) from P(t) and C(t);
t=t+1;
}
End
}
End
```

Q_7: Based on the flowchart explains how genetic algorithms work?



Q_8: List Six Applications of Genetic Algorithms

Answer:

1. Optimization
2. Automatic Programming
3. Machine and robot learning
4. Economic models
5. Immune system models
6. Population genetics models

Q_9: Mention the most Advantages and Limitations of Genetic Algorithm

The advantages of genetic algorithm includes,

1. Parallelism
2. Liability
3. Solution space is wider
4. The fitness landscape is complex
5. Easy to discover global optimum

The limitation of genetic algorithm includes,

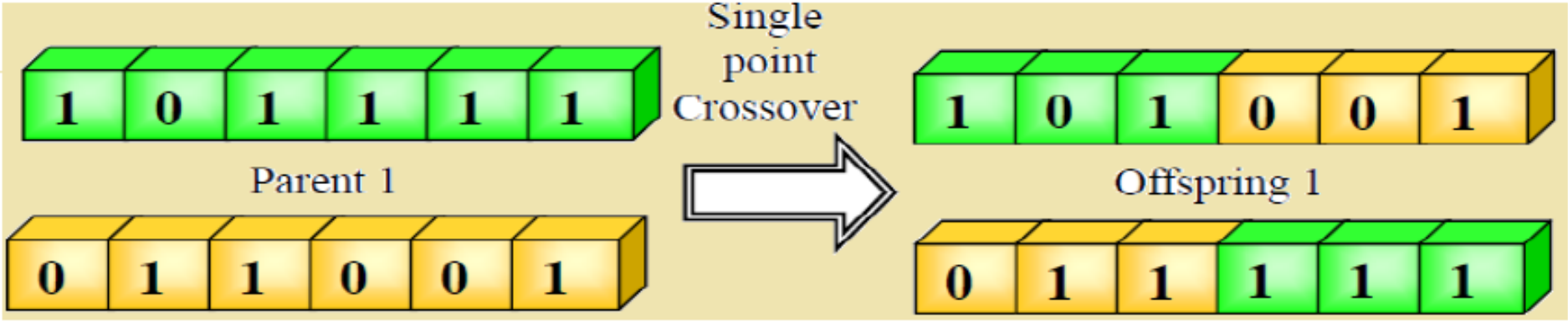
1. The problem of identifying fitness function
2. Definition of representation for the problem
3. Premature convergence occurs
4. The problem of choosing the various parameters like the size of the population, Mutation rate, cross over rate, the selection method and its strength.
5. Cannot use gradients

$Q_{10:}$ State the types of crossover and give **only one** example for any type.

Answer:

1. **Single Point Crossover**
2. **Multi Point Crossover**
3. **Uniform Crossover**

Example: Single Point Crossover

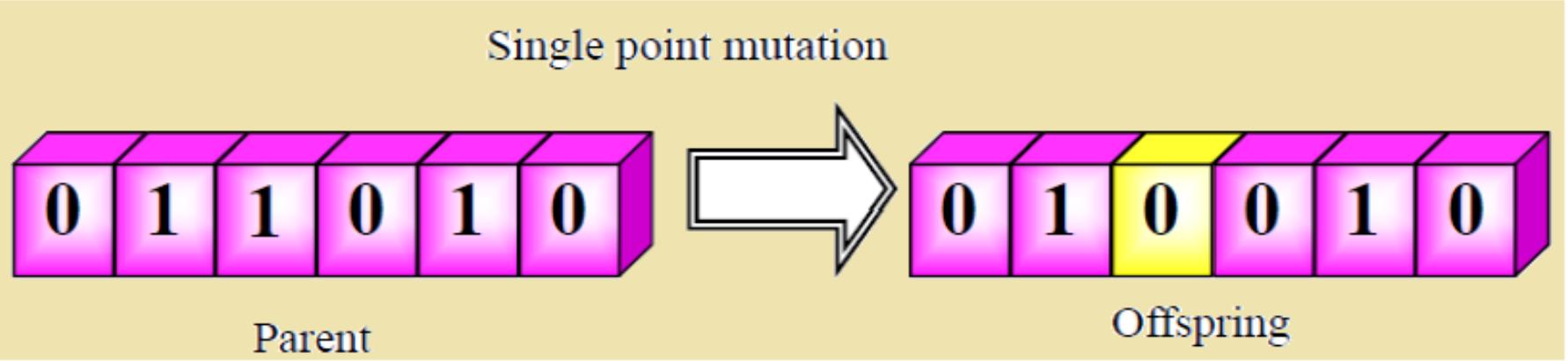


Q_{11}: State the types of mutation and give **only one** example for any type

Answer:

1. **Single Point Mutation**
2. **Multi Point Mutation**

Example: Single Point Mutation



Q_{12}: Choose the suitable words for the following point:

1. GA works on two types of spaces …….. & ……
2. The …… is the link between chromosomes and the performance decoded solutions.
3. GA provides …… search in complex landscapes

4. There are two important issues with respect to search strategies

   ..... & .....

5. Infeasibility refers to the phenomenon that a solution decoded from

   chromosome lies ..... the feasible region of given problem

6. Genetic operators .....& .....work on genotype space
7. To cross the Hamming cliffs in ..... all bits have to be

   .....

8. Genetic operators .....& .....work on phenotype space
9. In ..... the structure of genotype space is ..... to that in

   phenotype

10. Integer or literal permutation encoding is best for .....

مع تمنياتنا لجميع الطلبة بالنجاح