Machine Learning

Machine Learning

Artificial Intelligence Branch

Third Class

Prof. Dr. Ayad R. Abbas

2024-2025

REFERENCES

1.	Fundamentals of Neural Networks: Architecture, Algorithms, and application. By Laurene Fausett
2.	Neural Networks. By Phil Picton
3.	Neural Networks. Fundamentals, Application, Examples. By Werner Kinnebrock
4.	Neural network for identification, prediction and control. By D. T. Pham and X. Liu.
5.	Machine Learning, Tom Mitchell, McGraw Hill, 1997.
6.	COS 511: Theoretical Machine Learning http://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0204.pdf
7.	http://people.revoledu.com/kardi/tutorial/DecisionTree/how-to-use-decision-tree.htm

CHAPTER ONE INTRODUCTION

1.1 DEFINITION OF LEARNING SYSTEM

WHAT IS MACHINE LEARNING?

Machine learning studies computer algorithms for learning to do stuff. We might, for instance, be interested in learning to complete a task, or to make accurate predictions, or to behave intelligently. The learning that is being done is always based on some sort of observations or data, such as examples (the most common case in this course), direct experience, or instruction. So in general, machine learning is about learning to do better in the future based on what was experienced in the past.

The emphasis of machine learning is on automatic methods. In other words, the goal is to devise learning algorithms that do the learning automatically without human intervention or assistance. The machine learning paradigm can be viewed as "programming by example."

Often we have a specific task in mind, such as spam filtering. But rather than program the computer to solve the task directly, in machine learning, we seek methods by which the computer will come up with its own program based on examples that we provide.

Machine learning is a core subarea of artificial intelligence. It is very unlikely that we will be able to build any kind of intelligent system capable of any of the facilities that we associate with intelligence, such as language or vision, without using learning to get there.

These tasks are otherwise simply too difficult to solve. Further, we would not consider a system to be truly intelligent if it were incapable of learning since learning is at the core of intelligence.

1.2 GOALS OF MACHINE LEARNING RESEARCH

The primary goal of machine learning research is to develop general purpose algorithms of practical value. Such algorithms should be efficient. As usual, as computer scientists, we care about time and space efficiency. But in the context of learning, we also care a great deal about another precious resource, namely, the amount of data that is required by the learning algorithm.

Learning algorithms should also be as general purpose as possible. We are looking for algorithms that can be easily applied to a broad class of learning problems.

Of primary importance, we want the result of learning to be a prediction rule that is as accurate as possible in the predictions that it makes.

Occasionally, we may also be interested in the interpretability of the prediction rules produced by learning. In other words, in some contexts (such as medical diagnosis), we want the computer to find prediction rules that are easily understandable by human experts.

As mentioned above, machine learning can be thought of as "programming by example."

What is the advantage of machine learning over direct programming? First, the results of using machine learning are often more accurate than what can be created through direct programming. The reason is that machine learning algorithms are data driven, and are able to examine large amounts of data. On the other hand, a human expert is likely to be guided by imprecise impressions or perhaps an examination of only a relatively small number of examples.

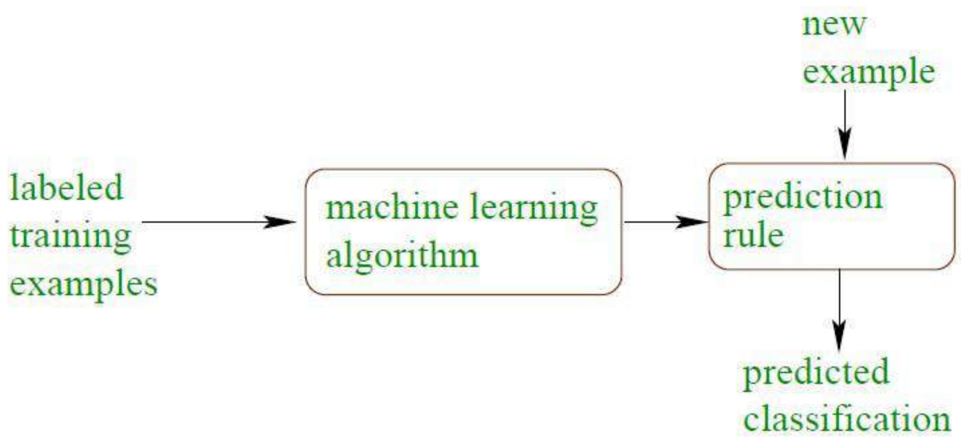


Figure 1: Diagram of a typical learning problem.

1.3 LEARNING MODELS

To study machine learning mathematically, we need to formally define the learning problem.

This precise definition is called a learning model. A learning model should be rich enough to capture important aspects of real learning problems, but simple enough to study the problem mathematically. As with any mathematical model, simplifying assumptions are unavoidable.

A learning model should answer several questions:

- What is being learned?
- How is the data being generated? In other words, where does it come from?
- How is the data presented to the learner? For instance, does the learner see all the data at once or only one example at a time?
- What is the goal of learning in this model?

CHAPTER TWO INDUCTIVE CLASSIFICATION

2.1CONCEPT LEARNING TASK

To ground our discussion of concept learning, consider the example task of learning the target concept "days on which my friend Aldo enjoys his favorite water sport." Table 2.1 describes a set of example days, each represented by a set of attributes. The attribute EnjoySport indicates whether or not Aldo enjoys his favorite water sport on this day. The task is to learn to predict the value of EnjoySport for an arbitrary day, based on the values of its other attributes.

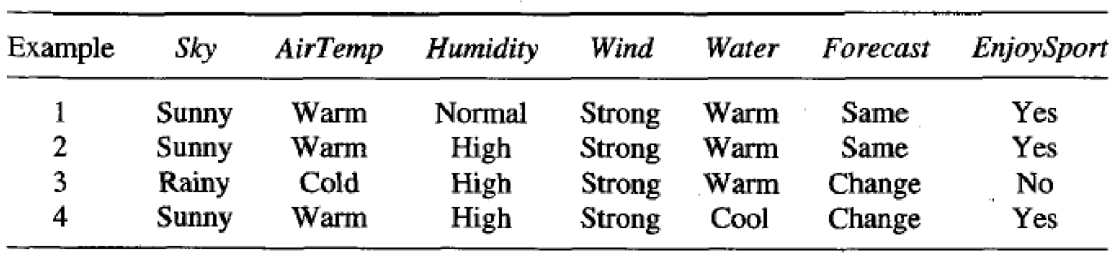
What hypothesis representation shall we provide to the learner in this case?

Let us begin by considering a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes. In particular, let each hypothesis be a vector of six constraints, specifying the values of the six attributes **Sky, AirTemp, Humidity, Wind, Water,** and **Forecast**. For each attribute, the hypothesis will either

- Indicate by a "?' that any value is acceptable for this attribute,
- Specify a single required value (e.g., ***Warm)*** for the attribute, or

Indicate by a *"Ø"* that no value is acceptable.

If some instance x satisfies all the constraints of hypothesis h, then h classifies x as a positive example (h(x) = 1). To illustrate, the hypothesis that Aldo enjoys his favorite sport only on cold days with high humidity (independent of the values of the other attributes) is represented by the expression (?, Cold, High, ?, ?, ?)



Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

The most general hypothesis-that every day is a positive example-is represented by (?, ?, ?, ?, ?, ?) and the most specific possible hypothesis-that no day is a positive example-is represented by (0,0,0,0,0,0) To summarize, the EnjoySport concept learning task requires learning the set of days for which EnjoySport = yes, describing this set by a conjunction of constraints over the instance attributes. In general, any concept learning task can be described by the set of instances over which the target function is defined, the target function, the set of candidate hypotheses considered by the learner, and the set of available training examples. The definition of the EnjoySport concept learning task in this general form.

2.2 CONCEPT LEARNING AS SEARCH

Concept learning can be viewed as the task of searching through a large space of hypothesis implicitly defined by the hypothesis representation. The goal of the concept learning search is to find the hypothesis that best fits the training examples.
Concept learning is a task of searching a hypotheses space the representation chosen for hypotheses determines the search space In the example we have:

$3 \times 2^5 = 96$ possible instances (6 attributes)

$1 + 4 \times 3^5 = 973$ possible hypothesis

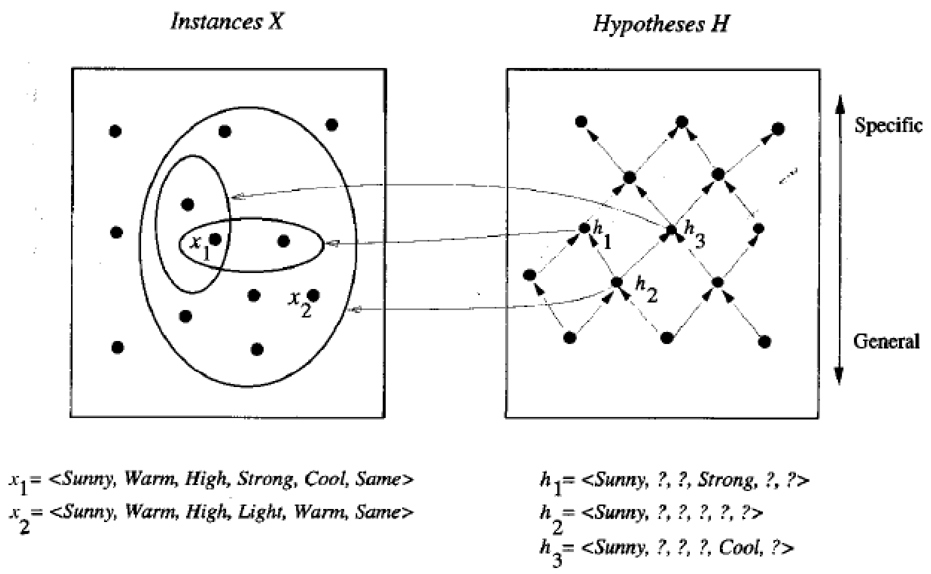
(considering that all the hypothesis with some ∅ are semantically equivalent)

2.3 GENERAL-TO-SPECIFIC ORDERING OF HYPOTHESES

Many algorithms for concept learning organize the search through the hypothesis space by relying on a very useful structure that exists for any concept learning problem: a general-to-specific ordering of hypotheses. By taking advantage of this naturally occurring structure over the hypothesis space, we can design learning algorithms that exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis. To illustrate the general-to-specific ordering, consider the two hypotheses

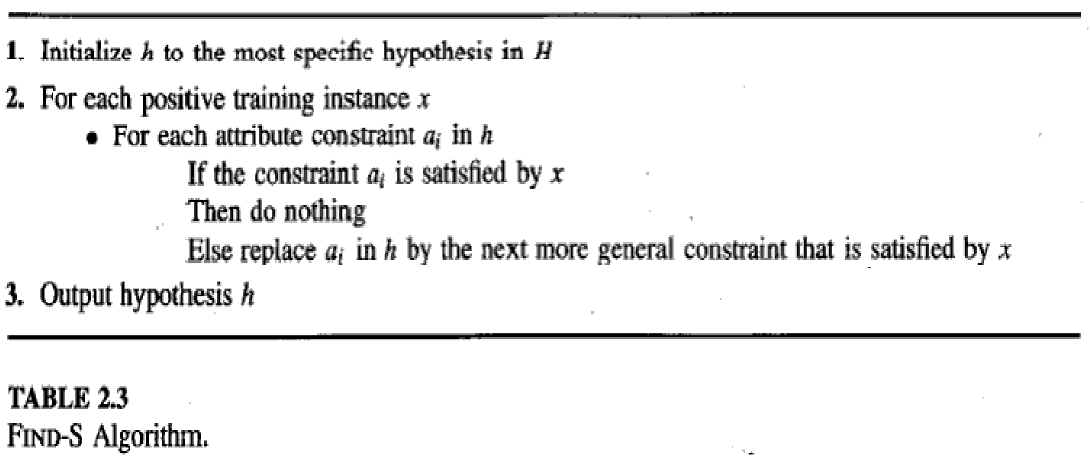
h1 = (Sunny, ?, ?, Strong, ?, ?)

h2 = (Sunny, ?, ?, ?, ?, ?)



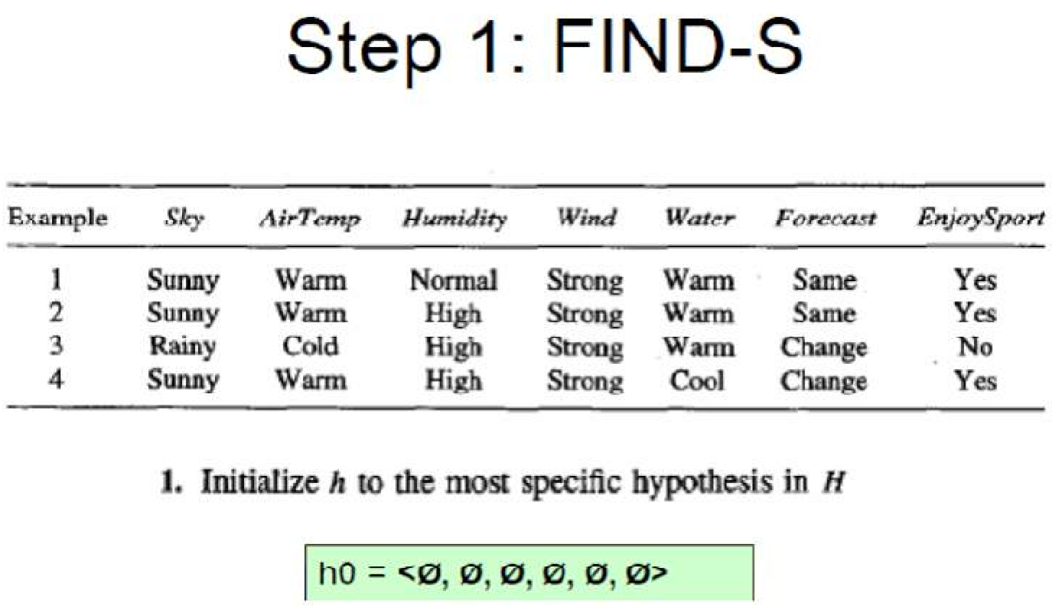
Instances X
Hypotheses H
Specific
General
x_1
x_2
h_1
h_2
h_3
x_1= <Sunny, Warm, High, Strong, Cool, Same>
x_2= <Sunny, Warm, High, Light, Warm, Same>
h_1= <Sunny, ?, ?, Strong, ?, ?>
h_2= <Sunny, ?, ?, ?, ?, ?>
h_3= <Sunny, ?, ?, ?, Cool, ?>

FIND-S: FINDING THE MOST SPECIFIC HYPOTHESIS



```
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
       • For each attribute constraint a_i in h
               If the constraint a_i is satisfied by x
               Then do nothing
               Else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h
```

TABLE 2.3
FIND-S Algorithm.



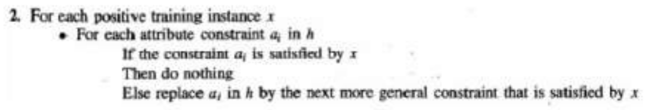
Step 1: FIND-S

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

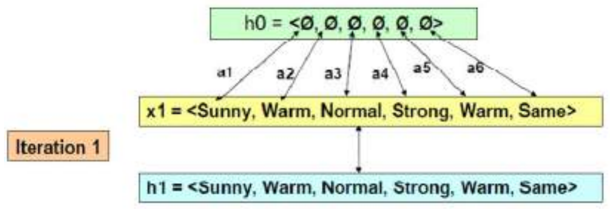
1. Initialize h to the most specific hypothesis in H

h0 = <Ø, Ø, Ø, Ø, Ø, Ø>

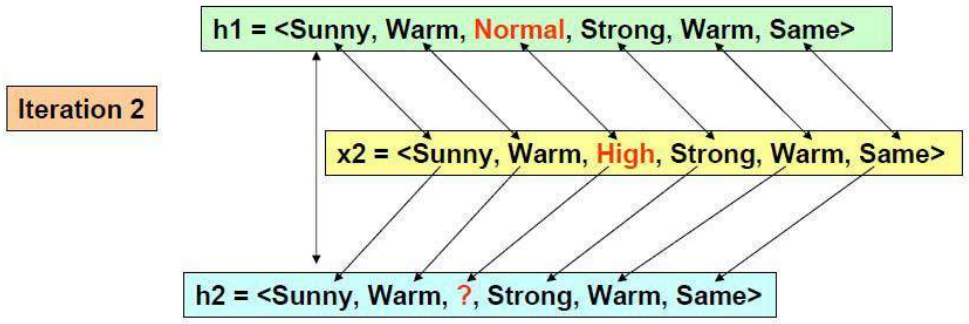
Step 2: FIND-S



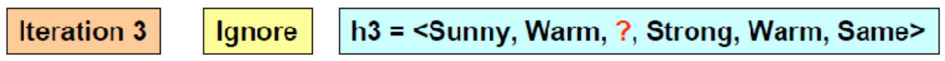
```
2. For each positive training instance x
     • For each attribute constraint a_i in h
           If the constraint a_i is satisfied by x
           Then do nothing
           Else replace a_i in h by the next more general constraint that is satisfied by x
```

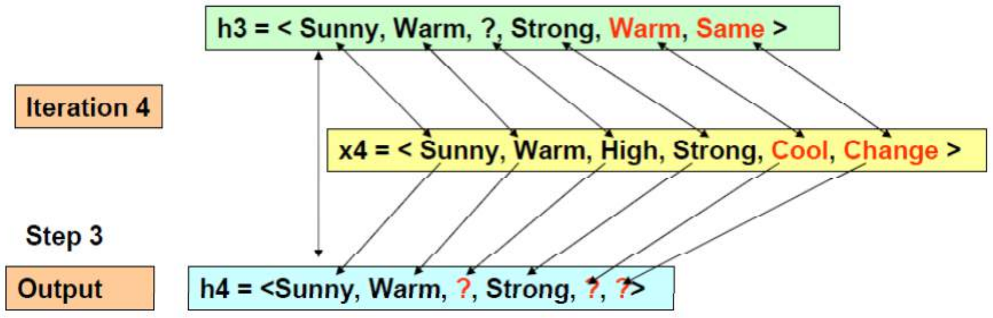


h0 = <Ø, Ø, Ø, Ø, Ø, Ø>
a1
a2
a3
a4
a5
a6
x1 = <Sunny, Warm, Normal, Strong, Warm, Same>
Iteration 1
h1 = <Sunny, Warm, Normal, Strong, Warm, Same>



h1 = <Sunny, Warm, Normal, Strong, Warm, Same>
Iteration 2
x2 = <Sunny, Warm, High, Strong, Warm, Same>
h2 = <Sunny, Warm, ?, Strong, Warm, Same>





h3 = < Sunny, Warm, ?, Strong, Warm, Same >
Iteration 4
x4 = < Sunny, Warm, High, Strong, Cool, Change >
Step 3
Output
h4 = <Sunny, Warm, ?, Strong, ?, ?>

CHAPTER THREE

DECISION TREE

3.1 DECISION-TREE LEARNING

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree. Learned trees can also be re-represented as sets of if-then rules to improve human readability.

These learning methods are among the most popular of inductive inference algorithms and have been successfully applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.

3.2 DECISION TREE REPRESENTATION

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending.

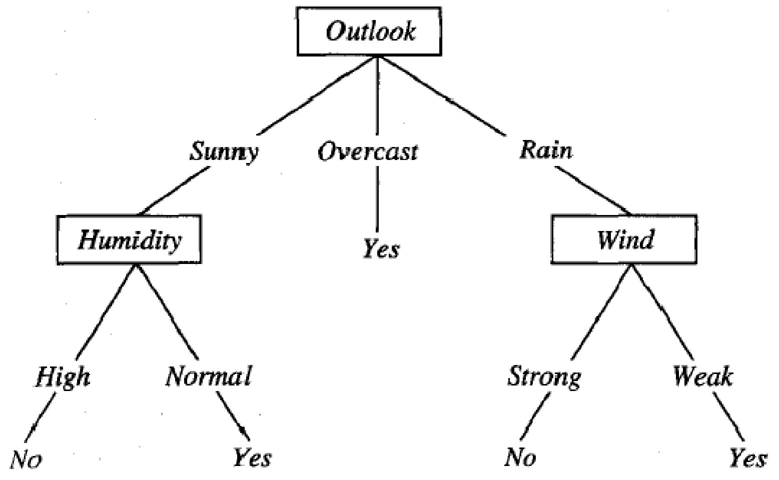


FIGURE 3.1
A decision tree for the concept *PlayTennis*. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf (in this case, *Yes* or *No*). This tree classifies Saturday mornings according to whether or not they are suitable for playing tennis.

From that node corresponds to one of the possible values for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the sub-tree rooted at the new node.

Figure ***3.1*** illustrates a typical learned decision tree. This decision tree classifies Saturday mornings according to whether they are suitable for playing tennis.

For example, the instance

$$\langle Outlook = Sunny,\ Temperature = Hot,\ Humidity = High,\ Wind = Strong\rangle$$

Would be sorted down the leftmost branch of this decision tree and would therefore be classified as a negative instance (i.e., the tree predicts that ***PlayTennis***

= no).

This tree and the example used in Table ***3.2*** to illustrate the ID3 learning algorithm are adapted from (Quinlan ***1986).***

Many **practical problems** have been found to fit these characteristics. Decision tree learning has therefore been applied to problems such as learning to classify medical patients by their disease, equipment malfunctions by their cause, and loan applicants by their likelihood of defaulting on payments. Such problems, in which the task is to classify examples into one of a discrete set of possible categories, are often referred to as ***classifications problems.***

3.3 THE BASIC DECISION TREE LEARNING ALGORITHM

Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees. This approach is exemplified by the ID3 algorithm (Quinlan 1986) and its successor C4.5 (Quinlan 1993), which form the primary focus of our discussion here.

Basic algorithm, ID3, learns decision trees by constructing them top down, beginning with the question "which attribute should be tested at the root of the tree?'To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples

3.4 WHICH ATTRIBUTE IS THE BEST CLASSIFIER?

The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree. We would like to select the attribute that is most useful for classifying examples. What is a good quantitative measure of the worth of an attribute? We will define a statistical property, called ***information gain that*** measures how well a given attribute separates the training examples according to their target classification. ID3 uses this information gain measure to select among the candidate attributes at each step while growing the tree.

3.5 ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

In order to define information gain precisely, we begin by defining a measure commonly used in information theory, called ***entropy,*** that characterizes the (im) purity of an arbitrary collection of examples. Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

where $p_{\oplus}$ is the proportion of positive examples in S and $p_{\ominus}$ is the proportion of negative examples in S. In all calculations involving entropy we define $0 \log 0$ to be 0.

To illustrate, suppose S is a collection of 14 examples of some boolean concept, including 9 positive and 5 negative examples (we adopt the notation [9+, 5−] to summarize such a sample of data). Then the entropy of S relative to this boolean classification is

$$\begin{aligned} Entropy([9+,5-]) &= -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) \\ &= 0.940 \end{aligned} \qquad (3.2)$$

Notice that the entropy is 0 if all members of S belong to the same class. For example, if all members are positive ($p_{\oplus} = 1$), then $p_{\ominus}$ is 0, and $Entropy(S) = -1 \cdot \log_2(1) - 0 \cdot \log_2 0 = -1 \cdot 0 - 0 \cdot \log_2 0 = 0$. Note the entropy is 1 when the collection contains an equal number of positive and negative examples. If the collection contains unequal numbers of positive and negative examples, the

3.6 INFORMATION GAIN MEASURES THE EXPECTED REDUCTION IN ENTROPY

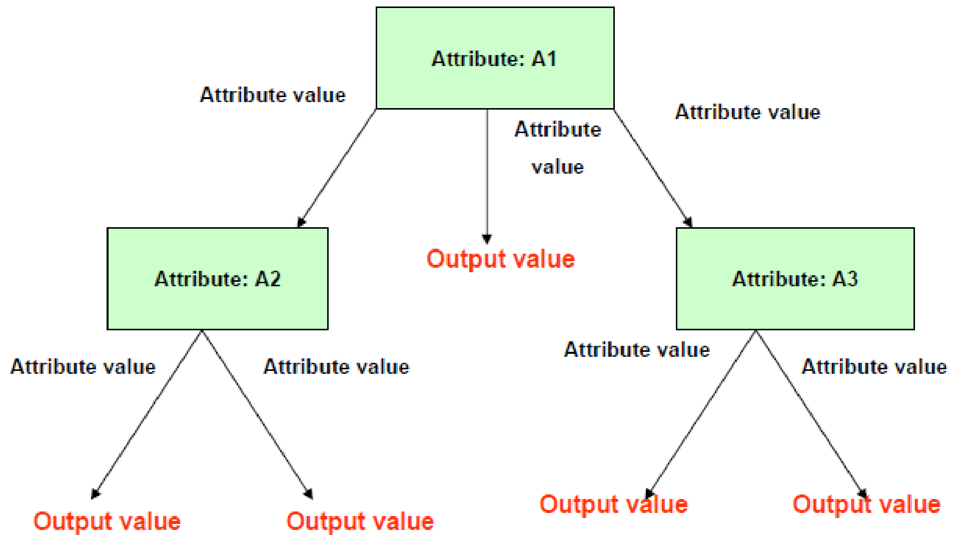
Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data. The measure we will use, called ***information gain,*** is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. More precisely, the information gain, ***Gain(S, A)*** of ***an*** attribute **A,** relative to a collection of examples ***S,*** is defined as

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

For example, suppose S is a collection of training-example days described by attributes including *Wind*, which can have the values *Weak* or *Strong*. As before, assume S is a collection containing 14 examples, $[9+, 5-]$. Of these 14 examples, suppose 6 of the positive and 2 of the negative examples have $Wind = Weak$, and the remainder have $Wind = Strong$. The information gain due to sorting the original 14 examples by the attribute *Wind* may then be calculated as

$$\begin{aligned}
Values(Wind) &= Weak, Strong \\
S &= [9+, 5-] \\
S_{Weak} &\leftarrow [6+, 2-] \\
S_{Strong} &\leftarrow [3+, 3-] \\
Gain(S, Wind) &= Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v) \\
&= Entropy(S) - (8/14) Entropy(S_{Weak}) \\
&\quad - (6/14) Entropy(S_{Strong}) \\
&= 0.940 - (8/14)0.811 - (6/14)1.00 \\
&= 0.048
\end{aligned}$$

Building Decision Tree



From **table D** and for each associated subset Si , we compute degree of impurity. We have discussed about how to compute these indices in the previous section.

To compute the degree of impurity, we must distinguish whether it is come from the parent table D or it come from a subset table Si with attribute i.

If the table is a parent table D, we simply compute the number of records of each class. For example, in the parent table below, we can compute degree of impurity based on **transportation mode**. In this case we have 4 Busses, 3 Cars and 3 Trains (in short 4B, 3C, 3T): Based on these data, we can compute probability of each class. Since probability is equal to frequency relative, we have

Prob (Bus) = 4 / 10 = 0.4

Prob (Car) = 3 / 10 = 0.3

Prob (Train) = 3 / 10 = 0.3

Observe that when to compute probability, we only focus on the *classes* , not on the *attributes* . Having the probability of each class, now we are ready to compute the quantitative indices of impurity degrees.

3.7 ENTROPY

One way to measure impurity degree is using entropy.

$$Entropy = \sum_j -p_j log_2 p_j$$

Example: Given that Prob (Bus) = 0.4, Prob (Car) = 0.3 and Prob (Train) = 0.3, we can now compute entropy as

Entropy = – 0.4 log (0.4) – 0.3 log (0.3) – 0.3 log (0.3) = 1.571 The logarithm is base 2.

Entropy of a pure table (consist of single class) is zero because the probability is 1 and log (1) = 0. Entropy reaches maximum value when all classes in the table have equal probability. Figure below plots the values of maximum entropy for different number of classes n, where probability is equal to p=1/n. I this case, maximum entropy is equal to -n*p*log p. Notice that the value of entropy is larger than 1 if the number of classes is more than 2.

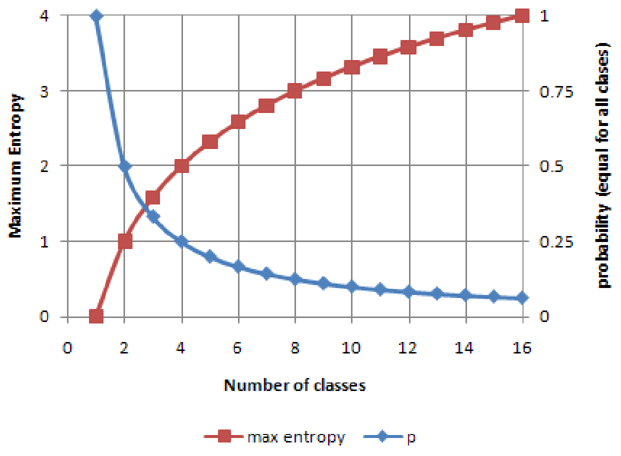


Table D

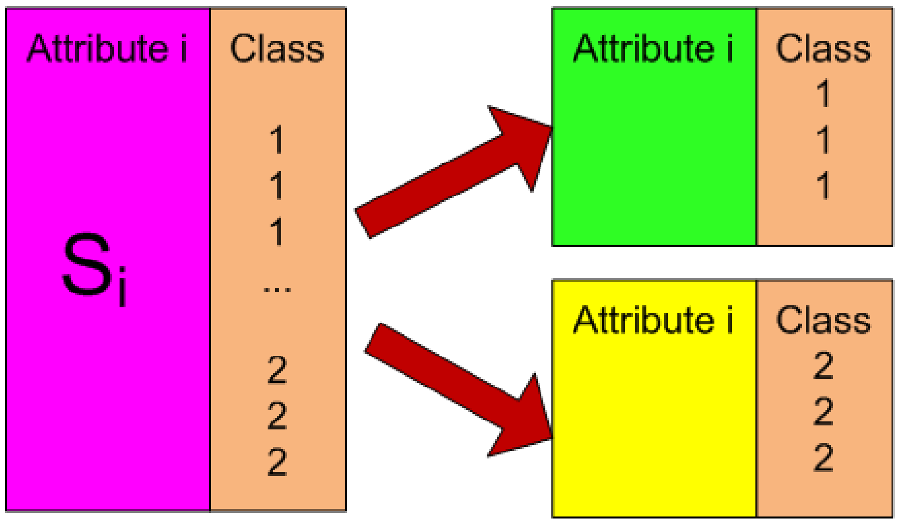
Data

Attributes				Classes
Gender	Car ownership	Travel Cost ($)/km	Income Level	Transportation mode
Male	0	Cheap	Low	Bus
Male	1	Cheap	Medium	Bus
Female	1	Cheap	Medium	Train
Female	0	Cheap	Low	Bus
Male	1	Cheap	Medium	Bus
Male	0	Standard	Medium	Train
Female	1	Standard	Medium	Train
Female	1	Expensive	High	Car
Male	2	Expensive	Medium	Car
Female	2	Expensive	High	Car

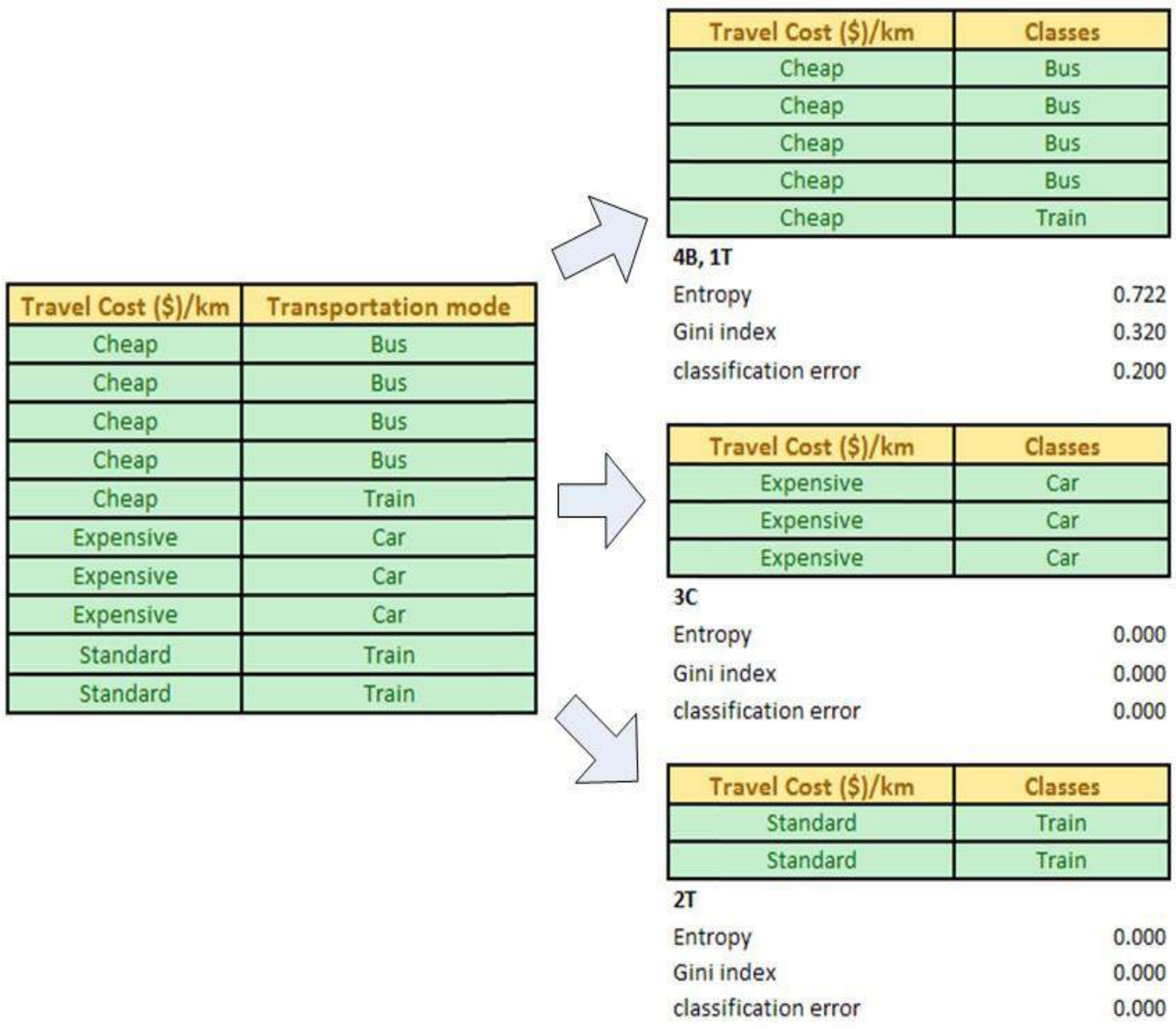
4B, 3C, 3T

Entropy	1.571
Gini index	0.660
Classification error	0.600

If the table is a subset of attribute table Si, we need to separate the computation of impurity degree for each value of the attribute i.



For example, attribute Travel cost per km has three values: Cheap, Standard and Expensive. Now we sort the table Si = [Travel cost/km, Transportation mode] based on the values of Travel cost per km. Then we separate each value of the travel cost and compute the degree of impurity (either using entropy, gini index or classification error).



Travel Cost ($)/km	Transportation mode
Cheap	Bus
Cheap	Bus
Cheap	Bus
Cheap	Bus
Cheap	Train
Expensive	Car
Expensive	Car
Expensive	Car
Standard	Train
Standard	Train

Travel Cost ($)/km	Classes
Cheap	Bus
Cheap	Bus
Cheap	Bus
Cheap	Bus
Cheap	Train

4B, 1T

Entropy	0.722
Gini index	0.320
classification error	0.200

Travel Cost ($)/km	Classes
Expensive	Car
Expensive	Car
Expensive	Car

3C

Entropy	0.000
Gini index	0.000
classification error	0.000

Travel Cost ($)/km	Classes
Standard	Train
Standard	Train

2T

Entropy	0.000
Gini index	0.000
classification error	0.000

3.8 INFORMATION GAIN

The reason for different ways of computation of impurity degrees between data table D and subset table S i is because we would like to compare the difference of impurity degrees *before* we split the table (i.e. data table D) and *after* we split the table according to the values of an attribute i (i.e. subset table Si) . The measure to compare the difference of impurity degrees is called **information gain** . We would like to know what our gain is if we split the data table based on some attribute values.

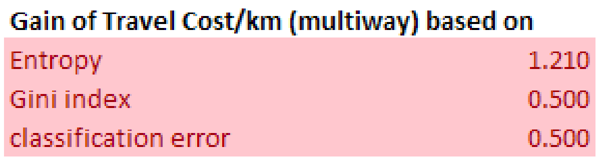
Information gain is computed as impurity degrees of the parent table and weighted summation of impurity degrees of the subset table. The weight is based on the number of records for each attribute values. Suppose we will use entropy as measurement of impurity degree, then we have:

Information gain (i) = Entropy of parent table D – Sum (n k /n * Entropy of each value k of subset table Si)

For example, our data table D has classes of 4B, 3C, 3T which produce entropy of

1.571. Now we try the attribute Travel cost per km which we split into three: Cheap that has classes of 4B, 1T (thus entropy of 0.722), Standard that has classes of 2T (thus entropy = 0 because pure single class) and Expensive with single class of 3C (thus entropy also zero).

The information gain of attribute Travel cost per km is computed as 1.571
– (5/10 * 0.722+2/10*0+3/10*0) = 1.210
You can also compute information gain based on Gini index or classification error in the same method. The results are given below.



Gain of Travel Cost/km (multiway) based on	
Entropy	1.210
Gini index	0.500
classification error	0.500

For each attribute in our data, we try to compute the information gain. The illustration below shows the computation of information gain for the first iteration (based on the data table) for other three attributes of Gender, Car ownership and Income level.

Subset

Gender	Classes
Female	Bus
Female	Car
Female	Car
Female	Train
Female	Train

1B, 2C, 2T

Entropy	1.522
Gini index	0.640
classification error	0.600

Gender	Classes
Male	Bus
Male	Bus
Male	Bus
Male	Car
Male	Train

3B, 1C, 1T

Entropy	1.371
Gini index	0.560
classification error	0.400

Gain of Gender based on

Entropy	0.125
Gini index	0.060
classification error	0.100

Car ownership	Classes
0	Bus
0	Bus
0	Train

2B, 1T

Entropy	0.918
Gini index	0.444
classification error	0.333

Car ownership	Classes
1	Bus
1	Bus
1	Car
1	Train
1	Train

2B, 1C, 2T

Entropy	1.522
Gini index	0.640
classification error	0.600

Car ownership	Classes
2	Car
2	Car

2C

Entropy	0.000
Gini index	0.000
classification error	0.000

Gain of Car ownership (multiway) based on

Entropy	0.534
Gini index	0.207
classification error	0.200

Income Level	Classes
High	Car
High	Car

2C

Entropy	0.000
Gini index	0.000
classification error	0.000

Income Level	Classes
Low	Bus
Low	Bus

2B

Entropy	0.000
Gini index	0.000
classification error	0.000

Income Level	Classes
Medium	Bus
Medium	Bus
Medium	Car
Medium	Train
Medium	Train
Medium	Train

2B, 1C, 3T

Entropy	1.459
Gini index	0.611
classification error	0.500

Gain of Income Level (multiway) based on

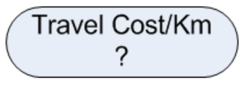
Entropy	0.695
Gini index	0.293
classification error	0.300

Table below summarizes the information gain for all four attributes. In practice, you don't need to compute the impurity degree based on three methods. You can use either one of Entropy or Gini index or index of classification error.

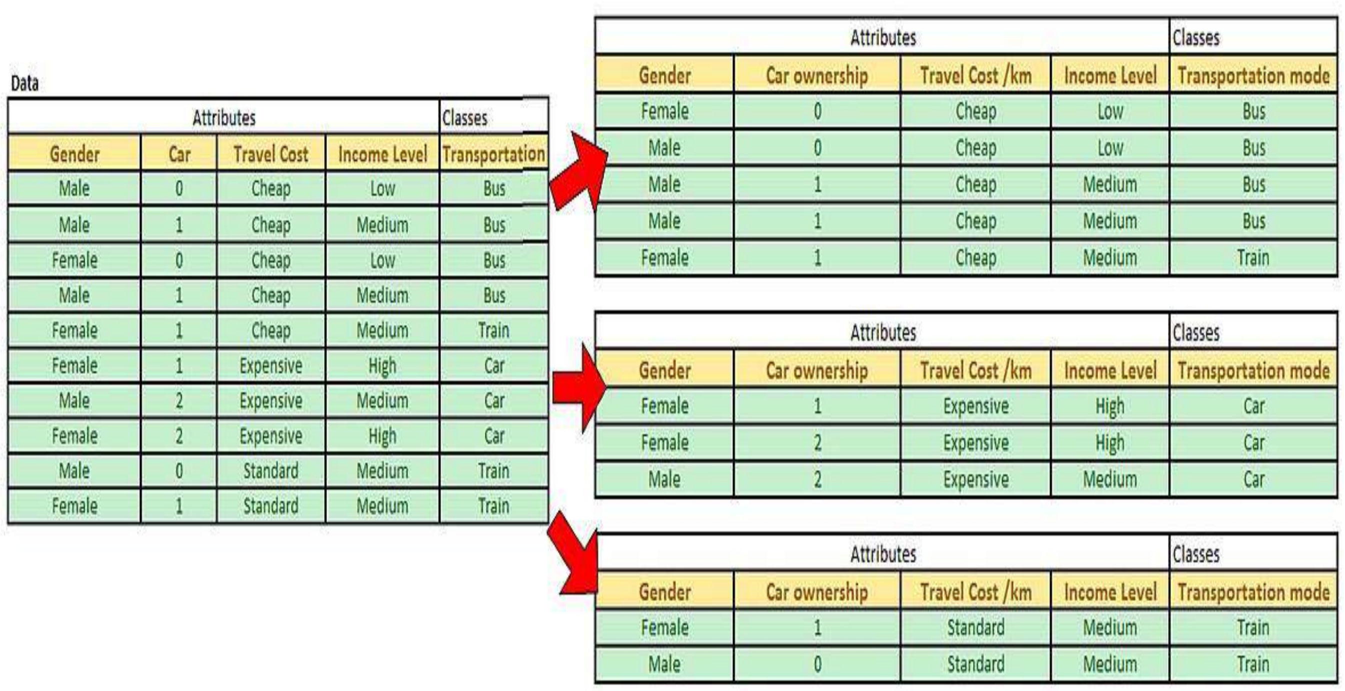
Results of first Iteration

Gain	Gender	Car ownership	Travel Cost/KM	Income Level
Entropy	0.125	0.534	**1.210**	0.695
Gini index	0.060	0.207	**0.500**	0.293
Classification error	0.100	0.200	**0.500**	0.300

Once you get the information gain for all attributes, then we find the optimum attribute that produce the maximum information gain (i* = argmax {information gain of attribute i}). In our case, travel cost per km produces the maximum information gain. We put this optimum attribute into the node of our decision tree. As it is the first node, then it is the root node of the decision tree. Our decision tree now consists of a single root node.



Once we obtain the optimum attribute, we can split the data table according to that optimum attribute. In our example, we split the data table based on the value of travel cost per km.



Data

Attributes				Classes
Gender	Car	Travel Cost	Income Level	Transportation
Male	0	Cheap	Low	Bus
Male	1	Cheap	Medium	Bus
Female	0	Cheap	Low	Bus
Male	1	Cheap	Medium	Bus
Female	1	Cheap	Medium	Train
Female	1	Expensive	High	Car
Male	2	Expensive	Medium	Car
Female	2	Expensive	High	Car
Male	0	Standard	Medium	Train
Female	1	Standard	Medium	Train

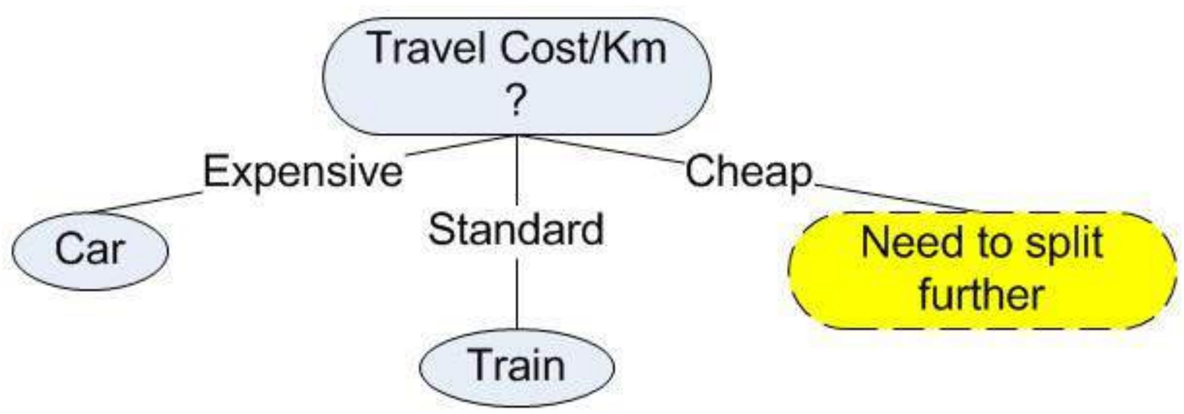
Attributes				Classes
Gender	Car ownership	Travel Cost /km	Income Level	Transportation mode
Female	0	Cheap	Low	Bus
Male	0	Cheap	Low	Bus
Male	1	Cheap	Medium	Bus
Male	1	Cheap	Medium	Bus
Female	1	Cheap	Medium	Train

Attributes				Classes
Gender	Car ownership	Travel Cost /km	Income Level	Transportation mode
Female	1	Expensive	High	Car
Female	2	Expensive	High	Car
Male	2	Expensive	Medium	Car

Attributes				Classes
Gender	Car ownership	Travel Cost /km	Income Level	Transportation mode
Female	1	Standard	Medium	Train
Male	0	Standard	Medium	Train

Using this information, we can now update our decision tree. We can add node Gender which has two values of male and female. The pure class is related to leaf node, thus Male gender has leaf node of Bus. For Female gender, we need to split further the attributes in the next iteration.

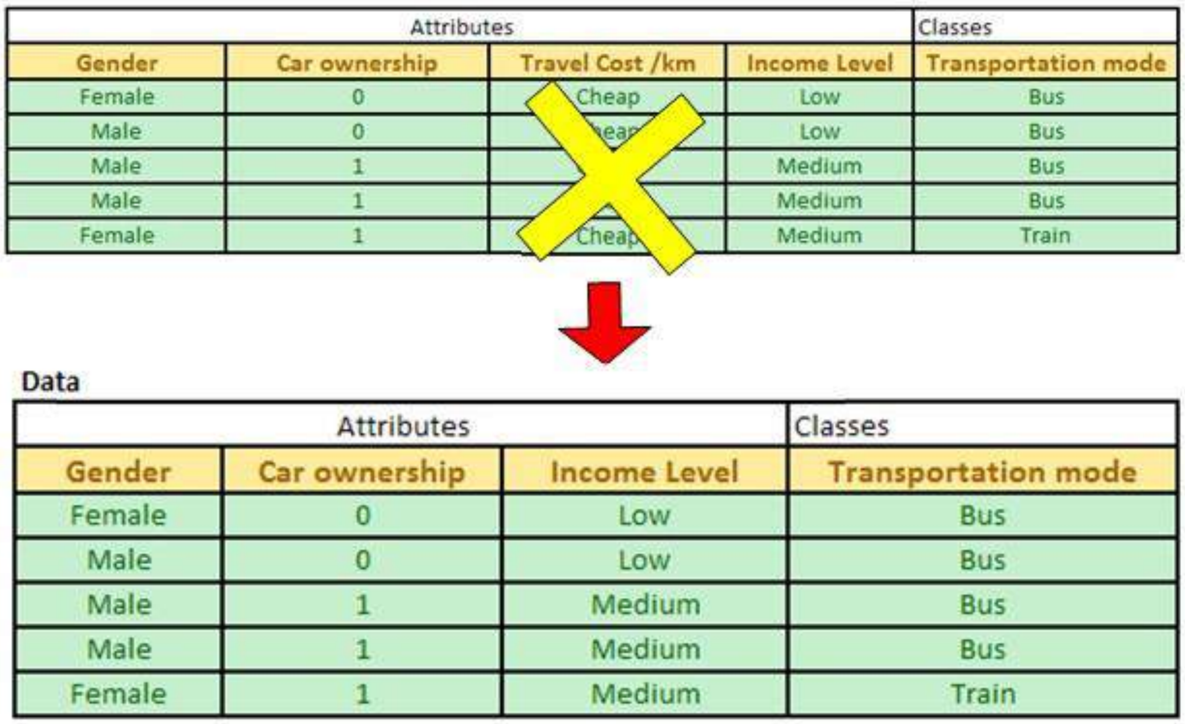
After the split of the data, we can see clearly that value of Expensive travel cost/km is associated only with pure class of Car while Standard travel cost/km is only related to pure class of Train. Pure class is always assigned into leaf node of a decision tree. We can use this information to update our decision tree in our first iteration into the following.



For Cheap travel cost/km, the classes are not pure, thus we need to split further.

Second Iteration

In the second iteration, we need to update our data table. Since Expensive and Standard Travel cost/km have been associated with pure class, we do not need these data any longer. For second iteration, our data table D is only come from the Cheap Travel cost/km. We remove attribute travel cost/km from the data because they are equal and redundant.

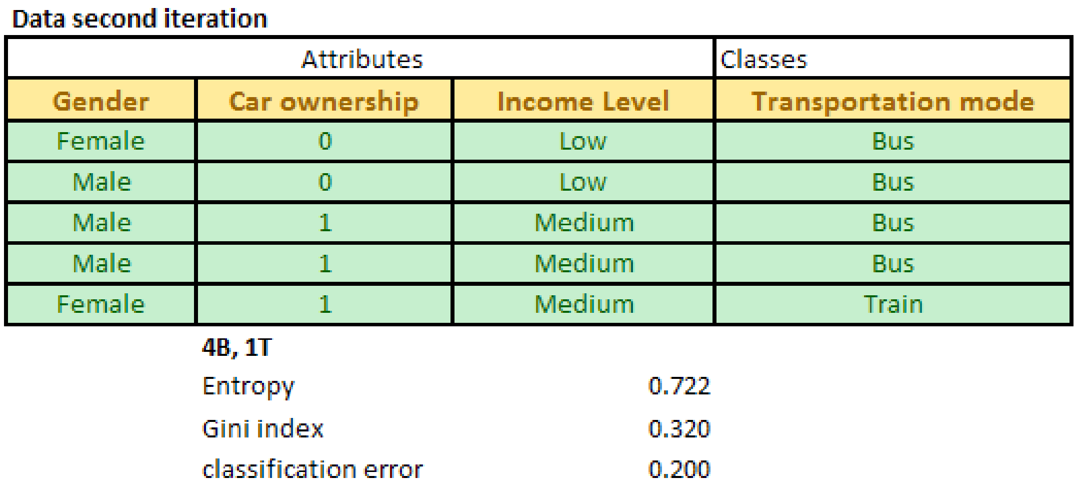


Attributes				Classes
Gender	Car ownership	Travel Cost /km	Income Level	Transportation mode
Female	0	Cheap	Low	Bus
Male	0	Cheap	Low	Bus
Male	1	[illegible]	Medium	Bus
Male	1	[illegible]	Medium	Bus
Female	1	Cheap	Medium	Train

Data

Attributes			Classes
Gender	Car ownership	Income Level	Transportation mode
Female	0	Low	Bus
Male	0	Low	Bus
Male	1	Medium	Bus
Male	1	Medium	Bus
Female	1	Medium	Train

Now we have only three attributes: Gender, car ownership and Income level. The degree of impurity of the data table D is shown in the picture below.



Data second iteration

Attributes			Classes
Gender	Car ownership	Income Level	Transportation mode
Female	0	Low	Bus
Male	0	Low	Bus
Male	1	Medium	Bus
Male	1	Medium	Bus
Female	1	Medium	Train

4B, 1T

Entropy	0.722
Gini index	0.320
classification error	0.200

Then, we repeat the procedure of computing degree of impurity and information gain for the three attributes. The results of computation are exhibited below.

Subsets of second iterations

Gender	Classes
Female	Bus
Female	Train

1B, 1T	
Entropy	1.000
Gini index	0.500
classification	0.500

Car ownership	Classes
0	Bus
0	Bus

2B	
Entropy	0.000
Gini index	0.000
classification error	0.000

Income Level	Classes
Low	Bus
Low	Bus

2B	
Entropy	0.000
Gini index	0.000
classification error	0.000

Gender	Classes
Male	Bus
Male	Bus
Male	Bus

3B	
Entropy	0.000
Gini index	0.000
classification	0.000

Car ownership	Classes
1	Bus
1	Bus
1	Train

2B, 1T	
Entropy	0.918
Gini index	0.444
classification error	0.333

Income Level	Classes
Medium	Bus
Medium	Bus
Medium	Train

2B, 1T	
Entropy	0.918
Gini index	0.444
classification error	0.333

Gain of Gender based on

Entropy	0.322
Gini index	0.120
classification	0.000

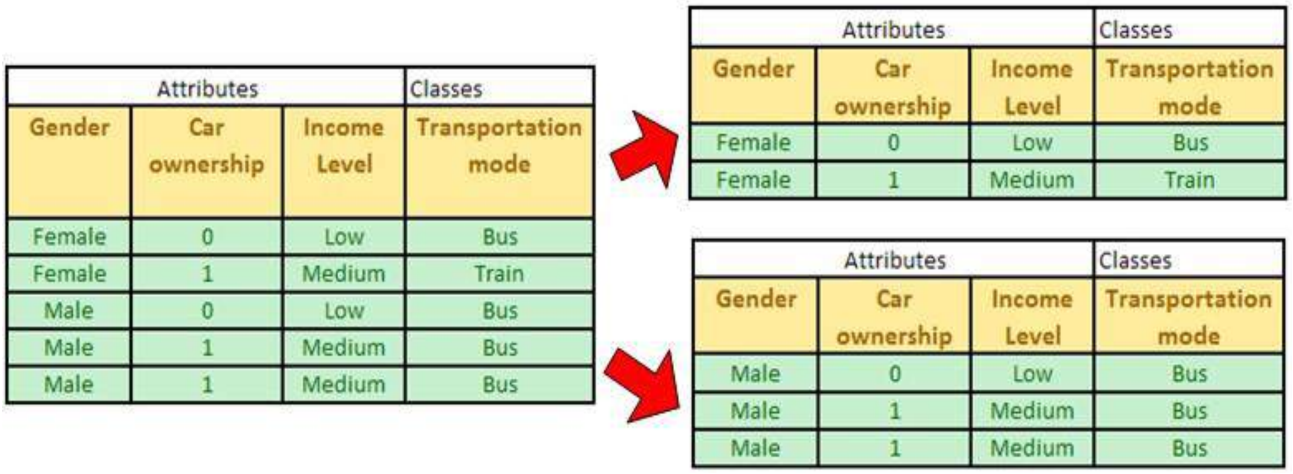
Gain of Car ownership based on

Entropy	0.171
Gini index	0.053
classification error	0.000

Gain of Income Level based on

Entropy	0.171
Gini index	0.053
classification error	0.000

The maximum gain is obtained for the optimum attribute Gender. Once we obtain the optimum attribute, the data table is split according to that optimum attribute. In our case, Male Gender is only associated with pure class Bus, while Female still need further split of attribute.

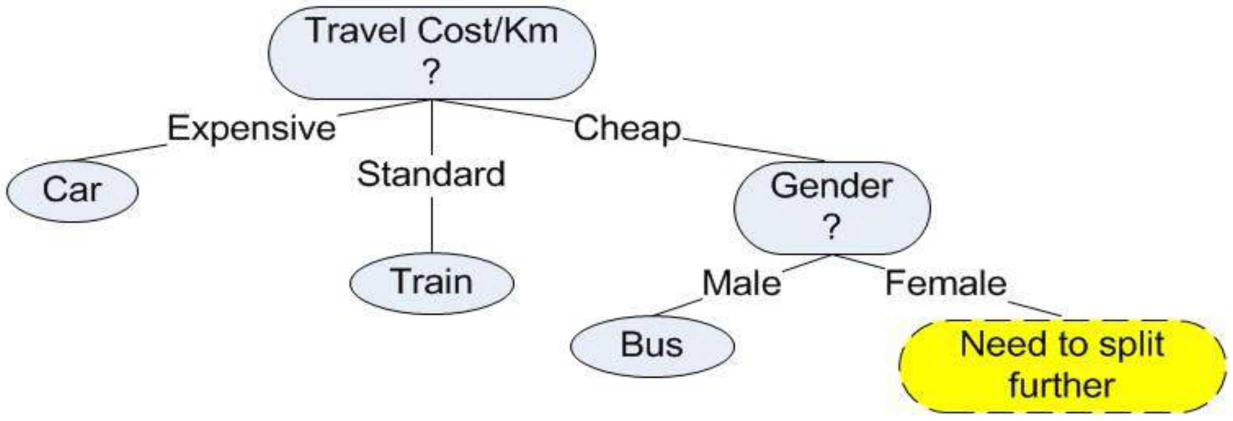


Attributes			Classes
Gender	Car ownership	Income Level	Transportation mode
Female	0	Low	Bus
Female	1	Medium	Train
Male	0	Low	Bus
Male	1	Medium	Bus
Male	1	Medium	Bus

Attributes			Classes
Gender	Car ownership	Income Level	Transportation mode
Female	0	Low	Bus
Female	1	Medium	Train

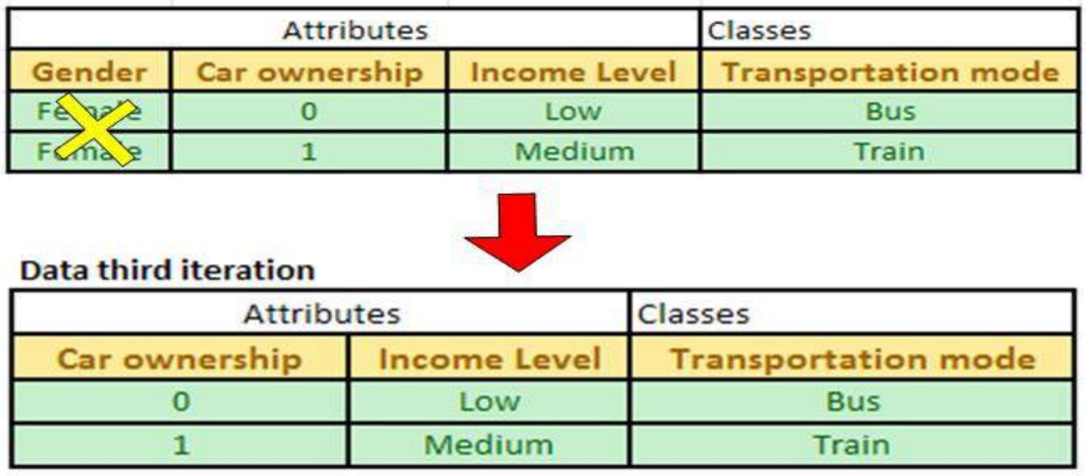
Attributes			Classes
Gender	Car ownership	Income Level	Transportation mode
Male	0	Low	Bus
Male	1	Medium	Bus
Male	1	Medium	Bus

Using this information, we can now update our decision tree. We can add node Gender which has two values of male and female. The pure class is related to leaf node, thus Male gender has leaf node of Bus. For Female gender, we need to split further the attributes in the next iteration.



Third iteration

Data table of the third iteration comes only from part of the data table of the second iteration with male gender removed (thus only female part). Since attribute Gender has been used in the decision tree, we can remove the attribute and focus only on the remaining two attributes: Car ownership and Income level.

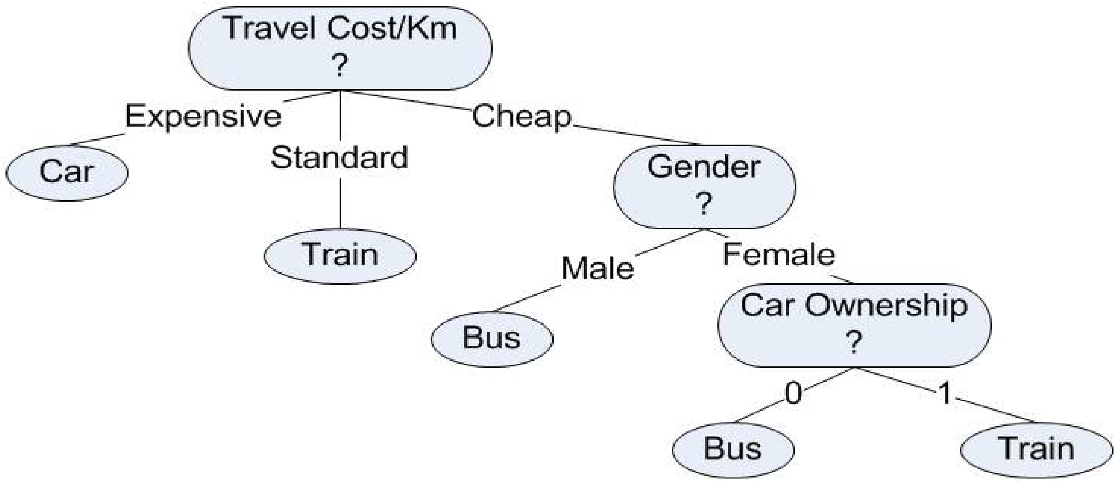


Attributes			Classes
Gender	Car ownership	Income Level	Transportation mode
Female	0	Low	Bus
Female	1	Medium	Train

Data third iteration

Attributes		Classes
Car ownership	Income Level	Transportation mode
0	Low	Bus
1	Medium	Train

If you observed the data table of the third iteration, it consists only two rows. Each row has distinct values. If we use attribute car ownership, we will get pure class for each of its value. Similarly, attribute income level will also give pure class for each value. Therefore, we can use either one of the two attributes. Suppose we select attribute car ownership, we can update our decision tree into the final version



Now we have grown the final full decision tree based on the data.

CHAPTER FOUR
ARTIFICIAL NEURAL NETWORKS

4.1 FUNDAMENTALS OF ARTIFICIAL NEURAL NETWORKS.

Artificial neural network (ANN) models have been studied for many years with the hope of achieving "Human-like performance", Different names were given to these models such as:

- Parallel distributed processing models
- Biological computers or Electronic Brains.
- Connectionist models
- Neural morphic system

After that, all these names settled on Artificial Neural Networks (ANN) and after it on neural networks (NN) only.

There are two basic different between computer and neural, these are:

1- These models are composed of many non-linear computational elements operating in parallel and arranged in patterns reminiscent of biological neural networks.

2- Computational Elements (or node s) are connected via weights that are typically adapted during use to improve performance just like human brain.

Computer $\longrightarrow$ logic Elements (1, 0)

Neural $\longrightarrow$ weighted performance

Areas of Neural Networks

The areas in which neural networks are currently being applied are:

1-Signal processing

2- Pattern Recognition.

3- Control problems

4- Medicine

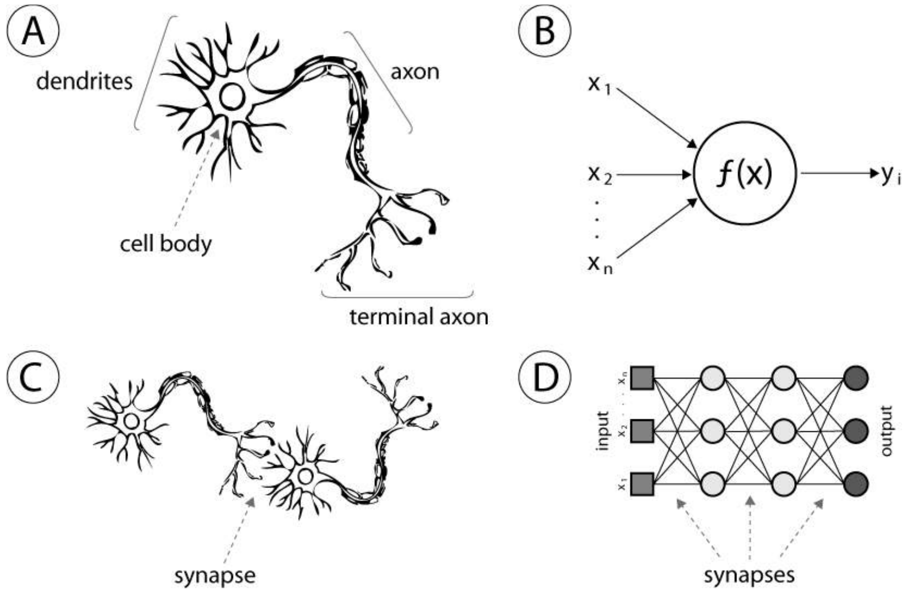
5- Speech production

6- Speech Recognition

7- Business

4.2 THEORY OF NEURAL NETWORKS (NN)

Human brain is the most complicated computing device known to a human being. The capability of thinking, remembering, and problem solving of the brain has inspired many scientists to model its operations. Neural network is an attempt to model the functionality of the brain in a simplified manner. These models attempt to achieve "good" performance via dense interconnections of simple computational elements. The term (ANN) and the connection of its models are typically used to distinguish them from biological network of neurons of living organism which can be represented systematically as shown in figure below



Biological Neural Network and Artificial Neural Network

Neclues is a simple processing unite which receives and combines signals from many other neurons through input paths called ***<u>dendrites</u>*** if the combined signal is strong enough, it activates the firing of neuron which produces an o/p signal. The path of the o/p signal is called the ***axon***, ***<u>synapse</u>*** is the junction between the (axon) of the neuron and the dendrites of the other neurons. The transmission across this junction is chemical in nature and the amount of signal transferred depends on the synaptic strength of the junction. This synoptic strength is modified when the brain is learning.

Weights (ANN) $\equiv$ synaptic strength (biological Networks)

4.3 TYPES OF LEARNING

In case a neural network is to be used for particle applications, a general procedure is to be taken, which in its various steps can be described as follows:-

1: A logical function to be represented is given. The input vector e_1 , e_2, e_3, …. , e_n are present, whom the output vectors a_1, a_2, a_3, …. , a_n assigned. These functions are to be represented by a network.

2: A topology is to be selected for the network.

3: The weights w_1, w_2, w_3, … are to be selected in such away that the network represents The given function (n) the selected topology. Learn procedures are to be used for determining the weights.

4: After the weights have been learned and the network becomes available, it can be used as after as desired.

<u>1- Supervised Learning:-</u>

The supervised is that, at every step the system is informed about the exact output vector. The weights are changed according to a formula (e.g. the delta-rule), if o/p is unequal to a. This method can be compared to learning under a teacher, who knows the contents to be learned and regulates them accordingly in the learning procedure.

<u>2- Unsupervised Learning:-</u>

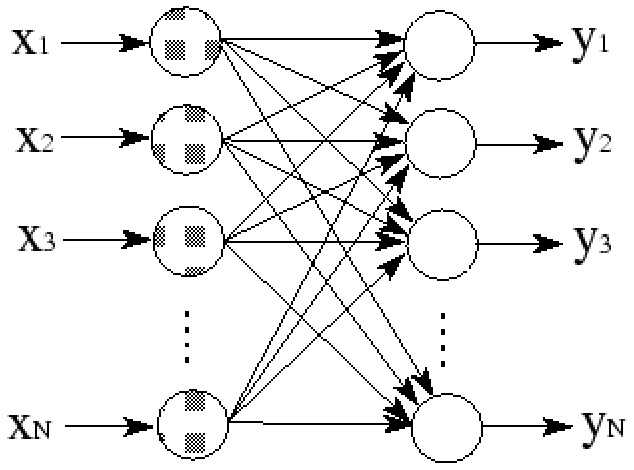
Here the correct final vector is not specified, but instead the weights are changed through random numbers. With the help of an evaluation function one can ascertain whether the output calculated with the changed weights is better than the previous one. In this case the changed weights are stored, else forgotten. This type of learning is also called reinforcement learning.

4.4 TYPICAL ARCHITECTURE OF ANN

Neural nets are often classified as single layer or multilayer. In determining the number of layers, the input units are not counted as a layer, because they perform no computation. Equivalently, the number of layers in the net can be defined to be the number of layers of weighted interconnects links between the slabs of neurons.

1- Single-Layer Net:-

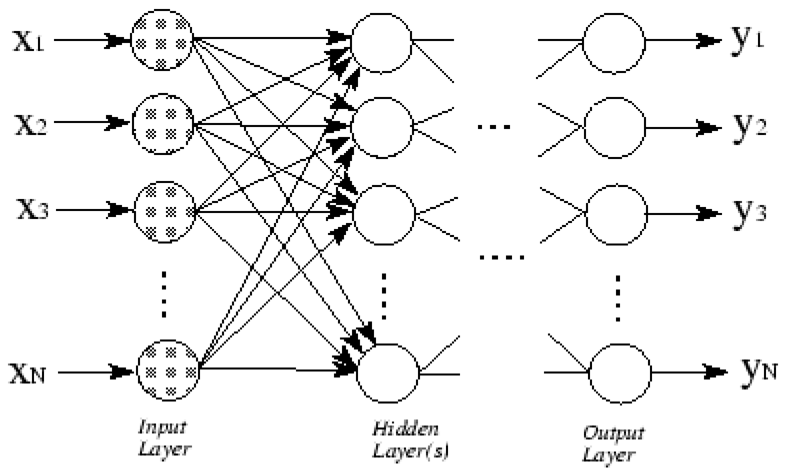
A single-layer net has one layer of connection weight. Often, the units can be distinguished as input units, which receive signals from the outside world, and output units, from which the response of the net can be read. In the typical single-layer net shown in figure bellow the input units are fully connected to output units but are not connected to other input units and the output units are not connected to other output units.



Single Layer Neural Network

2-Multilayer net

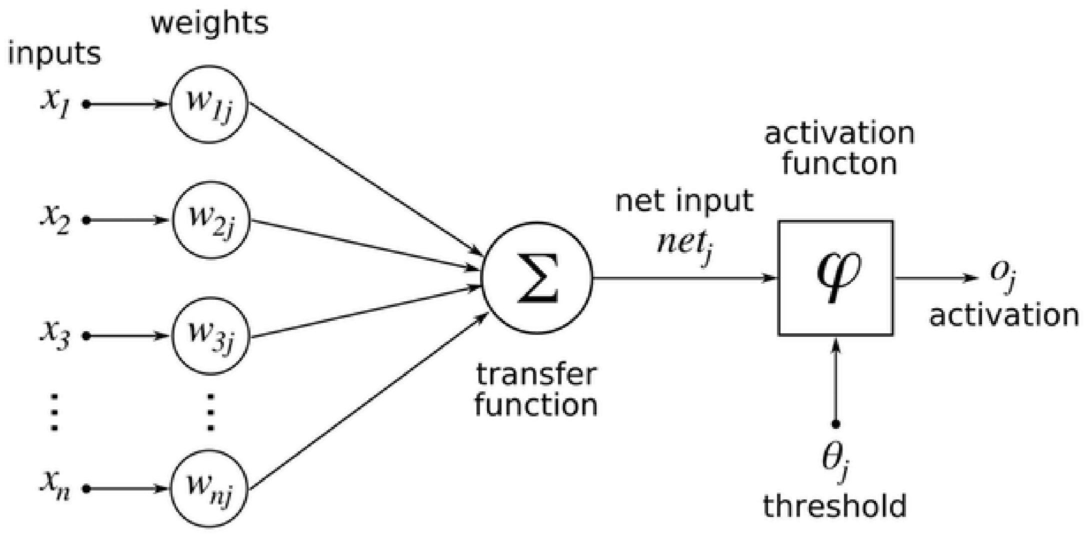
A Multilayer net is a net with one or more layers (or levels) of nodes which is called hidden units, between the input units and the output units. Typically, there is a layer of weights between two adjacent levels of units (input, hidden, or output). Multilayer nets can solve more complicated problems than can single-layer nets, but training may be more difficult. However, in some cases, training may be more successful because it is possible to solve a problem that a single-layer net can not be trained to perform correctly at all. The figure bellow shows the multilayer neural net.



Multiple Layer Neural Network

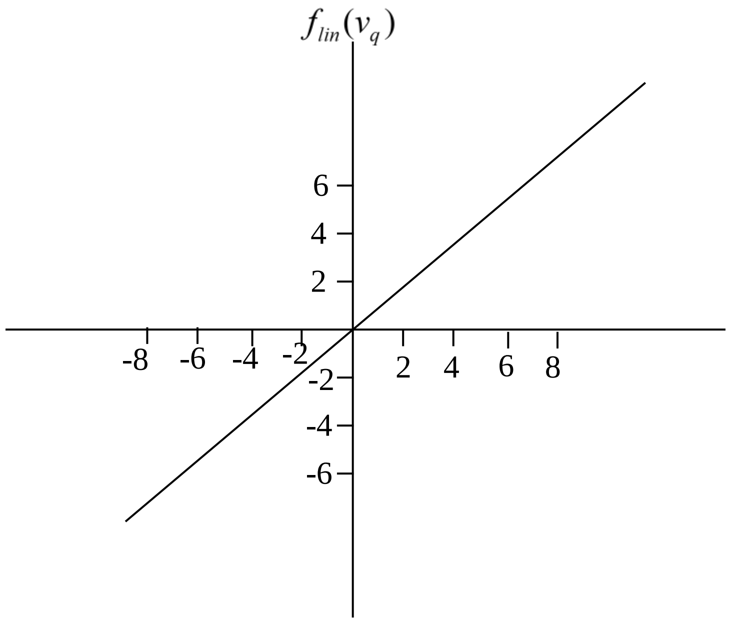
4.5 BASIC ACTIVATION FUNCTIONS

The activation function (Sometimes called a transfers function) shown in figure below can be a linear or nonlinear function. There are many different types of activation functions. Selection of one type over another depends on the particular problem that the neuron (or neural network) is to solve. The most common types of activation function are:-



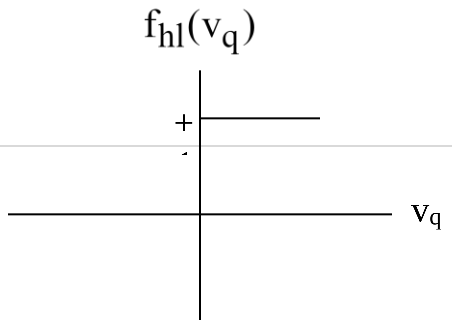
1- The first type is ***the linear*** (or ***identity***) function.

$y_q\, f_{lin}(v_q) = v_q$



2-The second type of activation function is a ***hard limiter***; this is a binary (or bipolar) function that hard-limits the input to the function to either a 0 or a 1 for the binary type, and a -1 or 1 for the bipolar type. The binary hard limiter is sometimes called the threshold function, and the bipolar hard limiter is referred to as the symmetric hard limiter.

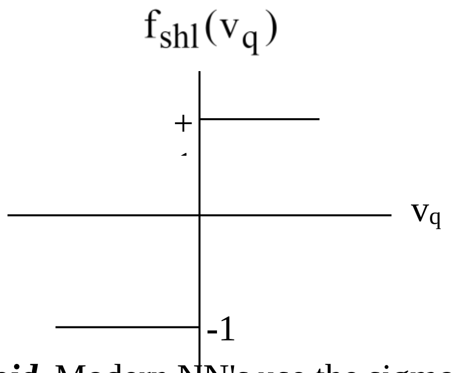
a- The o/p of the binary hard limiter:-



$$y_q = f_{hl}(v_q) = \begin{cases} 0 & \text{if} \quad v_q < 0 \\ 1 & \text{if} \quad v_q >= 0 \end{cases}$$

b-The o/p for the symmetric hard limiter (shl):-

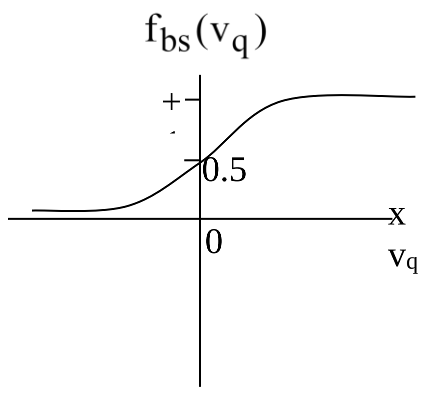
$$y_q = f_{shl}(v_q) = \begin{cases} -1 & \text{if} \quad net < 0 \\ 0 & \text{if} \quad v_q = 0 \\ 1 & \text{if} \quad v_q > 0 \end{cases}$$



3-The fourth type is ***sigmoid***. Modern NN's use the sigmoid nonlinearity which is also known as logistic, semi linear, or squashing function.

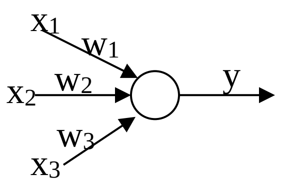
$$y_q = f_{bs}(v_q) = \frac{1}{1+e^{-\propto v_q}}$$

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



Ex.1 find y for the following neuron if :- $x_1=0.5$, $x_2=1$, $x_3=0.7$

$w_1=0$, $w_2=-0.3$, $w_3=0.6$



Sol

net = $= X_1W_1 + X_2W_2 + X_3W_3$

=0.5*0+1*-0.3+(-0.7*0.6)= -0.72

1- if f is linear

y = -0.72

2- if f is hard limiter (on-off)

y = -1

3-if f is sigmoid

$$y = \frac{1}{1+e^{-(-0.72)}} = 0.32$$

4.6 THE BIAS

قيمة ثابتة تضاف لتحسين التعلم

Some networks employ a bias unit as part of every layer except the output layer.

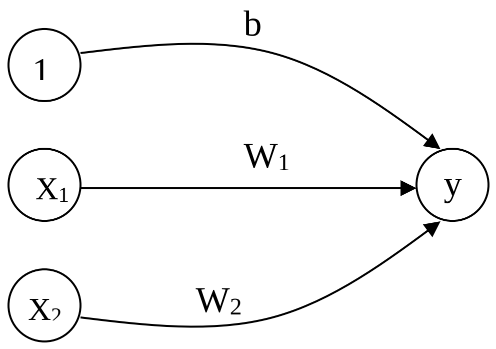
This units have a constant activation value of 1 or -1, it's weight might be adjusted during learning. The bias unit provides a constant term in the weighted sum which results in an improvement on the convergence properties of the network.

A bias acts exactly as a weight on a connection from a unit whose activation is always increasing the bias increases the net input to the unit. If a bias is included, the activation function is typically taken to be:

$$f(\text{net})\begin{cases} 1 & \text{if net} >= 0; \\ -1 & \text{if net} < 0; \end{cases}$$

Where

$$net = b + \sum_i X_i W_i$$



Input unit output unit

4.7 LEARNING ALGORITHMS

The NN's mimic the way that a child learns to identify shapes and colors NN algorithms are able to adapt continuously based on current results to improve performance. Adaptation or learning is an essential feature of NN's in order to handle the new "environments" that are continuously encountered. In contrast to NN's algorithms, traditional statistical techniques are not adoption but typically process all training data simultaneously before being used with new data. The performance of learning procedure depends on many factors such as:-

1- The choice of error function.

2- The net architecture.

3- Types of nodes and possible restrictions on the values of the weights.

4- An activation function.

The convergent of the net:-

Depends on the:-

1- Training set
2- The initial conditions
3- Learning algorithms.

4.8 PERCEPTRON

the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

The algorithm is as follows:

1. Initialize the weights and threshold to small random numbers.
2. Present a vector **x** to the neuron inputs and calculate the output.
3. Update the weights according to:

$$w(t+1)=w(t)+c(d-y)*x$$

where

- **d** is the desired output,
- **t** is the iteration number, and
- **c** is the learning rate or step size, where $0.0 < c <= 1.0$

4. Repeat steps 2 and 3 until:
   - the iteration error is less than a user-specified error threshold or
   - a predetermined number of iterations have been completed.

Illustration of the perceptron learning algorithm

Example: Consider the following training set:

x1	x2	x3	d
1	0	1	-1
0	-1	-1	1
-1	-0.5	-1	1

The learning rate is assumed to be 0.1. The initial weight vector is $w^0 = (1, -1, 0)$. Then the learning according to the perceptron learning rule progresses as follows.

Solution:

[Step 1] Input x at t=1, desired output d is -1:

$$\text{Net1}=(1,-1,0)\begin{pmatrix}1\\0\\1\end{pmatrix} = 1$$

Correction in this step is needed since

$y1 = -1 \neq \text{sign}(1)$.

We thus obtain the updated vector

$w^1 = w^0 + 0.1(-1 - 1)\, x^1$

Plugging in numerical values we obtain

$$w^1 = \begin{pmatrix}1\\-1\\0\end{pmatrix} - 0.2\begin{pmatrix}1\\0\\1\end{pmatrix} = \begin{pmatrix}0.8\\-1\\-0.2\end{pmatrix}$$

[Step 2] Input x at t=2 , desired output d is 1. For the present w^1 we compute the following:

$$\text{Net2}=(0.8,-1,-0.2)\begin{pmatrix}0\\-1\\-1\end{pmatrix} = 1.2$$

Correction is not performed in this step since 1 = sign(1.2), so we let $w^2 := w^1$

[Step 3] Input is x at t=3, desired output d is 1.

Net3=(0.8,-1,-0.2) $\begin{pmatrix} -1 \\ -0.5 \\ -1 \end{pmatrix}$ = -0.1

Correction in this step is needed since y3 = 1 6 ≠ sign(−0.1). We thus obtain the updated vector

$w^3 = w^2 + 0.1(1+1)\, x^3$

Plugging in numerical values we obtain

$W^3 = \begin{pmatrix} 0.8 \\ -1 \\ -0.2 \end{pmatrix} + 0.2 \begin{pmatrix} -1 \\ -0.5 \\ -1 \end{pmatrix} = \begin{pmatrix} 0.6 \\ -1.1 \\ -0.4 \end{pmatrix}$

[Step 4] Input x^1 , desired output d is -1:

Net4=(0.6,-1.1,-0.4) $\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$ = 0.2

Correction in this step is needed since y1 = −1 6 ≠ sign(0.2). We thus obtain the updated vector

$w^4 = w^3 + 0.1(-1 - 1)$ x 1

Plugging in numerical values we obtain

$W^4 = \begin{pmatrix} 0.6 \\ -1.1 \\ -0.4 \end{pmatrix} - 0.2 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.4 \\ -1.1 \\ -0.6 \end{pmatrix}$

Terminates this learning process at t=6, the correct weights (0.4, −1.1, −0.6).

4.9 BACK PROPAGATION

The determination of the error is a recursive process which start with the o/p units and the error is back propagated to the I/p units. Therefore the rule is called error Back propagation (EBP) or simply Back Propagation (BP). The weight is changed exactly in the same form of the standard DR

$$\Delta w_{ij} = \xi \ \delta_j \ x_i$$

$$\Rightarrow \quad w_{ij}(t+1) = w_{ij}(t) \ + \ \xi \ \delta_j \ x_i$$

There are two other equations that specify the error signal. If a unite is an o/p unit, the error signal is given by:-

$$\delta = (d_j - y_j) \ f_j(\text{net} \ \ j)$$

$$\text{Where} \quad \text{net} \quad j \ = \sum w_{ij} \ x_i + \theta$$

The GDR minimize the squares of the differences between the actual and the desired o/p values summed over the o/p unit and all pairs of I/p and o/p vectors. The rule minimize the overall error $E = \sum E_p$ by implementing a gradient descent in E: - where, $E_p = 1/2 \sum_j (d_j - y_j)^2$

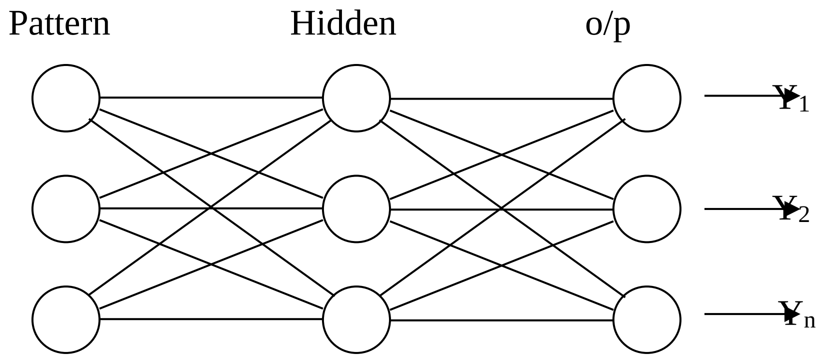
.

The BP consists of two phases:-

1- Forward Propagation:-

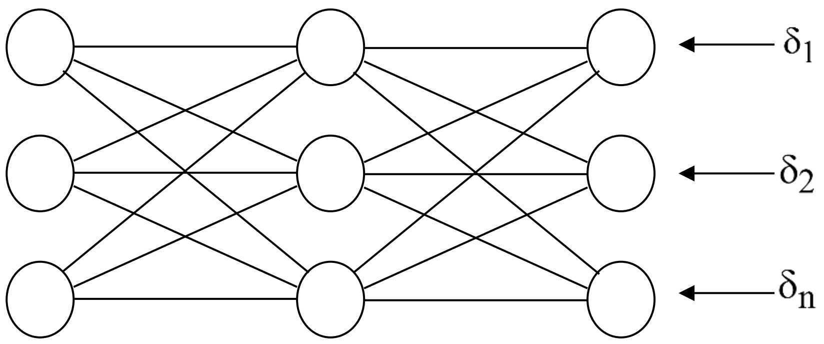
During the forward phase, the I/p is presented and propagated towards the o/p.

المرحلة الأولى



2- Backward Propagation:-

During the backward phase, the ***errors*** are formed at the o/p and propagated towards the I/p



3- Compute the error in the hidden layer.

If $y = f(x) = \dfrac{1}{1+e^{-x}}$

$f' = y(1-y)$

Equation is can rewrite as:-

$$\delta_j = y(1-y)(d_j - y_j)$$

The error signal for hidden units for which there is no specified target (desired o/p) is determined recursively in terms of the error signals of the units to which it directly connects and the weights of those connections:-

That is

$$\delta_j = f'(net_j)\sum_k \delta_k w_{ik}$$

<u>Or</u>

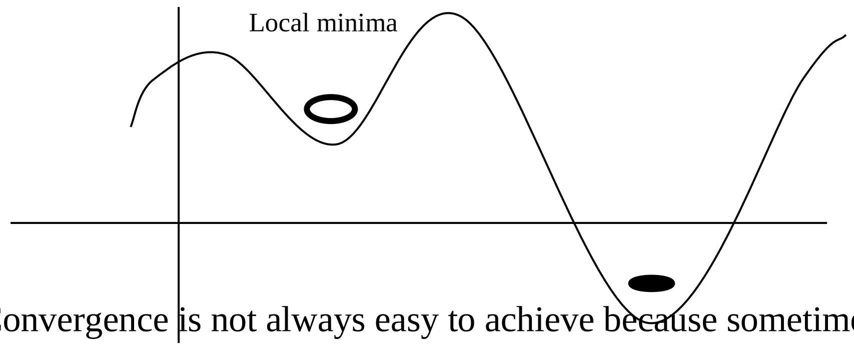
$$\delta_j = y_j(1-y_j)\sum_k \delta_k w_{ik}$$

B.P learning is implemented when hidden units are embedded between input and output units.

Convergence:-

A quantitative measure of the learning is the :Root Mean Square (RMS) error which is calculated to reflect the "degree" of learning.

Generally, an RMS bellow (0.1) indicates that the net has learned its training set. Note that the net does not provide a yes /no response that is "correct" or "incorrect" since the net get closer to the target value incrementally with each step. It is possible to define a cut off point when the nets o/p is said to match the target values.



- Convergence is not always easy to achieve because sometimes the net gets stuck in a "Local minima" and stops learning algorithm.
- Convergence can be represented intuitively in terms of walking about mountains.

Momentum term

The choice of the learning rate plays important role in the stability of the process. It is possible to choose a learning rate as large as possible without leading to oscillations. This offers the most rapid learning. One way to increase the learning rate without leading to oscillations is to modify the GDR to include momentum term.

This can be achieved by the following rule:-

$$W_{ij}(t+1) = W_{ij}(t) + \zeta\delta_j x_i + \propto (W_{ij}(t) - W_{ij}(t-1))$$

Where $\propto$ $(0 < \propto < 1)$ is a constant which determines the effect of the past weight changes on the current direction of movement in weight space.

A "global minima" unfortunately it is possible to encounter a local minima, avally that is not the lowest possible in the entire terrain. The net does not leave a local minima by the standard BP algorithm and special techniques should be used to get out of a local minima such as:-

1- Change the learning rate or the momentum term.

2- Change the no. of hidden units (10%).

3- Add small random value to the weights.

4- Start the learning again with different initial weights.

Back propagation training algorithm

Training a network by back propagation involves three stages:-

1-the feed forward of the input training pattern

2-the back propagation of the associated error

3-the adjustment of the weights

let n = number of input units in input layer,

let p = number of hidden units in hidden layer

let m = number of output units in output layer

let V_{ij} be the weights between i/p layer and the hidden layer,

let W_{ij} be the weights between hidden layer and the output layer,

we refer to the i/p units as X_i , i=1, 2, ….,n. and we refer to the hidden units as Z_j , j=1,….,p. and we refer to the o/p units as y_k, k=1,….., m.

δ_{1j} is the error in hidden layer,

δ_{2k} is the error in output layer,

ζ is the learning rate

$\propto$ is the momentum coefficient (learning coefficient, $0.0 < \propto < 1.0$,

y_k is the o/p of the net (o/p layer),

Z_j is the o/p of the hidden layer,

X_i is the o/p of the i/p layer.

η is the learning coefficient.

The algorithm is as following :-

Step 0 : initialize weights (set to small random value).

Step 1 : while stopping condition is false do steps 2-9

Step 2: for each training pair, do steps 3-8

Feed forward :-

Step 3:- Each i/p unit (X_i) receives i/p signal X_i & broad casts this signal to all units in the layer above (the hidden layer)

Step 4:- Each hidden unit (Z_j) sums its weighted i/p signals,

$$Z-\text{inj} = \text{Vaj} + \sum_{i=1}^{n} x_i v_{ij} \quad (\text{Vaj is abias})$$

and applies its activation function to compute its output signal (the activation function is the binary sigmoid function),

$$Z_j f(Z-\text{inj}) = 1 \ / \ (1+\exp-(Z-\text{inj}))$$

and sends this signal to all units in the layer above (the o/p layer).

Step 5:- Each output unit (Yk)sums its weighted i/p signals,

$$y-\text{ink} = \text{wok} + \sum_{j=1}^{p} \text{Zjwjk} \quad (\text{where wok is abias})$$

and applies its activation function to compute its output signal.

$$y_k = f(y-\text{ink}) = 1/(1+\exp-(y-\text{ink})$$

back propagation of error:-

step 6 : Each output unit (y_k , k= 1 tom) receive a target pattern corresponding to the input training pattern, computes its error information term and calculates its weights correction term used to update W_{jk} later,

$$\delta_{2k} = y_k(1-y_k)*(T_k - y_k),$$

where T_k is the target pattern & k=1 to m .

step 7 : Each hidden unit (Z_j, j= 1 top) computes its error information term and calculates its weight correction term used to update Vij later,

$$\delta_{1j} = Zj*(1-Zj)*\sum_{k=1}^{m}\delta 2kWjk$$

Update weights and bias :-

step 8: Each output unit (y_k, k =1 tom) updates its bias and weights:

$$Wjk(new) = \eta*\delta 2k*Zj+\propto*[Wjk(dd)],$$

j= 1 to p

Each hidden unit (Z_j, j= 1 to p) update its bias and weights:

$$Vij(new) = \eta*\delta 1j*Xi+\propto[vij(dd)],$$

I = 1 to n

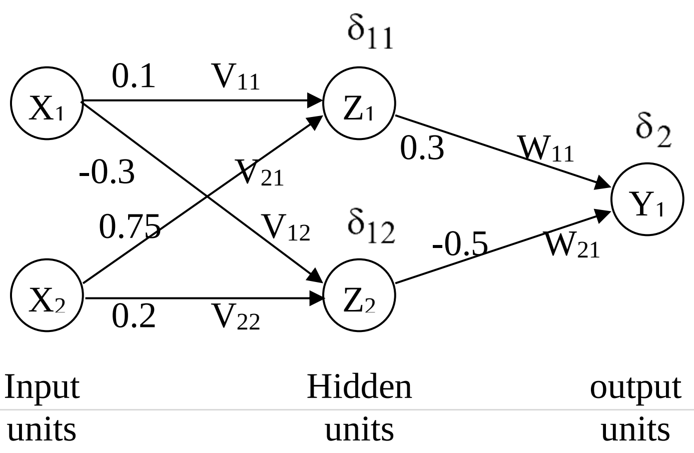
Step 9 : Test stopping condition.

Example: Suppose you have BP- ANN with 2-input , 2-hiddden , 1-output nodes with sigmoid function and the following matrices weight, trace with 1-iteration.

$$V = \begin{bmatrix} 0.1 & -0.3 \\ 0.75 & 0.2 \end{bmatrix} \qquad w = \begin{bmatrix} 0.3 & -0.5 \end{bmatrix}$$

Where $\propto = 0.9, \quad \eta = 0.45, \quad x = (1,0), \text{and } T_k = 1$

Solution:-



1-Forword phase :-

$$Z-in1 = X_1V_{11} + X_2V_{21} = 1*0.1 + 0*0.75 = 0.1$$

$$Z-in2 = X_1V_{12} + X_2V_{22} = 1*-0.3 + 0*0.2 = -0.3$$

$$Z_1 = f(Z-in1) = 1/(1+\exp-(Z-in1)) = 0.5$$

$$Z_2 = f(Z-in2) = 1/(1+\exp-(Z-in2)) = 0.426$$

$$y-in1 = Z_1W_{11} + Z_2W_{21}$$
$$= 0.5*0.3 + 0.426*(-0.5) = -0.063$$

$$y_1 = f(y-in1) = 1/(1+\exp-(y-in1) = 0.484$$

2-Backward phase :-

$$\delta_2k = yk(1-yk)*(Tk-yk)$$

$$\delta_{21} = 0.484(1-0.484)*(1-0.484)0.129$$

$$\delta_{1j} = Z_j*(1-Z_j)*\sum_{k=1}^{m}\delta_{2k}W_{jk}$$

$$\delta_{11} = Z_1(1-Z_1)*(\delta_{21}W_{11})$$
$$= 0.5(1-0.5)*(0.129*0.3) = 0.0097$$

$$\delta_{12} = Z_2(1-Z_2)*(\delta_{21}W_{21})$$
$$= 0.426(1-0.426)*(0.129*(-0.5)) = -0.015$$

3-Update weights:-

$$W_{jk}(new) = \eta*\delta_{2k}*Z_j + \propto *\left[W_{jk}(old)\right]$$

$$W_{11} = \eta*\delta_{21}*Z_1 + \propto *\left[W_{11}(old)\right]$$
$$= 0.45*0.129*0.5 + 0.9*0.3 = 0.299$$

$$W_{21} = \eta*\delta_{21}*Z_2 + \propto *\left[W_{21}(old)\right]$$
$$= 0.45*0.129*0.426 + 0.9*-0.5 = -0.4253$$

$$V_{ij}(new) = \eta*\delta_{1j}*X_i + \propto *\left[V_{ij}(old)\right]$$

$$V_{11} = \eta*\delta_{11}*X_1 + \propto *\left[V_{11}(old)\right]$$
$$= 0.45*0.0097*1 + 0.9*0.1 = 0.0944$$

$$V_{12} = \eta * \delta_{12} * X_1 + \propto * \left[V_{12}(\text{old})\right]$$
$$= 0.45 * 0.0158 * 1 + 0.9 * \text{-}0.3 = -0.2771$$
$$V_{21} = \eta * \delta_{11} * X_2 + \propto * \left[V_{21}(\text{old})\right]$$
$$= 0.45 * 0.0097 * 0 + 0.9 * 0.75 = 0.675$$
$$V_{22} = \eta * \delta_{12} * X_2 + \propto * \left[V_{22}(\text{old})\right]$$
$$= 0.45 * \text{-}0.0158 * 0 + 0.9 * 0.2 = 0.18$$

$$\therefore V = \begin{bmatrix} 0.0944 & -0.2771 \\ 0.675 & 0.18 \end{bmatrix} \qquad W = \begin{bmatrix} 0.299 & \text{-}0.4253 \end{bmatrix}$$

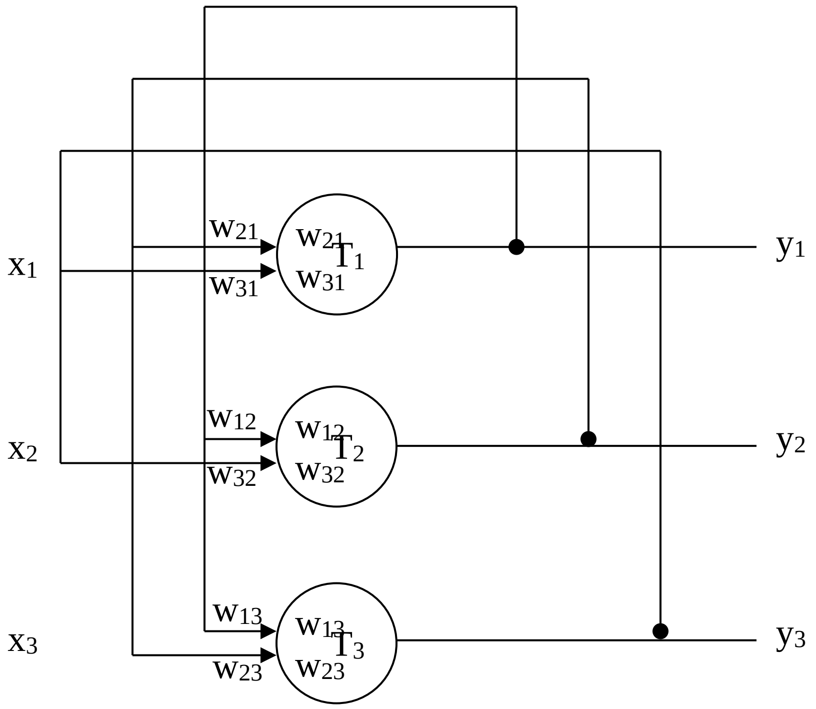
4.10 THE HOPFIELD NETWORK

The Nobel prize winner (in physics) John Hopfield has developed the discrete Hopfield net in (1982-1984). The net is a fully interconnected neural net, in the sense that each unit is connected to every other unit. The discrete Hopfield net has symmetric weights with no self-connections, i.e,

$$W_{ij} = W_{ji}$$

$$\text{And} \quad W_{ii} = 0$$

In this NN, inputs of 0 or 1 are usually used, but the weights are initially calculated after converting the inputs to -1 or +1 respectively.



"The Hopfield network"

The outputs of the Hopfield are connected to the inputs as shown in Figure, Thus feedback has been introduced into the network. The present output pattern is no longer solely dependent on the present inputs, but is also dependent on the previous outputs. Therefore the network can be said to have some sort of memory, also the Hopfield network has only one layer of neurons. The response of an individual neuron in the network is given by :-

$$y_j = 1 \quad \text{if} \quad \sum_{i=1\ i\neq j}^{n} W_{ij} X_i > T_j$$

$$y_j = 0 \quad \text{if} \quad \sum_{i=1\ i\neq j}^{n} W_{ij} X_i < T_j$$

This means that for the jth neuron, the inputs from all other neurons are weighted and summed.

Note $i \neq j$, which means that the output of each neuron is connected to the input of every other neuron, but not to itself. The output is a hard-limiter which gives a 1 output if the weighted sum is greater than T_j and an output of 0 if the weighted sum is less than T_j. it will be assumed that the output does not change when the weighted sum is equal to T_j.

Thresholds also need to be calculated. This could be included in the matrix by assuming that there is an additional neuron, called neuron 0, which is permanently stuck at 1. All other neurons have input connections to this neuron's output with weight W01, W02, W03,…etc. this provides an offset which is added to the weighted sum. The relation ship between the offset and the threshold T_j is therefore:- $Tj = -W0j$

The output [y] is just the output of neuron 0 which is permanently stuck at 1, so the formula becomes:- $[W_0] = [X]^t [Y_0]$

For example, if the patterns $X_1 = [0011]$ and $X_2 = [0101]$ are to be stored, first convert them to

$$X_1 = [-1 \quad -1 \quad 1 \quad 1]$$

$$X_2 = [-1 \quad 1 \quad -1 \quad 1]$$

To find the threshold:-

1- The matrix $\begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$

2-The transpose of the matrix is $\begin{bmatrix} -1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$

3- y_0 is permanently stuck at +1 , so the offsets are calculated as follows

$$W_0 = \begin{bmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{bmatrix} \begin{bmatrix} +1 \\ +1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ +2 \end{bmatrix}$$

4-These weights could be converted to thresholds to give:-

$$\begin{aligned} T_1 &= 2 \\ T_2 &= 0 \\ T_3 &= 0 \\ T_4 &= -2 \end{aligned} \qquad Tj = -W0j$$

Example: Consider the following samples are stored in a net:-

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & +1 & -1 & -1 \\ +1 & +1 & -1 & -1 \\ -1 & -1 & +1 & +1 \end{bmatrix}$$

binary $\rightarrow$ convert $\rightarrow$ bipolar

The binary input is (1110). We want the net to know which of samples is the i/p near to?

Note :-

A binary Hopfield net can be used to determine whether an input vector is a “known” vector (i.e., one that was stored in the net) or “unknown” vector.

Solution:-1-use Hebb rule to find the weights matrix

$$W = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix}$$

$W_{ii}=0$ (diagonal)

$W_{ij}=W_{ji}$

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & W_{12} & W_{13} & W_{14} \\ 2 & W_{21} & 0 & W_{23} & W_{24} \\ 3 & W_{31} & W_{32} & 0 & W_{34} \\ 4 & W_{41} & W_{42} & W_{43} & 0 \end{array}$$

$$W_{12} = (-1*1) + (1*1) + (-1*-1) = 1$$
$$W_{13} = (-1*-1) + (1*-1) + (-1*1) = -1$$
$$W_{14} = (-1*-1) + (1*-1) + (-1*1) = -1$$

$$W_{21} = W_{12} = 1$$
$$W_{23} = (1*-1) + (1*-1) + (-1*1) = -3$$
$$W_{24} = (-1*-1) + (1*-1) + (-1*1) = -3$$

$$W_{31} = W_{32} = 1$$
$$W_{32} = W_{23} = -3$$
$$W_{34} = (-1*-1) + (-1*-1) + (1*1) = 3$$

$$W_{41} = W_{14} = -1$$
$$W_{42} = W_{24} = -3$$
$$W_{43} = W_{34} = 3$$

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

2-The i/p vector x = (1 1 1 0). For this vector, y= (1 1 1 0)

Choose unit y_1 to update its activation

$$y-in1 = X_1 + \sum_{j}^{m} y_j w_{j1}$$

$$y-in1 = 1 + [(0*1) + (1*1) + (-1*1) + (-1*0)]$$
$$= 1 + 0 = 1$$
$$\therefore y = (1110)$$

Choose unit y_2 to up date its activation:-

$$y-in2 = x_2 + \sum_{j} y_j w_{j2}$$

$$= 1 + [(1*1) + (1*0) + (1*-3) + (0*-3)]$$
$$= 1 + (-2) = -1$$
$$y-in2 < 0 \qquad \therefore y_2 = 0$$
$$\therefore y = (1010)$$

Choose unit y_3 to update its activation:-

$$y-in3 = x_3 + \sum_{j} y_j w_{j3}$$

$$= 1 + [(1*-1) + (1*-3) + (1*0) + (0*3)]$$
$$= 1 + (-4) = -3$$
$$y-in3 < 0 \qquad \therefore y_3 = 0$$
$$\therefore y = (1000)$$

Choose unit y_4 to update its activation:-

$$y-in4 = x_4 + \sum_{j} y_j w_{j4}$$

= 0+ [(1*-1) + (1*-3) + (1*3) + (0*0)]

= 0+ (-1) = -1

y-in4 < 0 y4=0

y = (1000)

3- Test for convergence, false

$\therefore$ The input vector x = (1000), for this vector, Y= (1 0 0 0)

$y - in1 = 1$

$y - in2 = 1$

$y - in3 = -1 = 0$

$y - in4 = -1 = 0$

$\therefore y = (1100)$

$\therefore$ The input vector x= (1 1 0 0)

Y= (1 1 0 0)

$y - in1 = 2 = 1$

$y - in2 = 2 = 1$

$y - in3 = -4 = 0$

$y - in4 = -4 = 0$

$\therefore y = (1100)$

The input is near to the second sample.

True.

Stop.

4.11 KOHONEN NETWORK

Teuvo kohonen presented the self-organizing feature map in 1982. it is an unsupervised, competitive learning , clustering network in which only one neuron (or only one neuron in a group) is “on” at a time.

The self-organizing neural networks, also called (topology –preserving maps), assume a topological structure among the cluster units. This property is observed in the brain, but is not found in other artificial neural networks.

There are m cluster units arranged in a one –or two – dimensional array.

Cluster. وهي مجاميع من المعلومات كل مجموعة لها صفة معينة :

The weight vector for cluster units serves as an exemplar of the input patterns associated with that cluster. During the self organizing process, the cluster unit whose weight vector matches the input pattern most closely (typically, the square of the minimum Euclidean distance) is chosen as the winner. The winning unit and its neighboring units update their weights. The weight vectors of neighboring units are not, in general, close to the input pattern.

Kohonen Network Architecture

A kohonen network has two layers, an input layer to receive the input and an output layer. Neurons in the output layer are usually arranged into a regular two dimensional array. The architecture of the kohonen self-organizing map is shown bellow.

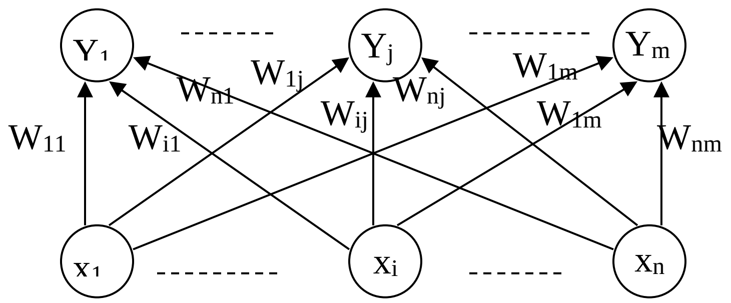


Figure (4.1)

(kohonen self-organizing map)

Kohonen Network Algorithm

Step 0 : initialize weights wij

Set topological neighborhood parameters

Set Learning rate parameters.

Step1:while stopping condition is false, do step 2-8

Step2: for each input vector x, do step 3-5

Step3: for each j, compute distance

$$D(j) = \sum_{i} (x_i - w_{ij})^2 \qquad \text{Euclidean distances}$$

Step4 : find index J such that D(J) is a minimum

Step5: for all units j within a specified neighborhood of J, and for all i:

$$Wij(new) = Wij(old) + \propto [Xi + Wij(old)]$$

Step6: update learning rate.

Step7: Reduce radius of topological neighborhood at specified times

Step8: Test stopping condition.

Example: A kohonen self-organizing map (SOM) to be cluster four vectors

$$\text{vector}1 = (1 \quad 1 \quad 0 \quad 0)$$
$$\text{vector}2 = (0 \quad 0 \quad 0 \quad 1)$$
$$\text{vector}3 = (1 \quad 0 \quad 0 \quad 0)$$
$$\text{vector}4 = (0 \quad 0 \quad 1 \quad 1)$$

The maximum no. of clusters to be formed is m=2 with learning rate $\propto = 0.6$

Solution: With only 2 clusters available, the neighborhood of nodJ is set so that only one cluster up dates its weight at each step

Initial weight matrix:

$$\begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}$$

1- for the first vector $\begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ (1 & 1 & 0 & 0) \end{array}$

$$D(i) = (1-0.2)^2 + (1-0.6)^2 + (0-0.5)^2 + (0-0.9)^2 = 1.86$$

$$D(2) = (1-0.8)^2 + (1-0.4)^2 + (0-0.7)^2 + (0-0.3)^2 = 0.98 \text{ (Minimum)}$$

$\therefore J = 2$ (The input vector) is closest to output node 2)

$\therefore$ The weight on the winning unit is update:-

$$W_{21}(\text{new}) = W_{12}(\text{old}) + 0.6(x_i - W_{12}(\text{old}))$$
$$= 0.8 + 0.6(1 - 0.8) = 0.92$$

$$W_{22}(new) = 0.4 + 0.6(1 - 0.4)$$
$$= 0.4 + 0.36 \quad = 0.76$$

$$W_{23}(\text{new}) = 0.7 + 0.6(0 - 0.7)$$
$$= 0.28$$

$$W_{24}(\text{new}) = 0.3 + 0.6(0 - 0.3)$$
$$= 0.12$$

This gives the weight matrix $\begin{bmatrix} 0.2 & 0.92 \\ 0.6 & 0.76 \\ 0.5 & 0.28 \\ 0.9 & 0.12 \end{bmatrix}$

2-for the second vector $\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}$

$$D(i) = (0-0.2)^2 + (0-0.6)^2 + (0-0.5)^2 + (1-0.9)^2 = 0.66 \text{ minimum}$$

$$D(2) = (0-0.92)^2 + (0-0.76)^2 + (0-0.28)^2 + (1-0.12)^2 = 2.2768$$

$\therefore J = 1$(The i/p vector is closest to o/p node 1)

After update the first column of the weight matrix:-

$$\begin{bmatrix} 0.08 & 0.92 \\ 0.24 & 0.76 \\ 0.20 & 0.28 \\ 0.96 & 0.12 \end{bmatrix}$$

3- for the third vector (1 0 0 0)

$$D(i) = (-0.08)^2 + (0-0.24)^2 + (0-0.20)^2 + (0-0.96)^2 = 1.856$$

$$D(2) = (1-0.92)^2 + (0-0.76)^2 + (0-0.28)^2 + (1-0.12)^2$$
$$= 2.2768 \quad \text{minimum}$$

$\therefore J = 2$ (The i/p vector is closest to o/p node (2))

After update the second column of the weight matrix:-

$$\begin{bmatrix} 0.08 & 0.968 \\ 0.24 & 0.304 \\ 0.20 & 0.112 \\ 0.96 & 0.0.48 \end{bmatrix}$$

4- for the fourth vector (0 0 1 1)

$$D(i) = (0-0.08)^2 + (0-0.24)^2 + (1-0.20)^2 + (1-0.96)^2 = 0.7056 \text{ minimum}$$

$$D(2) = (0-0.968)^2 + (0-0.304)^2 + (1-0.112)^2 + (1-0.048)^2 = 2.724$$

$\therefore$ J $=1$(the i/p vector is closest to o/p node 1)

After update the first column of the weight matrix :-

$$\begin{bmatrix} 0.032 & 0.968 \\ 0.096 & 0.304 \\ 0.680 & 0.112 \\ 0.984 & 0.0.48 \end{bmatrix}$$

$\therefore$ Reduce the learning rate

$\propto (t+1) * \propto (t) = 0.5 * (0.6) = 0.3$

$\therefore$ After one iteration the weight matrix will be:-

$$\begin{bmatrix} 0.032 & 0.970 \\ 0.096 & 0.300 \\ 0.680 & 0.110 \\ 0.984 & 0.048 \end{bmatrix}$$

5.1 Genetic Algorithm (GA)

A genetic algorithm is a search procedure modelled on the mechanics of natural selection rather than a simulated reasoning process. Domain Knowledge is embedded in the abstract representation of a candidate solution termed an organism. Organisms are grouped into sets called populations. Successive population are called generation. The aim of GA is search for goal.

A generational GA creates an initial generation G(0) , and for each generation ,G(t) , generates a new one ,G(t+1) . An abstract view of the algorithm is:-

```
Generate initial population, G(0);
Evaluate G(0);
t:=0;
Repeat
t:=t+ 1
Generate G(t) using G(t-1);
Evaluate G(t);
 Until solution is found.
```

5.1.1 Genetic Operators

The process of evolving a solution to a problem involves a number of operations that are loosely modeled on their counterparts from genetics .

Modeled after the processes of biological genetics , pairs of vectors in the population are allowed to " mate" with a probability that is proportional to their fitness . the mating procedure typically involves one or more genetic operators . The most commonly applied genetic operators are :-

1- Crossover.
2- Mutation.
3- Reproduction.

1- Crossover

Is the process where information from two parents is combined to form children. It takes two chromosomes and swaps all genes residing after a randomly selected crossover point to produce new chromosomes.

This operator does not add new genetic information to the population chromosomes but manipulates the genetic information already present in the mating pool (MP).

The hope is to obtain new more fit children It works as follows :-

1- Select two parents from the MP (The best two chromosomes) .
2- Find a position K between two genes randomly in the range (1, M-1)
   M = length of chromosome
3- Swap the genes after K between the two parents.

The output will be the both children <u>or</u> the more fit one.

2- Mutation

The basic idea of it is to add new genetic information to chromosomes. It is important when the chromosomes are similar and the GA may be yet stuck in Local maxima. A way to introduce new information is by changing the a of some genes. Mutation can be applied to :-

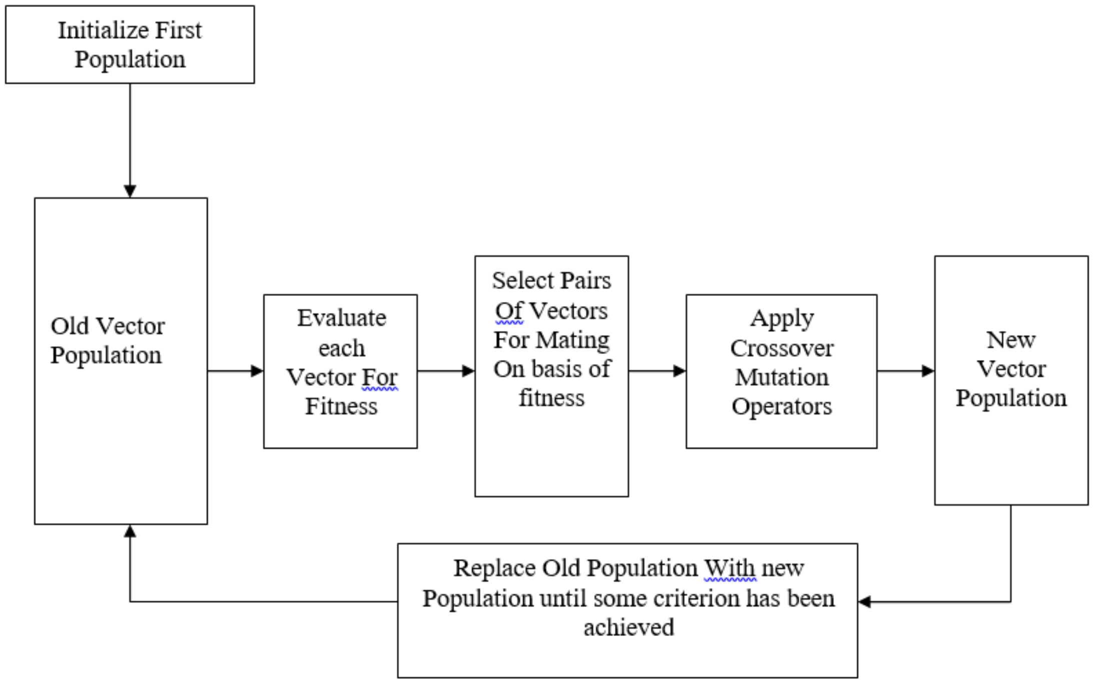
1- Chromosomes selected from the MP.
2- Chromosomes that have already subject to crossover.

The Figure bellow illustrates schematically the GA approach.

3- Reproduction

After manipulating the genetic information already present in the MP . by fitness function the reproduction operator add new genetic information to the population of the chromosomes by combining strong parents with strong children , the hope is to obtain new more fit children . Reproduction imitate to the natural selection.

This schematic diagram of a genetic algorithm shows the functions that are carried out in each generation. Over a number of such generation the initial population is evolved to the point where it can meet some criterion with respect the problem at hand.


Initialize First Population
Old Vector Population
Evaluate each Vector For Fitness
Select Pairs Of Vectors For Mating On basis of fitness
Apply Crossover Mutation Operators
New Vector Population
Replace Old Population With new Population until some criterion has been achieved

Genetic Algorithm approach

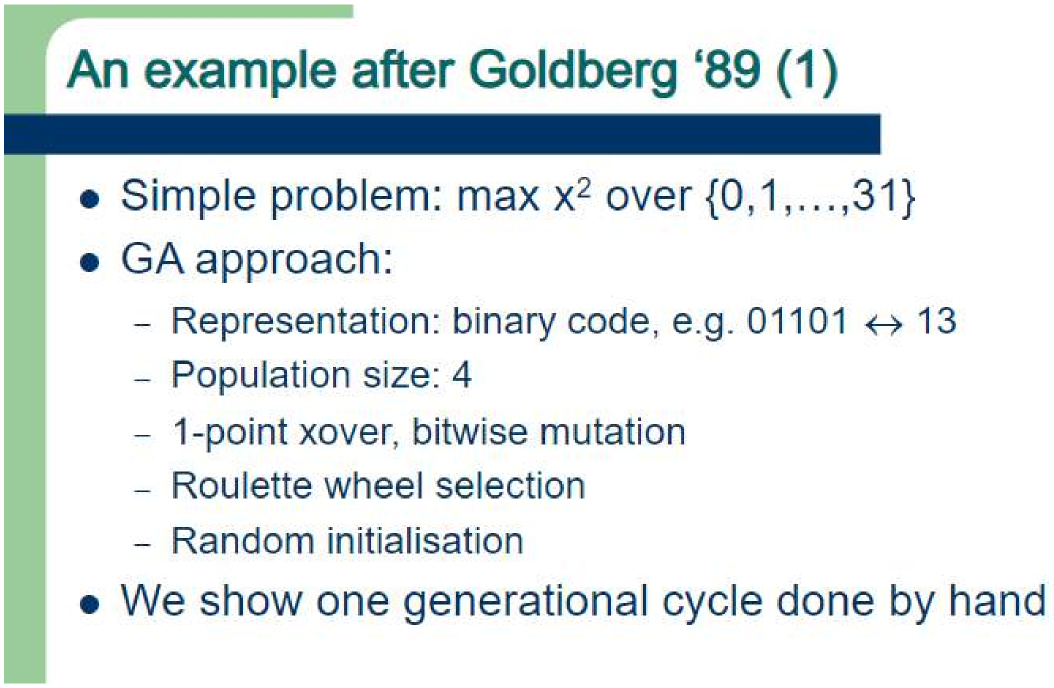
5.2 Genetic Programming (GP)

Genetic programming (GP) is a domain – independent problem – solving approach in which computer programs are evolved to solve, or approximately solve problems. Thus, it addresses one of the central goals of computer science namely automatic programming. The goal of automatic programming is to create, in an automated way, a computer program that enables a computer to solve a problem.

GP is based on reproduction and survival of the fittest genetic operations such as crossover and mutation. Genetic operation are used to create new offspring population of individual computer programs from the current population of programs .

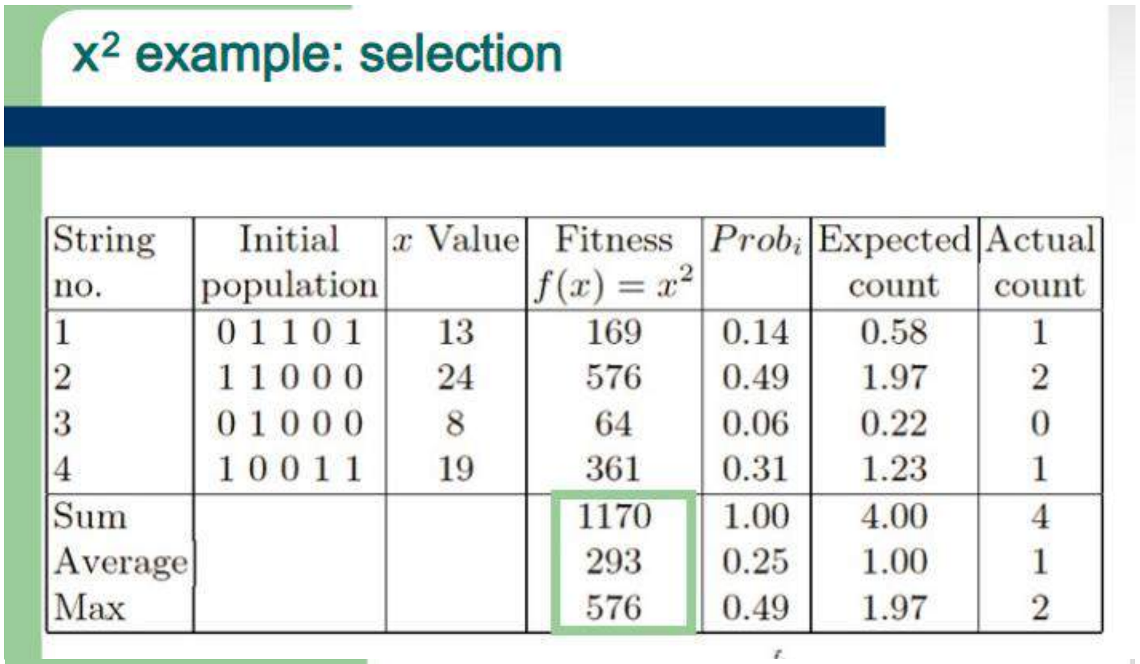
GP has several properties that make it more suitable than other paradigms (e.g. . best – first search , heuristic search , hill climbing etc .) , these properties are :-

1- GP produces a solution to a problem as a computer program. Thus GP is automatic programming.
2- Adaptation in GP is general hierarchical computer programs of dynamically varying size & shape.
3- It is probabilistic algorithm.
4- Another important feature of GP is role of pre processing of inputs and post processing of outputs .



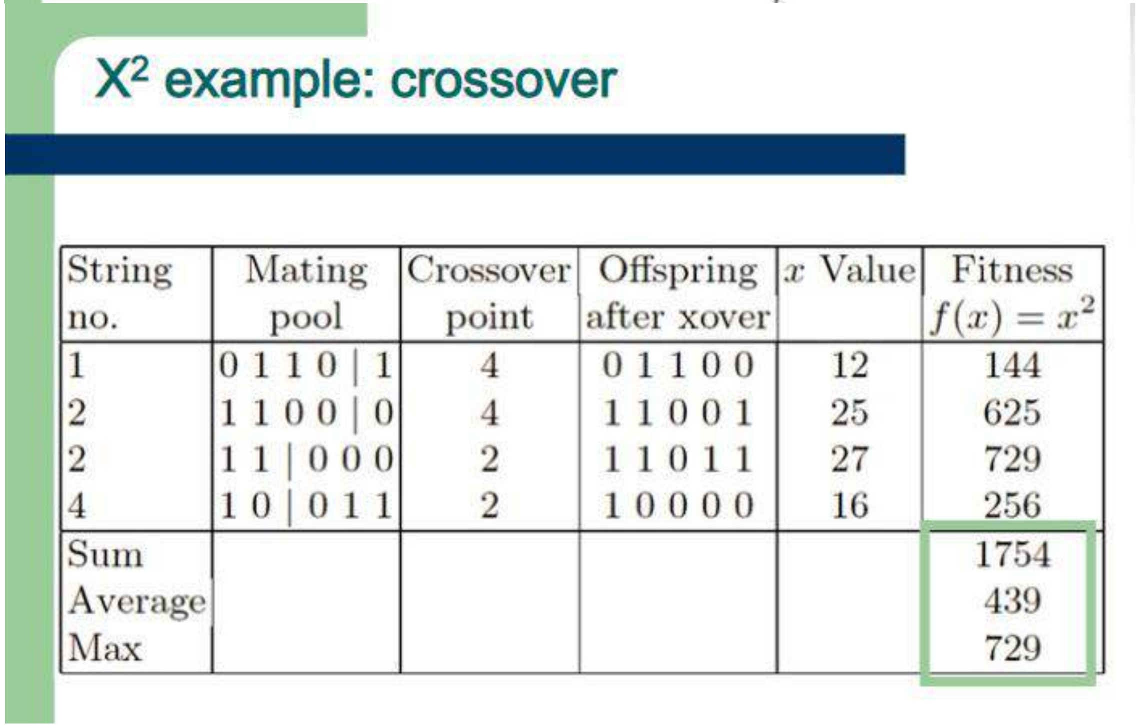
An example after Goldberg '89 (1)

- Simple problem: max x^2 over {0,1,…,31}
- GA approach:
  - Representation: binary code, e.g. 01101 $\leftrightarrow$ 13
  - Population size: 4
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- We show one generational cycle done by hand



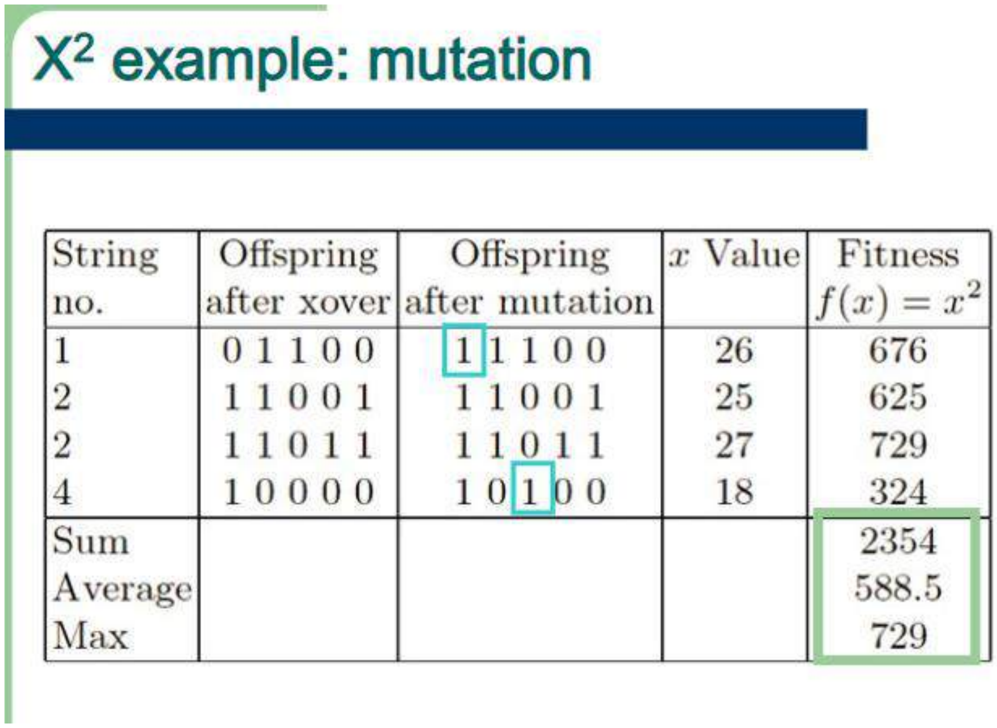
x^2 example: selection

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



X^2 example: crossover

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



X^2 example: mutation

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