

Machine Learning

تعلم الماكنة

الفصل الأول

First Course

أعداد

الأستاذ الدكتور

زياد طارق الطائي

Professor Dr.

Ziyad Tariq Al-Ta'i

المرحلة الرابعة

علوم الحاسوب

REFERENCES

- 1 - Machine Learning, Tom M. Mitchell, McGraw-Hill Science/Engineering/Math; ISBN: 0070428077, 1997.
- 2 - COS 511: Theoretical Machine Learning, http://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0204.pdf
- 3 - <http://people.revoledu.com/kardi/tutorial/DecisionTree/how-to-usedecision-tree.htm>
- 4- Introduction to Machine Learning, Alex Smola and S.V.N. Vishwanathan, Cambridge University Press 2008.

1.1 WHAT IS MACHINE LEARNING?

The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.

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.

Although a subarea of AI, machine learning also intersects broadly with other fields, especially statistics, but also mathematics, physics, theoretical computer science and more.

1.2 EXAMPLES OF MACHINE LEARNING PROBLEMS

There are many examples of machine learning problems. Much of this course will focus on classification problems in which the goal is to categorize objects into a fixed set of categories. Here are several examples:

- **Optical character recognition:** categorize images of handwritten characters by the letters represented.
- **face detection:** find faces in images (or indicate if a face is present).
- **Spam filtering:** identify email messages as spam or non-spam.
- **Topic spotting:** categorize news articles (say) as to whether they are about politics, sports, entertainment, etc.
- **Spoken language understanding:** within the context of a limited domain, determine the meaning of something uttered by a speaker to the extent that it can be classified into one of a fixed set of categories.
- **Medical diagnosis:** diagnose a patient as a sufferer or non-sufferer of some disease.
- **Customer segmentation:** predict, for instance, which customers will respond to a particular promotion.
- **Fraud detection:** identify credit card transactions (for instance) which may be fraudulent in nature.
- **Weather prediction:** predict, for instance, whether or not it will rain tomorrow (In this last case, we most likely would actually be more interested in estimating the probability of rain tomorrow). Although much of what we will talk about will be about classification problems, there are other important learning problems. In classification, we want to categorize objects into fixed categories. In regression, on the other hand, we are trying to predict a real value. For instance, we may wish to predict how much it will rain tomorrow. Or, we might want to predict how much a house will sell for.

A richer learning scenario is one in which the goal is actually to behave intelligently, or to make intelligent decisions. For instance, a robot needs to learn to navigate through its environment without colliding with anything. To use machine learning to make money on the stock market, we might treat investment as a classification problem (will the stock go up or down) or a regression problem (how much will the stock go up), or, dispensing with these intermediate goals, we might want the computer to learn directly how to decide to make investments so as to maximize wealth. A final example is game playing where the goal is for the computer to learn to play well through experience.

1.3 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.

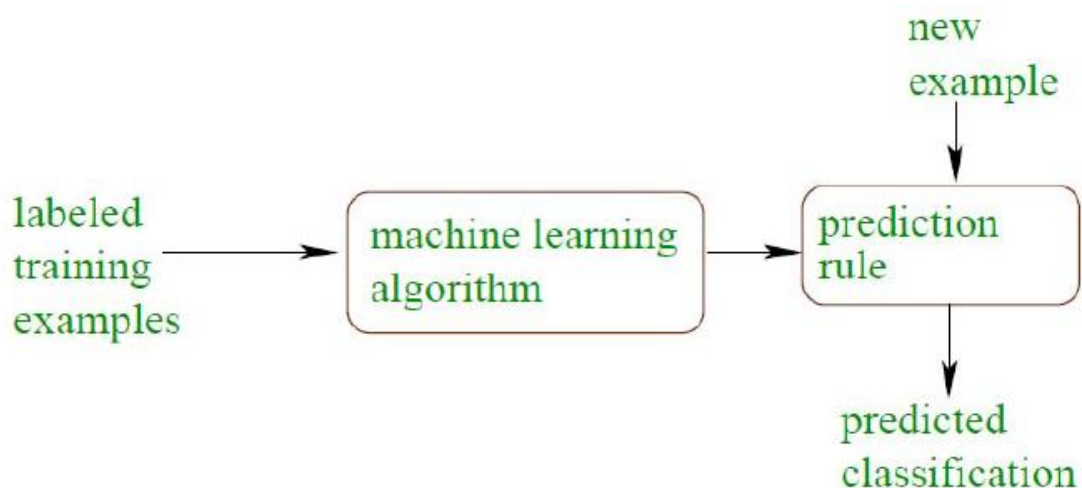


Figure 1: Diagram of a typical learning problem.

1.4 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?

1.5 A CONCEPT LEARNING TASK

Much of learning involves acquiring general concepts from specific training examples. People, for example, continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc. Each such concept can be viewed as describing some subset of objects or events defined over a larger set (e.g., the subset of animals that constitute birds). Alternatively, each concept can be thought of as a boolean-valued function defined over this larger set (e.g., a function defined over all animals, whose value is true for birds and false for other animals).

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.

The most general hypothesis—that every day is a positive example—is represented by

$$\langle ?, ?, ?, ?, ?, ? \rangle$$

and the most specific possible hypothesis—that *no* day is a positive example—is represented by

$$\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

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 is given in Table 2.2.

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

$$\langle ?, \text{Cold}, \text{High}, ?, ?, ? \rangle$$

| 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 |

TABLE 2.1

1.6 CONCEPT LEARNING AS SEARCH

The goal of the concept learning search is to find the hypothesis that best fits the training examples.

1.7 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

$$h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ? \rangle$$

$$h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ? \rangle$$

Now consider the sets of instances that are classified positive by h_1 and by h_2 . Because h_2 imposes fewer constraints on the instance, it classifies more instances as positive. In fact, any instance classified positive by h_1 will also be classified positive by h_2 . Therefore, we say that h_2 is more general than h_1 .

- $h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ? \rangle$
- $h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ? \rangle$
- Any instance classified positive by h_1 will also be classified positive by h_2
- h_2 is more general than h_1
- Definition: $h_j \geq_g h_k$ iff $(\forall x \in X) [(h_j = 1) \rightarrow (h_k = 1)]$
 - \geq_g *more general than or equal to*
 - $>_g$ *strictly more general than*
- Most general hypothesis: $\langle ?, ?, ?, ?, ?, ? \rangle$
- Most specific hypothesis: $\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

1.8 Find S Algorithm:

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

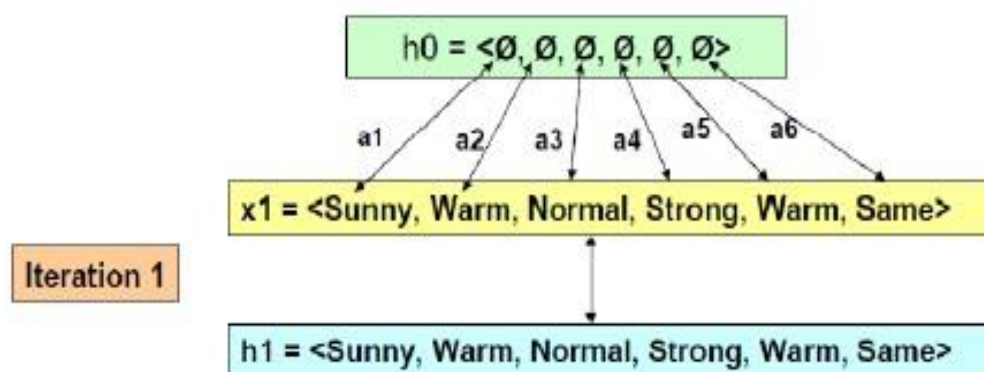
| Example | Sky | AirTemp | Humidity | Wind | Water | Forecast | EnjoySport |
|---------|-------|---------|----------|--------|-------|----------|------------|
| 1 | Sunny | Warm | Normal | Strong | Warm | Same | Yes |
| 2 | Sunny | Warm | High | Strong | Warm | Same | Yes |
| 3 | Rainy | Cold | High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

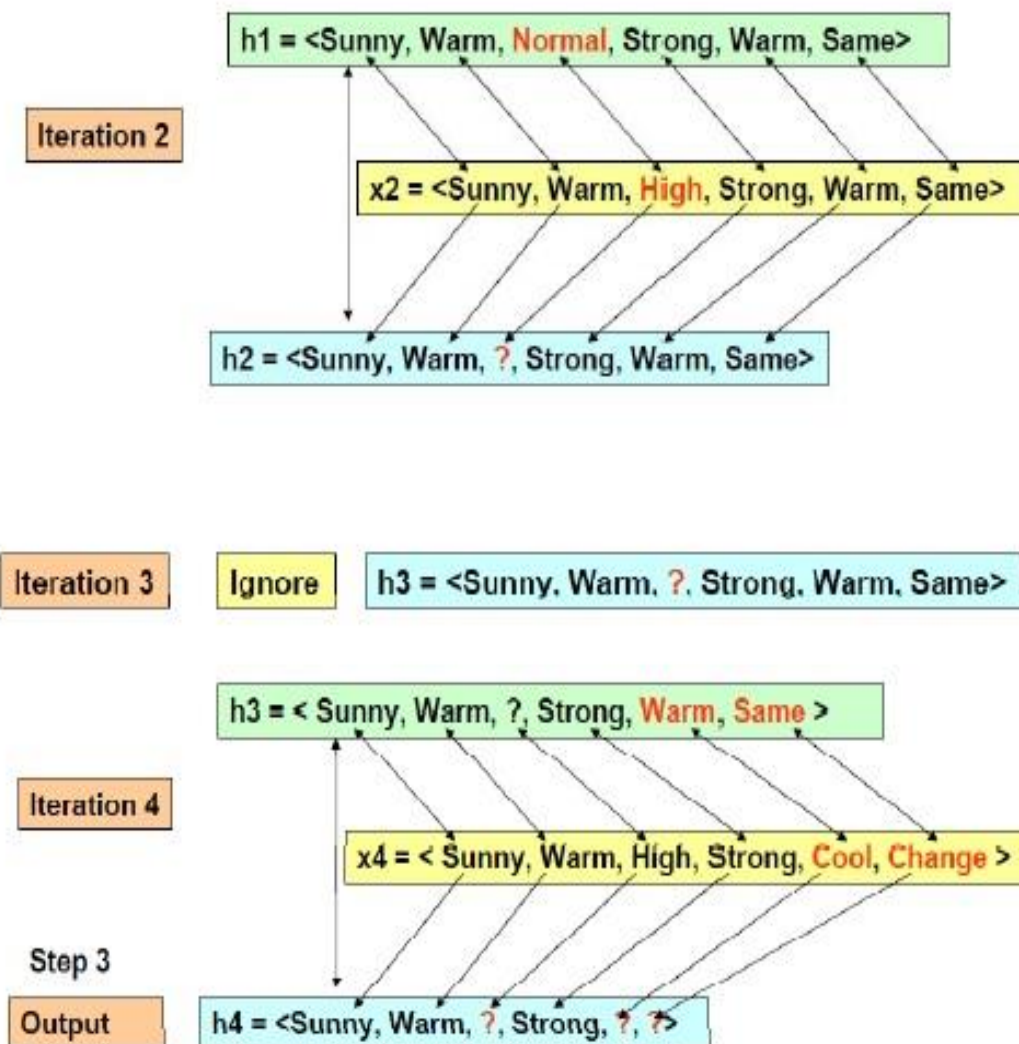
1. Initialize h to the most specific hypothesis in H

$h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

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





1.9 VERSION SPACE

The set of all valid hypotheses provided by an algorithm is called version space (VS) with respect to the hypothesis space H and the given example set D.

1.10 CANDIDATE-ELIMINATION ALGORITHM TO OBTAIN VERSION SPACE:

The Candidate-Elimination algorithm finds all describable hypotheses that are consistent with the observed training examples.

Candidate-Elimination Algorithm

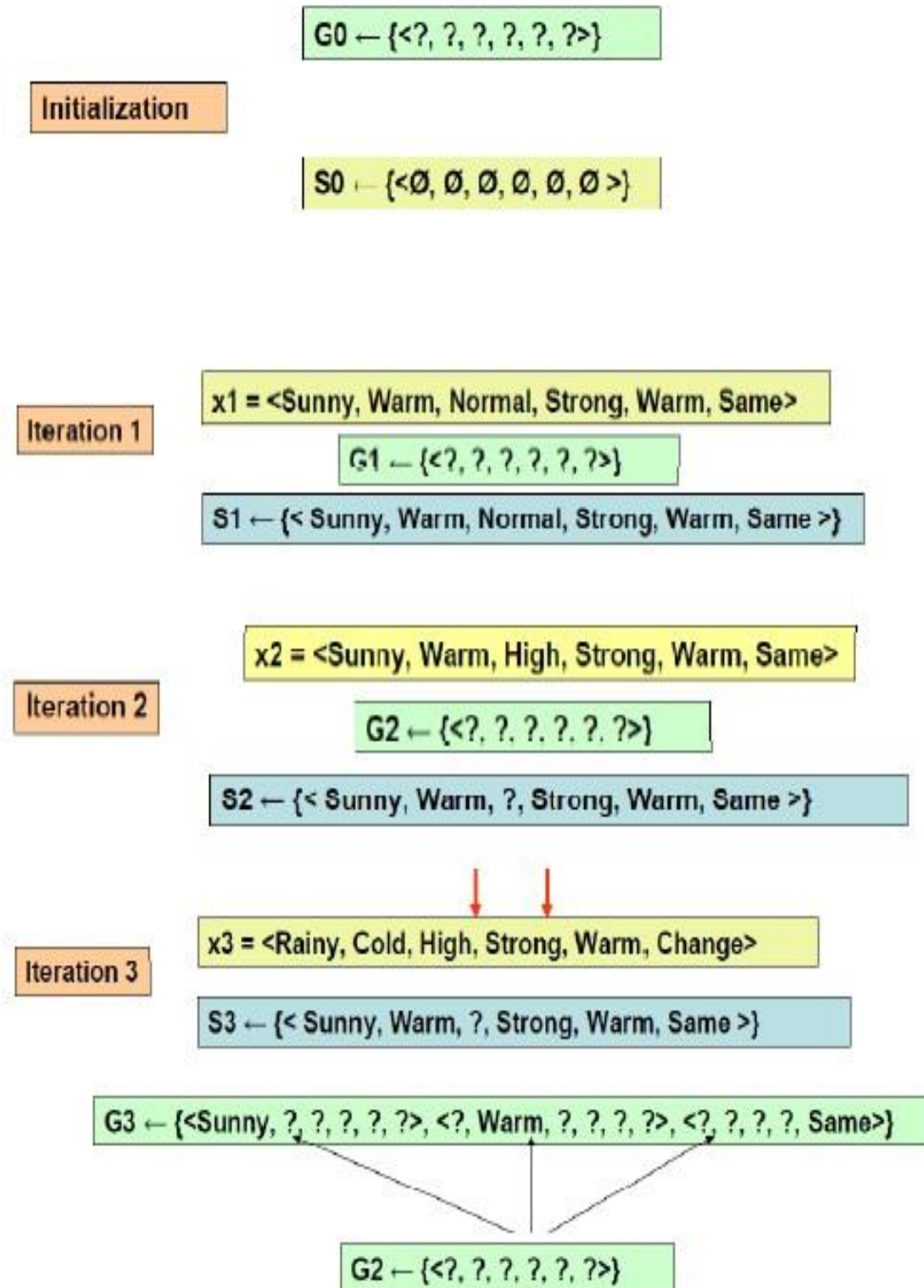
Initialize G to the set of maximally general hypotheses in H

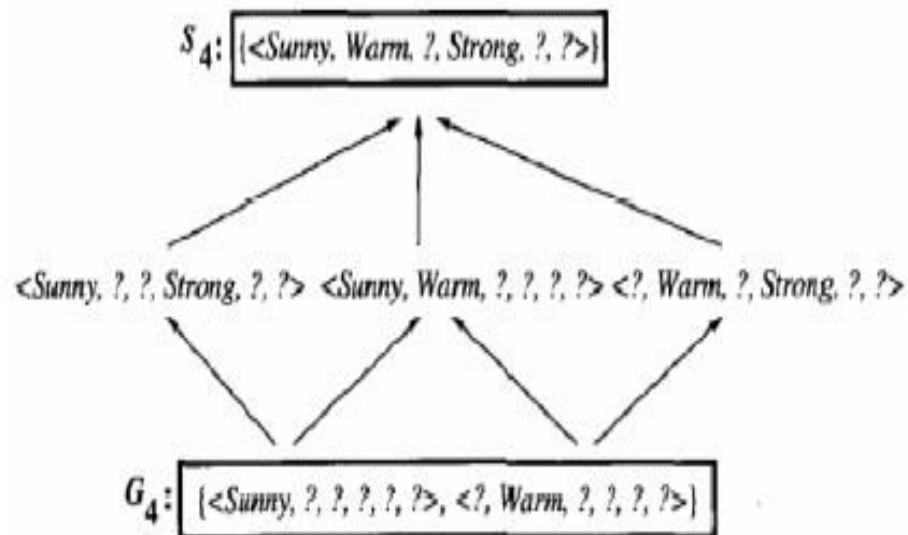
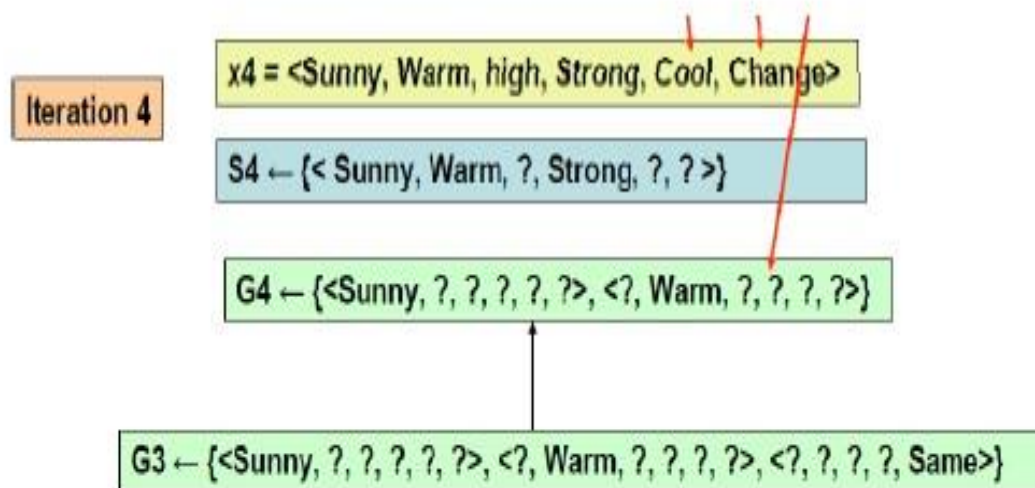
Initialize S to the set of maximally specific hypotheses in H

For each training example d , do

- If d is a positive example
 - Remove from G any hypothesis inconsistent with d
 - For each hypothesis s in S that is not consistent with d
 - Remove s from S
 - Add to S all minimal generalizations h of s such that
 - h is consistent with d , and some member of G is more general than h
 - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
 - Remove from S any hypothesis inconsistent with d
 - For each hypothesis g in G that is not consistent with d
 - Remove g from G
 - Add to G all minimal specializations h of g such that
 - h is consistent with d , and some member of S is more specific than h
 - Remove from G any hypothesis that is less general than another hypothesis in G

EXAMPLE: BY USING THE FORECAST TABLE





1.10 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.

1.10.1 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.

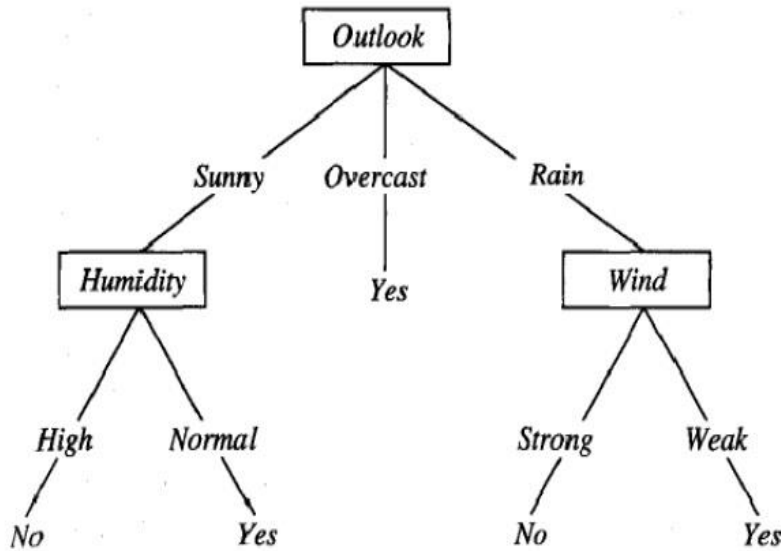


FIGURE 3.1

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

(Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong)

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*).

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*.

1.10.2 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 and its successor C4.5, 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.

1. 10.3 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.

1.10.4 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

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

ID3(*Examples*, *Target_attribute*, *Attributes*)

Examples are the training examples. *Target_attribute* is the attribute whose value is to be predicted by the tree. *Attributes* is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given *Examples*.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = -
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
 - $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*
 - The decision attribute for *Root* $\leftarrow A$
 - For each possible value, v_i , of A ,
 - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
 - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
 - If $Examples_{v_i}$ is empty
 - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
 - Else below this new branch add the subtree
 $ID3(Examples_{v_i}, Target_attribute, Attributes - \{A\})$
- End
- Return *Root*

* The best attribute is the one with highest *information gain*, as defined in Equation (3.4).

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

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

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

1.10.5 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

$$Values(Wind) = Weak, Strong$$

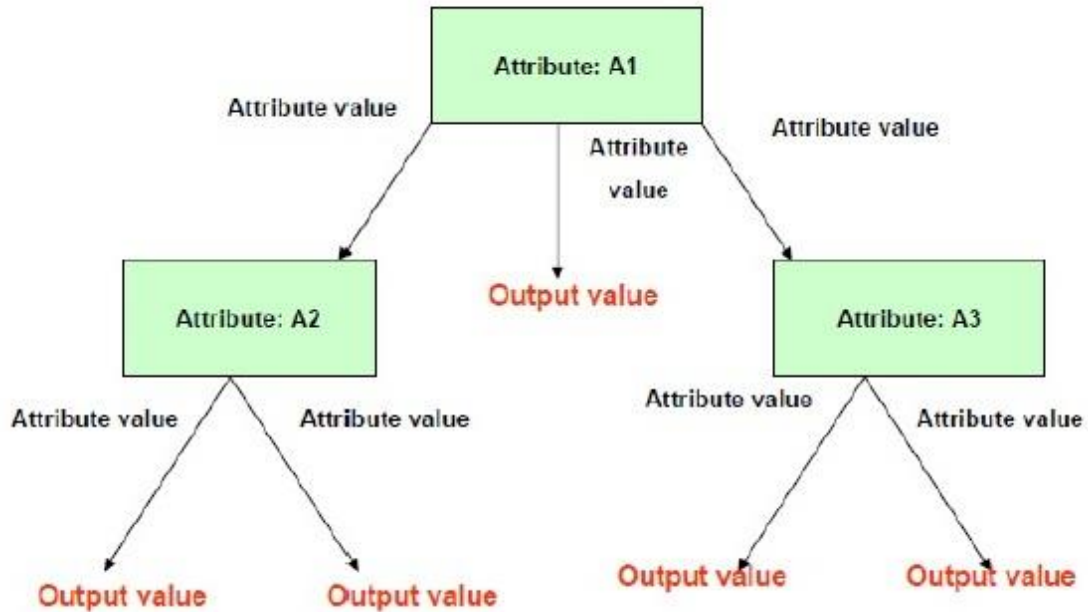
$$S = [9+, 5-]$$

$$S_{Weak} \leftarrow [6+, 2-]$$

$$S_{Strong} \leftarrow [3+, 3-]$$

$$\begin{aligned} Gain(S, Wind) &= Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v) \\ &= Entropy(S) - (8/14)Entropy(S_{Weak}) \\ &\quad - (6/14)Entropy(S_{Strong}) \\ &= 0.940 - (8/14)0.811 - (6/14)1.00 \\ &= 0.048 \end{aligned}$$

1.10.6 Building Decision Tree:



From table D and for each associated subset S_i , 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 S_i 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.

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

$$\text{Entropy} = -0.4 \log(0.4) - 0.3 \log(0.3) - 0.3 \log(0.3) = 1.571$$

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 $p=1/n$. In this case, maximum entropy is equal to $-n \cdot p \cdot \log p$. Notice that the value of entropy is larger than 1 if the number of classes is more than 2.

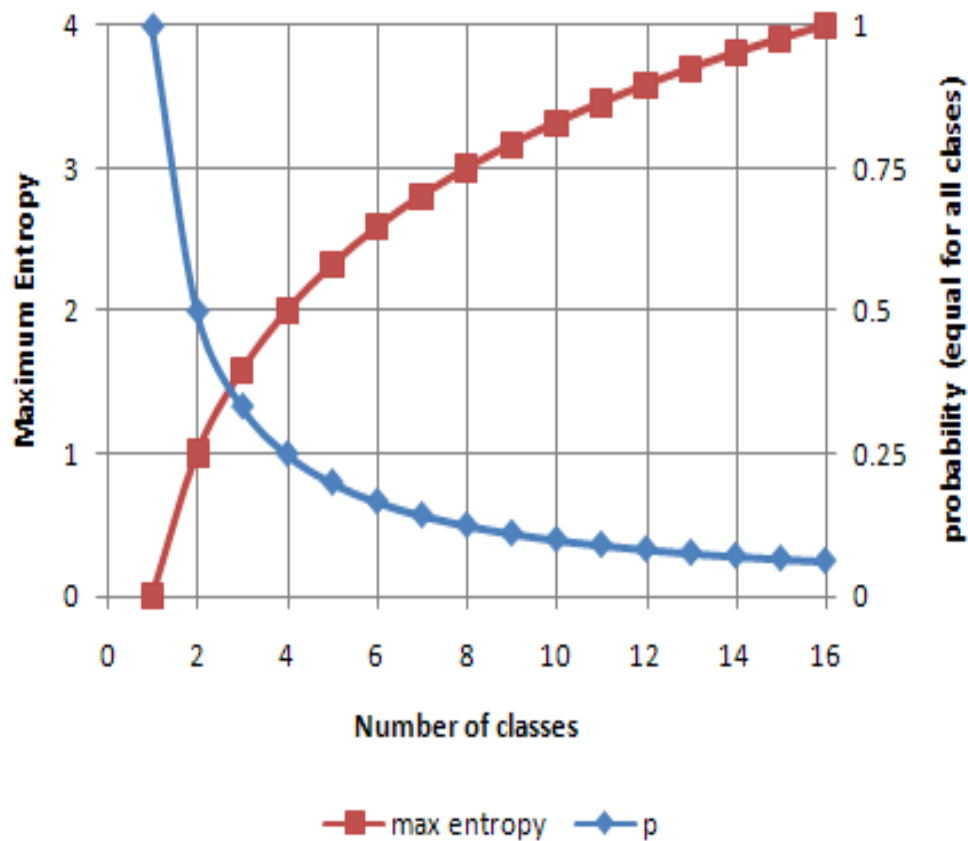


Table D

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 |

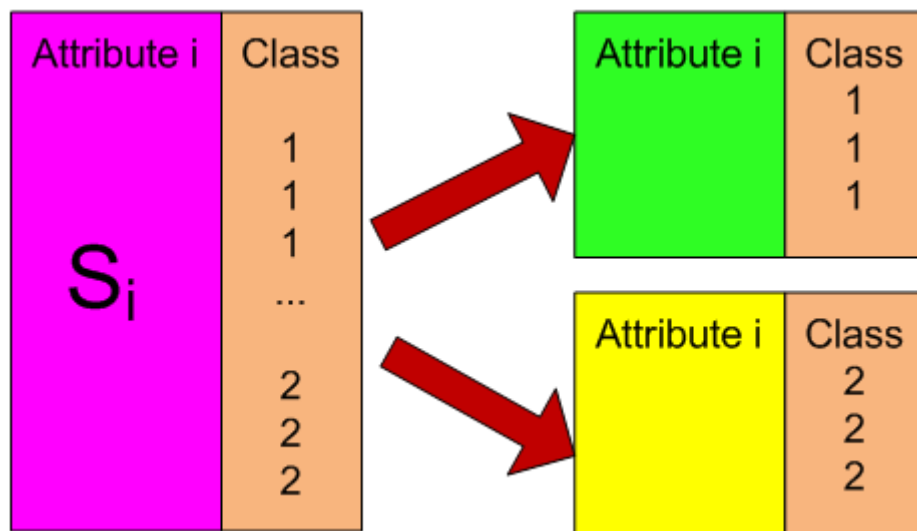
4B, 3C, 3T

Entropy 1.571

Gini index 0.660

Classification error 0.600

If the table is a subset of attribute table S_i , 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 $S_i = [\text{Travel cost/km}, \text{Transportation mode}]$ 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 |

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 S_i). 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 S_i)

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 |

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 S_i) 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.

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 | 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 ($i^* = \text{argmax} \{\text{information gain of attribute } i\}$). 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.

Travel Cost/Km
?

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

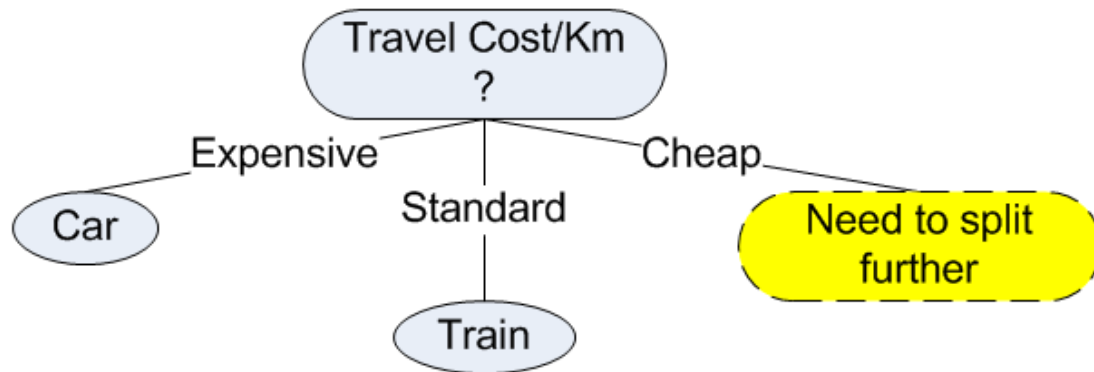
| Attributes | | | | Classes |
|------------|-----|-------------|--------------|----------------|
| Gender | Car | Travel Cost | Income Level | Transportation |
| Male | 0 | Cheap | Low | Bus |
| Male | 1 | Cheap | Medium | Bus |
| Female | 0 | Cheap | Low | Bus |
| Male | 1 | Cheap | Medium | Bus |
| 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 |

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 | Cheap | Medium | Bus |
| Male | 1 | Cheap | 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

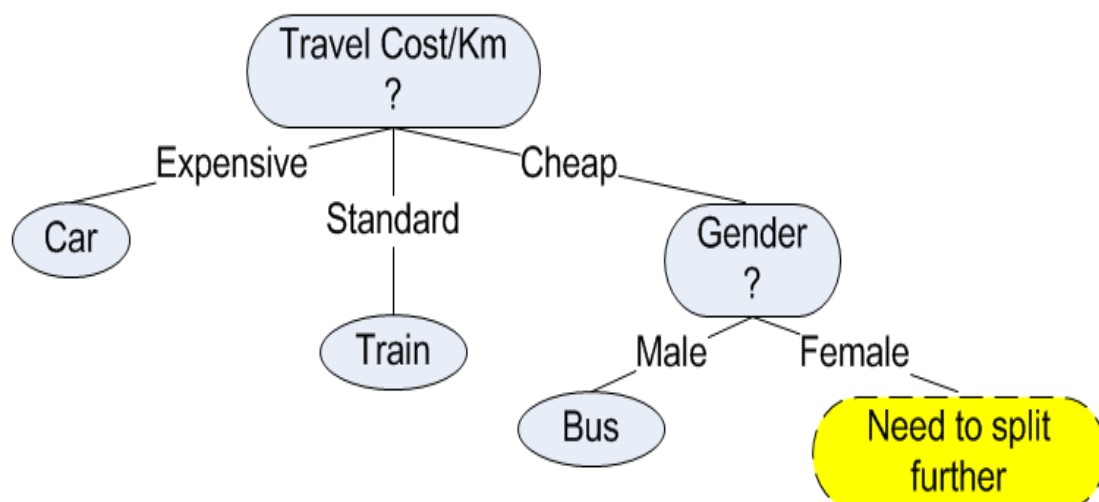
The maximum gain is obtained for the optimum attribute Gender. Once we obtain the optimum attribute, the data table is split according to that optimum attribute. In our case, Male Gender is only associated with pure class Bus, while Female still need further split of attribute.

| 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.

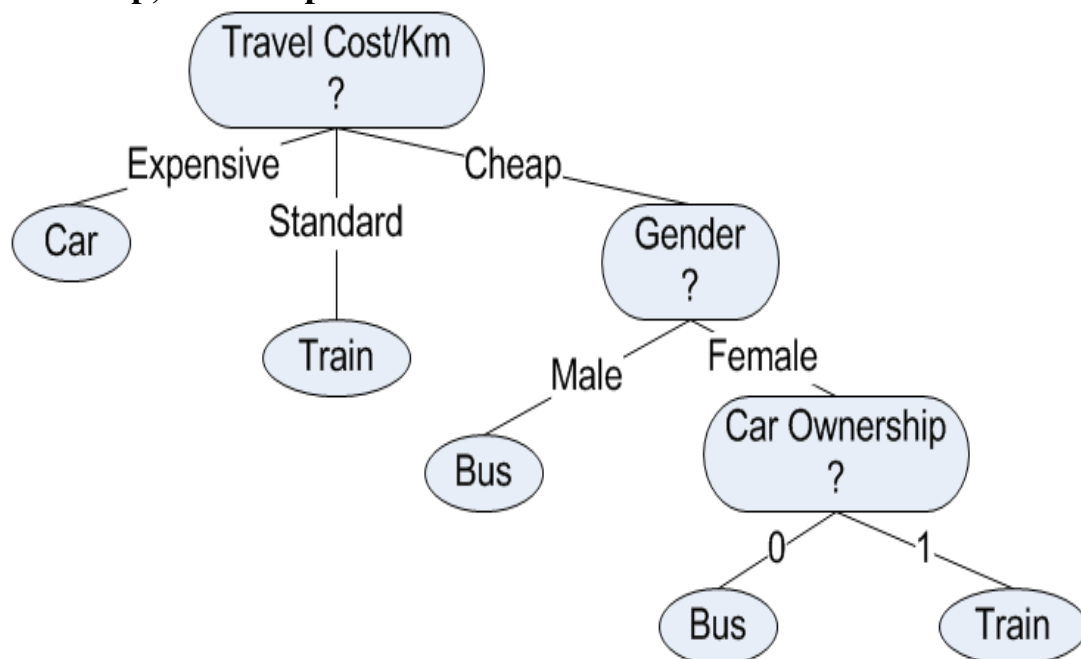
| Attributes | | | Classes |
|------------|---------------|--------------|---------------------|
| Gender | Car ownership | Income Level | Transportation mode |
| Female | 0 | Low | Bus |
| Female | 1 | Medium | Train |



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.

Bayesian Learning:

Learning models can be used for many tasks such as:

- 1- Prediction: Given the Inputs: which Outputs?
- 2- Diagnosis: Given the Outputs: which Inputs?
- 3- Unsupervised: Given Inputs and Outputs: Which structure?

Bayes Theorem:

What is the most probable hypothesis h , given training data D ?

A method to calculate the probability of a hypothesis based on:

- Its prior probability.
- The probability of observing the data given the hypothesis.
- The data itself.

$$P(h|D) = \frac{P(h)P(D|h)}{P(D)}$$

- $P(h)$ = prior probability of hypothesis h
- $P(D)$ = prior probability of training data D
- $P(h|D)$ = probability of h given D
- $P(D|h)$ = probability of D given h

Illustrative Example:

The Win envelope has 1 dollar and four beads in it. The Lose envelope has three beads in it. Someone draws an envelope at random and offers to sell it to you. Which one should we choose?



Before deciding, you are allowed to see one bead in one envelope. If it is black, Which one should we choose? And if it is red?

Prior Probabilities:

$$P(\text{Win}) = 1/2$$

$$P(\text{Lose}) = 1/2$$

$$P(\text{red}) = 3/7$$

$$P(\text{black}) = 4/7$$

$$P(\text{black}|\text{Win}) = 1/2$$

$$P(\text{red}|\text{Win}) = 1/2$$

After seeing the bead:

If bead = black:

$$P(\text{Win}|\text{black}) = \frac{P(\text{Win})P(\text{black}|\text{Win})}{P(\text{black})} = \frac{1/2 * 1/2}{4/7} = 0.4375$$

$$\text{If bead = red: } P(\text{Win}|\text{red}) = \frac{P(\text{Win})P(\text{red}|\text{Win})}{P(\text{red})} = \frac{1/2 * 1/2}{3/7} = 0.583$$

Choosing Hypotheses:

$$P(h|D) = \frac{P(h)P(D|h)}{P(D)}$$

Machine learning is interested in the *best hypothesis* h from some space H , given observed training data D

Any maximally probable hypothesis is called *maximum a posteriori (MAP) hypothesis* (\mathbf{h}_{MAP}).

$$\begin{aligned} h_{\text{MAP}} &= \underset{h \in H}{\operatorname{argmax}} P(h|D) \\ &= \underset{h \in H}{\operatorname{argmax}} \frac{P(D|h)P(h)}{P(D)} \\ &= \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h) \end{aligned}$$

note that $P(D)$ can be dropped, because it is a constant independent of h .

Example:

- consider a medical diagnosis problem in which there are two alternative hypotheses
 - the patient has a particular form of cancer (denoted by *cancer*)
 - the patient does not (denoted by $\neg\text{cancer}$)
- the available data is from a particular laboratory with two possible outcomes:
 \oplus (positive) and \ominus (negative)

$$P(\text{cancer}) = .008 \quad P(\neg\text{cancer}) = 0.992$$

$$P(\oplus|\text{cancer}) = .98 \quad P(\ominus|\text{cancer}) = .02$$

$$P(\oplus|\neg\text{cancer}) = .03 \quad P(\ominus|\neg\text{cancer}) = .97$$

- suppose a new patient is observed for whom the lab test returns a positive (\oplus) result
- Should we diagnose the patient as having cancer or not?

$$P(\oplus|\text{cancer})P(\text{cancer}) = (.98).008 = .0078$$

$$P(\oplus|\neg\text{cancer})P(\neg\text{cancer}) = (.03).992 = .0298$$

$$\Rightarrow h_{MAP} = \neg\text{cancer}$$

- the exact posterior probabilities can be determined by normalizing the above properties to 1

$$P(cancer|\oplus) = \frac{.0078}{.0078+0.0298} = .21$$

$$P(\neg cancer|\oplus) = \frac{.0298}{.0078+0.0298} = .79$$

⇒ the result of Bayesian inference depends strongly on the prior probabilities, which must be available in order to apply the method directly

Artificial Neural Networks

References

- 1- Fundamentals of Neural Networks: Architecture, Algorithms, and application, By Laurene Fausett.**
- 2- Introduction to Artificial Neural Systems, By Jacek M.Zurada,1997.**
- 3- Neural Networks. Fundamentals, Application, Examples, By Werner Kinnebrock.1995.**
- 4- Neural network for identification, prediction and control. By D. T. Pham and X. Liu.**

1.1 Introduction

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.

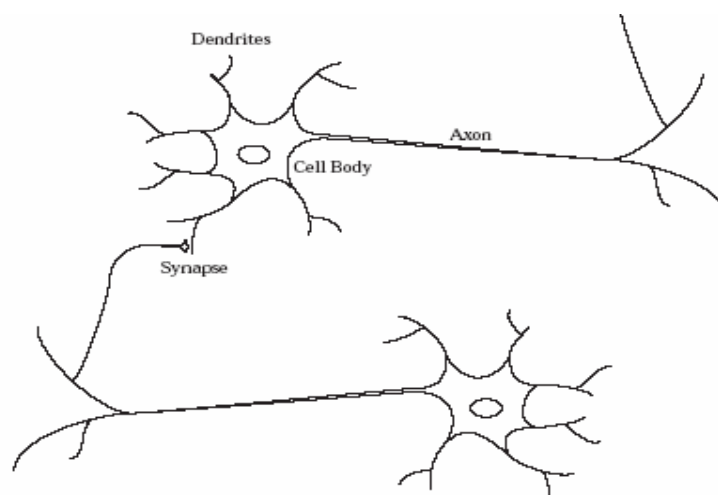
1.2 Areas of Neural Networks

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.**

1.3 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 :



Schematic Drawing of Biological Neurons

Neural Networks signals from many other neurons through input paths called dendrites 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, synapse 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.

1.4 Artificial Neural Networks (ANN)

An artificial neural network is an information processing system that has certain performance characters 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:

1-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 action function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A neural network is characterized by:

1- Architecture: - its pattern of connections between the neurons.

2- Training Learning Algorithm: - its method of determining the weights on the connections.

3- Activation function.

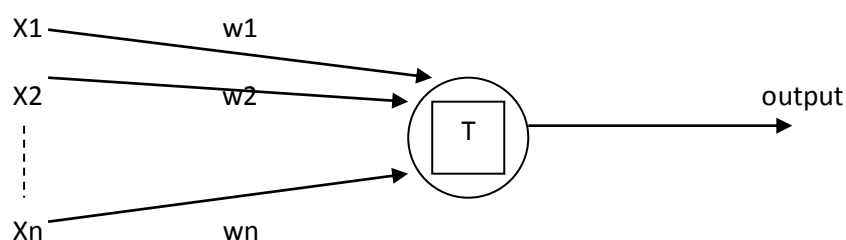
1.5 Properties of ANN

- 1- Parallelism.
- 2-Capacity for adaptation "learning rather programming".
- 3-Capacity of generalization.
- 4-No problem definition.
- 5- Abstraction & solving problem with noisy data.
- 6- Ease of constriction & learning.
- 7-Distributed memory.
- 8- Fault tolerance.

2. McCulloch-Pitts Neuron Model:

The modern view of neural networks began in the 1940s with the work of Warren McCulloch and Walter Pitts, who showed that networks of artificial neurons could, in principle, compute any arithmetic or logical function. Their work is often acknowledged as the origin of the neural network field.

McCulloch-Pitts neuron model is called the perceptron model in 1943, which is shown in following figure:



X_i : input which is either 0 or 1.

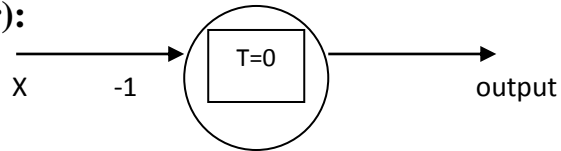
W_i : weight which is either +1 or -1.

T or θ : threshold value.

Output= 1 if $\sum_{i=1}^n w_i x_i \geq T$; **Output=0** if $\sum_{i=1}^n w_i x_i < T$.

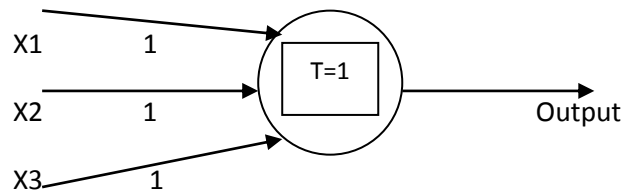
This model is simple, but it has substantial computing potential. It perform the basic logic operations; such as:

NOT Gate (Inverter):



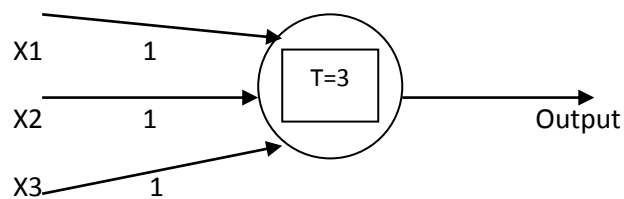
| X | Output |
|---|--------|
| 0 | 1 |
| 1 | 0 |

OR Gate (3-input):



| X3 | X2 | X1 | Output |
|----|----|----|--------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

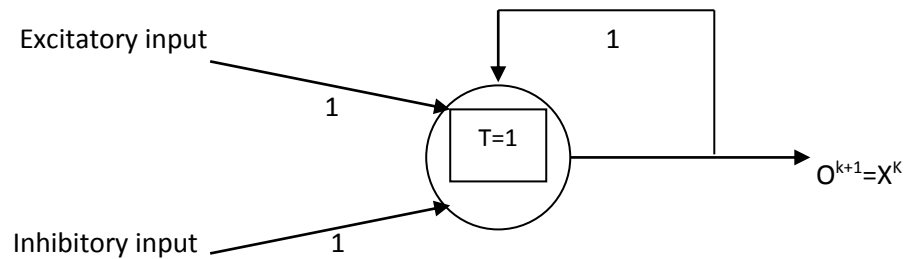
AND Gate (3-input):



| X3 | X2 | X1 | Output |
|----|----|----|--------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Memory Cell:

Once a feed back loop is closed around the neuron as shown in figure below we obtain a memory cell (R-S FF: when $R=S=0$ and $R=S=1$ no change):

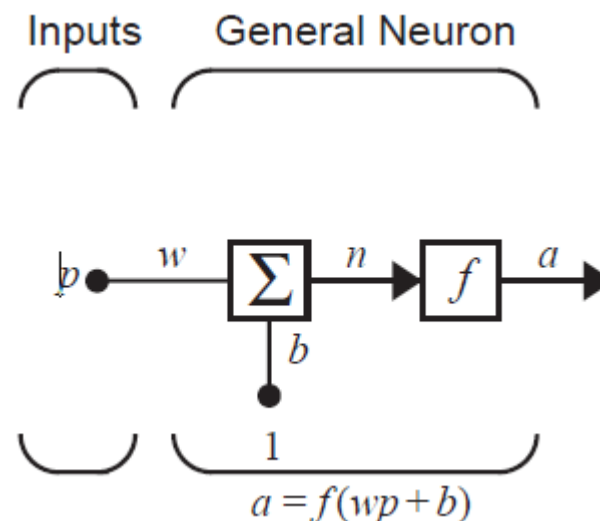


| Inh. i/p | Exc. i/p | X^K | O^{k+1} |
|----------|----------|-------|-----------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

3. Neuron Model

Single-Input Neuron

A single-input neuron is shown in Figure below.



The scalar input p is 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 f , which produces the scalar neuron output a . (Some authors use the term (activation function) rather than transfer function and (offset) rather than bias). If we relate this simple model back to the biological neuron that we discussed in Chapter 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 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. However, if you do not want to have a bias in a particular neuron, it can be omitted. Note that w and b are both adjustable scalar parameters of the neuron. Typically the transfer function is chosen by the designer and then the parameters w and b will be adjusted by

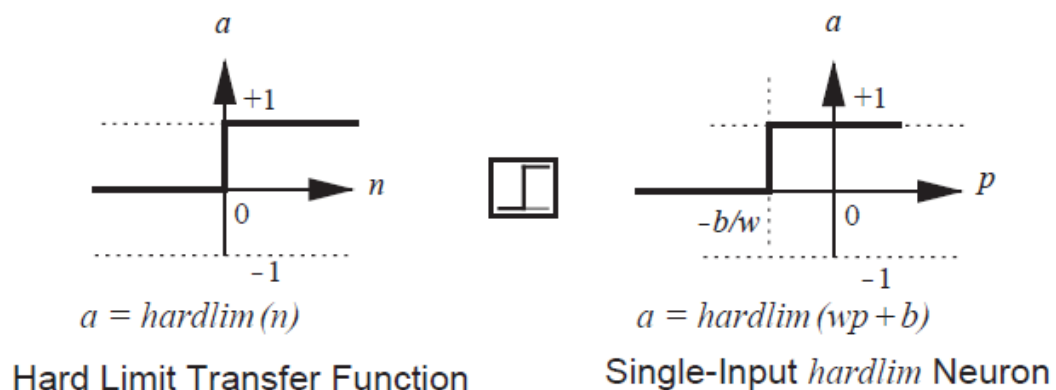
some learning rule so that the neuron input/output relationship meets some specific goal.

3.1 Transfer Functions

The transfer function in previous figure may be a linear or a nonlinear function of n . A particular transfer function is chosen to satisfy some specification of the problem that the neuron is attempting to solve. Three of the most commonly used functions are discussed below.

Hard Limit Transfer Function:

The hard limit transfer function, shown on the left side of Figure below:

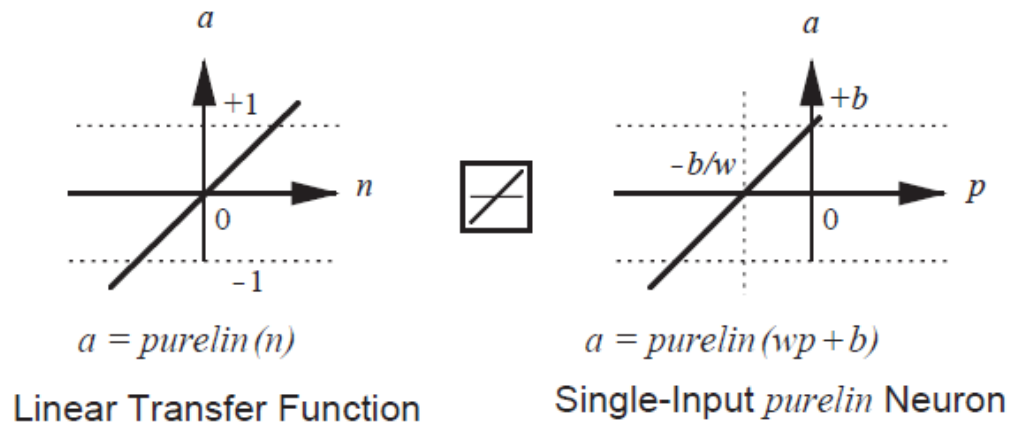


sets the output of the neuron to 0 if the function argument is less than 0, or 1 if its argument is greater than or equal to 0.

The graph on the right side of previous figure illustrates the input/output characteristic of a single-input neuron that uses a hard limit transfer function. Here we can see the effect of the weight and the bias. Note that an icon for the hard limit transfer function is shown between the two figures. Such icons will replace the general f in network diagrams to show the particular transfer function that is being used.

Linear Transfer Function:

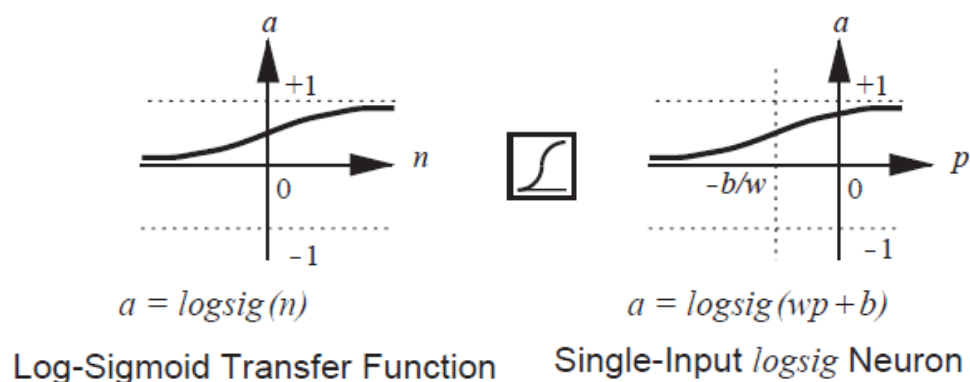
The output of a linear transfer function is equal to its input: $a = n$, as illustrated in following figure:



Neurons with this transfer function are used in the ADALINE networks. The output (a) versus input (p) characteristic of a single-input linear neuron with a bias is shown on the right of the above figure.

Log-Sigmoid Transfer Function:










The log-sigmoid transfer function is shown in following figure .



This transfer function takes the input (which may have any value between plus and minus infinity) and squashes the output into the range 0 to 1, according to the expression:

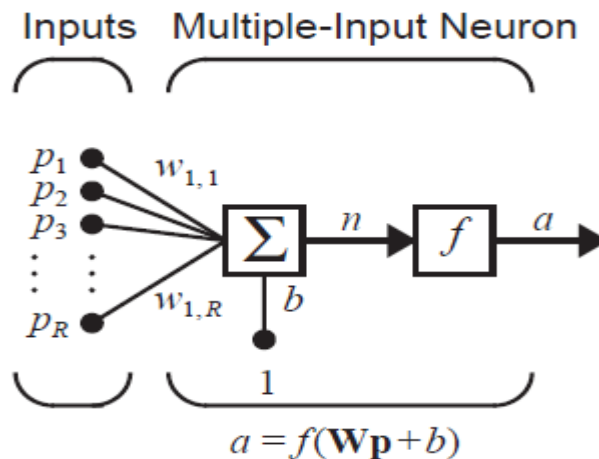
$$a = \frac{1}{1 + e^{-n}}$$

The log-sigmoid transfer function is commonly used in multilayer networks that are trained using the back propagation. Most of the transfer functions are summarized in following table.

| Name | Input/Output Relation | Icon | MATLAB Function |
|-----------------------------|--|---|-----------------|
| Hard Limit | $a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$ |  | hardlim |
| Symmetrical Hard Limit | $a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$ |  | hardlims |
| Linear | $a = n$ |  | purelin |
| Saturating Linear | $a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$ |  | satlin |
| Symmetric Saturating Linear | $a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$ |  | satlins |
| Log-Sigmoid | $a = \frac{1}{1 + e^{-n}}$ |  | logsig |
| Hyperbolic Tangent Sigmoid | $a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$ |  | tansig |
| Positive Linear | $a = 0 \quad n < 0$ $a = n \quad 0 \leq n$ |  | poslin |
| Competitive | $a = 1 \quad \text{neuron with max } n$ $a = 0 \quad \text{all other neurons}$ |  | compet |

3.2 Multiple-Input Neuron:

Typically, a neuron has more than one input. A neuron with R inputs is shown in following figure. The individual inputs p_1, p_2, \dots, p_R are each weighted by corresponding elements $w_{1,1}, w_{1,2}, \dots, w_{1,R}$ of the weight matrix W .



The neuron has a bias b , which is summed with the weighted inputs to form the net input n : $n = w_{1,1}p_1 + w_{1,2}p_2 + \dots + w_{1,R}p_R + b$.

This expression can be written in matrix form: $n = W\mathbf{p} + b$, where the matrix W for the single neuron case has only one row. Now the neuron output can be written as $a = f(W\mathbf{p} + b)$.

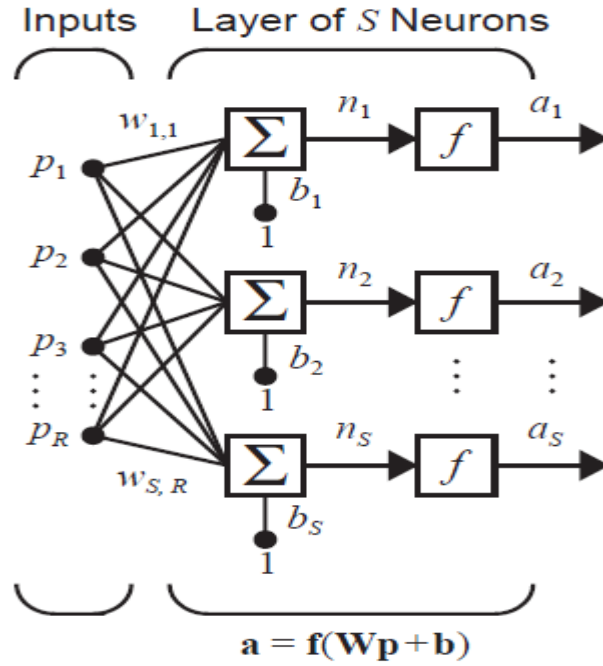
Fortunately, neural networks can often be described with matrices. This kind of matrix expression will be used throughout this lectures.

4. 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.

4.1 A Layer of Neurons:

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



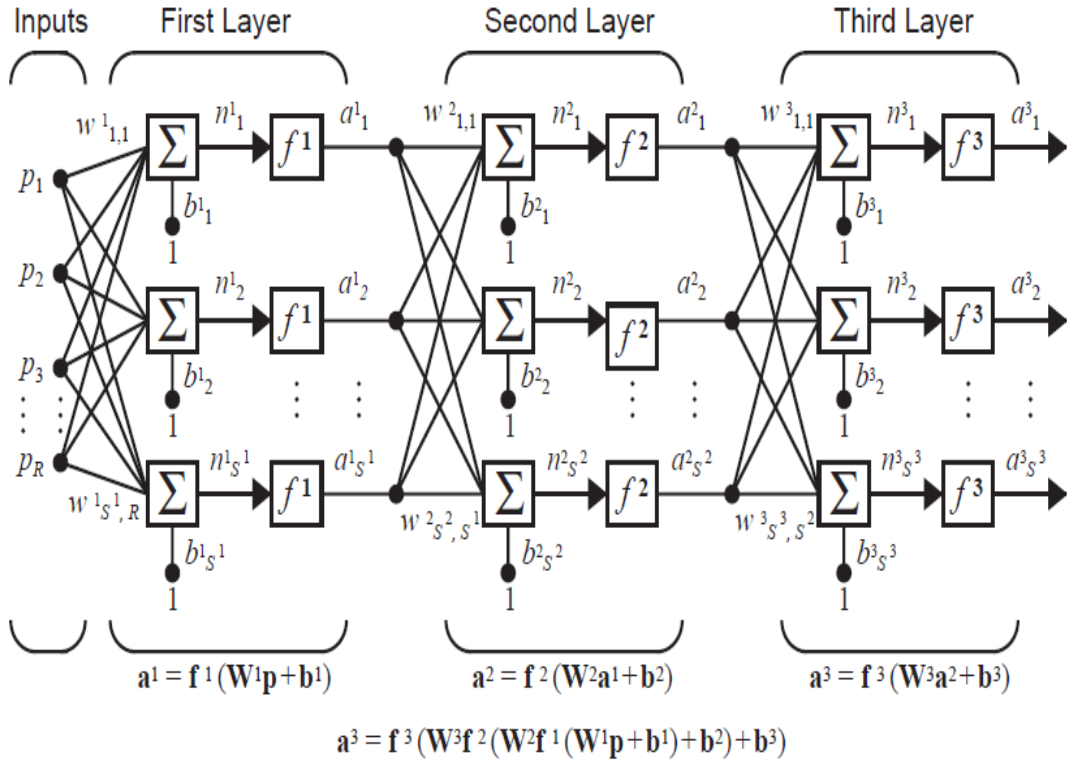
The layer includes the weight matrix, the summers, the bias vector \mathbf{b} , the transfer function boxes and the output vector \mathbf{a} . Some authors refer to the inputs as another layer, but we will not do that here. Each element of the input vector \mathbf{p} is connected to each neuron through the weight matrix \mathbf{W} . Each neuron has a bias b_i , a summer, a transfer function f and an output a_i . Taken together, the outputs form the output vector \mathbf{a} . It is common for the number of inputs to a layer to be different from the number of neurons (i.e., $R \neq S$). You might ask if all the neurons in a layer must have the same transfer function. The answer is no; you can define a single (composite) layer of neurons having different transfer functions by combining two of the networks shown above in parallel. Both networks would have the same inputs, and each network would create some of the outputs. The input vector elements enter the network through the weight matrix \mathbf{W} :

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

As noted previously, the row indices of the elements of matrix W indicate the destination neuron associated with that weight, while the column indices indicate the source of the input for that weight. Thus, the indices in $w_{3,2}$ say that this weight represents the connection to the third neuron from the second source.

4.2 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 a . 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 W^1 , and the weight matrix for the second layer is written as W^2 . This notation is used in the three-layer network shown in Figure below.



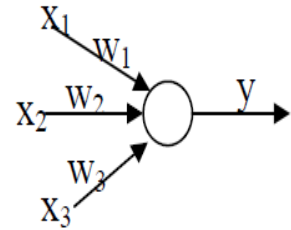
As shown, there are R inputs, S^1 neurons in the first layer, S^2 neurons in the second layer, etc. As noted, different layers can have different numbers of neurons. The outputs of layers one and two are the inputs for layers two and three. Thus layer 2 can be viewed as a one-layer

network with $R = S^1$ inputs, $S = S^2$ neurons, and an $S^1 \times S^2$ weight matrix W^2 . The input to layer 2 is a^1 , and the output is a^2 . A layer whose output is the network output is called an output layer. The other layers are called hidden layers. The network shown above has an output layer (layer 3) and two hidden layers (layers 1 and 2).

Multilayer networks are more powerful than single-layer networks. For instance, a two-layer network having a sigmoid first layer and a linear second layer can be trained to approximate most functions arbitrarily well. Single-layer networks cannot do this. At this point the number of choices to be made in specifying a network may look overwhelming, so let us consider this topic. The problem is not as bad as it looks. First, recall that the number of inputs to the network and the number of outputs from the network are defined by external problem specifications. So if there are four external variables to be used as inputs, there are four inputs to the network. Similarly, if there are to be seven outputs from the network, there must be seven neurons in the output layer. Finally, the desired characteristics of the output signal also help to select the transfer function for the output layer. If an output is to be either -1 or 1, then a symmetrical hard limit transfer function should be used. Thus, the architecture of a single-layer network is almost completely determined by problem specifications, including the specific number of inputs and outputs and the particular output signal characteristic. Now, what if we have more than two layers? Here the external problem does not tell you directly the number of neurons required in the hidden layers. In fact, there are few problems for which one can predict the optimal number of neurons needed in a hidden layer. This problem is an active area of research. As for the number of layers, most practical neural networks have just two or three layers. Four or more layers are used rarely. We should say something about the use of biases. One can choose neurons with or without biases. The bias gives the network an extra variable, and so you might expect that networks with biases would be more powerful.

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

$$\text{net} = X_1 W_1 + X_2 W_2 + X_3 W_3$$

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

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-if f is tan h

$$y = \frac{e^{-0.72} - e^{0.72}}{e^{-0.72} + e^{+0.72}} = -0.6169$$

Ex.2

The output of a simulated neural using a sigmoid function is 0.5 find the value of threshold when the input $x_1 = 1$, $x_2 = 1.5$, $x_3 = 2.5$. and have initial weights value = 0.2.

Sol

$$\text{Output} = F(\text{net} + \theta)$$

$$F(\text{net}) = \frac{1}{1 + e^{-\text{net}}}$$

$$\text{Net} = \sum W_i X_i$$

$$= X_1 W_1 + X_2 W_2 + X_3 W_3$$

$$= (1 \cdot 0.2) + (1.5 \cdot 0.2) + (2.5 \cdot 0.2) = 0.2 + 0.30 + 0.50 = 1$$

$$0.5 = \frac{1}{1 + e^{-(1+\theta)}}$$

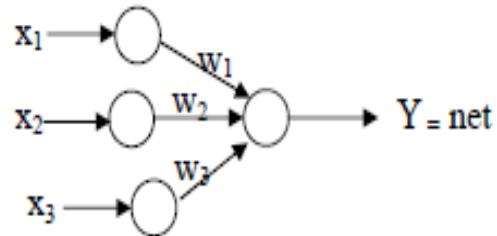
$$0.5 (1 + e^{-(1+\theta)}) = 1$$

$$0.5 + 0.5 e^{-(1+\theta)} = 1$$

$$0.5 e^{-(1+\theta)} = 0.5$$

$$e^{-(1+\theta)} = 1$$

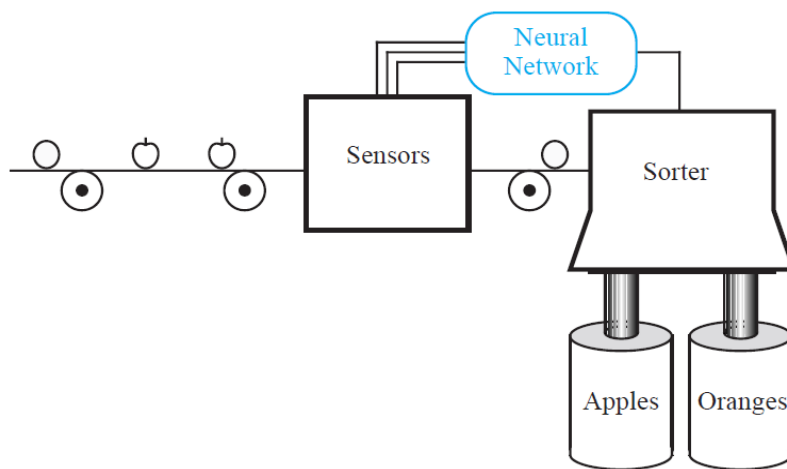
$$-(1+\theta) = \ln 1 \Rightarrow -1 - \theta = 0 \Rightarrow -\theta = 1 \Rightarrow \therefore \theta = -1$$



Example 3:

A produce dealer has a warehouse that stores a variety of fruits and vegetables. When fruit is brought to the warehouse, various types of fruit may be mixed together. The dealer wants a machine that will sort the fruit according to type. There is a conveyor belt on which the fruit is loaded. This conveyor passes through a set of sensors, which measure three properties of the fruit: shape, texture and weight.

These sensors are somewhat primitive. The shape sensor will output a 1 if the fruit is approximately round and a - 1 if it is more elliptical. The texture sensor will output a 1 if the surface of the fruit is smooth and a - 1 if it is rough. The weight sensor will output a 1 if the fruit is more than one pound and a - 1 if it is less than one pound. The three sensor outputs will then be input to a neural network. The purpose of the network is to decide which kind of fruit is on the conveyor, so that the fruit can be directed to the correct storage bin. To make the problem even simpler, let's assume that there are only two kinds of fruit on the conveyor: apples and oranges.



As each fruit passes through the sensors it can be represented by a three dimensional vector. The first element of the vector will represent shape, the second element will represent texture and the third element will represent weight:

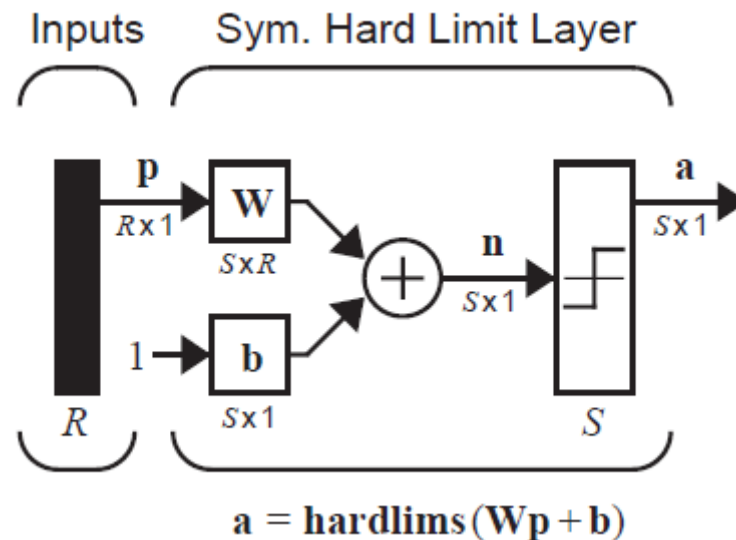
$$P = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix}$$

$$\therefore Orange = P1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

$$\therefore Apple = p2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

The neural network will receive one three-dimensional input vector for each fruit on the conveyer and must make a decision as to whether the fruit is an orange (p1) or an apple (p2) .

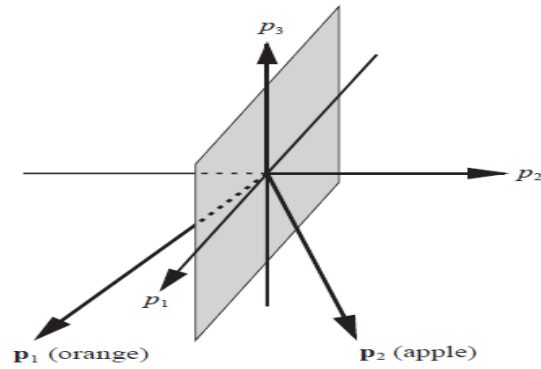
The network we will discuss is the perceptron. Following figure illustrates a single-layer perceptron with a symmetric hard limit transfer function hardlims.



Now consider the apple and orange pattern recognition problem. Because there are only two categories, we can use a single-neuron perceptron. The vector inputs are three-dimensional ($R = 3$), therefore the perceptron equation will be:

$$a = \text{hardlims} \left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + b \right).$$

We want to choose the bias b and the elements of the weight matrix so that the perceptron will be able to distinguish between apples and oranges. For example, we may want the output of the perceptron to be 1 when an apple is input and - 1 when an orange is input. Using the concept illustrated in figure below, let's find a linear boundary that can separate oranges and apples.



From this figure we can see that the linear boundary that divides these two vectors symmetrically is the p_1, p_3 plane.

The p_1, p_3 plane, which will be our decision boundary, can be described by the equation: $p_2 = 0$, but the weight matrix is orthogonal to the decision boundary. Therefore the weight matrix and bias will be $W = [0 \ 1 \ 0]$, $b = 0$, in order to get :

$$[0 \ 1 \ 0] \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + 0 = 0$$

The bias is 0 because the decision boundary passes through the origin. the prototype pattern p_2 (apple) for which we want the perceptron to produce an output of 1.

Now let's test the operation of our perceptron pattern classifier. It classifies perfect apples and oranges correctly since

Orange:

$$a = \text{hardlims} \left([0 \ 1 \ 0] \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + 0 \right) = -1(\text{orange}) ,$$

Apple:

$$a = \text{hardlims} \left([0 \ 1 \ 0] \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + 0 \right) = 1(\text{apple}) .$$

But what happens if we put a not-so-perfect orange into the classifier? Let's say that an orange with an elliptical shape is passed through the sensors. The input vector would then be:

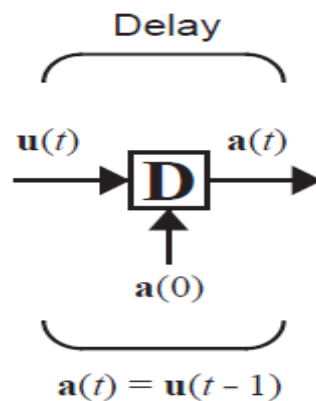
$$\mathbf{p} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}.$$

The response of the network would be

$$a = \text{hardlims} \left(\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} + 0 \right) = -1(\text{orange}) .$$

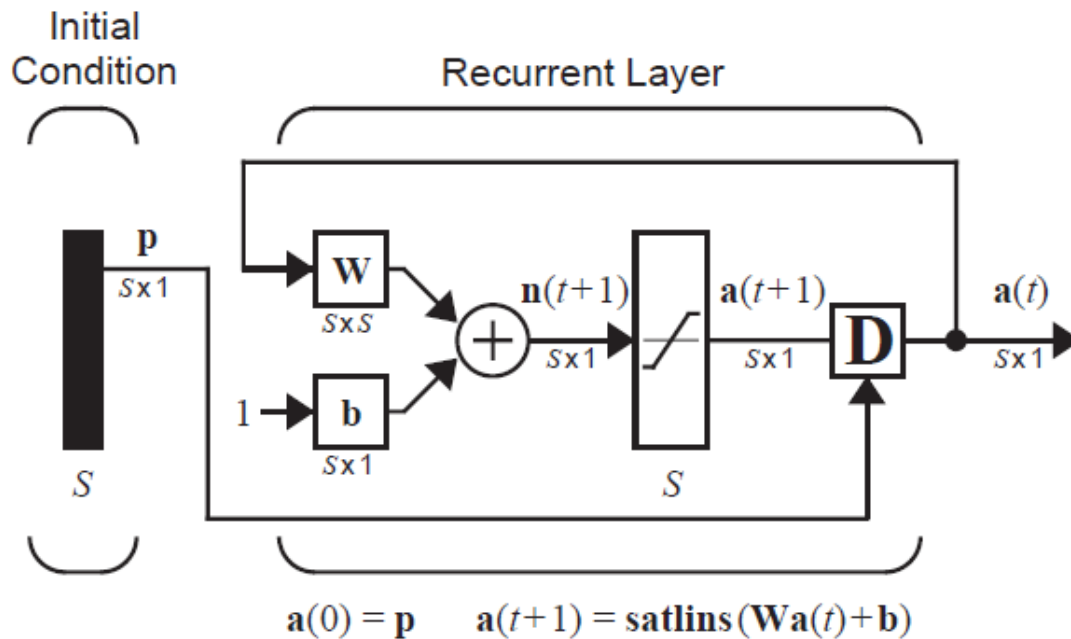
5. Recurrent (Feedback) Networks:

Delay Before we discuss recurrent networks, we need to introduce some simple building blocks. The first is the delay block, which is illustrated in Figure below.



The delay output $a(t)$ is computed from its input $u(t)$ according to $a(t) = u(t - 1)$. Thus the output is the input delayed by one time step. (This assumes that time is updated in discrete steps and takes on only integer values.) above Eq. requires that the output be initialized at time $t = 0$. This initial condition is indicated in above figure by the arrow coming into the bottom of the delay block.

Recurrent Network We are now ready to introduce recurrent networks. A recurrent network is a network with feedback; some of its outputs are connected to its inputs. This is quite different from the networks that we have studied thus far, which were strictly feed forward with no backward connections. One type of discrete-time recurrent network is shown in figure below.



In this particular network the vector p supplies the initial conditions (i.e., $a(0) = p$). Then future outputs of the network are computed from previous outputs:

$$a(1) = \text{satlins}(Wa(0) + b),$$

$$a(2) = \text{satlins}(Wa(1) + b), \dots$$

Recurrent networks are potentially more powerful than feed forward networks and can exhibit temporal behavior.

6. Learning and Adaptation

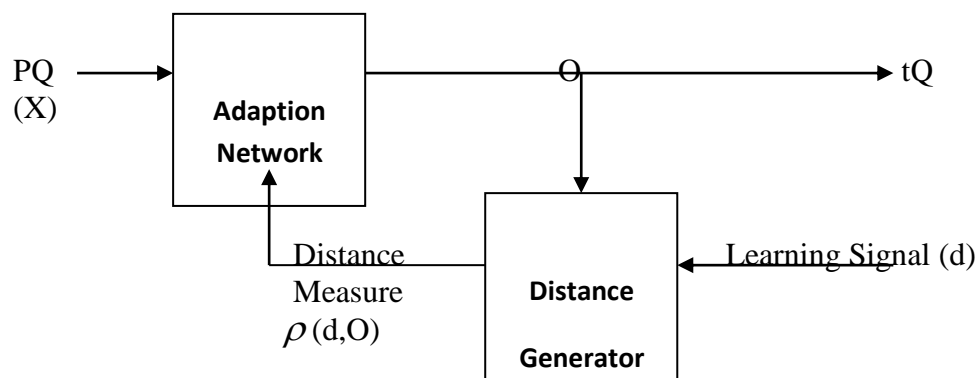
Learning: Is a relatively permanent change in behavior brought about by experience. Learning in human beings and animals is an inferred process.

By *learning rule* we mean a procedure for modifying the weights and biases of a network. (This procedure may also be referred to as a training algorithm.) The purpose of the learning rule is to train the network to perform some task. There are many types of neural network learning rules. They fall into three broad categories: supervised learning, unsupervised learning and reinforcement (or graded) learning.

Supervised Learning:

In *supervised learning*, the learning rule is provided with a set of examples (the *training set*) of proper network behavior: $\{p_1, t_1\}$, $\{p_2, t_2\}$,, $\{p_Q, t_Q\}$,

where p_Q is an input to the network and t_Q is the corresponding correct (*target*) output. As the inputs are applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets.



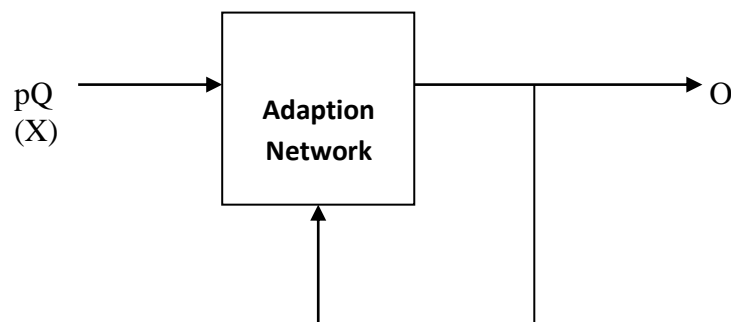
Reinforcement learning:

Reinforcement learning is similar to supervised learning, except that, instead of being provided with the correct output for each network input, the algorithm is only given a grade. The grade (or score) is a measure of the network performance over some sequence of inputs.

This type of learning is currently much less common than supervised learning. It appears to be most suited to control system applications.

Unsupervised Learning:

In *unsupervised learning*, the weights and biases are modified in response to network inputs only. There are no target outputs available. At first glance this might seem to be impractical. How can you train a network if you don't know what it is supposed to do? Most of these algorithms perform some kind of clustering operation. They learn to categorize the input patterns into a finite number of classes. This is especially useful in such applications as vector quantization.



Neural Networks Learning Rules:

$$W(\text{new}) = W(\text{old}) + \Delta W$$

Where ΔW is the weight change.

$$\Delta W_i(t) = c \cdot r[w_i(t), x_i(t), d_i] \cdot X_i(t)$$

Where c is learning constant that determines the rate of learning.

$r[w_i, x_i, d_i]$ is general learning rule.

$$W_i(t+1) = W_i(t) + c \cdot r[w_i(t), x_i(t), d_i] \cdot X_i(t) \quad \text{for continuous time.}$$

$$W_i^{K+1} = W_i^K + c r(w_i^K, x_i^K, d_i^K) X_i^K \quad \text{for discrete time.}$$

6.1 Hebbian Learning Rule:

The earliest and simplest learning rule for a neural net is generally known as the Hebb rule. Hebbian learning rule suggested by Hebb in 1949. Hebb's basic idea is that if a unit U_j receives an input from a unit U_i and both units are highly active (positive), then the weight W_{ij} (from unit i to unit j) should be strengthened (increase), otherwise the weight decrease.

Hebbian learning rule is unsupervised learning, with continuous and discrete transfer functions .

Example:

Apply Hebbian learning rule on neuron with four inputs. And initial weight $W^1 = [1 \ -1 \ 0 \ 0.5]$, learning constant $c = 1$. The training set is:

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix}, x_2 = \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix}, x_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix}$$

Using symmetrical hard limit transfer function:

$$n1 = net1 = [W^{1t} x_1] = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = 3$$

$$O1 = \text{sgn}[3] = 1; \quad W^2 = W^1 + c \cdot O1 x_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 1 \times 1 \times \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 1.5 \\ 0.5 \end{bmatrix}$$

$$n2 = net2 = [W^{2t} x_2] = \begin{bmatrix} 2 & -3 & 1.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = -0.25$$

$$O2 = \text{sgn}[-0.25] = -1; \quad W^3 = W^2 + c \cdot O2 x_2 = \begin{bmatrix} 2 \\ -3 \\ 1.5 \\ 0.5 \end{bmatrix} + 1 \times -1 \times \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -2.5 \\ 3.5 \\ 2 \end{bmatrix}$$

$$n_3 = \text{net}_3 = [W^{3t} x_3] = \begin{bmatrix} 1 & -2.5 & 3.5 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = -3$$

$$O_3 = \text{sgn}[-3] = -1 ; W^4 = W^3 + c O_3 x_3 = \begin{bmatrix} 1 \\ -2.5 \\ 3.5 \\ 2 \end{bmatrix} + 1 \times -1 \times \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 1 \\ -3.5 \\ 4.5 \\ 0.5 \end{bmatrix}$$

Using log sigmoid transfer function:

$$n_1 = \text{net}_1 = [W^{1t} x_1] = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = 3$$

$$O_1 = f(n_1) = f(\text{net}_1) = \frac{1}{1 + e^{-n_1}} = \frac{1}{1 + e^{-3}} = 0.953$$

$$W^2 = W^1 + c O_1 x_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 1 \times 0.953 \times \begin{bmatrix} 1 \\ -2 \\ 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 1.953 \\ -2.906 \\ 1.4295 \\ 0.5 \end{bmatrix}$$

$$n_2 = \text{net}_2 = [W^{2t} x_2] = \begin{bmatrix} 1.953 & -2.906 & 1.4295 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = -0.203$$

$$O_2 = f(n_2) = f(\text{net}_2) = \frac{1}{1 + e^{-n_2}} = \frac{1}{1 + e^{0.203}} = 0.449$$

$$W^3 = W^2 + c \cdot O_2 x_2 = \begin{bmatrix} 1.953 \\ -2.906 \\ 1.4295 \\ 0.5 \end{bmatrix} + 1 \times 0.449 \times \begin{bmatrix} 1 \\ -0.5 \\ -2 \\ -1.5 \end{bmatrix} = \begin{bmatrix} 2.402 \\ -3.131 \\ 0.5315 \\ -0.1735 \end{bmatrix}$$

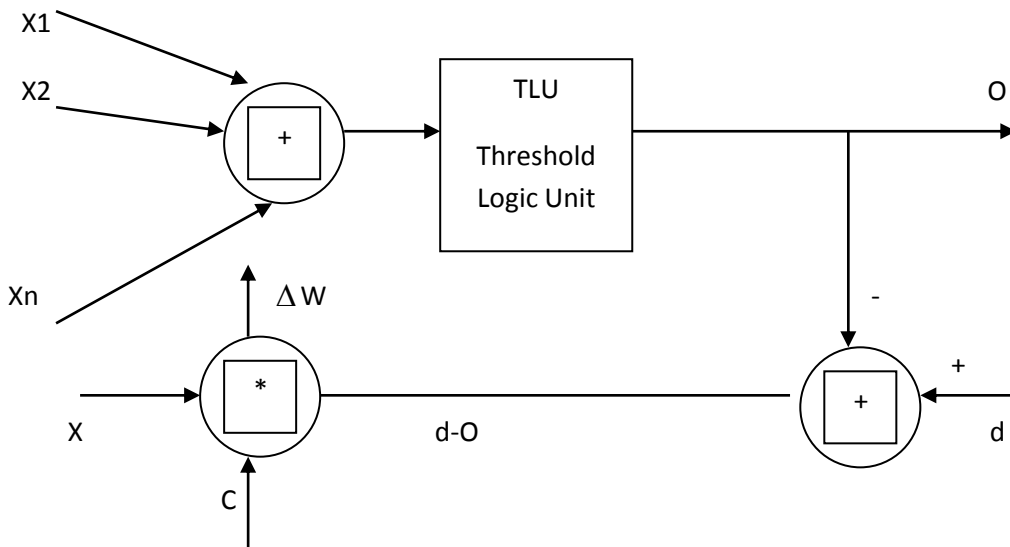
$$n_3 = \text{net}_3 = [W^3]^t x_3 = \begin{bmatrix} 2.402 & -3.131 & 0.5315 & -0.1735 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = -3.923$$

$$O_3 = f(n_3) = f(\text{net}_3) = \frac{1}{1 + e^{-n_3}} = \frac{1}{1 + e^{3.923}} = 0.0194$$

$$W^4 = W^3 + c \cdot O_3 x_3 = \begin{bmatrix} 2.402 \\ -3.131 \\ 0.5315 \\ -0.1735 \end{bmatrix} + 1 \times 0.0194 \times \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 2.402 \\ -3.112 \\ 0.5121 \\ -0.1444 \end{bmatrix}$$

6.2 Perceptron Learning Rule:

This learning rule is applicable only for neurons with discrete transfer functions, and the weights are adjusted if and only if the output is incorrect. This learning is supervised learning rule as shown in following figure.



At Perceptron learning rule:

$$r = d_i - O_i$$

$$\text{Then: } \Delta W = c (d_i - O_i) X_j$$

Example:

Apply Perceptron learning rule on neuron with three inputs. And initial weight:

$W^1 = [1 \ -1 \ 0 \ 0.5]$, learning constant $c = 0.1$. The training sets are:

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}; d_1 = -1, x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}; d_2 = -1, x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}; d_3 = 1.$$

Using symmetrical hard limit transfer function:

$$n1 = \text{net1} = [W^{1t} x_1] = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

$$O1 = \text{sgn}[2.5] = 1; \quad d_1 \neq O_1$$

$$W^2 = W^1 + c [d_1 - O_1] x_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 0.1 \times [-1 - 1] \times \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

$$n2 = \text{net2} = [W^{2t} x_2] = \begin{bmatrix} 0.8 & -0.6 & 0 & 0.7 \end{bmatrix} \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} = -1.6$$

$$O2 = \text{sgn}[-1.6] = -1; \text{ since } d_2 = O_2 = -1; \therefore d_2 - O_2 = 0; \therefore \text{ NO need for learning (correction).}$$

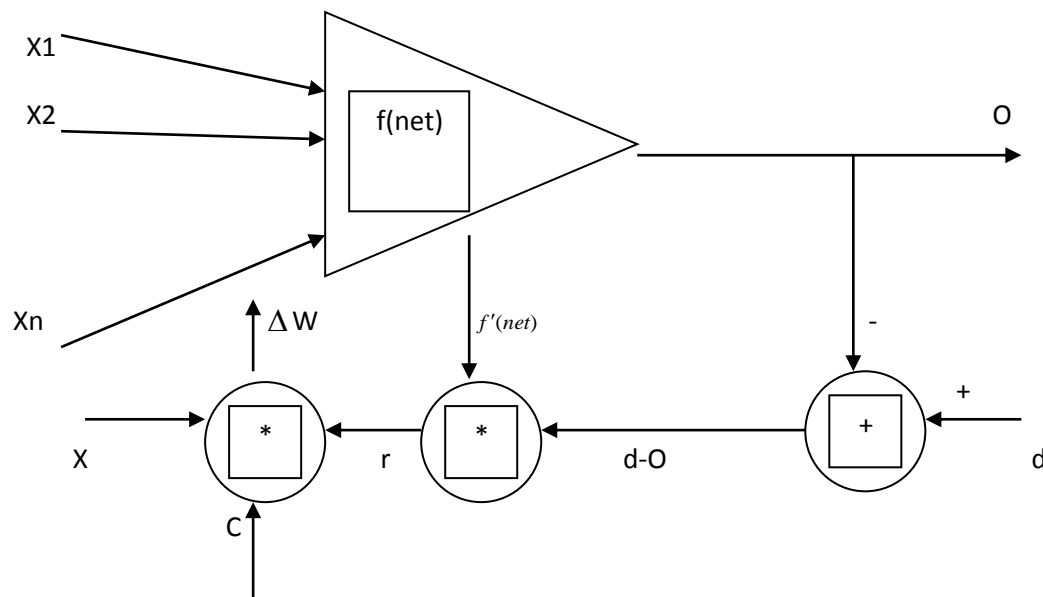
$$n3 = \text{net3} = [W^{3t} x_3] = \begin{bmatrix} 0.8 & -0.6 & 0 & 0.7 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = -2.1$$

$$O_3 = \text{sgn} [-2.1] = -1 ; d_3 = 1 \neq O_3$$

$$W^3 = W^2 + c [d_3 - O_3] x_3 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + 0.1 \times [1 - (-1)] \times \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

6.3 Delta Learning Rule:

This learning rule is applicable only for neurons with continuous transfer functions. This learning is supervised learning rule as shown in following figure.



At delta learning rule:

$$r = (d_i - f(w_i x)) f'(w_i x)$$

$$f'(w_i x) = \frac{1}{2} (1 - o^2)$$

$$\text{Then: } \Delta w_i = c(d_i - o_i) f'(\text{net}_i) x$$

$$\text{Or: } \Delta w_i = c(d_i - o_i) \frac{1}{2} (1 - o^2) x$$

Example:

Apply Delta learning rule on neuron with three inputs. And initial weight:

$W^1 = [1 \ -1 \ 0 \ 0.5]$, learning constant $c = 0.1$. The training sets are:

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}; d_1 = -1, x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}; d_2 = -1, x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}; d_3 = 1.$$

Using log sigmoid transfer function:

$$n_1 = \text{net}_1 = [W^{1t} x_1] = \begin{bmatrix} 1 & -1 & 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

$$O_1 = f(n_1) = f(\text{net}_1) = \frac{1}{1 + e^{-n_1}} = \frac{1}{1 + e^{-2.5}} = 0.924$$

$$W^2 = W^1 + c (d_1 - O_1) \frac{1}{2} (1 - O_1^2) x_1 =$$

$$\begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 0.1 \times (-1 - 0.924) \times 0.5 \times (1 - (0.924)^2) \times \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.986 \\ -0.972 \\ 0 \\ 0.514 \end{bmatrix}$$

$$n_2 = \text{net}_2 = [W^{2t} x_2] = \begin{bmatrix} 0.986 & -0.972 & 0 & 0.514 \end{bmatrix} \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} = -1.972$$

$$O_2 = f(n_2) = f(\text{net}_2) = \frac{1}{1 + e^{-n_2}} = \frac{1}{1 + e^{1.972}} = 0.122$$

$$W^3 = W^2 + c (d_2 - O_2) \frac{1}{2} (1 - o_2^2) x_2 =$$

$$\begin{bmatrix} 0.986 \\ -0.972 \\ 0 \\ 0.514 \end{bmatrix} + 0.1 \times (-1 - 0.122) \times 0.5 \times (1 - (0.122)^2) \times \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.986 \\ -0.889 \\ -0.028 \\ 0.459 \end{bmatrix}$$

$$n_3 = \text{net}_3 = [W^3 x_3] = \begin{bmatrix} 0.986 & -0.889 & -0.028 & 0.459 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = -2.348$$

$$O_3 = f(n_3) = f(\text{net}_3) = \frac{1}{1 + e^{-n_3}} = \frac{1}{1 + e^{-2.348}} = 0.087$$

$$W^4 = W^3 + c (d_3 - O_3) \frac{1}{2} (1 - o_3^2) x_3 =$$

$$\begin{bmatrix} 0.986 \\ -0.889 \\ -0.028 \\ 0.459 \end{bmatrix} + 0.1 \times (1 - 0.087) \times 0.5 \times (1 - (0.087)^2) \times \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.94 \\ -0.843 \\ -0.005 \\ 0.413 \end{bmatrix}$$

6.4 Widrow-Hoff (Least Mean Square) Learning Rule:

This learning rule is applicable for neurons without transfer functions. This learning is supervised learning rule. This learning rule can be considered a special case of Delta learning rule assuming that:

$$f(w_i^t x) = w_i^t x; f'(w_i^t x) = 1$$

Then: $r = d_i - w_i x$

i.e.:

$$\Delta w_i = c(d_i - w_i x)x$$

Example:

Perform two training steps using Widrow-Hoff learning rule? Assume the following training data:

$$x_1 = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}; d_1 = -1; x_2 = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}; d_2 = 1; w^1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}; c = 0.25.$$

$$n1 = net1 = [w^{1t} x_1] = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} = 1$$

$$w^2 = w^1 + c(d_1 - net_1)x_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + (0.25)(-1 - 1) \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1.5 \end{bmatrix}$$

$$n2 = net2 = [w^{2t} x_2] = \begin{bmatrix} 0 & 0 & 1.5 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix} = -1.5$$

$$w^3 = w^2 + c(d_2 - net_2)x_2 =$$

$$= \begin{bmatrix} 0 \\ 0 \\ 1.5 \end{bmatrix} + (0.25)(1 + 1.5) \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.625 \\ -1.25 \\ 0.875 \end{bmatrix}$$

Notes:

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.**

The convergence in the case of complete information is better than in the case of incomplete information.

Training a NN is to perform weights assignment in a net to minimize the o/p error. The net is said to be trained when convergence is achieved or in other words the weights stop changing.

Some Other Learning Rules:

- 1. correlation learning rule**
- 2. Winner –Take-All learning rule**
- 3. Outstar learning rule**