[Chapter 5: Applications of AI]
Artificial Intelligence (CSC 355)

Arjun Singh Saud

Central Department of Computer Science & Information Technology
Tribhuvan University

EXPERT SYSTEMS

Introduction

An Expert system is a set of program that manipulate encoded knowledge to solve problem in a specialized domain that normally requires human expertise.

An expert system is an AI computer program specially designed to represent human expertise in a particular domain.

An expert system's knowledge is obtained from expert sources and coded in a form suitable for the system to use in its inference or reasoning processes. The expert knowledge must be obtained from specialists or other sources of expertise, such as texts, journals, articles and data bases. Once a sufficient body of expert knowledge has been acquired , it must be encoded in some form, loaded into a knowledge base then tested and refined continually throughout the life of the system.

Who is generally acknowledged as an expert ?

Anyone can be considered a domain expert if he or she has deep knowledge(of booth facts and rules) and strong practical experience in a particular domain. The area of the domain may be limited . For example , experts in electrical engineering may have only general knowledge about transformers, while experts in life insurance marketing might have limited understanding of a real estate insurance policy. In general , an expert is a skillful person who can do things other people cannot.

What is Knowledge ?

Knowledge is a theoretical or practical understanding of a subject or a domain. Knowledge is also the sum of what is currently known, and apparently knowledge is power. Those who possess knowledge are called experts. They are the most powerful and important people in their organizations. Any successful company has at least a few first-class experts and it cannot remain in business without them.

How do experts think ?

The human mental process is internal , and it is too complex to be represented as a algorithm. However, most experts are capable of expressing their knowledge as an algorithm. However, most experts are capable of expressing their knowledge in the form of rules for problem solving. Consider a simple example. Imagine, you meet an alien ! He wants to cross a road. Can you help him ? You are an expert in crossing roads- you've been on this job for several years. Thus you are able to teach the alien. How would you do this ?

You explain to the alien that he can cross the road safely when the traffic light is green, and he must stop when the traffic light is red. These are the basic rules. Your knowledge can be formulated as the following simple statements:

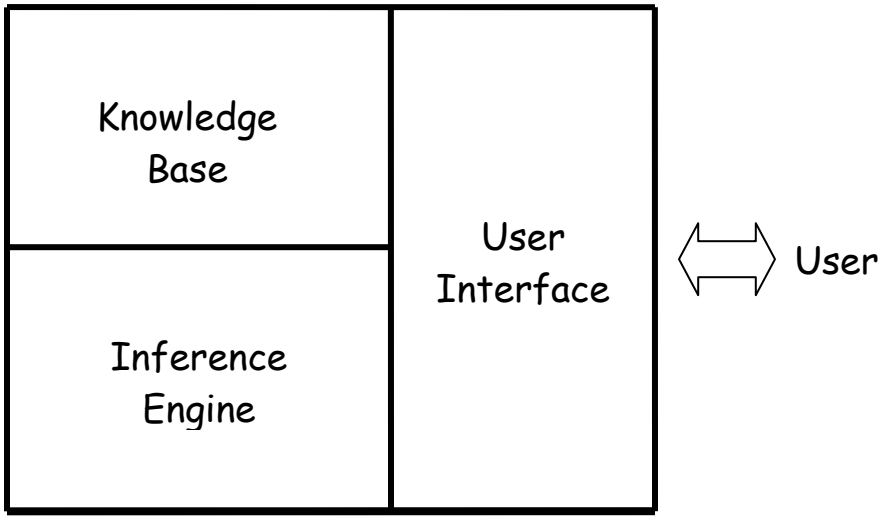
IF the 'traffic light' is green
THEN the action is go

IF the 'traffic light' is red
THEN The action is stop

These statements represented in the IF-THEN form are called **production rules** or just **rules**. The term 'rule' in AI, which is the most commonly used type of knowledge representation, can be defined as an IF-THEN structure that relates given information or facts in the IF part to some action in the THEN part. A rule provides some description of how to solve a problem. Rules are relatively easy to create and understand.

Components of an Expert System

There is currently no such thing as "standard" expert system. Because a variety of techniques are used to create expert systems, they differ as widely as the programmers who develop them and the problems they are designed to solve. However, the principal components of most expert systems are knowledge base, an inference engine, and a user interface, as illustrated in the figure.



(Fig: Block Diagram of expert system)

1. **Knowledge Base**

The component of an expert system that contains the system's knowledge is called its knowledge base. This element of the system is so critical to the way most expert systems are constructed that they are also popularly known as *knowledge-based systems*

A knowledge base contains both declarative knowledge(facts about objects, events and situations) and procedural knowledge (information about courses of action). Depending on the form of knowledge representation chosen, the two types of knowledge may be separate or integrated. Although many knowledge representation techniques have been used in expert systems, the most prevalent form of knowledge representation currently used in expert systems is the *rule-based production* system approach.

To improve the performance of an expert system , we should supply the system with some knowledge about the knowledge it posses, or in other words, meta-knowledge.

Meta-knowledge can be simply defined as **knowledge about knowledge**. Meta-knowledge is knowledge about the use and control of domain knowledge in an expert system . In rule-based expert systems, meta-knowledge is represented by meta-

rules. A meta-rule determines a strategy for the use of task-specific rules in the expert system.

2. Inference Engine

Simply having access to a great deal of knowledge does not make you an expert; you also must know **how** and **when** to apply the appropriate knowledge. Similarly, just having a knowledge base does not make an expert system intelligent. The system must have another component that directs the implementation of the knowledge . That element of the system is known variously as the *control structure*, the *rule interpreter*, or the *inference engine*.

The inference engine decides which heuristic search techniques are used to determine how the rules in the knowledge base are to be applied to the problem. In effect , an inference engine "runs" an expert system, determining which rules are to be invoked, accessing the appropriate rules in the knowledge base, executing the rules , and determining when an acceptable solution has been found.

3. User Interface

The component of an expert system that communicates with the user is known as the *user interface*. The communication performed by a user interface is bidirectional. At the simplest level, we must be able to describe our problem to the expert system, and the system must be able to respond with its recommendations. We may want to ask the system to explain its "reasoning" , or the system may request additional information about the problem from us.

Although the designer of expert systems generally have a great deal of experience with computers, the intended users of expert systems are frequently computer novices. It is therefore critically important to ensure that an expert system is especially easy to use.

Features of an expert system

What are the features of a good expert system ? Although each expert system has its own particular characteristics, there are several features common to many systems. The following list from Rule-Based Expert Systems suggests seven criteria that are important prerequisites for the acceptance of an expert system .

1. "The program should be **useful**." An expert system should be developed to meet a specific need, one for which it is recognized that assistance is needed.

2. "The program should be **usable**." An expert system should be designed so that even a novice computer user finds it easy to use .

3. "The program should be **educational when appropriate**." An expert system may be used by non-experts, who should be able to increase their own expertise by using the system.

4. "The program should be able to **explain its advice**." An expert system should be able to explain the "reasoning" process that led it to its conclusions, to allow us to decide whether to accept the system's recommendations.

5. "The program should be able to **respond to simple questions**." Because people with different levels of knowledge may use the system , an expert system should be able to answer questions about points that may not be clear to all users.

6. "The program should be able to **learn new knowledge**." Not only should an expert system be able to respond to our questions, it also should be able to ask questions to gain additional information.

7. "The program's knowledge should be **easily modified**." It is important that we should be able to revise the knowledge base of an expert system easily to correct errors or add new information.

The main players in the expert system development team

As soon as knowledge is provided by human expert, we can input it into a computer. We expect the computer to act as an intelligent assistant in some specific domain of expertise or to solve a problem that would otherwise have to be solved by an expert. We also would like the computer to be able to integrate new knowledge and to show its knowledge in form that is easy to read and understand, and to deal with simple sentences in a natural language rather than an artificial programming language. Finally, we want our computer to explain how it reaches a particular conclusion. In other words, we have to build an expert system, a computer program capable of performing at the level of a human expert in a narrow problem area.

The most popular expert systems are rule-based systems. a great number have been built and successfully applied in such areas as business and engineering, medicine and geology, power systems and mining. A large number of companies produce and market software for rule-based expert system development - expert system shells for personal computers.

Expert system shells are becoming particularly popular for developing rule based systems. Their main advantage is that the system builder can now concentrate on the knowledge rather than on programming language.

What is an expert system shell?

An expert system shell can be considered as an expert system with the knowledge removed. Therefore, all the user has to do is to add the knowledge in the form of rules and provide relevant data to solve a problem.

Let us now look at who is needed to develop an expert system and what skills are needed.

In general, there are five members of the expert system development team: the domain expert, the knowledge engineer, the programmer, the project manager and the end-user. The success of their expert system entirely depends on how well the members work together. The basic relations in the development team are summarized in figure.

Domain expert

The domain expert is a knowledgeable and skilled person capable of solving problems in a specific area or **domain. T**his person has the greatest expertise in a given domain. This expertise is to be captured in the expert system. Therefore, the expert must be able to communicate his or her knowledge, be willing to participate in the expert system development and commit a substantial amount of time to the project. The domain expert is the most important player in the expert system development team.

Knowledge engineer

The Knowledge engineer is some one who is capable of designing , building and testing an expert system. This person is responsible for selecting an appropriate task for the expert system. He or she interviews the domain expert to find out how a particular problem is solved. Through interaction with the expert, the knowledge engineer establishes what reasoning methods the expert uses to handle facts and rules and decide how to represent them in the expert system. The knowledge engineer then chooses some development software or an expert system shell, or looks at programming languages for

encoding the knowledge. And finally, the knowledge engineer is responsible for testing, revising and integrating the expert system into the workspace. Thus, the knowledge engineer is committed to the project from the initial design stage to the final delivery of the expert system, and even after the project is completed, he or she may also be involved in maintaining the system.

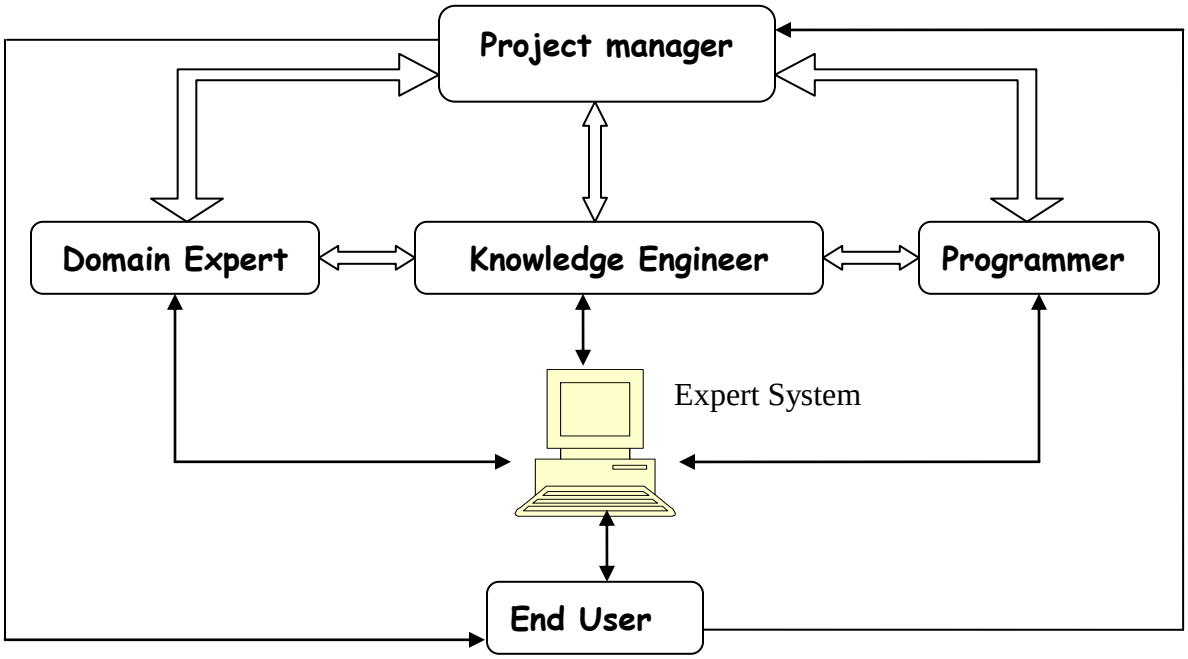


Fig: The main players of the expert system development team

Programmer

The programmer is the person responsible for the actual programming, describing the domain knowledge in terms that a computer can understand. The programmer needs to have skills in symbolic programming in such AI languages as LISP, Prolog and OPS5 and also some experience in the application of different types of expert system shells. In addition, the programmer should know conventional programming languages like C, Pascal, Fortran and Basic. If an expert system shell is used, the knowledge engineer can easily encode the knowledge into the expert system shell and thus eliminate the need for the programmer. However, if a shell cannot be used , a programmer must develop the knowledge and data representation structures (knowledge and database), control structure(inference engine) and dialogue structure (user interface). The programmer may also be involved in testing the expert system.

Project manager

The project manager is the leader of the expert system development team, responsible for keeping the project on track. He or she makes sure that all deliverables

and milestones are met, interacts with the expert, knowledge engineer, programmer and end-user.

End-User

The end-user**,** often called just the user, is a person who uses the expert system when it is developed. The user might be an analytical chemist determining the molecular structure of soil from mars, a junior doctor diagnosing an infectious blood disease, an exploration geologist trying to discover a new mineral deposit, or a power system operator needing advice in an emergency. Each of these users of expert systems has different needs, which the system must meet; the system's final acceptance will depend on the user's satisfaction. The user must not only be confident in the expert system performance but also feel comfortable using it. Therefore, the design of the user interface of the expert system is also vital for the project's success; the end-user's contribution here can be crucial.

The development of an expert system can be started when all five players have joined the team. However, many expert systems are now developed on personal computers using expert system shells. This can eliminate the need for the programmer and also might reduce the role of the knowledge engineer, programmer and even the expert could be same person. But all team players are required when large expert systems are developed.

Stages of Expert System Development

Although great strides have been made in expediting the process of developing an expert system, it often remains an extremely time consuming task. It may be possible for one or two people to develop a small expert system in a few months; however the development of a sophisticated system may require a team of several people working together for more than a year.

An expert system typically is developed and refined over a period of several years. We can divide the process of expert system development into five distinct stages. In practice, it may not be possible to break down the expert system development cycle precisely. However, an examination of these five stages may serve to provide us with some insight into the ways in which expert systems are developed.

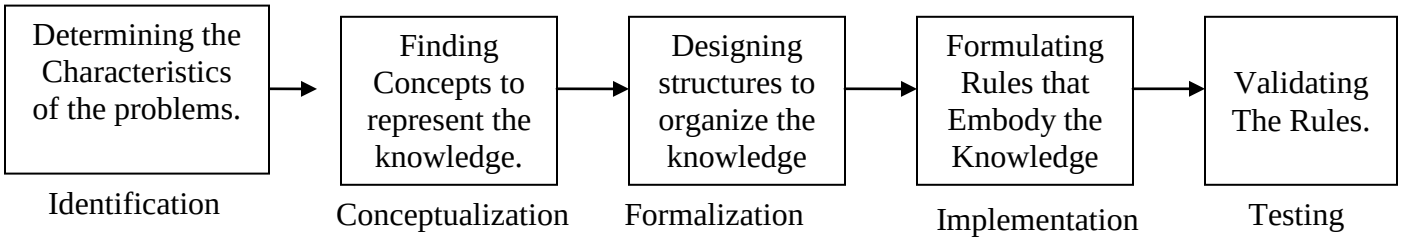


Fig: Different phases of expert system development

Identification:

Beside we can begin to develop an expert system, it is important that we describe, with as much precision as possible, the problem that the system is intended to solve. It is not enough simply to feel that the system would be helpful in certain situation; we must determine the exact nature of the problem and state the precise goals that indicate exactly how we expect the expert system to contribute to the solution.

Conceptualization:

Once we have formally identified the problem that an expert system is to solve, the next stage involves analyzing the problem further to ensure that its specifics, as well as it generalities, are understood.

In the conceptualization stage the knowledge engineer frequently creates a diagram of the problem to depict graphically the relationships between the objects and processes in the problem domain. It is often helpful at this stage to divide the problem into a series of sub-problems and to diagram both the relationships among the pieces of each sub-problem and the relationships among the various sub-problems.

Formalization:

In the preceding stages, no effort has been made to relate the domain problem to the artificial intelligence technology that may solve it. During the identification and the conceptualization stages, the focus is entirely on understanding the problem. Now, during the formalization stage, the problem is connected to its proposed solution, an expert system, by analyzing the relationships depicted in the conceptualization stage.

During formalization, it is important that the knowledge engineer be familiar with the following:

- The various techniques of knowledge representation and heuristic search used in expert systems.
- The expert system "tools" that can greatly expedite the development process. And
- Other expert systems that may solve similar problems and thus may be adequate to the problem at hand.

Implementation:

During the implementation stage, the formalized concepts are programmed onto the computer that has been chosen for system development, using the predetermined techniques and tools to implement a "first pass" prototype of the expert system.

Theoretically, if the methods of the previous stage have been followed with diligence and care, the implementation of the prototype should be as much an art as it is a science, because following all rules does not guarantee that the system will work the first time it is implemented. Many scientists actually consider the first prototype to be a "throw-away' system, useful for evaluating progress but hardly a usable expert system.

Testing:

Testing provides opportunities to identify the weakness in the structure and implementation of the system and to make the appropriate corrections. Depending on the types of problems encountered, the testing procedure may indicate that the system was

Natural Language Processing:

Communication:
Communication is the intentional exchange of information through production and perception of signs drawn from a shared system of conventional signs. Communication is important in partially observable environments where two or more agents works together to achieve a goal.

Speech Act:
Any activity done by the agent to produce language is called speech act. Speech act does not always mean producing sound; it may be any other activity for communication.

Formal Languages:
Formal language is defined as a (possibly finite) set of strings; each string is a concatenation of terminal symbols (sometimes called words). Formal languages have strict mathematical definitions. This is in contrast to the natural languages like Nepali, English, and Chinese etc.

Grammar:
A grammar is a finite set of rules that specifies a language. Formal languages always have official grammar but natural language have no official grammar. Linguistics are trying to discover the properties of natural language through scientific inquiry and trying to codify them. Toll to date no linguist is succeeding completely.

Both formal and natural languages have syntax and semantics to each valid string.
Syntax defines the rules for writing sentences.
e.g. in mathematics x+ y is valid sentence
but xy+ is not valid sentence.
Semantics defines the meaning of sentence.
E,g. x + y means sum of x and y.

In natural language it is also important to understand the pragmatics of the sentence: the actual meaning of the sentence that is spoken in a situation.
Strings are composed of substrings called phrases.

$S \rightarrow NP\ VP$

Here S (sentence) NP (Noun Phrase) and VP (Verb Phrase) are non-terminal symbols.

Component steps of Communication:

Intention:
S (speaker) wants to inform H (hearer) about the proposition P. e.g. Speaker wants to inform H that Wumpus is no longer alive.

Generation
Speaker (S) selects words W to express a proposition (P) in current situation(C) so that it becomes more meaningful to hearer.

Synthesis
Speaker makes physical realization W' of selected word W either by producing sound or by writing in a paper or by any other means.

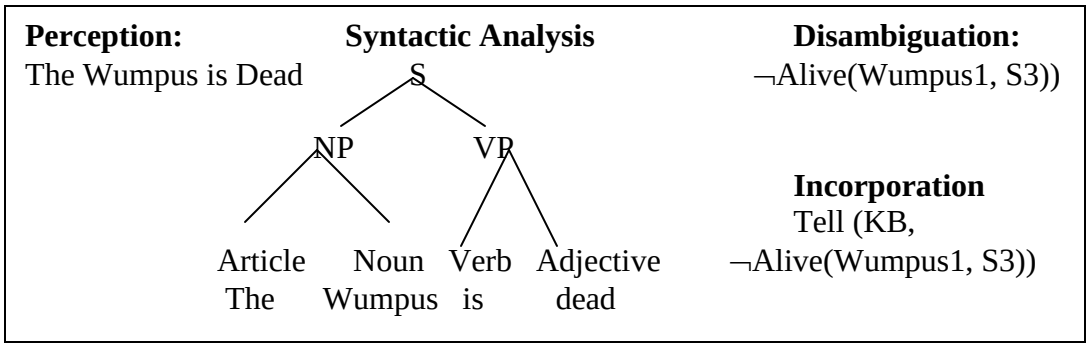
Perception
Hearer perceives physical realization as W2" and decodes it as the word W2. When the medium is speech, percept step is speech recognition, when it is printing; it is called optical character recognition.

Analysis
Hearer infers possible meanings P1; : : : Pn from W2. Analysis step can be divided into three parts: syntactic interpretation, Semantic Interpretation and pragmatic interpretation. e.g.

Speaker

Intention:	Generation	Synthesis
Know(H, ¬Alive(Wumpus, S3))	The Wumpus is Dead	[thaxwhampaxsihzdehd]

Hearer



Disambiguation:

Hearer infers intended meaning Pi that a speaker indented (where ideally Pi=P) Disambiguation process is heavily depends upon uncertain reasoning.

Incorporation
Hearer incorporates Pi into Knowledge base if it believes to Speaker. A totally naïve agents believes everything that it hears.

Grammar types

Regular:
The grammar in which left hand side contains a single non-terminal but right hand side contains a terminal optionally followed by any number of non-terminals is called regular grammar. i.e
nonterminal → terminal[nonterminal]
e.g.
S → **a**S
S → **b**

Context-free:
The grammar in which left hand side contains a single non-terminal but right hand side contains any sequence of terminals and non-terminals is called context-free grammar.
i.e
nonterminal → anything
e.g.
S → aSb

Context-sensitive: more nonterminals on left-hand side
ASB → AAaBB

Recursively enumerable: no constraints

Natural languages probably context-free, parsable in real time

Note:

- If sentence has more than one interpretation (parse tree), such sentence is called ambiguous sentence.
- The grammar which generates more than one parse tree of a sentence is called ambiguous grammar.
- Parsing/ syntax analysis is the mechanism of identifying whether a given sentence is grammatically correct or not.
- Semantic analysis is the mechanism of extracting meanings of the sentence.
- Pragmatic analysis is used to extract the contextual/ situational meaning of a sentence. It is used when a sentence has more than one semantic interpretation.
- Parse tree is a tree that is used in syntax analysis of parsing of a sentence.

CFG of English Language:

S → NP VP | S Conjunction S
NP → Pronoun
Name
Noun
Article Noun
Digit Digit
NP PP
VP → Verb
VP NP
VP Adjective
VP PP
VP Adverb
PP → Preposition NP

Neural Networks:

A neuron is a cell in brain whose principle function is the collection, Processing, and dissemination of electrical signals. Brains Information processing capacity comes from networks of such neurons. Due to this reason some earliest AI work aimed to create such artificial networks. (Other Names are Connectionism; Parallel distributed processing and neural computing).

What is a Neural Network?

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

Why use neural networks?

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage

Neural networks versus conventional computers

Neural networks take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do.

Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements(neurones) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable.

On the other hand, conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small unambiguous instructions. These instructions are then converted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.

Units of Neural Network:

Nodes(units):

Nodes represent a cell of neural network.

Links:

Links are directed arrows that show propagation of information from one node to another node.

Activation:

Activations are inputs to or outputs from a unit.

Weight:

Each link has weight associated with it which determines strength and sign of the connection.

Activation function:

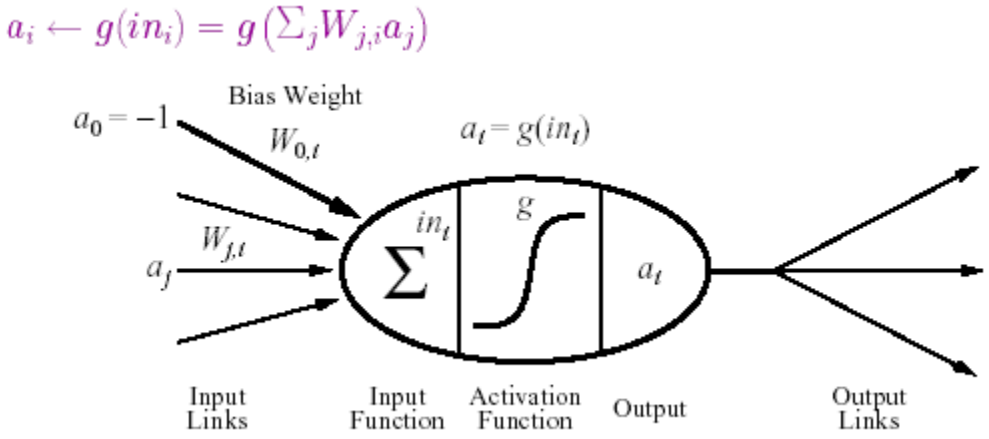
A function which is used to derive output activation from the input activations to a given node is called activation function.

Bias Weight:

Bias weight is used to set the threshold for a unit. Unit is activated when the weighted sum of real inputs exceeds the bias weight.

Simple Model of Neural Network

A simple mathematical model of neuron is devised by McCulloch and Pit is given in the figure given below:



It fires when a linear combination of its inputs exceeds some threshold.

A neural network is composed of nodes (units) connected by directed links A link from unit j to i serve to propagate the activation a_j from j to i. Each link has some numeric weight $W_{j,i}$ associated with it, which determines strength and sign of connection.

Each unit first computes a weighted sum of it's inputs:

$$in_i = \sum_{J=0}^{n} W_{j,i}\ a_j$$

Then it applies activation function g to this sum to derive the output:

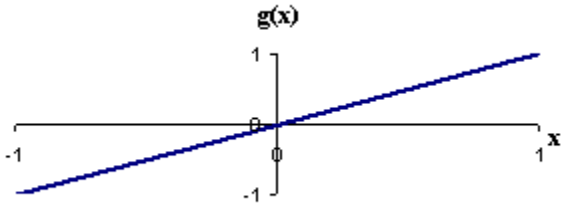
$$a_i = g(in_i) = g\left(\sum_{J=0}^{n} W_{j,i}\ a_j\right)$$

Here, a_j output activation from unit j and $W_{j,i}$ is the weight on the link j to this node. Activation function typically falls into one of three categories:

- Linear
- Threshold (*Heaviside function)*
- Sigmoid
- Sign

For **linear activation functions**, the output activity is proportional to the total weighted output.

g(x) = k x + c, where k and x are constant



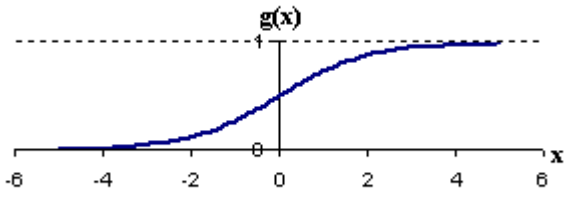
For **threshold activation functions**, the output are set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

g(x) = 1 if x>= k

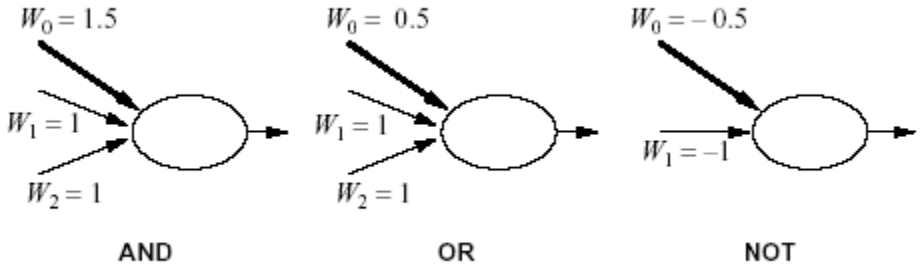
= 0 if x < k

For **sigmoid activation functions**, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units. It has the advantage of differentiable.

$g(x) = 1/ (1 + e^{-x})$



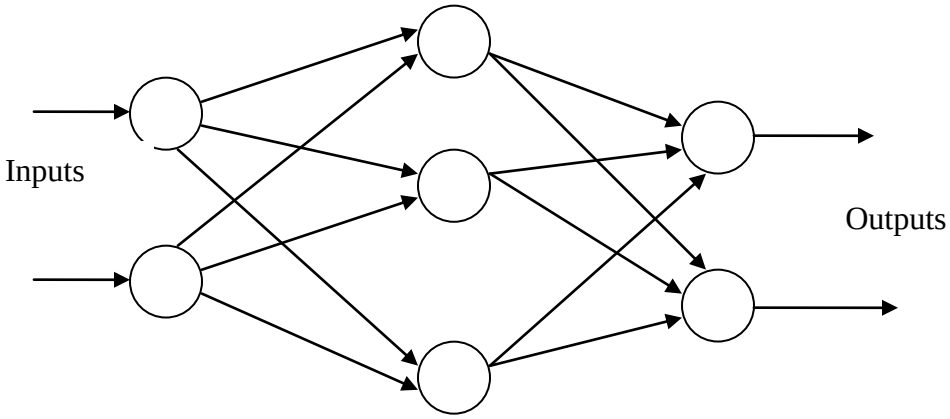
Realizing logic gates by using Neurons:



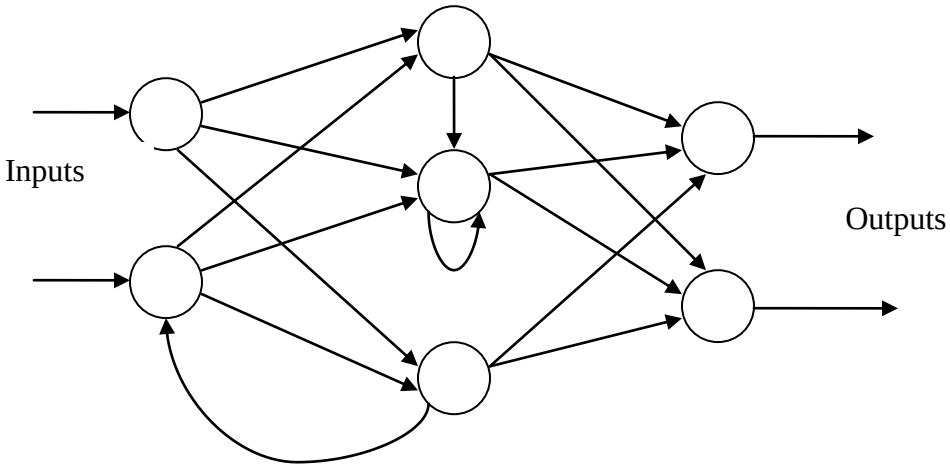
Network structures:

Feed-forward networks:

Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down.

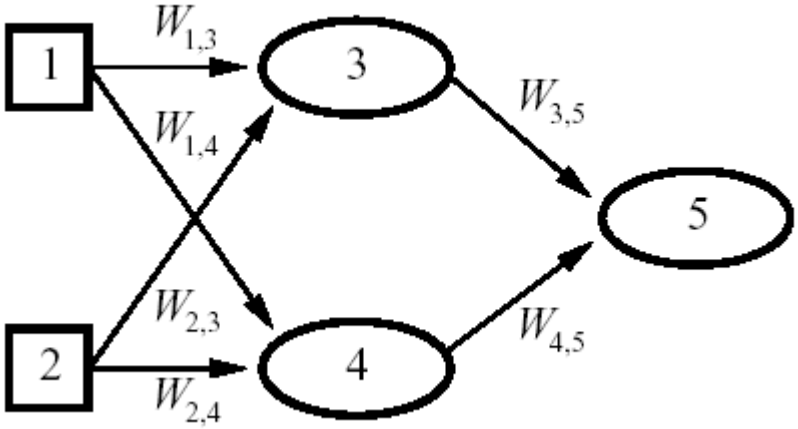


Feedback networks (Recurrent networks:)



Feedback networks (figure 1) can have signals traveling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent.

Feed-forward example



Here;

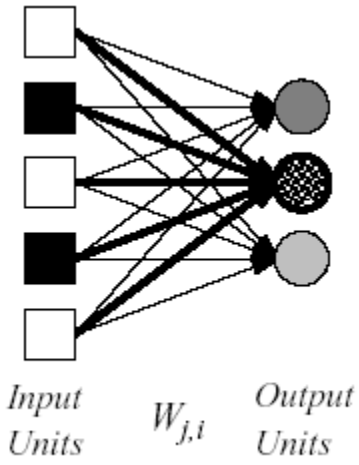
$a5 = g(W_{3;5}\ a_3 + W_{4;5}\ a_4)$

$= g(W_{3;5}\ g(W_{1;3}\ a_1 + W_{2;3}\ a_2) + W4;5\ g(W_{1;4}\ a_1 + W_{2;4}\ a_2)$

Types of Feed Forward Neural Network:

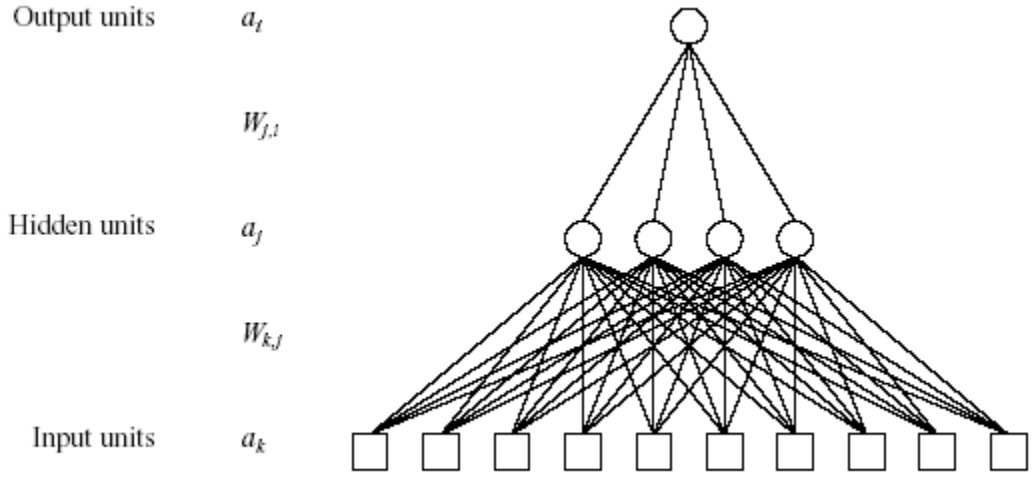
Single-layer neural networks (perceptrons)

A neural network in which all the inputs connected directly to the outputs is called a single-layer neural network, or a perceptron network. Since each output unit is independent of the others each weight affects only one of the outputs.



Multilayer neural networks (perceptrons)

The neural network which contains input layers, output layers and some hidden layers also is called multilayer neural network. The advantage of adding hidden layers is that it enlarges the space of hypothesis. Layers of the network are normally fully connected.



Once the number of layers, and number of units in each layer, has been selected, training is used to set the network's weights and thresholds so as to minimize the prediction error made by the network

Training is the process of adjusting weights and threshold to produce the desired result for different set of data.

Learning in Neural Networks:

Learning: One of the powerful features of neural networks is learning. **Learning in neural networks is carried out by adjusting the connection weights among neurons**. It is similar to a biological nervous system in which learning is carried out by changing synapses connection strengths, among cells.

The operation of a neural network is determined by the values of the interconnection weights. There is no algorithm that determines how the weights should be assigned in order to solve specific problems. Hence, the weights are determined by a learning process

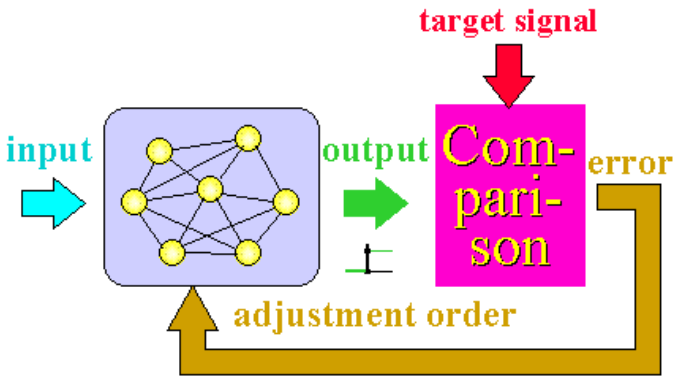
Learning may be classified into two categories:

(1) Supervised Learning
(2) Unsupervised Learning

Supervised Learning:

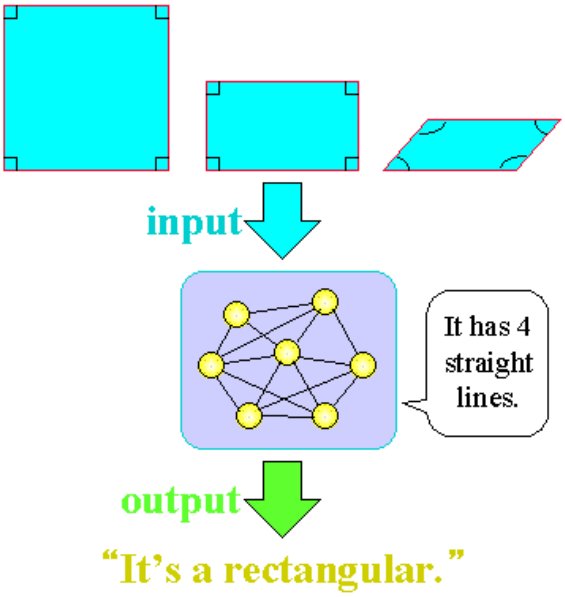
In supervised learning, the network is presented with inputs together with the target (teacher signal) outputs. Then, the neural network tries to produce an output as close as possible to the target signal by adjusting the values of internal weights. The most common supervised learning method is the "error correction method".

Error correction method is used for networks which their neurons have discrete output functions. Neural networks are trained with this method in order to reduce the error (difference between the network's output and the desired output) to zero.



Unsupervised Learning:

In unsupervised learning, there is no teacher (target signal) from outside and the network adjusts its weights in response to only the input patterns. A typical example of unsupervised learning is **Hebbian learning.**



Consider a machine (or living organism) which receives some sequence of inputs x1, x2, x3, . . ., where xt is the sensory input at time t. In supervised learning the machine is given a sequence of input & a sequence of desired outputs y1, y2, . . . , and the goal of the machine is to learn to produce the correct output given a new input. While, in unsupervised learning the machine simply receives inputs x1, x2, . . ., but obtains neither supervised target outputs, nor rewards from its environment. It may seem somewhat mysterious to imagine what the machine could possibly learn given that it doesn't get any feedback from its environment. However, it is possible to develop of formal framework for unsupervised learning based on the notion that the machine's goal is to build representations of the input that can be used for decision making, predicting future inputs, efficiently communicating the inputs to another machine, etc. In a sense, unsupervised learning can be thought of as finding patterns in the data above and beyond what would be considered pure unstructured noise.

Hebbian Learning:

The oldest and most famous of all learning rules is Hebb's postulate of learning:

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B is increased"

From the point of view of artificial neurons and artificial neural networks, Hebb's principle can be described as a method of determining how to alter the weights between model neurons. **The weight between two neurons increases if the two neurons activate simultaneously—and reduces if they activate separately.** Nodes that tend to be either both positive or both negative at the same time have strong positive weights, while those that tend to be opposite have strong negative weights.

Hebb's Algorithm:

Step 0: initialize all weights to 0

Step 1: Given a training input, s, with its target output, t, set the activations of the input units: $x_i = s_i$

Step 2: Set the activation of the output unit to the target value: $y = t$

Step 3: Adjust the weights: $w_i(new) = w_i(old) + x_i y$

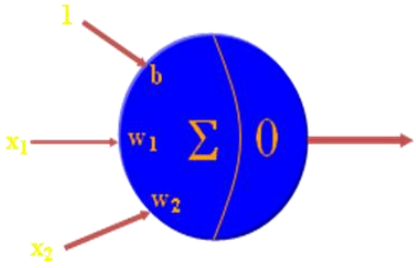
Step 4: Adjust the bias (just like the weights): $b(new) = b(old) + y$

Example:

PROBLEM: Construct a Hebb Net which performs like an AND function, that is, only when both features are "active" will the data be in the target class.

TRAINING SET (with the bias input always at 1):

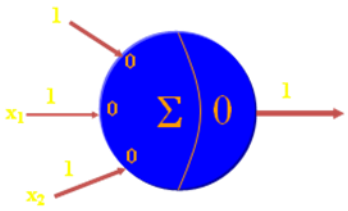
x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



Training-First Input:

- Initialize the weights to 0

Present the first input:
(1 1 1) with a target of 1



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

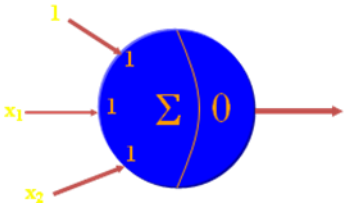
$= 0 + 1 = 1$

$w_2(new) = w_2(old) + x_2 t$

$= 0 + 1 = 1$

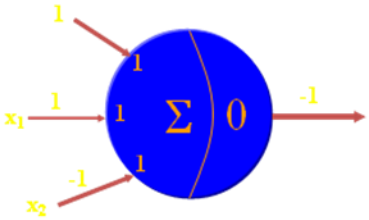
$b(new) = b(old) + t$

$= 0 + 1 = 1$



Training- Second Input:

- **Present the second input:**
  (1 -1 1) with a target of -1



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

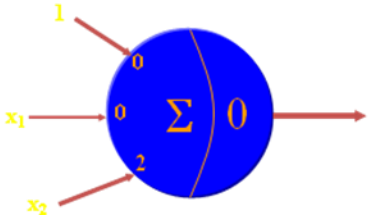
$= 1 + 1(-1) = 0$

$w_2(new) = w_2(old) + x_2 t$

$= 1 + (-1)(-1) = 2$

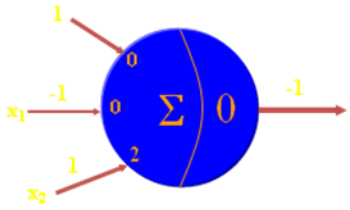
$b(new) = b(old) + t$

$= 1 + (-1) = 0$



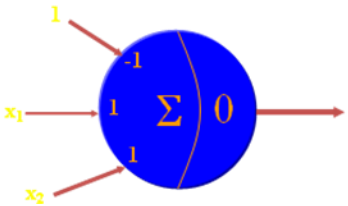
Training- Third Input:

- **Present the third input: (-1 1 1) with a target of -1**



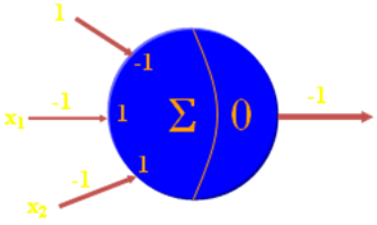
Update the weights:

$$w_1(new) = w_1(old) + x_1 t$$
$$= 0 + (-1)(-1) = 1$$
$$w_2(new) = w_2(old) + x_2 t$$
$$= 2 + 1(-1) = 1$$
$$b(new) = b(old) + t$$
$$= 0 + (-1) = -1$$



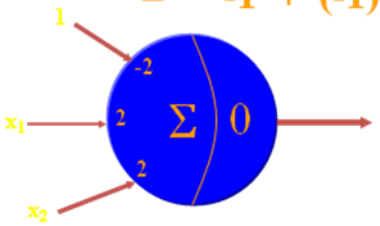
Training- Fourth Input:

- **Present the fourth input: (-1 -1 1) with a target of -1**



Update the weights:

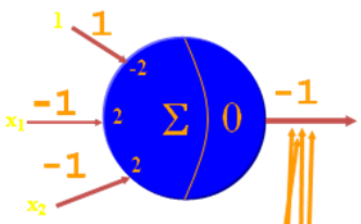
$$w_1(new) = w_1(old) + x_1 t$$
$$= 1 + (-1)(-1) = 2$$
$$w_2(new) = w_2(old) + x_2 t$$
$$= 1 + (-1)(-1) = 1$$
$$b(new) = b(old) + t$$
$$= -1 + (-1) = -2$$



Final Neuron:

- **This neuron works:**

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



1*2 + 1*2 + 1*(-2) = 2 > 0
(-1)*2 + 1*2 +1*(-2) = -2 < 0
1*2 + (-1)*2 + 1*(-2) = -2 < 0
(-1)*2 + (-1)*2 + 1 *(-2) = -6 <0

Perceptron Learning Theory:

The term "Perceptrons" was coined by Frank RosenBlatt in 1962 and is used to describe the connection of simple neurons into networks. These networks are simplified versions of the real nervous system where some properties are exagerrated and others are ignored. For the moment we will concentrate on Single Layer Perceptrons.

So how can we achieve learning in our model neuron? We need to train them so they can do things that are useful. To do this we must allow the neuron to learn from its mistakes. There is in fact a learning paradigm that achieves this, it is known as supervised learning and works in the following manner.

i. set the weight and thresholds of the neuron to random values.
ii. present an input.
iii. caclulate the output of the neuron.
iv. alter the weights to reinforce correct decisions and discourage wrong decisions, hence reducing the error. So for the network to learn we shall increase the weights on the active inputs when we want the output to be active, and to decrease them when we want the output to be inactive.
v. Now present the next input and repeat steps iii. - v.

Perceptron Learning Algorithm:

The algorithm for Perceptron Learning is based on the supervised learning procedure discussed previously.

Algorithm:

i. Initialize weights and threshold.

Set $w_i(t)$, $(0 <= i <= n)$, to be the weight i at time t, and $ø$ to be the threshold value in the output node. Set w_0 to be $-ø$, the bias, and x_0 to be always 1.

Set $w_i(0)$ to small random values, thus initializing the weights and threshold.

ii. Present input and desired output

Present input $x_0, x_1, x_2, ..., x_n$ and desired output $d(t)$

iii. Calculate the actual output

$y(t) = g\,[w_0(t)x_0(t) + w_1(t)x_1(t) + ... + w_n(t)x_n(t)]$

iv. Adapts weights

$w_i(t+1) = w_i(t) + \alpha[d(t) - y(t)]x_i(t)$, where $0 <= \alpha <= 1$ (learning rate) is a positive gain function that controls the adaption rate.

Steps iii. and iv. are repeated until the iteration error is less than a user-specified error threshold or a predetermined number of iterations have been completed.

Please note that the weights only change if an error is made and hence this is only when learning shall occur.

Delta Rule:

The **delta rule** is a gradient descent learning rule for updating the weights of the artificial neurons in a single-layer perceptron. It is a special case of the more general backpropagation algorithm. For a neuron j with activation function $g(x)$ the delta rule for j's ith weight $w_{ji}$is given by

$$\Delta w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i,$$

where α is a small constant called *learning rate*, $g(x)$ is the neuron's activation function, $t_j$is the target output, h_j is the weighted sum of the neuron's inputs, y_j is the actual output, and x_i is the ith input. It holds $h_j = \sum x_i w_{ji}$and $y_j = g(h_j)$.

The delta rule is commonly stated in simplified form for a perceptron with a linear activation function as

$$\Delta w_{ji} = \alpha(t_j - y_j)x_i$$

Backpropagation

It is a supervised learning method, and is an implementation of the **Delta rule**. It requires a teacher that knows, or can calculate, the desired output for any given input. It is most useful for feed-forward networks (networks that have no feedback, or simply, that have no connections that loop). The term is an abbreviation for "backwards propagation of errors". Backpropagation requires that the activation function used by the artificial neurons (or "nodes") is differentiable.

As the algorithm's name implies, the errors (and therefore the learning) propagate backwards from the output nodes to the inner nodes. So technically speaking, backpropagation is used to calculate the gradient of the error of the network with respect to the network's modifiable weights. This gradient is almost always then used in a simple *stochastic gradient descent algorithm, is a general optimization algorithm, but is typically used to fit the parameters of a machine learning model, to find weights that minimize the error*. Often the term "backpropagation" is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient descent. Backpropagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

Backpropagation networks are necessarily multilayer perceptrons (usually with one input, one hidden, and one output layer). In order for the hidden layer to serve any useful function, multilayer networks must have non-linear activation functions for the multiple layers: a multilayer network using only linear activation functions is equivalent to some single layer, linear network.

Summary of the backpropagation technique:

1. Present a training sample to the neural network.
2. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a *scaling factor*, how much lower or higher the output must be adjusted to match the desired output. This is the local error.
4. Adjust the weights of each neuron to lower the local error.

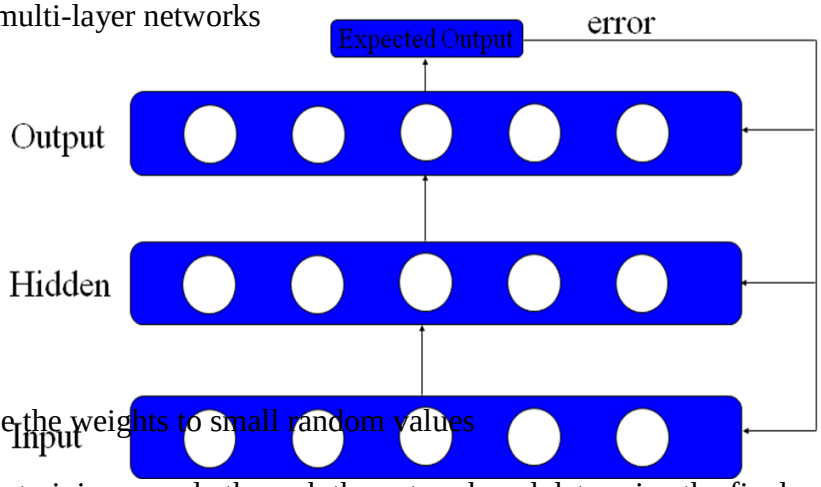
5. Assign "blame" for the local error to neurons at the previous level, giving greater responsibility to neurons connected by stronger weights.
6. Repeat from step 3 on the neurons at the previous level, using each one's "blame" as its error.

Characteristics:

- A multi-layered perceptron has three distinctive characteristics
  - The network contains one or more layers of hidden neurons
  - The network exhibits a high degree of connectivity
  - Each neuron has a smooth (differentiable everywhere) nonlinear activation function, the most common is the sigmoidal nonlinearity:

$$y_j = \frac{1}{1 + e^{s_j}}$$

- The backpropagation algorithm provides a computational efficient method for training multi-layer networks



Algorithm:

Step 0: Initialize the weights to small random values

Step 1: Feed the training sample through the network and determine the final output

Step 2: Compute the error for each output unit, for unit k it is:

$$\delta_k = (t_k - y_k) f'(y\_in_k)$$

Actual output

Required output

Derivative of f

Step 3: Calculate the weight correction term for each output unit, for unit k it is:

$$\Delta w_{jk} = \alpha \delta_k z_j$$

Hidden layer signal

A small constant

Step 4: Propagate the delta terms (errors) back through the weights of the hidden units where the delta input for the j^{th} hidden unit is:

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

The delta term for j^{th} hidden unit is: $\delta_j = \delta\_in_j f'(z\_in_j)$

Step 5: Calculate the weight correction term for the hidden units: $\Delta w_{ij} = \alpha \delta_j x_i$

Step 6: Update the weights: $w_{ik}(new) = w_{ik}(old) + \Delta w_{ik}$

Step 7: Test for stopping (maximum cylces, small changes, etc)

Note: There are a number of options in the design of a backprop system;

- Initial weights – best to set the initial weights (and all other free parameters) to random numbers inside a small range of values (say –0.5 to 0.5)
- Number of cycles – tend to be quite large for backprop systems
- Number of neurons in the hidden layer – as few as possible

Machine Vision:

Machine vision (MV) is the application of computer vision to industry and manufacturing. Whereas computer vision is the general discipline of making computers see (understand what is perceived visually), machine vision, being an engineering discipline, is interested in digital input/output devices and computer networks to control other manufacturing equipment such as robotic arms and equipment to eject defective products.

Machine vision is the ability of a computer to "see." A machine-vision system employs one or more video cameras, analog-to-digital conversion (ADC), and digital signal processing (DSP). The resulting data goes to a computer or robot controller. Machine vision is similar in complexity to voice recognition . The machine vision systems use video cameras, robots or other devices, and computers to visually analyze an operation or activity. Typical uses include automated inspection, optical character recognition and other non-contact applications.

Two important specifications in any vision system are the sensitivity and the resolution. Sensitivity is the ability of a machine to see in dim light, or to detect weak impulses at invisible wavelengths. Resolution is the extent to which a machine can differentiate between objects. In general, the better the resolution, the more confined the field of vision. Sensitivity and resolution are interdependent. All other factors held constant,

increasing the sensitivity reduces the resolution, and improving the resolution reduces the sensitivity.

One of the most common applications of Machine Vision is the inspection of manufactured goods such as semiconductor chips, automobiles, food and pharmaceuticals. Just as human inspectors working on assembly lines visually inspect parts to judge the quality of workmanship, so machine vision systems use digital cameras, smart cameras and image processing software to perform similar inspections.

Machine vision systems have two primary hardware elements: the camera, which serves as the eyes of the system, and a computer video analyser. The recent rapid acceleration in the development of machine vision for industrial applications can be attributed to research in the areas of computer technologies. The first step in vision analysis is the conversion of analog pixel intensity data into digital format for processing. Next, an appropriate computer algorithm is employed to understand the image data and provide appropriate analysis or action.

Machine vision encompasses computer science, optics, mechanical engineering, and industrial automation. Unlike computer vision which is mainly focused on machine-based image processing, machine vision integrates image capture systems with digital input/output devices and computer networks to control manufacturing equipment such as robotic arms. Manufacturers favour machine vision systems for visual inspections that require high-speed, high-magnification, 24-hour operation, and/or repeatability of measurements.

A typical machine vision system will consist of most of the following components:

- One or more digital or analogue cameras (black-and-white or colour) with suitable optics for acquiring images, such as lenses to focus the desired field of view onto the image sensor and suitable, often very specialized, light sources
- Input/Output hardware (e.g. digital I/O) or communication links (e.g. network connection or RS-232) to report results
- A synchronizing sensor for part detection (often an optical or magnetic sensor) to trigger image acquisition and processing and some form of actuators to sort, route or reject defective parts
- A program to process images and detect relevant features.

The aim of a machine vision inspection system is typically to check the compliance of a test piece with certain requirements, such as prescribed dimensions, serial numbers, presence of components, etc. The complete task can frequently be subdivided into independent stages, each checking a specific criterion. These individual checks typically run according to the following model:

1. Image Capture

2. Image Preprocessing
3. Definition of one or more (manual) regions of interest
4. Segmentation of the objects
5. Computation of object features
6. Decision as to the correctness of the segmented objects

Naturally, capturing an image, possible several for moving processes, is a pre-requisite for analysing a scene. In many cases these images are not suited for immediate examination and require pre-processing to change certain sizing specific structures etc. In most cases it is at least approximately known which image areas have to be analysed, i.e. the location of a mark to be read or a component to be verified. These are called Regions of Interest (ROIs) (sometimes Area of Interest or AOIs). Of course, such a region can also comprise the entire image if required.

Machine vision is used in various industrial and medical applications. Examples include:

- Electronic component analysis
- Signature identification
- Optical character recognition
- Handwriting recognition
- Object recognition
- Pattern recognition
- Materials inspection
- Currency inspection
- Medical image analysis

Computer Vision:

Computer vision is the science and technology of machines that see, where *see* in this case means that the machine is able to extract information from an image that is necessary to solve some task. As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner.

As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems. Examples of applications of computer vision include systems for:

- Controlling processes (e.g., an industrial robot or an autonomous vehicle).
- Detecting events (e.g., for visual surveillance or people counting).
- Organizing information (e.g., for indexing databases of images and image sequences).

- Modeling objects or environments (e.g., industrial inspection, medical image analysis or topographical modeling).
- Interaction (e.g., as the input to a device for computer-human interaction).

Computer vision is closely related to the study of biological vision. The field of biological vision studies and models the physiological processes behind visual perception in humans and other animals. Computer vision, on the other hand, studies and describes the processes implemented in software and hardware behind artificial vision systems. Interdisciplinary exchange between biological and computer vision has proven fruitful for both fields.

Computer vision is, in some ways, the inverse of computer graphics. While computer graphics produces image data from 3D models, computer vision often produces 3D models from image data. There is also a trend towards a combination of the two disciplines, e.g., as explored in augmented reality.

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, learning, indexing, motion estimation, and image restoration.