[Unit 7: Application of AI]
Artificial Intelligence (CSC 355)

Bal Krishna Subedi

Central Department of Computer Science & Information Technology

Tribhuvan University

Expert Systems:

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

A computer system that simulates the **decision- making process** of a human expert in a specific 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.

An expert system is an "intelligent" program that solves problems in a narrow problem area by using high-quality, specific knowledge rather than an algorithm.

Block Diagram

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.

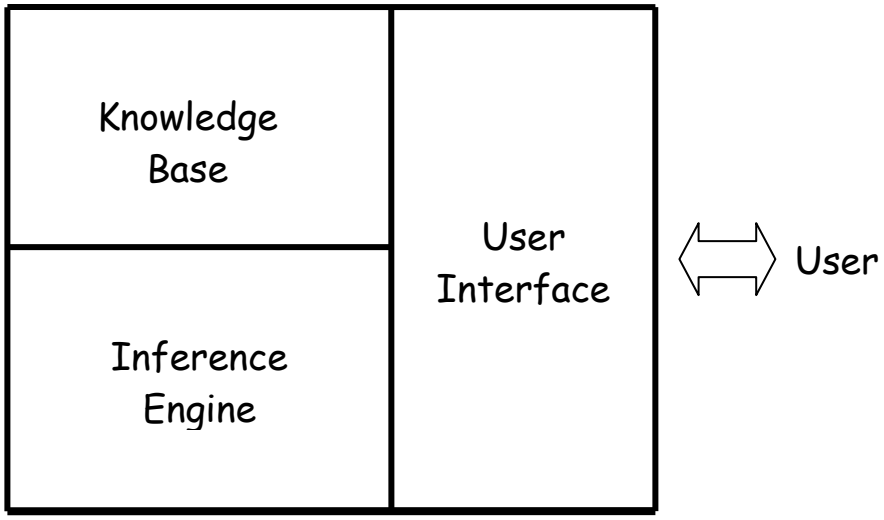


Fig: Block Diagram of expert system

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.

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.

Beside these three components, there is a Working Memory - a data structure which stores information about a specific run. It holds current facts and knowledge.

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.

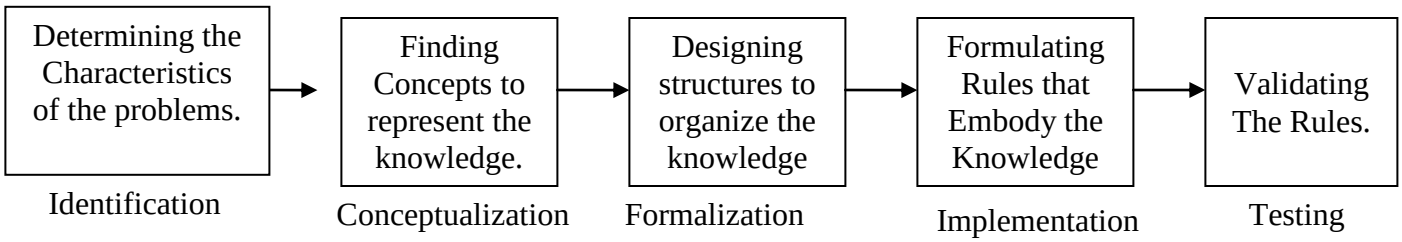


Fig: Different phases of expert system development

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

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.

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.

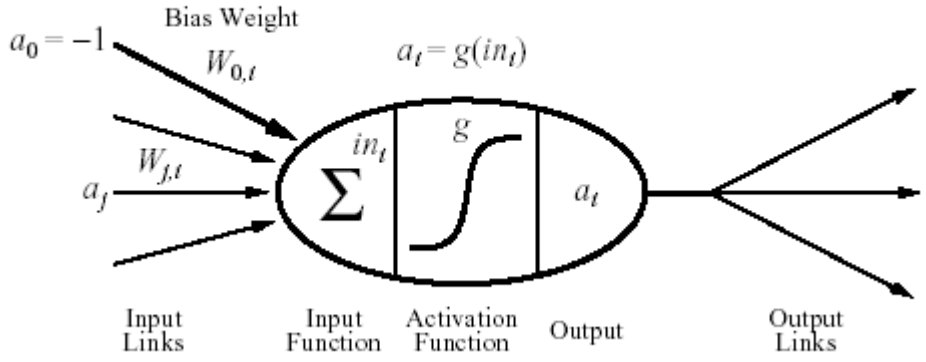
Activation function:
A function which is used to derive output activation from the input activations to a given node is called activation function.

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:

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



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:

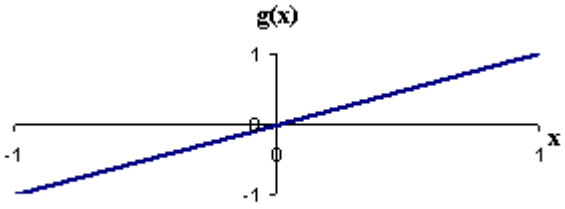
$$a_i = g(\ in_i) = g(\ \sum_{J=0}^{n} W_{j,i}\ a_j)$$

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



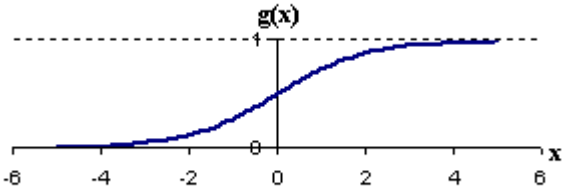
For **threshold activation functions**, the output are set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

g(x) = 1 if x>= k

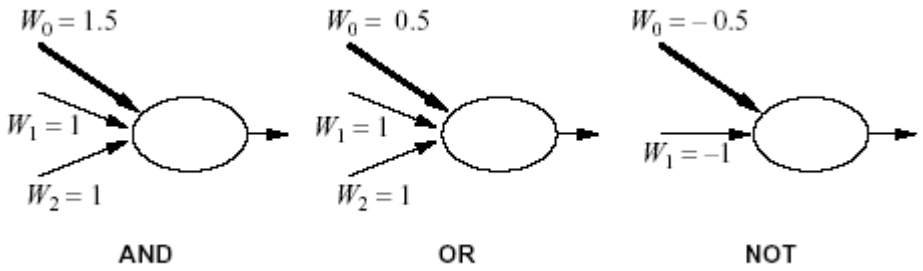
= 0 if x < k

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.

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



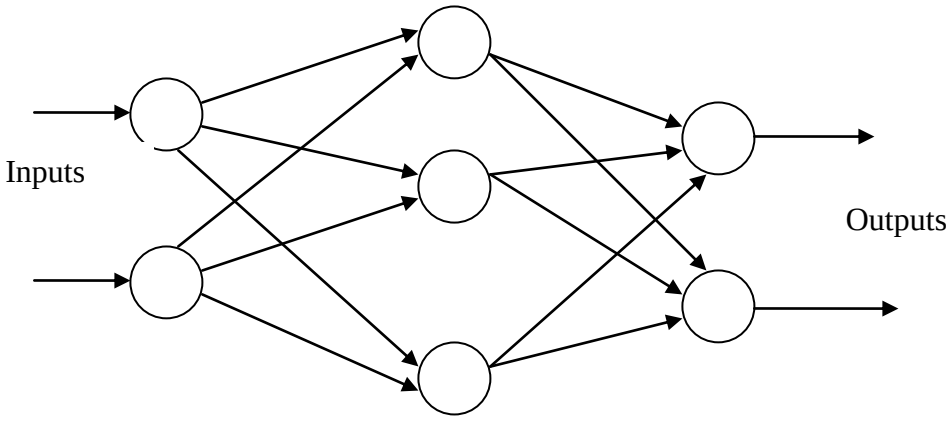
Realizing logic gates by using Neurons:



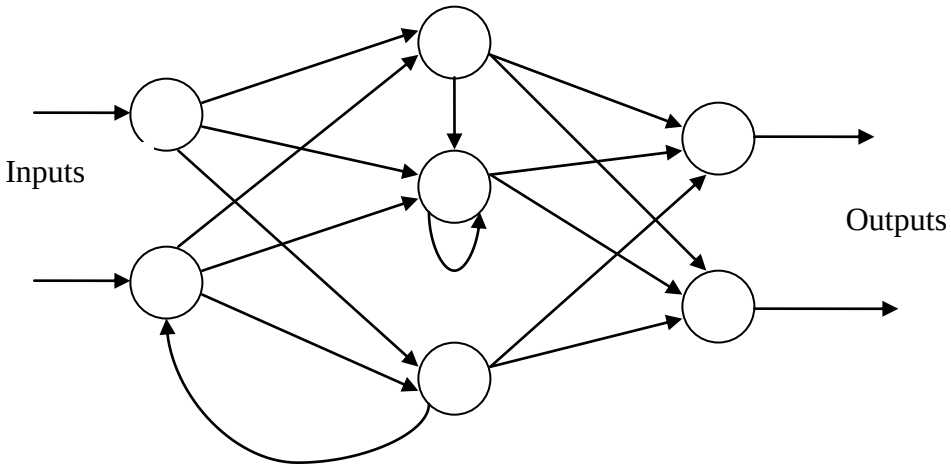
Network structures:

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.

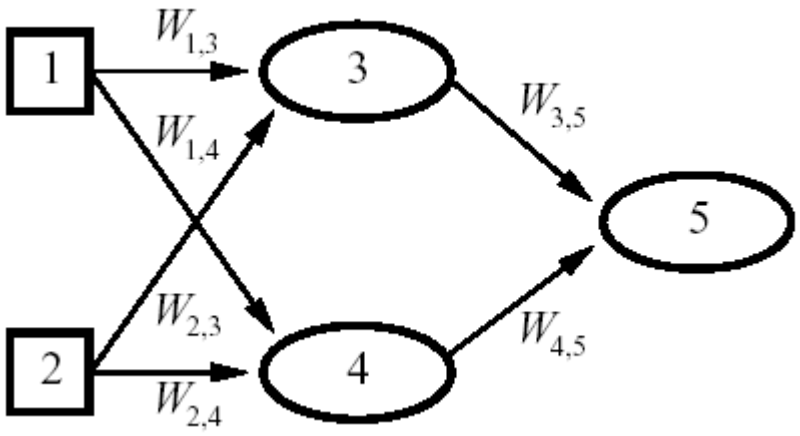


Feedback networks (Recurrent networks:)



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;

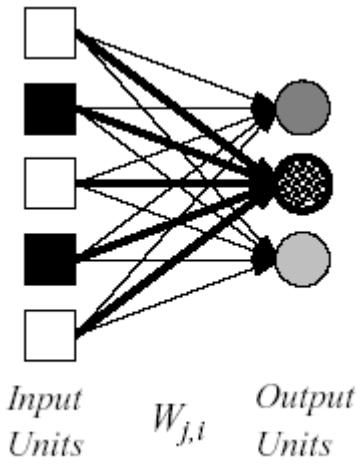
$a5 = g(W_{3;5}\ a_3 + W_{4;5}\ a_4)$

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

Types of Feed Forward Neural Network:

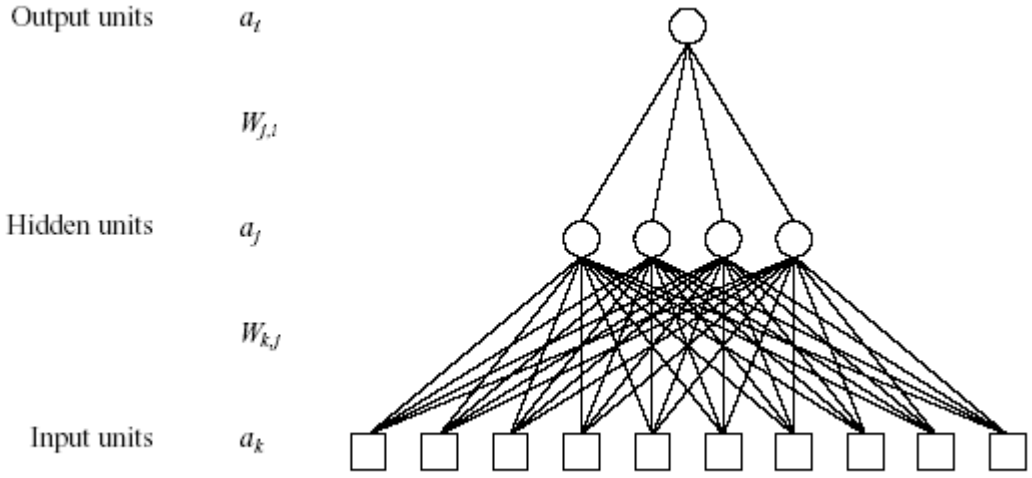
Single-layer neural networks (perceptrons)

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

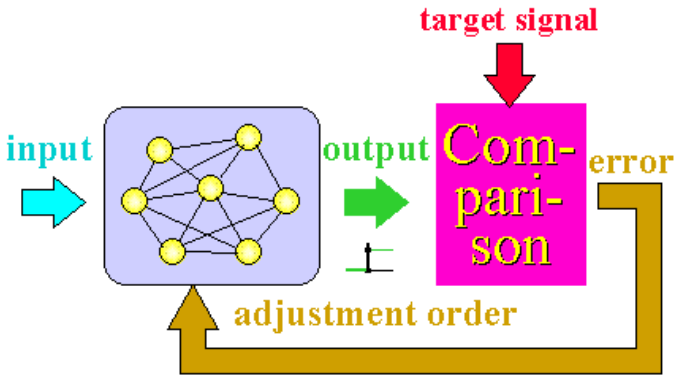
Learning may be classified into two categories:

(1) Supervised Learning
(2) Unsupervised Learning

Supervised Learning:

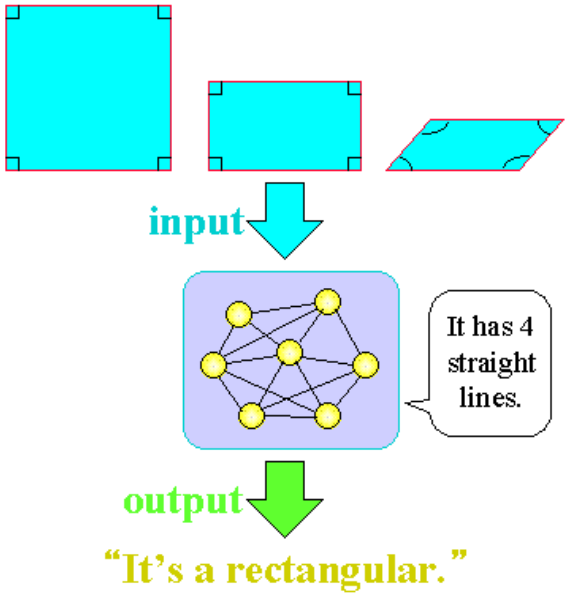
In supervised learning, the network is presented with inputs together with the target (teacher signal) outputs. Then, the neural network tries to produce an output as close as possible to the target signal by adjusting the values of internal weights. The most common supervised learning method is the "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: $w_i(new) = w_i(old) + x_i y$

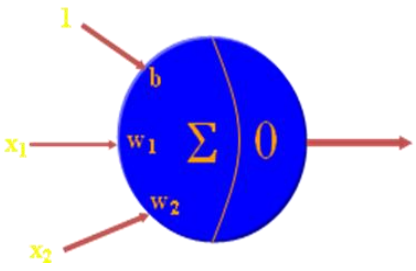
Step 4: Adjust the bias (just like the weights): $b(new) = b(old) + y$

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:

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

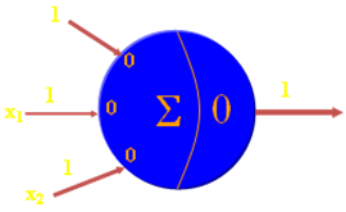
$= 0 + 1 = 1$

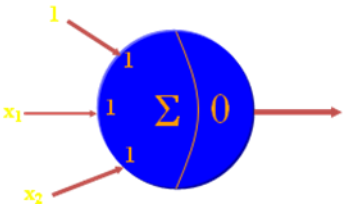
$w_2(new) = w_2(old) + x_2 t$

$= 0 + 1 = 1$

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

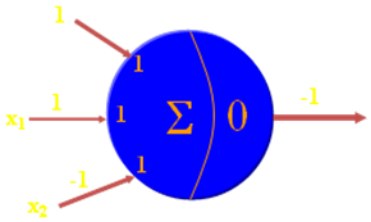
$= 0 + 1 = 1$





Training- Second Input:

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



Update the weights:

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

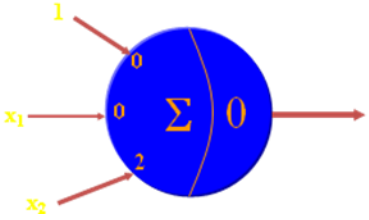
$= 1 + 1(-1) = 0$

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

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

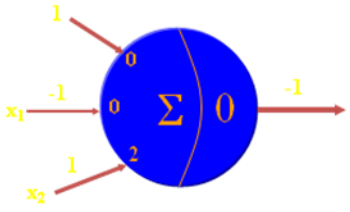
$b(new) = b(old) + t$

$= 1 + (-1) = 0$



Training- Third Input:

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



Update the weights:

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

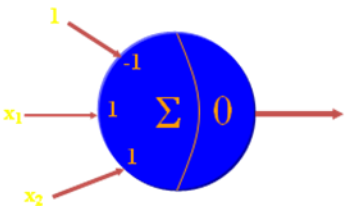
$= 0 + (-1)(-1) = 1$

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

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

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

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



Training- Fourth Input:

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

Update the weights:

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

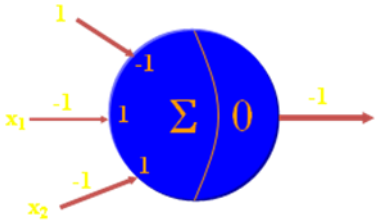
$= 1 + (-1)(-1) = 2$

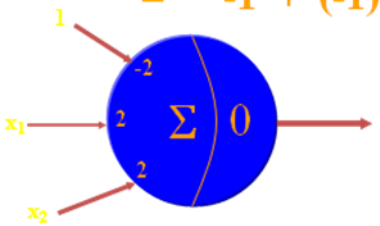
$w_2(new) = w_2(old) + x_2 t$

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

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

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

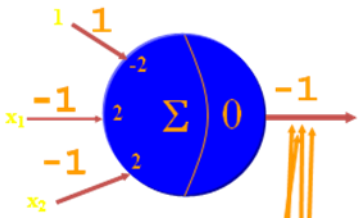




Final Neuron:

- **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 <= α <= 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:

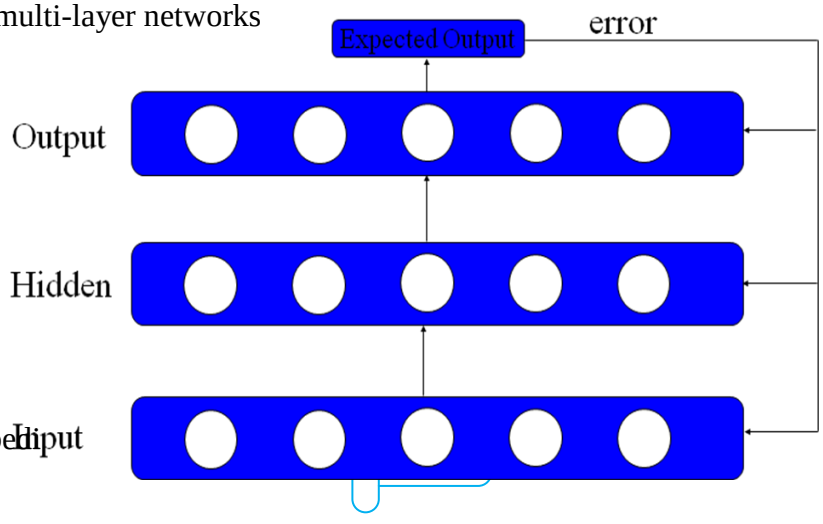
1. Present a training sample to the neural network.
2. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a *scaling factor*, how much lower or higher the output must be adjusted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

Actual output

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

Required output Derivative of f

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

Hidden layer signal

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

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

Natural Language Processing:

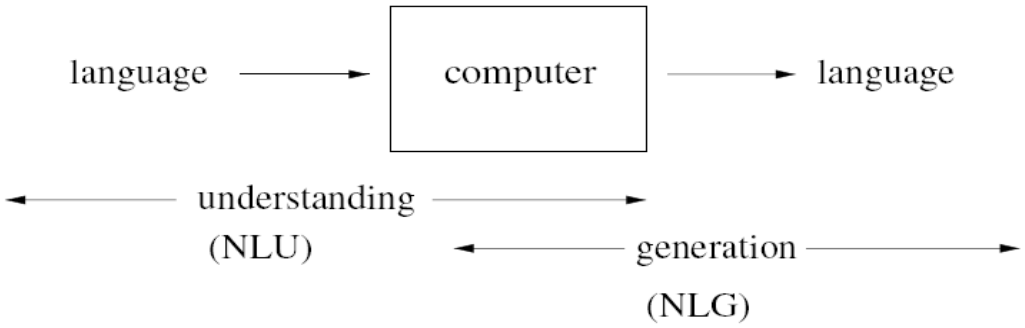
Perception and communication are essential components of intelligent behavior. They provide the ability to effectively interact with our environment. Humans perceive and communicate through their five basic senses of sight, hearing, touch, smell and taste, and their ability to generate meaningful utterances. Developing programs that understand a natural language is a difficult problem. Natural languages are large. They contain infinity of different sentences. No matter how many sentences a person has heard or seen, new ones can always be produced. Also, there is much ambiguity in a natural language. Many words have several meanings and sentences can have different meanings in different contexts. This makes the creation of programs that understand a natural language, one of the most challenging tasks in AI.

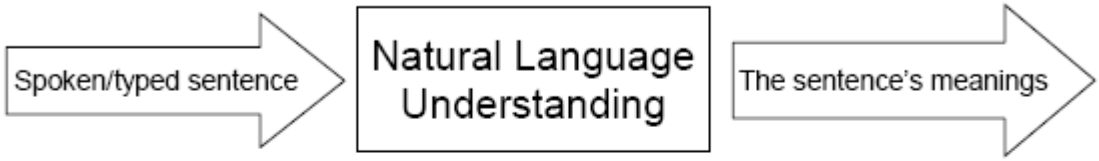
Developing programs to understand natural language is important in AI because a natural form of communication with systems is essential for user acceptance. AI programs must be able to communicate with their human counterparts in a natural way, and natural language is one of the most important mediums for that purpose. So, Natural Language Processing (NLP) is the field that deals with the computer processing of natural languages, mainly evolved by people working in the field of Artificial Intelligence.

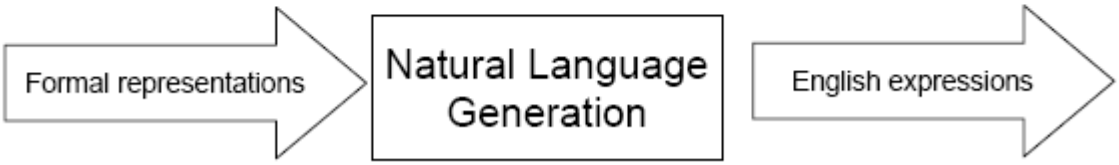
Natural Language Processing (NLP), is the attempt to extract the fuller meaning representation from the free text. Natural language processing is a technology which involves converting spoken or written human language into a form which can be processed by computers, and vice versa. Some of the better-known applications of NLP include:

- **Voice recognition software** which translates speech into input for word processors or other applications;
- **Text-to-speech synthesizers** which read text aloud for users such as the hearing-impaired;
- **Grammar and style checkers** which analyze text in an attempt to highlight errors of grammar or usage;
- **Machine translation systems** which automatically render a document such as a web page in another language.

computers using natural language as input and/or output







Natural Language Generation:

"Natural Language Generation (NLG), also referred to as text generation, is a subfield of natural language processing (NLP; which includes computational linguistics)

Natural Language Generation (NLG) is the natural language processing task of generating natural language from a machine representation system such as a knowledge base or a logical form.

In a sense, one can say that an NLG system is like a translator that converts a computer based representation into a natural language representation. However, the methods to produce the final language are very different from those of a compiler due to the inherent expressivity of natural languages.

NLG may be viewed as the opposite of natural language understanding. The difference can be put this way: whereas in natural language understanding the system needs to disambiguate the input sentence to produce the machine representation language, in NLG the system needs to make decisions about how to put a concept into words.

The different types of generation techniques can be classified into four main categories:

- Canned text systems constitute the simplest approach for single-sentence and multi-sentence text generation. They are trivial to create, but very inflexible.
- Template systems, the next level of sophistication, rely on the application of pre-defined templates or schemas and are able to support flexible alterations. The template approach is used mainly for multi-sentence generation, particularly in applications whose texts are fairly regular in structure.
- Phrase-based systems employ what can be seen as generalized templates. In such systems, a phrasal pattern is first selected to match the top level of the input, and then each part of the pattern is recursively expanded into a more specific phrasal pattern that matches some subportion of the input. At the sentence level, the phrases resemble phrase structure grammar rules and at the discourse level they play the role of text plans.
- Feature-based systems, which are as yet restricted to single-sentence generation, represent each possible minimal alternative of expression by a single feature. Accordingly, each sentence is specified by a unique set of features. In this framework, generation consists in the incremental collection of features appropriate for each portion of the input. Feature collection itself can either be based on unification or on the traversal of a feature selection network. The expressive power of the approach is very high since any distinction in language can be added to the system as a feature. Sophisticated feature-based generators, however, require very complex input and make it difficult to maintain feature interrelationships and control feature selection.

Many natural language generation systems follow a hybrid approach by combining components that utilize different techniques.

Natural Language Understanding:

Developing programs that understand a natural language is a difficult problem. Natural languages are large. They contain infinity of different sentences. No matter how many sentences a person has heard or seen, new ones can always be produced. Also, there is much ambiguity in a natural language. Many words have several meaning such as can, bear, fly, bank etc, and sentences have different meanings in different contexts.

Example :- A *can* of juice. I *can* do it.

This makes the creation of programs that understand a natural language, one of the most challenging tasks in AI. Understanding the language is not only the transmission of words. It also requires inference about the speakers' goal, knowledge as well as the context of the

interaction. We say a program understand natural language if it behaves by taking the correct or acceptable action in response to the input. A word functions in a sentence as a part of speech. Parts of the speech for the English language are nouns, pronouns, verbs, adjectives, adverbs, prepositions, conjunctions and interjections. Three major issues involved in understanding language.

- A large amount of human knowledge is assumed.
- Language is pattern based, phonemes are components of the words and words make phrases and sentences. Phonemes, words and sentences order are not random.
- Language acts are the product of agents (human or machine).

Levels of knowledge used in Language Understanding

A language understanding knowledge must have considerable knowledge about the structures of the language including what the words are and how they combine into phrases and sentences. It must also know the meanings of the words and how they contribute to the meanings of the sentence and to the context within which they are being used. The component forms of knowledge needed for an understanding of natural languages are sometimes classified according to the following levels.

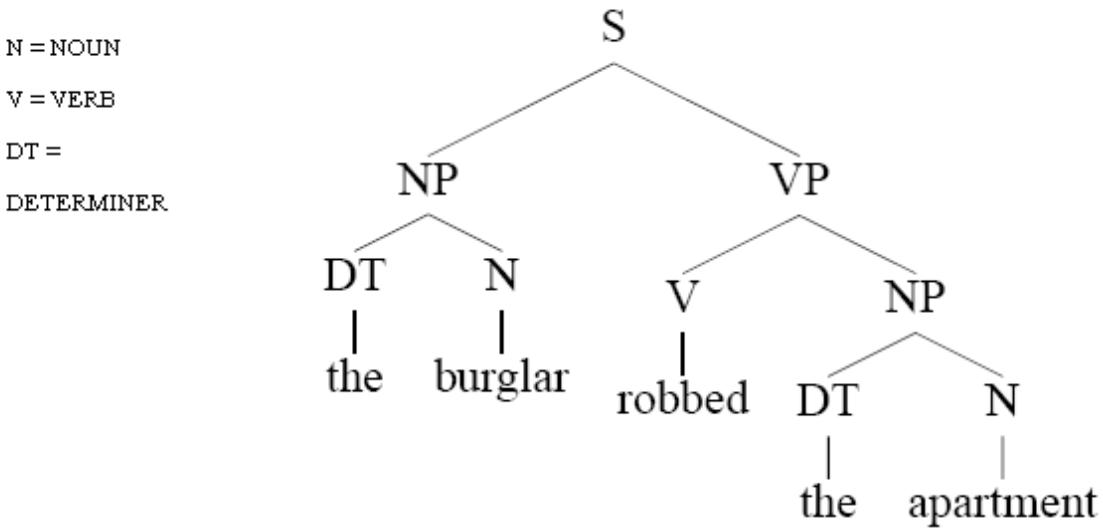
- **Phonological**
  - Relates sound to the words we recognize. A phoneme is the smallest unit of the sound. Phones are aggregated to the words.
- **Morphological**
  - This is lexical knowledge which relates to the word construction from basic units called morphemes. A morpheme is the smallest unit of meaning. Eg:- *friend* + ly = friendly
- **Syntactic**
  - This knowledge relates to how words are put together or structure red together to form grammatically correct sentences in the language.
- **Semantic**
  - This knowledge is concerned with the meanings of words and phrases and how they combine to form sentence meaning.
- **Pragmatic**

- This is high – level knowledge which relates to the use of sentences in different contexts and how the context affects the meaning of the sentence.

- **World**
  - Includes the knowledge of the physical world, the world of human social interaction, and the roles of goals and intentions in communication.

Basic Parsing Techniques

Before the meaning of a sentence can be determined, the meanings of its constituent parts must be established. This requires knowledge of the structure of the sentence, the meaning of the individual words and how the words modify each other. The process of determining the syntactical structure of a sentence is known as parsing. Parsing is the process of analyzing a sentence by taking it apart word – by – word and determining its structure from its constituent parts and sub parts. The structure of a sentence can be represented with a syntactic tree. When given an input string, the lexical parts or terms (root words), must first be identified by type and then the role they play in a sentence must be determined. These parts can be combined successively into larger units until a complete tree has been computed.



Noun Phrases (NP): "the burglar", "the apartment"

Verb Phrases (VP): "robbed the apartment"

Sentences (S): "the burglar robbed the apartment"

To determine the meaning of a word, a parser must have access to a lexicon. When the parser selects the word from the input stream, it locates the world in the lexicon and obtains the word's possible functions and features, including the semantic information.

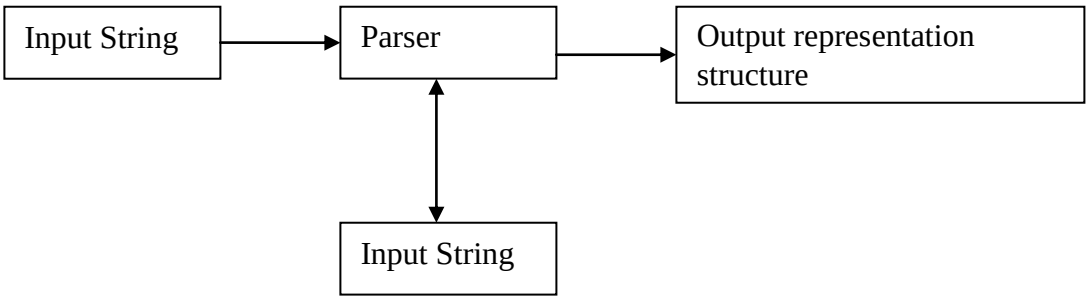


Figure :- Parsing an input to create an output structure

Lexeme (Lexicon) & word forms:

The distinction between these two senses of "word" is arguably the most important one in morphology. The first sense of "word", the one in which *dog* and *dogs* are "the same

word", is called a lexeme. The second sense is called *word form*. We thus say that *dog* and *dogs* are different forms of the same lexeme. *Dog* and *dog catcher*, on the other hand, are different lexemes, as they refer to two different kinds of entities. The form of a word that is chosen conventionally to represent the canonical form of a word is called a lemma, or citation form.

A lexicon defines the words of a language that a system knows about. This is includes common words and words that are specific to the domain of the application. Entries include meanings for each word and its syntactic and morphological behavior.

Morphology:

Morphology is the identification, analysis and description of the structure of words (words as units in the lexicon are the subject matter of lexicology). While words are generally accepted as being (with clitics) the smallest units of syntax, it is clear that in most (if not all) languages, words can be related to other words by rules. For example, English speakers recognize that the words *dog*, *dogs*, and *dog catcher* are closely related. English speakers recognize these relations from their tacit knowledge of the rules of word formation in English. They infer intuitively that *dog* is to *dogs* as *cat* is to *cats*; similarly, *dog* is to *dog catcher* as *dish* is to *dishwasher* (in one sense). The rules understood by the speaker reflect specific patterns (or regularities) in the way words are formed from smaller units and how those smaller units interact in speech. In this way, morphology is the branch of linguistics that studies patterns of word formation within and across languages, and attempts to formulate rules that model the knowledge of the speakers of those languages.

Morphological analysis is the process of recognizing the suffixes and prefixes that have been attached to a word.

We do this by having a table of affixes and trying to match the input as:
prefixes +root + suffixes.

- For example: adjective + ly -> adverb. E.g.: [Friend + ly]=friendly
- We may not get a unique result.
- "-s, -es" can be either a plural noun or a 3ps verb
- "-d, -ed" can be either a past tense or a perfect participle

Morphological Information:

- Transform part of speech
  - *green, greenness (adjective, noun)*
  - *walk, walker (verb, noun)*
- Change features of nouns
  - *boat, boats (singular, plural)*
- Bill slept , Bill's bed

  - (subjective case, possessive case)

- Change features of verbs
  - Aspect
    - *I walk. I am walking.* (present, progressive)
  - Tense
    - *I walked. I will walk. I had been walking. (past, future, past progressive)*
  - Number and person
    - *I walk. They walk. (first person singular, third person plural)*

Syntactic Analysis:

Syntactic analysis takes an input sentence and produces a representation of its grammatical structure. A grammar describes the valid parts of speech of a language and how to combine them into phrases. The grammar of English is nearly context free.

A computer grammar specifies which sentences are in a language and their parse trees. A parse tree is a hierarchical structure that shows how the grammar applies to the input. Each level of the tree corresponds to the application of one grammar rule.

It is the starting point for working out the meaning of the whole sentence. Consider the following two sentences.

1. "The dog ate the bone."
2. "The bone was eaten by the dog."

Understanding the structure (via the syntax rules) of the sentences help us work out that it's the bone that gets eaten and not the dog. Syntactic analysis determines possible grouping of words in a sentence. In other cases there may be many possible groupings of words. Consider the sentence "John saw Mary with a telescope". Two different readings based on the groupings.

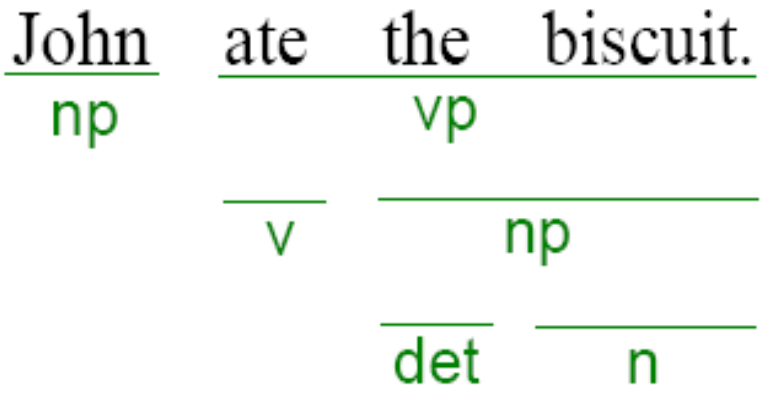
1. John saw (Mary with a telescope).
2. John (saw Mary with a telescope).

A sentence is syntactically ambiguous if there are two or more possible groupings. Syntactic analysis helps determining the meaning of a sentence by working out possible word structure. Rules of syntax are specified by writing a *grammar* for the language. A parser will check if a sentence is correct according to the grammar. It returns a representation (parse tree) of the sentence's structure. A grammar specifies allowable sentence structures in terms of basic categories such as noun and verbs. A given grammar, however, is unlikely to cover all possible grammatical sentences. Parsing sentences is to help determining their meanings, not just to check that they are correct. Suppose we want a grammar that recognizes sentences like the following.

John ate the biscuit.
The lion ate the zebra.
The lion kissed John

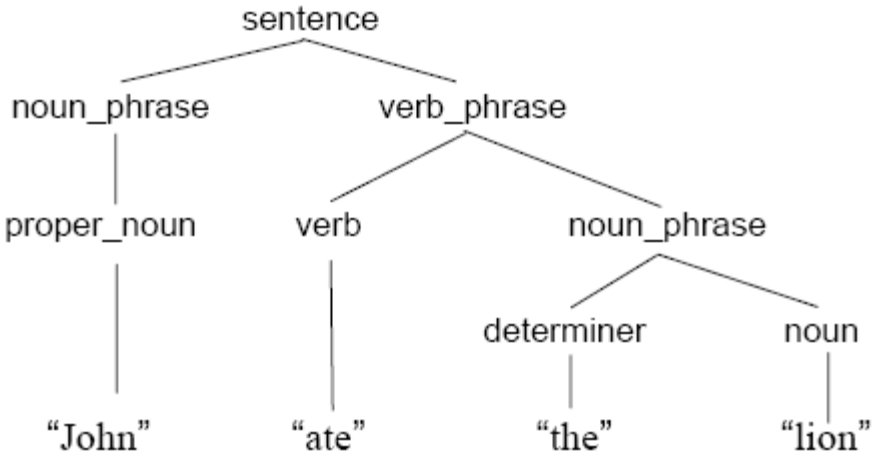
But reject incorrect sentences such as
Ate John biscuit the.
Zebra the lion the ate.
Biscuit lion kissed.

A simple grammar that deals with this is given below
sentence --> noun_phase, verb phrase.
noun_phrase --> proper_noun.
noun_phrase --> determiner, noun.
verb_phrase --> verb, noun_phrase.
proper_noun --> [mary].
proper_noun --> [john].
noun --> [zebra].
noun --> [biscuit].
verb --> [ate].
verb --> [kissed].
determiner --> [the].



Incorrect sentences like “biscuit lion kissed” will be excluded by the grammar.

- A ***parse trees*** illustrates the syntactic structure of the sentence.



Semantic Analysis:

Semantic analysis is a process of converting the syntactic representations into a meaning representation.

This involves the following tasks:

- Word sense determination
- Sentence level analysis
- Knowledge representation

- Word sense

Words have different meanings in different contexts.

Mary had a bat in her office.

- bat = `a baseball thing'
- bat = `a flying mammal'

- Sentence level analysis

Once the words are understood, the sentence must be assigned some meaning

I saw an astronomer with a telescope.

- Knowledge Representation

Understanding language requires lots of knowledge**.**

- Using predicate logic, for example, one can represent sentences like

"John likes Mary" likes(john,mary)

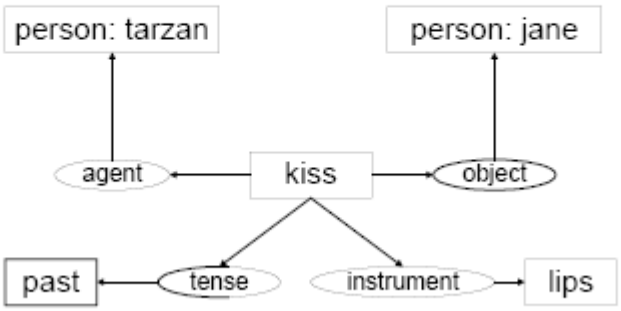
"The man likes Mary" man(m1)∧likes(m1,mary)

"A man likes Mary" ∃X(man(X)∧likes(X,mary)

"A tall bearded man likes Mary"

∃X(man(X)∧tall(X)∧bearded(X)∧likes(X,mary)

- Using semantic net, one can represent sentence "Tarzan kissed Jane" as



Parameters in Natural Language Processing:

- Auditory Inputs
- Segmentation
- Syntax Structure
- Semantic Structure
- Pragmatic Analysis

- Auditory Inputs:

Three of our five senses – sight, hearing and touch – are used as *major inputs*. These are usually referred to as the visual, auditory and tactile inputs respectively. They are sometimes called input channels; however, as previously mentioned, the term "channel" is used in various ways, so I will avoid it.

In the fashion of video devices, audio devices are used to either capture or create sound. In some cases, an audio output device can be used as an input device, in order to capture produced sound.

- Microphone
- MIDI keyboard or other digital musical instrument

- Segmentation:

Text segmentation is the process of dividing written text into meaningful units, such as words, sentences, or topics. The term applies both to mental processes used by humans when reading text, and to artificial processes implemented in computers, which are the subject of natural language processing. The problem is non-trivial, because while some written languages have explicit word boundary markers, such as the word spaces of written English and the distinctive initial, medial and final letter shapes of Arabic, such signals are sometimes ambiguous and not present in all written languages.

Word segmentation is the problem of dividing a string of written language into its component words. In English and many other languages using some form of the Latin alphabet, the space is a good approximation of a word delimiter. (Some examples where the space character alone may not be sufficient include contractions like *can't* for *can not*.)

However the equivalent to this character is not found in all written scripts, and without it word segmentation is a difficult problem. Languages which do not have a trivial word segmentation process include Chinese, Japanese, where sentences but not words are delimited, and Thai, where phrases and sentences but not words are delimited.

In some writing systems however, such as the Ge'ez script used for Amharic and Tigrinya among other languages, words are explicitly delimited (at least historically) with a non-whitespace character.

Word splitting is the process of parsing concatenated text (i.e. text that contains no spaces or other word separators) to infer where word breaks exist.

Sentence segmentation **is the problem of dividing a string of written language into its component sentences. In English and some other languages, using punctuation, particularly the full stop character is a reasonable approximation. However, even in English this problem is not trivial due to the use of the full stop character for abbreviations, which may or may not also terminate a sentence. For example *Mr.* is not its own sentence in "*Mr. Smith went to the shops in Jones Street.*" When processing plain text, tables of abbreviations that contain periods can help prevent incorrect assignment of sentence boundaries. As with word segmentation, not all written languages contain punctuation characters which are useful for approximating sentence boundaries.**

Other segmentation problems: Processes may be required to segment text into segments besides words, including morphemes (a task usually called morphological analysis), paragraphs, topics or discourse turns.

A document may contain multiple topics, and the task of computerized text segmentation may be to discover these topics automatically and segment the text accordingly. The topic boundaries may be apparent from section titles and paragraphs. In other cases one needs to use techniques similar to those used in document classification. Many different approaches have been tried.

- Syntax Structure:

Same concept as in the syntactic analysis above

- Semantic Structure:

Same concept as in the semantic analysis above

- Pragmatic Analysis:

This is high level knowledge which relates to the use of sentences in different contexts and how the context affects the meaning of the sentences. It is the study of the ways in which language is used and its effect on the listener. Pragmatic comprises aspects of meaning that depend upon the context or upon facts about real world.

Pragmatics – Handling Pronouns

Handling pronouns such as "he", "she" and "it" is not always straight forward. Let us see the following paragraph.

"John buys a new telescope. He sees Mary in the distance. He gets out his telescope. He looks at her through it".

Here, "*her*" refers to *Mary* who was not mentioned at all in the previous sentences. John's telescope was referred to as "*a new telescope*", "*his telescope*" and "*it*".

Let us see one more example

"When is the next flight to Sydney?"
"Does it have any seat left?"

Here, "*it*", refers to a particular flight to Sydney, not Sydney itself.

Pragmatics – Ambiguity in Language

A sentence may have more than one structure such as

"I saw an astronomer with a telescope."

This English sentence has a prepositional phrase "*with a telescope*" which may be attached with either with verb to make phrase "*saw something with telescope*" or to object noun phrase to make phrase "*a astronomer with a telescope*". If we do first, then it can be interpreted as "*I saw an astronomer who is having a telescope*", and if we do second, it can be interpreted as "*Using a telescope I saw an astronomer*".

Now, to remove such ambiguity, one possible idea is that we have to consider the context. If the knowledge base (KB) can prove that whether the telescope is with astronomer or not, then the problem is solved.

Next approach is that; let us consider the real scenario where the human beings communicate. If A says the same sentence "*I saw an astronomer with a telescope*." To B, then in practical, it is more probable that, B (listener) realizes that "*A has seen astronomer who is having a telescope*". It is because, normally, the word "*telescope*" belongs to "*astronomer*", so it is obvious that B realizes so.

If A has says that "*I saw a lady with a telescope*." In this case, B realizes that "*A has seen the lady using a telescope*", because the word "*telescope*" has not any practical relationship with "*lady*" like "*astronomer*".

So, we may be able to remove such ambiguity, by defining a data structure, which can efficiently handle such scenario. This idea may not 100% correct but seemed more probable.

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.