APPLICATION OF AI

Unit 6

Contents

Contents

Neural Networks

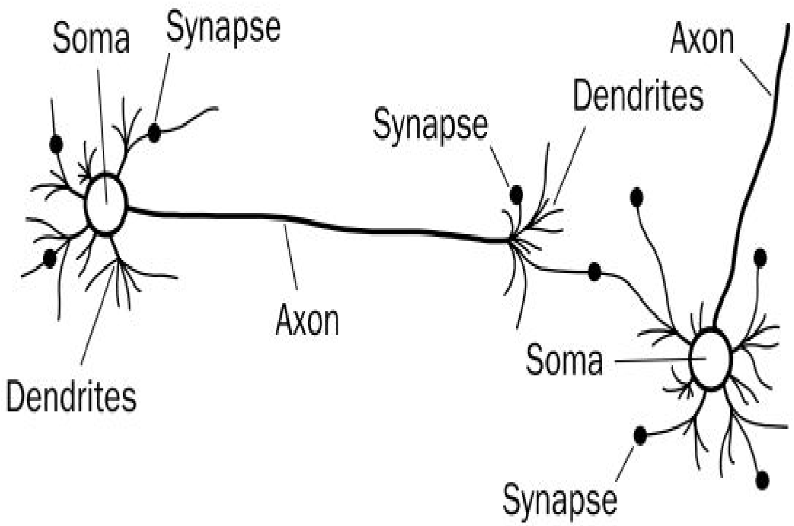
- ‘The computer hasn’t proved anything yet,’ angry Garry Kasparov, the world chess champion, said after his defeat in New York in May 1997. ‘If we were playing a real competitive match, I would tear down Deep Blue into pieces.’

Network Structure

- A neural network can be defined as a model of reasoning based on the human brain. The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called **neurons**
- The human brain incorporates nearly 10 billion neurons and 60 trillion connections, **synapses**, between them. By using multiple neurons simultaneously, the brain can perform its functions much faster than the fastest computers in existence today.

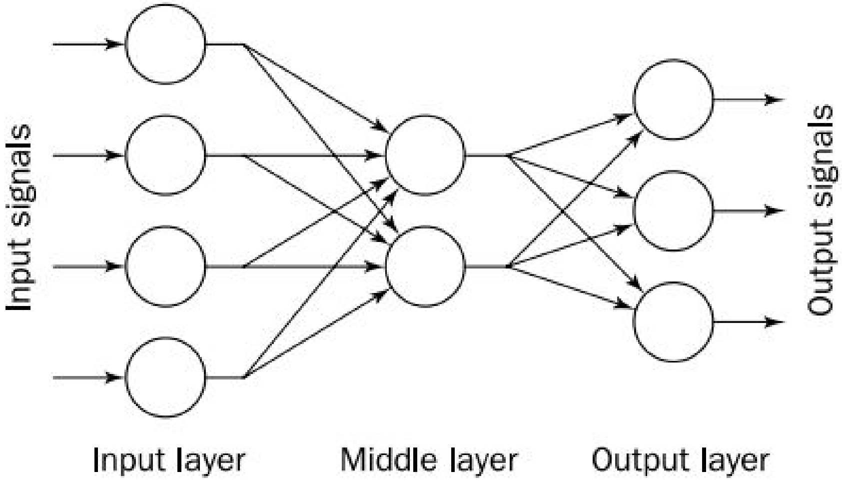
Network Structure

- Each neuron has a very simple structure, but an army of such elements constitutes a tremendous processing power
- **Neuron:** fundamental functional unit of all nervous system tissue
- **Soma:** cell body, contain nucleus
- **Dendrites:** a number of fibres, input
- **Axon:** single long fibre with many branches, output
- **Synapse:** junction of dendrites and axon, each neuron form synapse with 10 to 100000 other neurons



Network Structure

- Consists of a number of very simple and highly interconnected processors called neurons
- The neurons are connected by weighted links passing signals from one neuron to another.
- The output signal is transmitted through the neuron's outgoing connection. The outgoing connection splits into a number of branches that transmit the same signal. The outgoing ranches terminate at the incoming connections of other neurons in the network



The Neuron as a simple computing element: Diagram of a neuron

- The neuron computes the weighted sum of the input signals and compares the result with a **threshold value**, θ. If the net input is less than the threshold, the neuron output is −1. But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains a value +1.
- The neuron uses the following transfer or **activation function**
- $X = \sum_{i=1}^{n} x_i w_i$ $\qquad Y = \begin{Bmatrix} +1\ if\ X \geq \theta \\ -1\ if\ X \leq \theta \end{Bmatrix}$
- This type of activation function is called a **sign function**

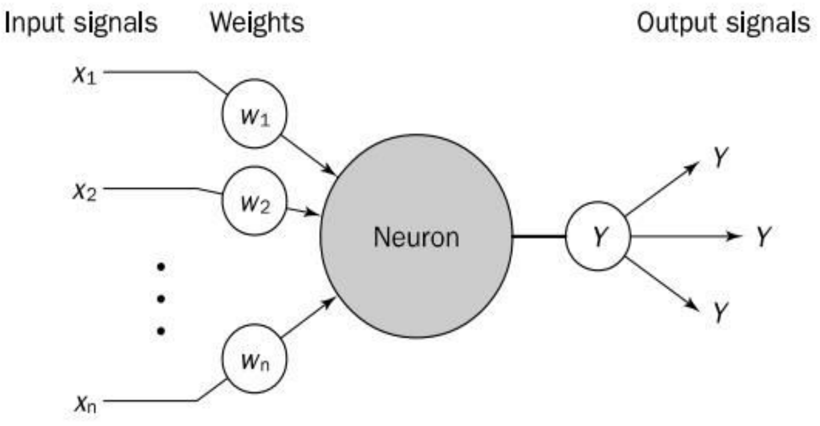
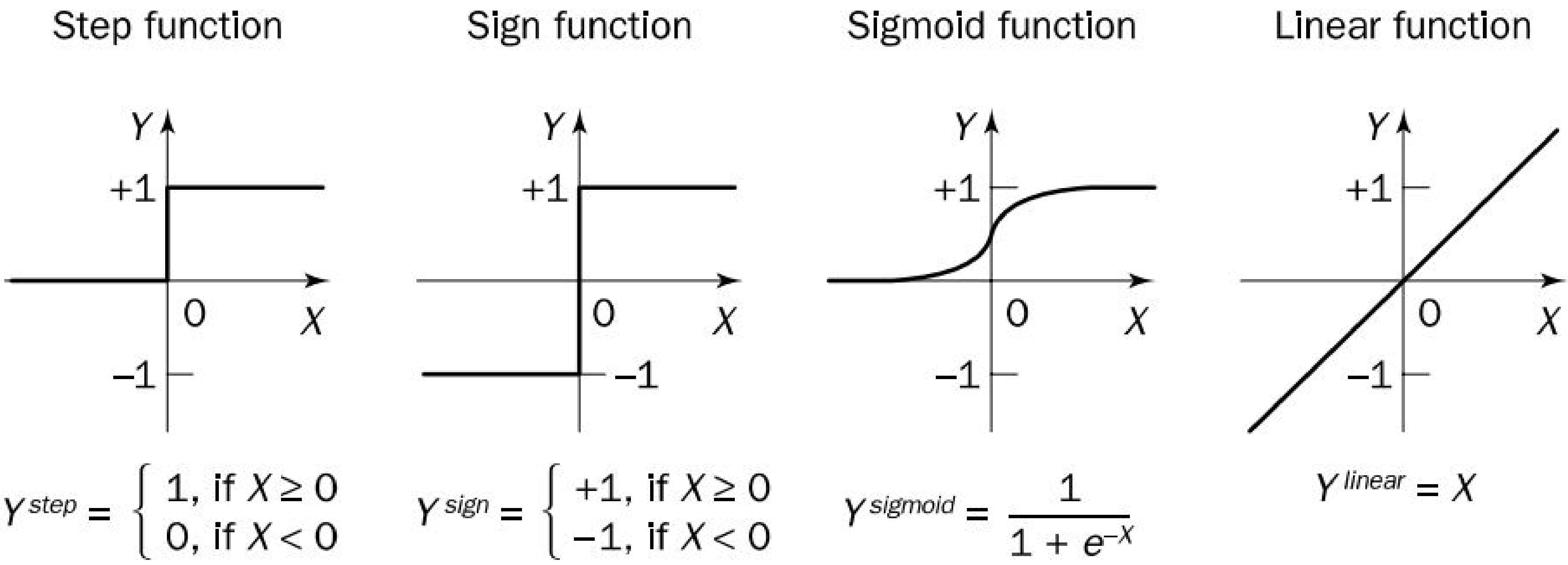


Diagram of a neuron



9

Network Structure

Fig: Activation Functions of Neuron

Perceptron

- In 1958, Frank Rosenblatt introduced a training algorithm that provided the first procedure for training a simple ANN : a **perceptron**
- The perceptron is the simplest form of a neural network. It consists of a single neuron with *adjustable* synaptic weights and a **hard limiter**
- The operation of Rosenblatt's perceptron is based on the McCulloch and Pitts neuron model. The model consists of a **linear combiner** followed by a hard limiter
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and -1 if its input is negative

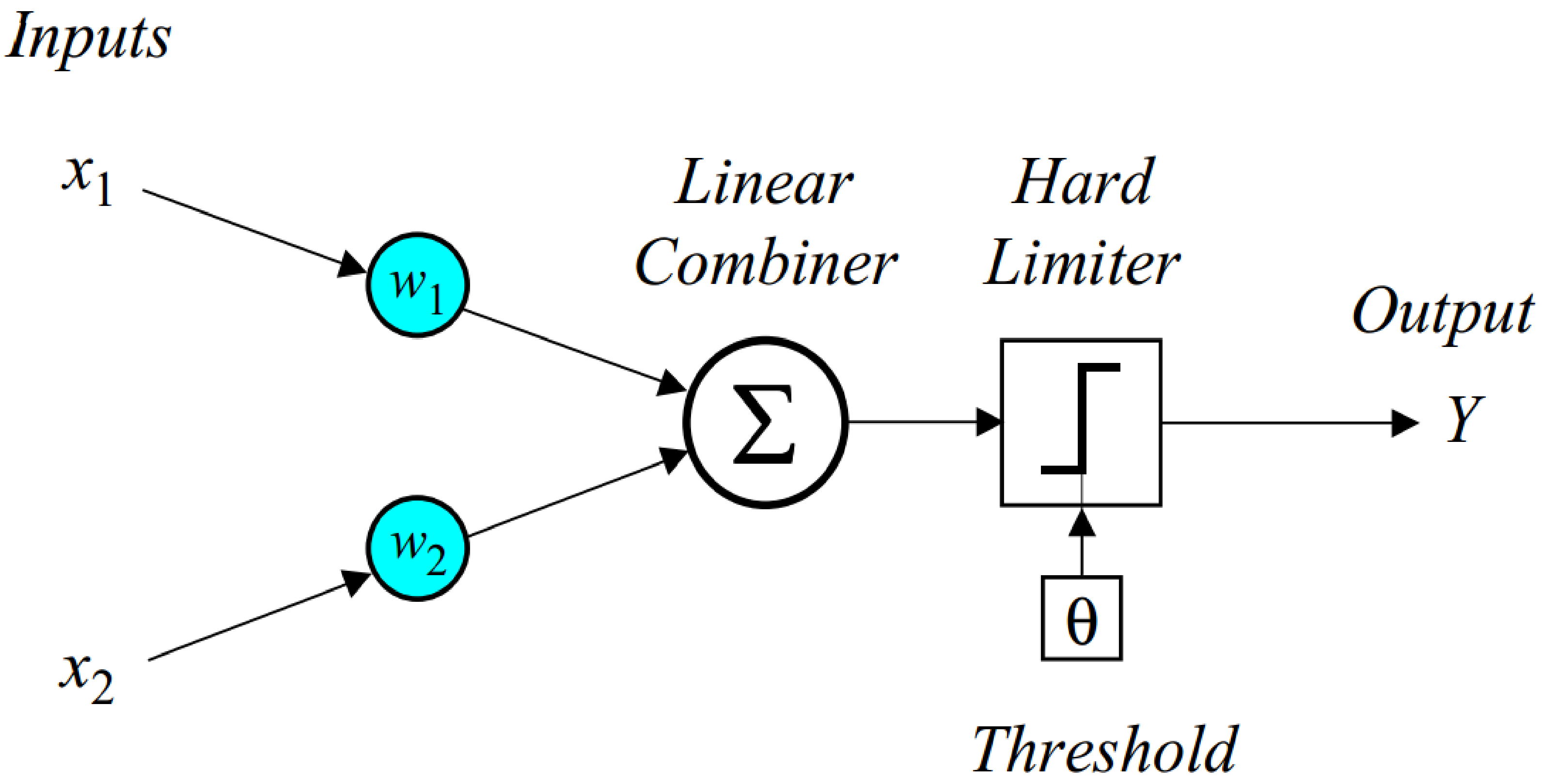


Fig: Single Layer two input Perceptron

Perceptron

- The aim of the perceptron is to classify inputs, $x_1, x_2, x_3, x_4, \ldots, x_n$. Into one of two classes, say A_1, A_2.
- In the case of an elementary perceptron, the n-dimensional space is divided into a *hyperplane* into two decision regions. The hyperplane is defined by ***linearly separable function***

$$\sum_{i=1}^{n}(x_i w_i - \theta) = 0$$

Perceptron

How does the perceptron learn its classification tasks?

- This is done by making small adjustment in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range [-0.5, +0.5], and then updated to obtain the output consistent with the training examples.
- If at iteration p, the actual output is , $Y_{(p)}$ and the desired output is , $Y_{d(p)}$, then the error is given by :

$$e_p = Y_{d(p)} - Y_{(p)} \text{ , where } p = 1, 2, 3 , \ldots$$

Iteration *p* here refers to the *pth* training example presented to the perceptron

- If the error, e_p is positive, we need to increase perceptron output $Y_{(p)}$, but if it is negative, we need to decrease $Y_{(p)}$

Perceptron

Perceptron Learning Rule

$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p)$$

where $p = 1, 2, 3, \ldots$

α is the **learning rate ,** a positive constant less than unity

The perceptron learning rule was first proposed by Rosenblatt in 1960. Using this rule we can derive perceptron training algorithm for classification task

Perceptron

Perceptron Training Algorithm

- **Step 1 : Initialization**

  Set initial weights $w_1, w_2, \ldots, w_n$ and threshold θ to random numbers in the range [-0.5, +0.5].

  If error e_p is positive, we need to increase perceptron output Y_p, but if it is negative, we need to decrease Y_p

Perceptron

Perceptron Training Algorithm

- **Step 2 : Activation**

  Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$, …, $x_n(p)$ and desired output $Y_d(p)$

  Calculate the actual output at iteration p = 1

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p) w_i(p) - \theta\right]$$

  Where *n* is the number of the perceptron inputs, and *step* is activation function

Perceptron

Perceptron Training Algorithm

- **Step 3: Weight Training**
- Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

- Where $\Delta w_i(p)$ is the weight correction at iteration *p*. The weight correction is computed by the delta rule:

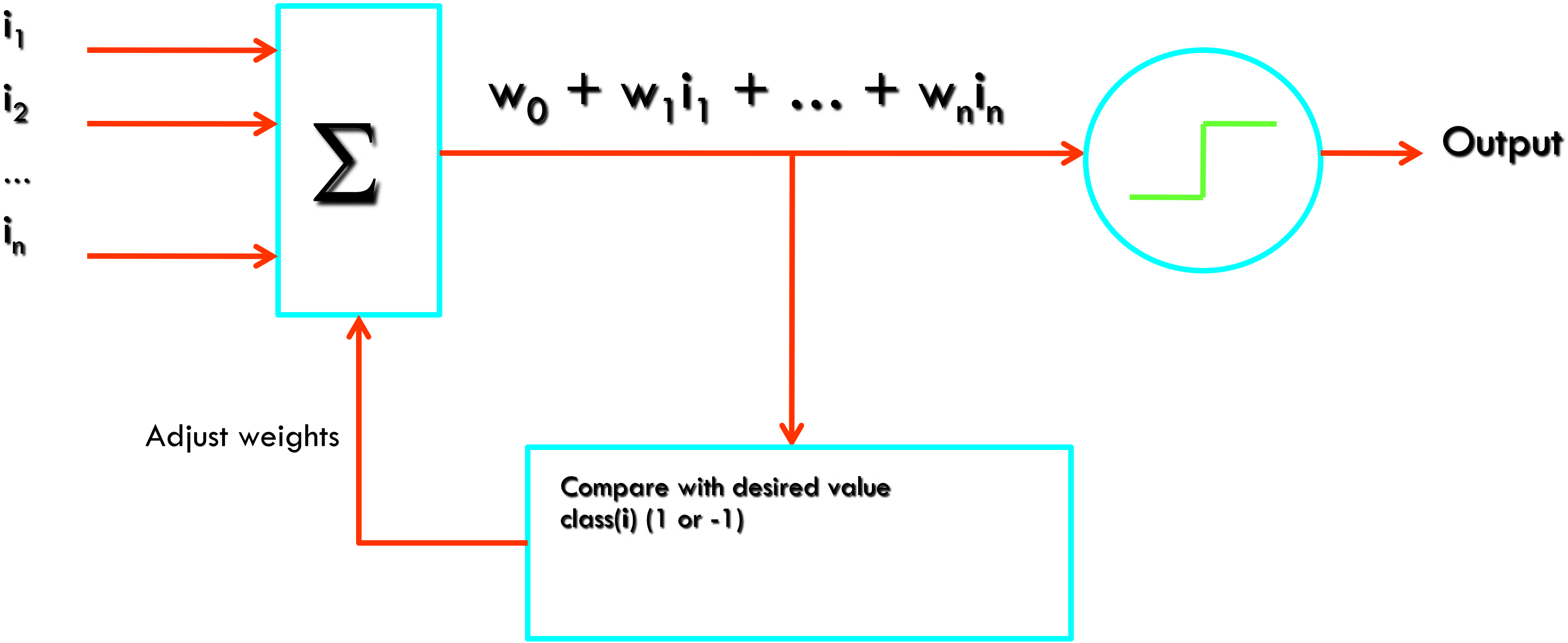
$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p)$$

- **Step 4: Iteration**
- Increase iteration *p* by 1, go back to *Step 2* and repeat the process until convergence

Adaline

- Stands for **Ada**ptive **Li**near **E**lement
- It is a simple perceptron-like system that accomplishes classification by modifying weights in such a way as to diminish the Mean Square Error at every iteration. This can be accomplished using gradient Adaptive Linear Element [Adaline]
- Used in Neural network for
  - Adaptive filtering
  - Pattern Recognition

Adaline

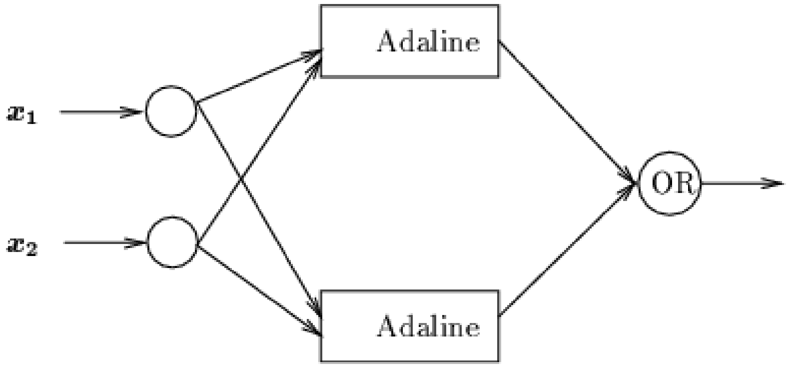


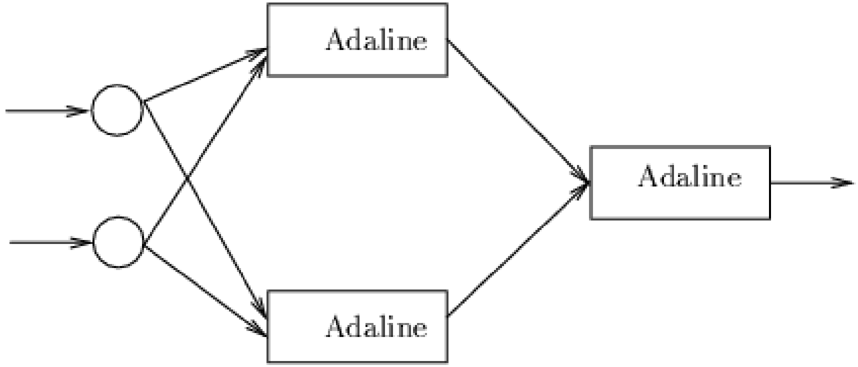
i_1
i_2
...
i_n
Σ
$w_0 + w_1 i_1 + \ldots + w_n i_n$
Output
Adjust weights
Compare with desired value
class(i) (1 or -1)

Madaline

- **Architectures**
  - Hidden layers of adaline nodes
  - Output nodes differ
- **Learning**
  - Error driven, but not by gradient descent
  - Minimum disturbance: smaller change of weights is preferred, provided it can reduce the error
- **Three Madaline models**
  - Different node functions
  - Different learning rules (MR I, II, and III)
  - MR I and II developed in 60's, MR III much later (88)

Madaline





MRI net:

Output nodes with logic function

MRII net:

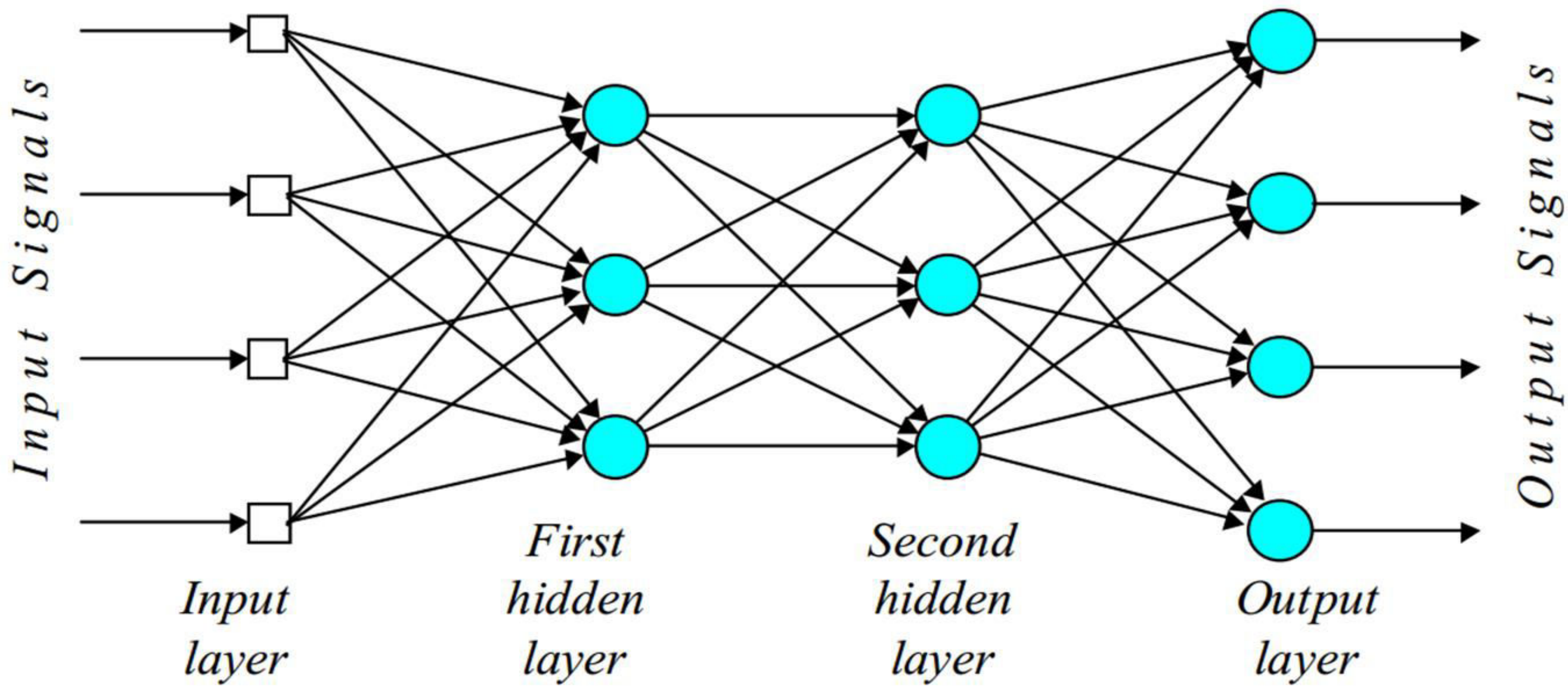
Output nodes are adalines

MRIII net:

Same as MRII, except the nodes with sigmoid function

Multilayer Perceptron

- A multi layer perceptron is a feed forward neural network with one or more hidden layers
- The network consists of :
  - Input Layer
  - Hidden Layer
  - Output Layer
- The input signal is propagated in a forward direction in a layer-by-layer basis



Multilayer Perceptron

Fig: Multilayer Perceptron with two hidden layers

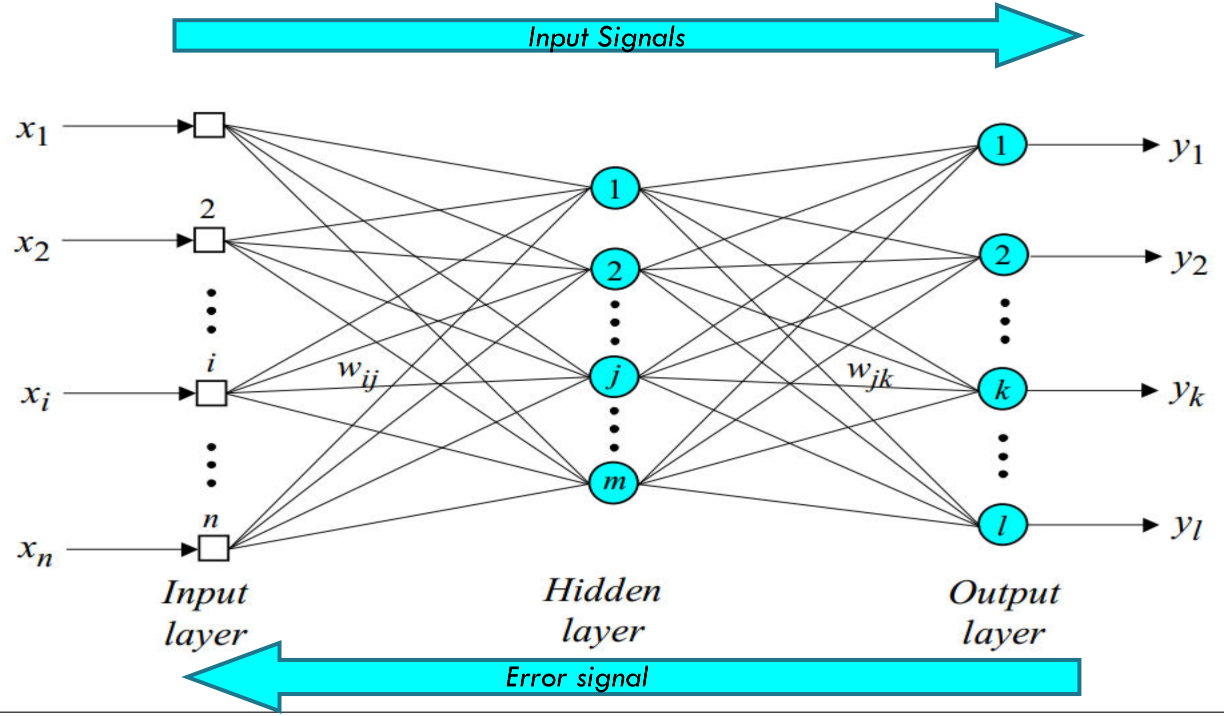
Multilayer Perceptron

- The hidden layer "hides" its desired output. Neurons in the hidden layer can not be observed through the input/output behavior of the network. There is no obvious way to know what the desired output of the hidden layer should be.
- Commercial ANNs incorporate three and sometimes four layers, including one or two hidden layers. Each layer can contain from 10 to 1000 neurons. Experimental neural networks may have fie or six layers, including three or four hidden layers, and utilize millions of neurons.

Back Propagation

- Learning in a multilayer network proceeds the same way as for a perceptron
- A training set of input patterns is presented to the network
- The network computes its output pattern, and if there is an error –or other word difference between actual and desired output pattern – the weight are adjusted to reduce the error
- In a back-propagation neural network, the learning algorithm has two phases
- First, a training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output pattern is generated by the out layer
- If this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated

Back Propagation



Back Propagation Training Algorithm

- **Step 1: Initialization**
- Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range:
- $\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i}\right)$
- where F_i is the total number of inputs of neuron i in the network. The weight initialization is done on a neuron-by-neuron basis.

Back Propagation Training Algorithm

- **Step 2: Activation**
- Activate the back-propagation neural network by applying inputs $x_1(p)$, $x_2(p)$, …, $x_n(p)$ and desired outputs $y_{d,1}(p)$, $y_{d,2}(p)$,…, $y_{d,n}(p)$
- A) Calculate the actual output of the neurons in the hidden layers:

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j\right]$$

- Where *n* is the number of inputs of neuron *j* in the hidden layer, and *sigmoid* is the *sigmoid* activation function

Back Propagation Training Algorithm

- **Step 2: Activation(contd…)**
- b) Calculate the actual outputs of the neurons in the output layer:

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \cdot w_{jk}(p) - \theta_k\right]$$

- Where *m* is the number of inputs of neuron *k* in the output layer.

Back Propagation Training Algorithm

- **Step 3: weight Training**
- Update the weights in the back-propagation network propagating backward the errors associated with output neurons.
- (a) Calculate the error gradient for the neurons in the output layer:

$$\boxed{\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)}$$

where $\boxed{e_k(p) = y_{d,k}(p) - y_k(p)}$

Calculate the weight corrections:

$$\boxed{\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)}$$

Update the weights at the output neurons:

$$\boxed{w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)}$$

Back Propagation Training Algorithm

- **Step 3: weight Training(contd…)**
- (b) Calculate the error gradient for the neurons in the hidden layer:

$$\boxed{\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p)\, w_{jk}(p)}$$

Calculate the weight corrections:

$$\boxed{\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)}$$

Update the weights at the hidden neurons:

$$\boxed{w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)}$$

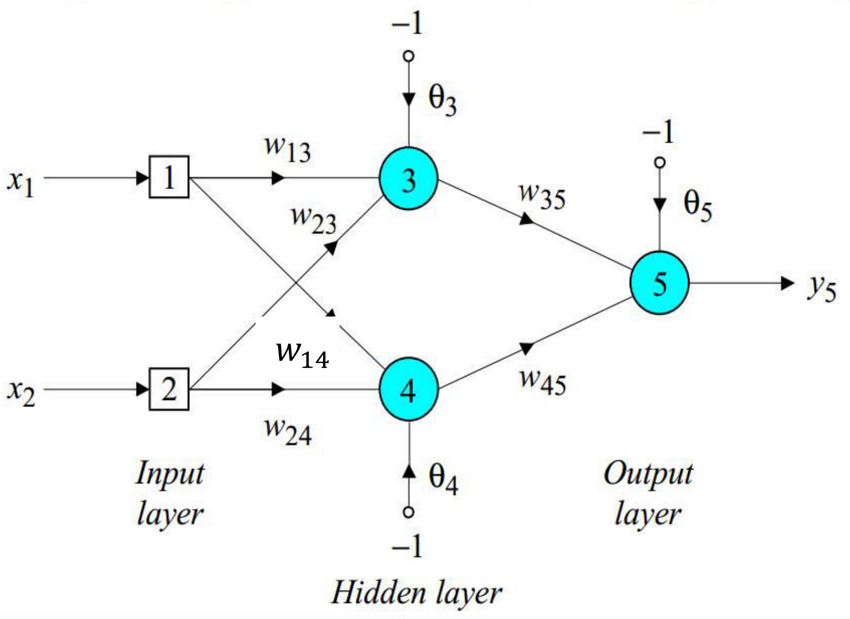
Back Propagation Training Algorithm

- **Step 3: Iteration**
- Increase iteration *p* by one, go back to *Step 2* and repeat the process until the selected error criterion is satisfied.
- As an example, we may consider the three layer back-propagation network. Suppose that the network is required to perform logical operation Exclusive-OR. Recall that a single-layer perceptron could not do this operation. Now we will apply the three layer net.

Example: Three-layer network for solving the Exclusive-OR operation

- The effect of the threshold applied to a neuron in the hidden layer is represented by its weight, θ, connected to a fixed input equal to -1
- The initial weights and threshold levels are set randomly as follows:

$w_{13} = 0.5, w_{14} = 0.9, w_{23} = 0.4, w_{24} = 1.0, w_{35} = -1.2,$
$w_{45} = 1.1, \theta_3 = 0.8, \theta_4 = -0.1$ and $\theta_5 = 0.3.$



Example: Three-layer network for solving the Exclusive-OR operation

- We consider a training set where inputs x_1 ,and x_2 are equal to 1 and desired output $y_{d,5}$ is 0. The actual output of neurons 3 and 4 in the hidden layers are calculated as:

$$y_3 = sigmoid\ (x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1 \cdot 0.5 + 1 \cdot 0.4 - 1 \cdot 0.8)}\right] = 0.5250$$

$$y_4 = sigmoid\ (x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1 \cdot 0.9 + 1 \cdot 1.0 + 1 \cdot 0.1)}\right] = 0.8808$$

- Now the actual output of neuron 5 in the output layer is determined as:

$$y_5 = sigmoid(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.52501.2 + 0.88081.1 - 1 \cdot 0.3)}\right] = 0.5097$$

Thus, the following error is obtained:

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$

Example: Three-layer network for solving the Exclusive-OR operation

- The next step is weight training. To update the weights and threshold levels in our network, we propagate the error, e , from the output layer backward to the input layer.
- First, we calculate the error gradient for neuron 5 in the output layer

$$\delta_5 = y_5\,(1-y_5)\,e = 0.5097\cdot(1-0.5097)\cdot(-0.5097) = -0.1274$$

- Then we determine the weight corrections assuming that the learning rate parameter, α, is equal to 0.1

$$\Delta w_{35} = \alpha\cdot y_3\cdot\delta_5 = 0.1\cdot 0.5250\cdot(-0.1274) = -0.0067$$
$$\Delta w_{45} = \alpha\cdot y_4\cdot\delta_5 = 0.1\cdot 0.8808\cdot(-0.1274) = -0.0112$$
$$\Delta\theta_5 = \alpha\cdot(-1)\cdot\delta_5 = 0.1\cdot(-1)\cdot(-0.1274) = -0.0127$$

Example: Three-layer network for solving the Exclusive-OR operation

- Now we calculate the error gradients for neuron 3 and 4 in the hidden layer:

$$\delta_3 = y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381$$
$$\delta_4 = y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.1274)\cdot 1.1 = -0.0147$$

- We, then, determine the weight corrections:

$$\Delta w_{13} = \alpha\cdot x_1\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha\cdot x_2\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta\theta_3 = \alpha\cdot(-1)\cdot\delta_3 = 0.1\cdot(-1)\cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \alpha\cdot x_1\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta w_{24} = \alpha\cdot x_2\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta\theta_4 = \alpha\cdot(-1)\cdot\delta_4 = 0.1\cdot(-1)\cdot(-0.0147) = 0.0015$$

Example: Three-layer network for solving the Exclusive-OR operation

- At last, we update all weights and thresholds
- The training process is updated till the sum of squared error is less than 0.001

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$
$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$
$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$
$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$
$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$
$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$
$$\theta_3 = \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962$$
$$\theta_4 = \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985$$
$$\theta_5 = \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127$$

Example: Three-layer network for solving the Exclusive-OR operation

- The Final results of three layer network learning is:

Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

Gradient Descent

- Gradient descent is an iterative minimisation method. The gradient of the error function always
  shows in the direction of the steepest ascent of the error function.
- It is determined as the derivative of the activation function multiplied by the error at the neuron output

For Neuron *k* in the output layer

$$\delta_k(p) = \frac{\partial y_k(p)}{\partial X_k(p)} \times e_k(p)$$

Where, yk(p) is the output of neuron k at iteration p, and Xk(p) is the net weighted input of neuron k at same iteration.

Hopfield Network

- neural networks with feedback – Hopfield networks
- presence of such loops has a profound impact on the learning capability of the network
- After applying a new input, the network output is calculated and feedback to adjust the input. Then the output is calculated again, and the process is repeated until the output becomes constant.[Working mechanism of Recurrent network]

Hopfield Network

- Refer Hopfield Network training algorithm in Negnevitskey’s Book [page 212]

Hopfield Network

- Storage memory, Energy based model
- Composed of binary threshold units with recurrent connections between them
- Recurrent network of non-linear units are generally hard to analyze. They behave in different ways:
  - Settle to stable state
  - Oscillates
  - Follows chaotic trajectories that cant be predicated fat into future [input might not assure solution]

Hopfield Network

- Hopfield realized the connections are symmetric, there is global energy function
  - Binary configuration of whole network has an energy
- Binary threshold decision rule cause the network to settle to minimum of this energy [if we apply downhill energy rule]

Hopfield Network

- Global energy is the sum of many contributions. Each contribution depends on "one connection weight" and binary state of two neurons
- $E = -\sum_i s_i b_i - \sum_{i<j} s_i s_j . w_i w_i$
- This quadratic energy function makes it possible for each unit to compute locally how it's state affects the global energy
- Energy gap = $\Delta E_i = E(s_i = 0) - E(s_i = 1) = b_i - \sum_j s_j w_{ij}$

Hopfield Network

- Example:

Kohonen Network

- self-organized maps (Kohonen, 1982)
- Learning of neural network without the presence of teacher
- Competitive learning

Kohonen Network

- In competitive learning, neurons compete among themselves to be activated
- While in Hebbian learning, several output neurons can be activated simultaneously, in competitive learning, only a single output neuron is active at any time.
- The output neuron that wins the “competition” is called the ***winner-takes-all*** **neuron.**

Kohonen Network

- The basic idea of competitive learning was introduced in the early 1970s.
- In the late 1980s, Teuvo Kohonen introduced a special class of artificial neural networks called **self-organizing feature maps**. These maps are based on competitive learning.

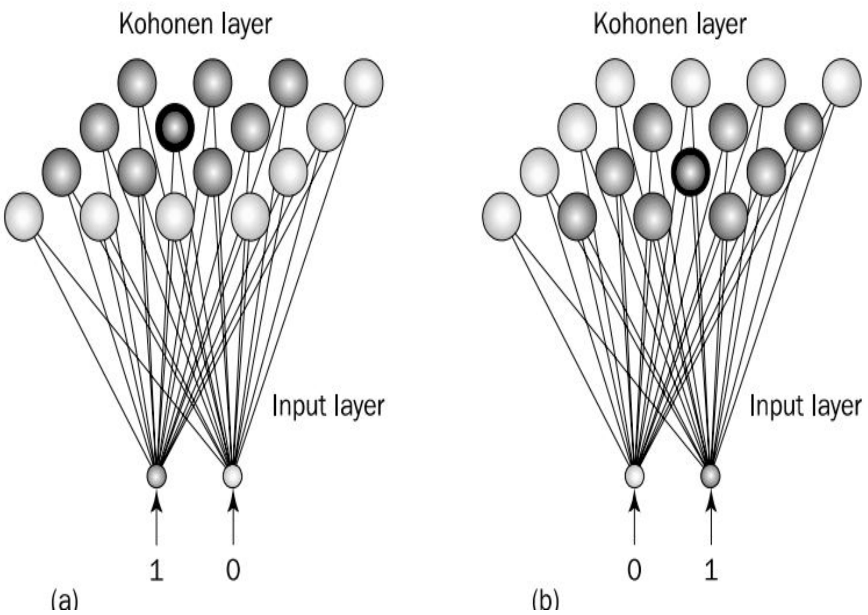
Kohonen Network

What is self organizing feature map?

Our brain is dominated by the cerebral cortex, a very complex structure of billions of neurons and hundreds of billions of synapses. The cortex includes areas that are responsible for different human activities (motor, visual, auditory, somatosensory, etc.), and associated with different sensory inputs. We can say that each sensory input is mapped into a corresponding area of the cerebral cortex. **The cortex is a self-organizing computational map in the human brain.**

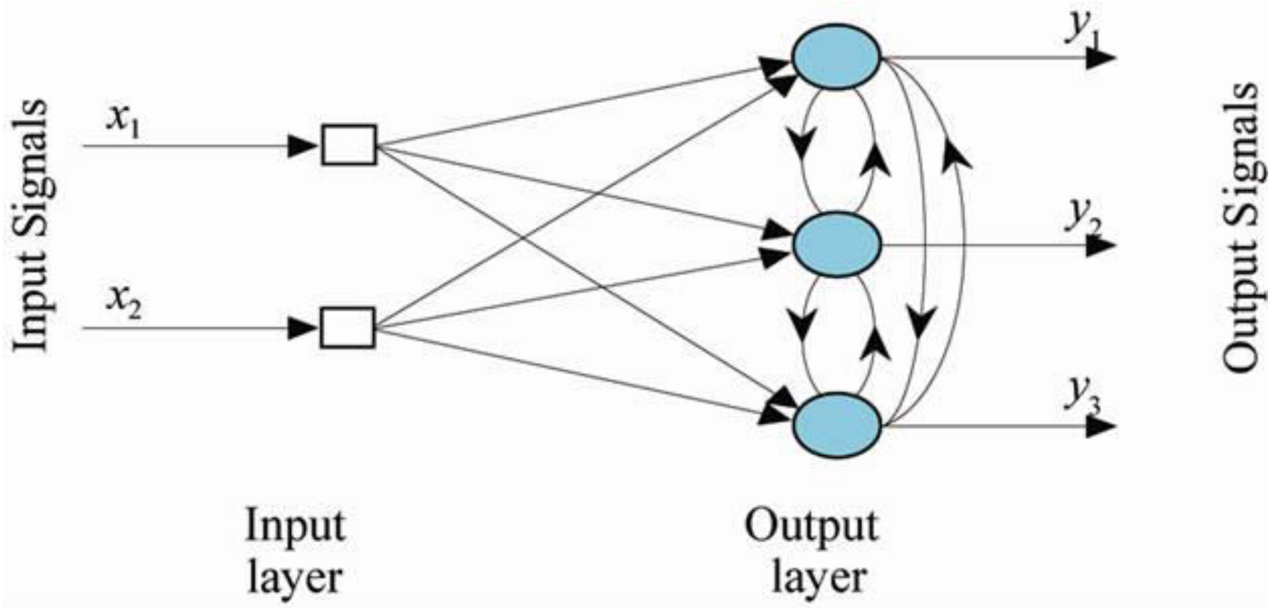
Kohonen Network

- The Kohonen model provides a topological mapping. It places a fixed number of input patterns from the input layer into a higher dimension output or Kohonen Layer
- Training in Kohonen Network begins with the winner's neighborhood of a fairly large size. Then, as training proceeds, the neighborhood size gradually decreases



Kohonen Network

Architecture of the Kohonen Network

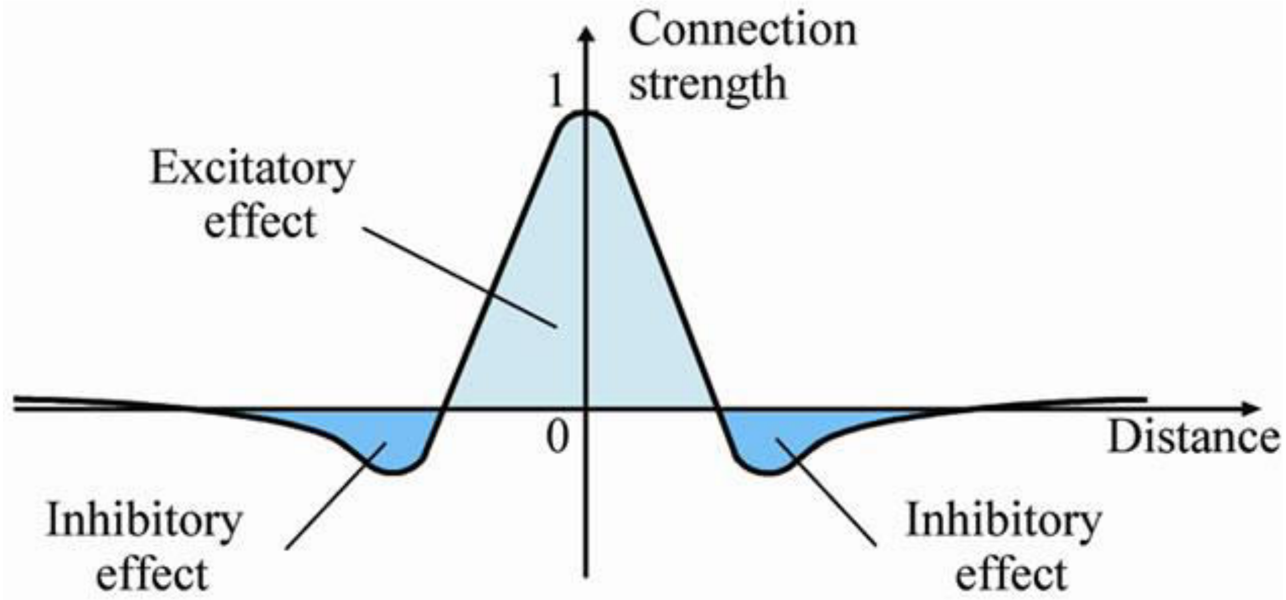


Kohonen Network

- The lateral connections are used to create a competition between neurons. The neuron with the largest activation level among all neurons in the output layer becomes the winner. This neuron is the only neuron that produces an output signal. The activity of all other neurons is suppressed in the competition.
- The lateral feedback connections produce excitatory or inhibitory effects, depending on the distance from the winning neuron. This is achieved by the use of a **Mexican hat function** which describes synaptic weights between neurons in the Kohonen layer.

Kohonen Network

Mexican hat function



Kohonen Network

- Mexican hat function represents the relationship between the distance from the winner-takes-all neuron and the strength of the connection within the Kohonen layer
- According to this function, the near neighborhood (a short-range lateral excitation area) has a strong excitatory effect, remote neighborhood (an inhibitory penumbra) has a mild inhibitory effect and very remote neighborhood (an area surrounding the inhibitory penumbra) has a weak excitatory effect, which is usually neglected

Kohonen Network

- In the Kohonen network, a neuron learns by shifting its weight from inactive connections to active ones. Only the winning neuron and its neighborhood are allowed to learn. If a neuron does not respond to a given input pattern, then learning can not occur in that particular neuron
- The competitive learning rule defines the change in Δw_{ij} applied to synaptic weight w_{ij} as

$$\Delta w_{ij} = \begin{cases} \alpha(x_i - w_{ij}), & \text{if neuron } j \text{ wins the competition} \\ 0, & \text{if neuron } j \text{ loses the competition} \end{cases}$$

- Where x_i is the input signal and α is the learning rate parameter

Kohonen Network

- The overall effect of the competitive learning rule resides in moving the synaptic weight vector W_i of the winning neuron j towards the input pattern X. the matching Euclidean Distance between vectors.
- The Euclidean Distance between pair of n-by-1 vectors X and W_i is defined by

$$d = \|\mathbf{X} - \mathbf{W}_j\| = \left[\sum_{i=1}^{n} (x_i - w_{ij})^2\right]^{1/2}$$

- Where x_i and w_j are ith elements of the vectors X and W_i respectively

Kohonen Network

- To identify the winning neuron , i_X , that best matchs the input vector X, we can apply following condition (Haykin, 1999)

$$j_{\mathbf{X}} = \min_{j} \|\mathbf{X} - \mathbf{W}_j\|, \qquad j = 1, 2, \ldots, m$$

where m is the number of neurons in the Kohonen layer

Kohonen Network

EX: Suppose that the two dimensional input vector X is presented to 3-dimensional Kohonen Network

$$\mathbf{X} = \begin{bmatrix} 0.52 \\ 0.12 \end{bmatrix}$$

The initial weight vectors, $\mathbf{W}_j$, are given by

$$\mathbf{W}_1 = \begin{bmatrix} 0.27 \\ 0.81 \end{bmatrix} \quad \mathbf{W}_2 = \begin{bmatrix} 0.42 \\ 0.70 \end{bmatrix} \quad \mathbf{W}_3 = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix}$$

Kohonen Network

We find the winning (best-matching) neuron $j_{\mathbf{X}}$ using the minimum-distance Euclidean criterion:

$$d_1 = \sqrt{(x_1 - w_{11})^2 + (x_2 - w_{21})^2} = \sqrt{(0.52 - 0.27)^2 + (0.12 - 0.81)^2} = 0.73$$

$$d_2 = \sqrt{(x_1 - w_{12})^2 + (x_2 - w_{22})^2} = \sqrt{(0.52 - 0.42)^2 + (0.12 - 0.70)^2} = 0.59$$

$$d_3 = \sqrt{(x_1 - w_{13})^2 + (x_2 - w_{23})^2} = \sqrt{(0.52 - 0.43)^2 + (0.12 - 0.21)^2} = 0.13$$

Neuron 3 is the winner and its weight vector $\mathbf{W}_3$ is updated according to the competitive learning rule.

$$\Delta w_{13} = \alpha\,(x_1 - w_{13}) = 0.1\,(0.52 - 0.43) = 0.01$$

$$\Delta w_{23} = \alpha\,(x_2 - w_{23}) = 0.1\,(0.12 - 0.21) = -0.01$$

Kohonen Network

- The updated weight if vector W_3 at (p+1) is given by

$$\mathbf{W}_3(p+1) = \mathbf{W}_3(p) + \Delta\mathbf{W}_3(p) = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix} + \begin{bmatrix} 0.01 \\ -0.01 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.20 \end{bmatrix}$$

The weight vector $\mathbf{W}_3$ of the wining neuron 3 becomes closer to the input vector $\mathbf{X}$ with each iteration.

Kohonen Network

- Competitive Learning Algorithm
  - Refer Page 209 of Negnevitsky

- "Radial Basis Function and Elastic Net Model" on your own

Characteristics of ANN

- Adaptive learning
- Self-organization
- Error tolerance
- Real-time operation
- Parallel information processing

Benefits and Limitations of ANN

Benefits	Limitations
Ability to tackle new kind of problems	Performs less well at tasks humans tend to find difficult
Robustness	Lack of explanation facilities
	Require large amount of test data

Expert System

Definition of Expert System:

An Expert System is a collection of programmes or Computer Software that solves problems in the domain of interest. It is called system because it consists of both problem solving component and a support component.

The process of building Expert System is called knowledge engineering and is done by knowledge Engineer

Expert System

Expert systems provide the following important features:

- Facility for non-expert personnel to solve problems that require some expertise
- Speedy solution
- Reliable solution
- Cost reduction
- Power to manage without human experts
- Wider areas of knowledge

Use of expert systems is specially recommended when:

- Human experts are difficult to find
- Human experts are expensive
- Knowledge improvement is important
- The available information is poor, partial, incomplete
- Problems are incompletely defined
- There is lack of knowledge among all those who need it
- The problem is rapidly changing legal rules and codes

Architecture of an Expert System

Architecture of expert systems reflects the knowledge engineers' understanding of the methods representing knowledge and how to perform intelligent decision making task with the support of computer-based systems

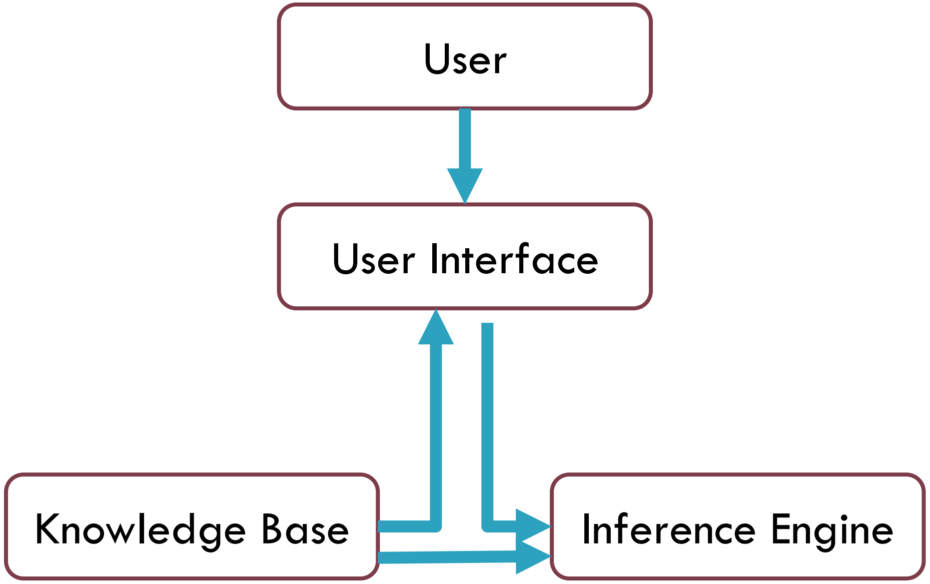


Fig: Architecture of Expert System

Expert System

- Declarative Knowledge
  - descriptive representation of knowledge. It tells us facts: what things are. It is expressed in a factual statement, such as “There is a positive association between smoking and cancer.” Domain experts tell us about truths and associations. This type of knowledge is considered shallow, or surface-level, information that experts can verbalize. Declarative knowledge is especially important in the initial stage of knowledge acquisition.

Expert System

- Procedural Knowledge
  - considers the manner in which things work under different sets of circumstances
  - The following is an example :“Compute the ratio between the price of a share and the earnings per share. If the ratio is larger than 12, stop your investigation. Your investment is too risky. If the ratio is less than 12, check the balance sheet.” Thus, procedural knowledge includes step-by-step sequences and how-to types of instructions; it may also include explanations.
  - Procedural knowledge involves automatic responses to stimuli.
  - It may also tell us how to use declarative knowledge and how to make inferences.

Expert System

Declarative Knowledge

- Declarative knowledge relates to a specific object.
- It includes information about the meaning, roles, environment, resources, activities, associations, and outcomes of the object

Procedural Knowledge

- Procedural knowledge relates to the procedures used in the problem-solving process (e.g., information about problem definition, data gathering, the solution process, evaluation criteria).

Development of an Expert System

Expert system design and development must be carefully programmed in success is desired.

The following are the main steps to be followed while developing expert system

a.	Outline Statement	b.	Knowledge acquisition
c.	Knowledge representation	d.	Prototype Development
e.	Testing	f.	Main Knowledge Acquisition
f.	Specification with detailed Information	g.	System Development
h.	Implementation	i.	Maintenance

Expert System: Features

- Goal Driven (Backward Chaining) or Data Driven (Forward Chaining) Reasoning
- Coping with uncertainty
- Data Representation
- User Interface
- Explanations (ability to explain solution w.r.t. problem specification
- Use knowledge rather than data
- Use symbolic representation for knowledge
- Should have meta knowledge

Expert System: Advantages

- Provide consistent answers for repetitive decisions, processors and tasks
- Hold and maintain significant levels of information
- Encourage organization to clarify the logic of their decision making
- Ask question like human expertise

Expert System: Disadvantages

- Lack of common sense needed in same decision making
- Cant make creative responses as human expert would in unusual circumstances
- Not able to explain their logic and reasoning
- Error may occur n the knowledge base and lead to wrong decision making
- Cant adopt to changing environment, unless knowledge base is changed

Expert System: Example → DENDRAL

- First ES developed in late 1960s
- Designed to analyse mass spectra
- Based on the mass of fragments seen in the spectra, it would be possible to make inference as the nature of molecule tested, identifying functional groups or even the entire molecule
- DENDRAL used heuristic knowledge obtained from experienced chemists
- Uses forward chaining for reasoning
- Discovered number of structures previously unknown to climate experts

Expert System - MYCIN

- An expert system for treating blood infections
- MYCIN would attempt to diagnose patients based on reported symptoms and medical test results
- Could ask some more information and lab test results for diagnosis
- It would recommend a course of treatment, if requested MYCIN would explain the reasoning that lead to its diagnosis and recommendations
- Uses about 500 production rules
- MYCIN operated at roughly the same level of competence as human specialists in blood infections
- Uses backward chaining for reasoning

Natural Language Processing

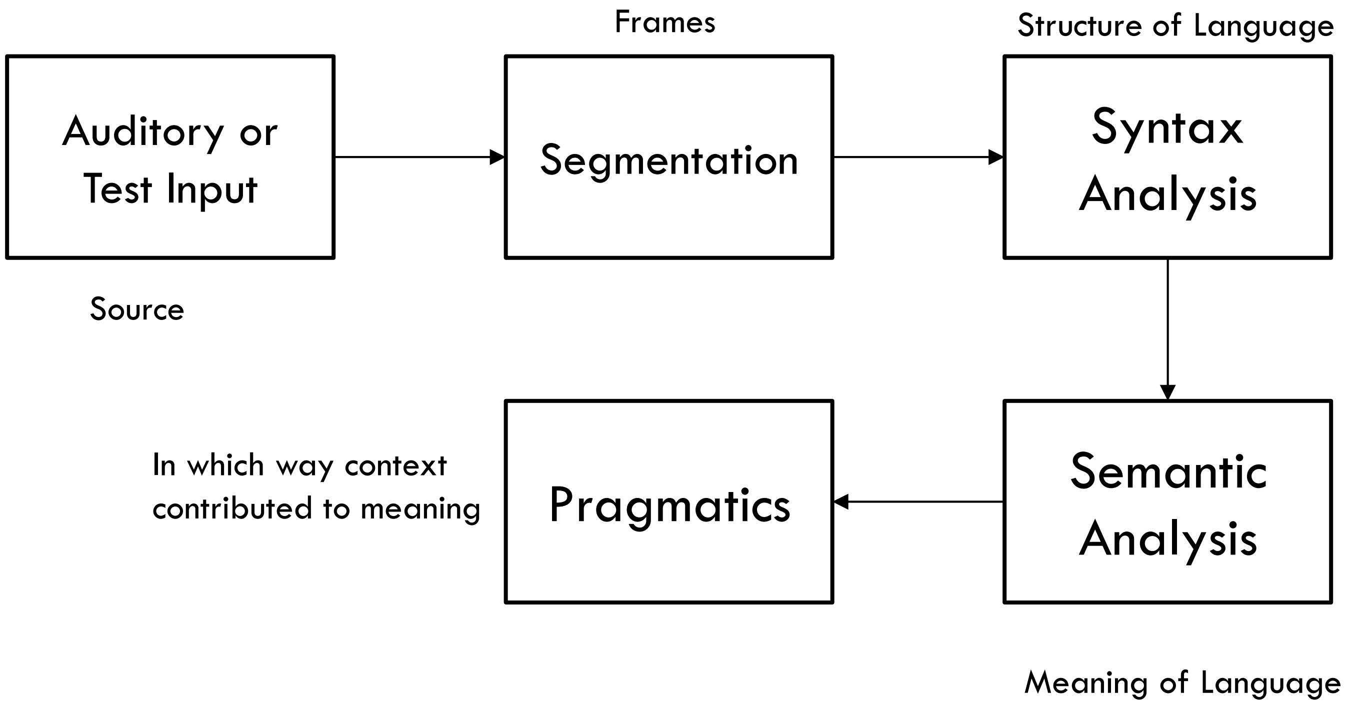
NLP

- Machine is considered intelligent if it can understand and manipulate Natural Language
- The language spoken by people : Natural Language

Natural Language Processing

- NLP is one of the field of AI that processes or analyses written or spoken language
- NLP=NLG+NLU, where NLG is about generation and NLU is about understanding the natural language
- Understanding language requires a lot of knowledge

NLP: Processes



NLP

- Machine is considered intelligent if it can understand and manipulate Natural Language
- The language spoken by people : Natural Language
- NLP → AI method of communicating with an intelligent systems using Natural Language
- NLP is required when an intelligent system like robot to perform as per your instructions
- EX: getting result from Dialog Based clinical ES

NLP

- NLP involves making computers to perform useful tasks with the natural languages human use
- I/O encompasses
  - Speech
  - Written Text

NLP [Components]

- Natural Language Understanding
  - Understanding involves the following tasks
    - Mapping the given input in natural language into useful representations.
    - Analyzing different aspects of the language

NLP [Component]

- Natural Language Generation
  - It is the process of producing meaningful phrases and sentences in the form of natural language from some internal representation. It involves :-
    - **Text planning** − It includes retrieving the relevant content from knowledge base.
    - **Sentence planning** − It includes choosing required words, forming meaningful phrases, setting tone of the sentence.
    - **Text Realization** − It is mapping sentence plan into sentence structure. The NLU is harder than NLG.

NLP [Difficulties in NLU]

- **Lexical ambiguity** – It is at very primitive level such as word-level. For example, treating the word "board" as noun or verb?
- **Syntax Level ambiguity** – A sentence can be parsed in different ways. For example, "He lifted the beetle with red cap." – Did he use cap to lift the beetle or he lifted a beetle that had red cap?
- **Referential ambiguity** – Referring to something using pronouns. For example, Rima went to Gauri. She said, "I am tired." – Exactly who is tired?
- One input can mean different meanings.
- Many inputs can mean the same thing

NLP [Steps]

- **Lexical Analysis** – It involves identifying and analyzing the structure of words. Lexicon of a language means the collection of words and phrases in a language. Lexical analysis is dividing the whole chunk of text into paragraphs, sentences, and words.
- **Syntactic Analysis** ***Parsing*** – It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words. The sentence such as "The school goes to boy" is rejected by English syntactic analyzer.

NLP [Steps]

- **Semantic Analysis** – It draws the exact meaning or the dictionary meaning from the text. The text is checked for meaningfulness. It is done by mapping syntactic structures and objects in the task domain. The semantic analyzer disregards sentence such as “hot icecream”.
- **Discourse Integration** – The meaning of any sentence depends upon the meaning of the sentence just before it. In addition, it also brings about the meaning of immediately succeeding sentence.
- **Pragmatic Analysis** – During this, what was said is re-interpreted on what it actually meant. It involves deriving those aspects of language which require real world knowledge.

NLP [Steps]

Lexical Analysis

↓

Syntactic Analysis

↓

Semantic Analysis

↓

Disclosure Integration

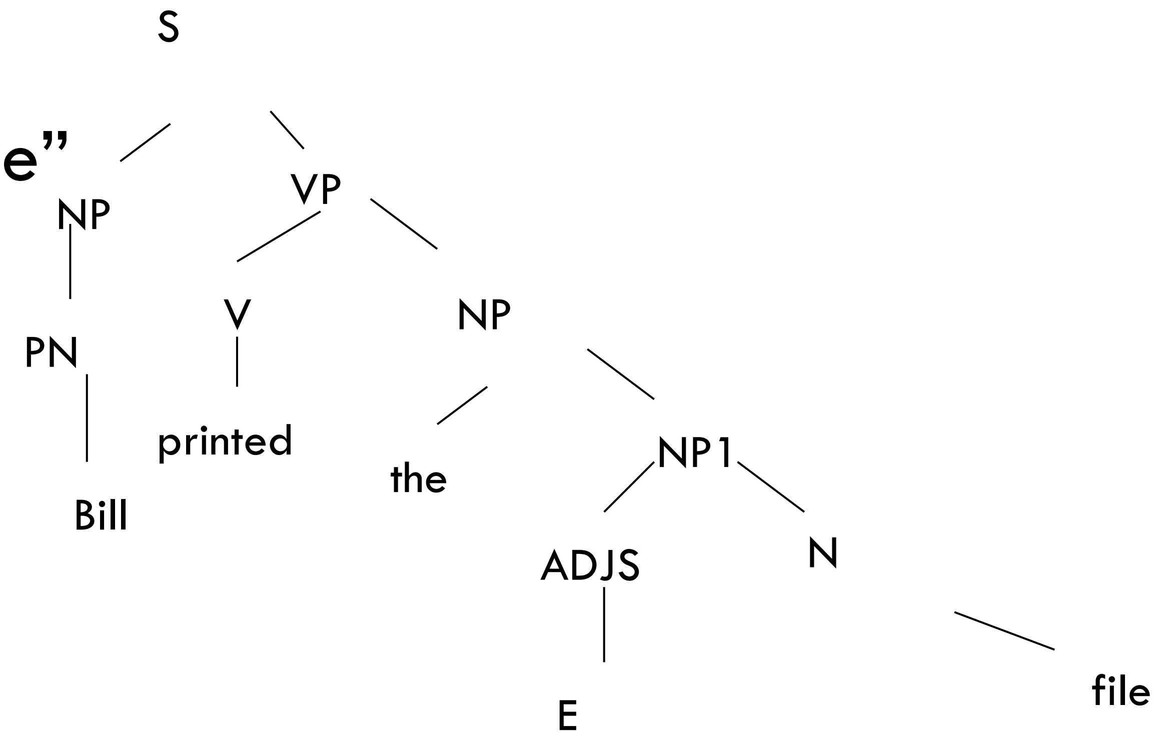
↓

Pragmatic Analysis

NLP

- Parse Tree

"Bill Printed the file"

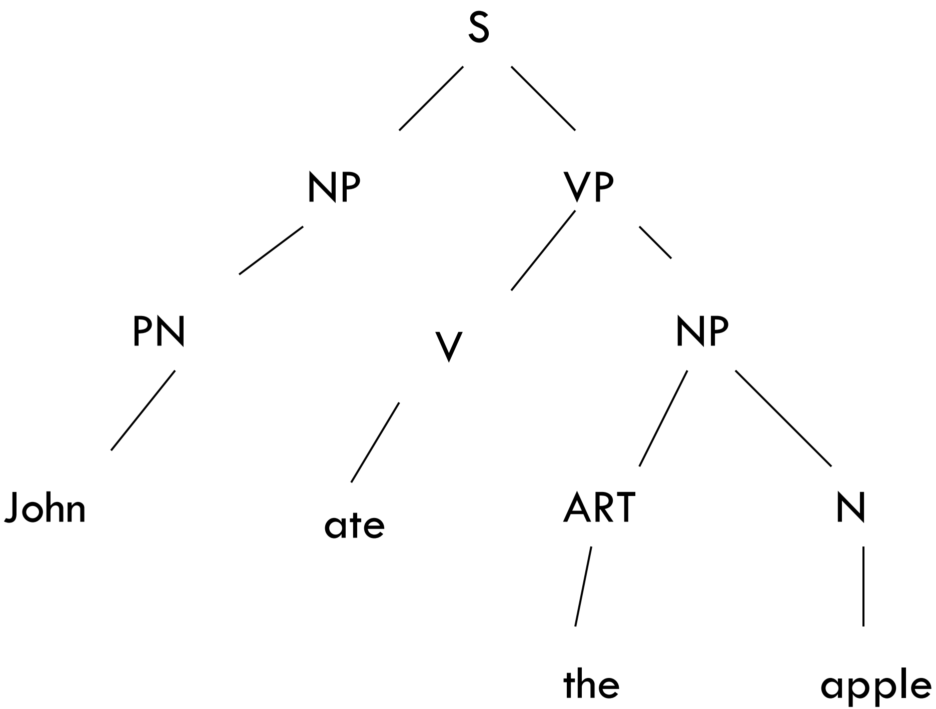


NLP

- A parse tree :

John ate the apple.

1. S -> NP VP
2. VP -> V NP
3. NP -> NAME
4. NP -> ART N
5. NAME -> John
6. V -> ate
7. ART-> the
8. N -> apple



References

- Russell, S. and Norvig, P., 2011, Artificial Intelligence: A Modern Approach, Pearson, India.
- Rich, E. and Knight, K., 2004, Artificial Intelligence, Tata McGraw hill, India.