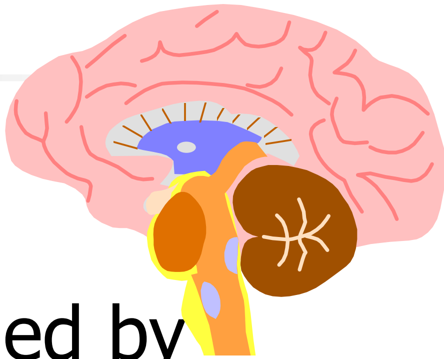
Artificial Neurons, Neural Networks and Architectures

Artificial Neural Networks

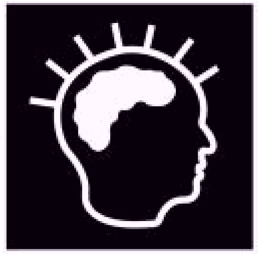
- Other terms/names
  - connectionist
  - parallel distributed processing
  - neural computation
  - adaptive networks..
- History
  - 1943-McCulloch & Pitts are generally recognised as the designers of the first neural network
  - 1949-First learning rule
  - 1969-Minsky & Papert - perceptron limitation - Death of ANN
  - 1980's - Re-emergence of ANN - multi-layer networks

The biological inspiration



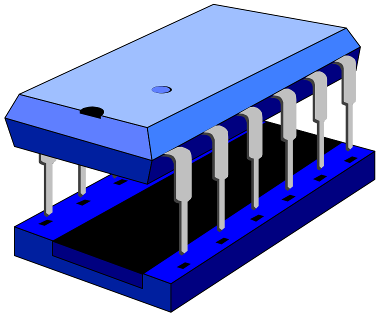
- The brain has been extensively studied by scientists.
- Vast complexity prevents understanding.
- Even the behaviour of an individual neuron is extremely complex

Features of the Brain



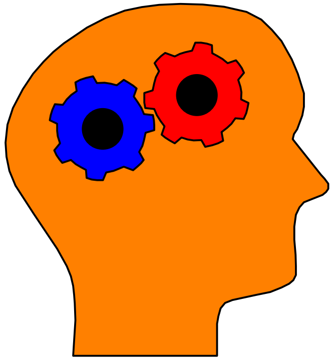
- Ten billion (10^{10}) neurons
- Face Recognition ~0.1secs
- On average, each neuron has several thousand connections
- Hundreds of operations per second
- High degree of parallel computation
- Distributed representations
- Compensated for problems by massive parallelism

The contrast in architecture



- The Von Neumann architecture uses a single processing unit;
  - Tens of millions of operations per second

- The brain uses many slow unreliable processors acting in parallel



Computer vs. Brain

- Computers are good at: 1/ Fast arithmetic and 2/ Doing precisely what the programmer programs them to do
- Computers are not good at: 1/ Interacting with noisy data or data from the environment, 2/ Massive parallelism, 3/ Fault tolerance, 4/ Adapting to circumstances

Neural Networks Defined

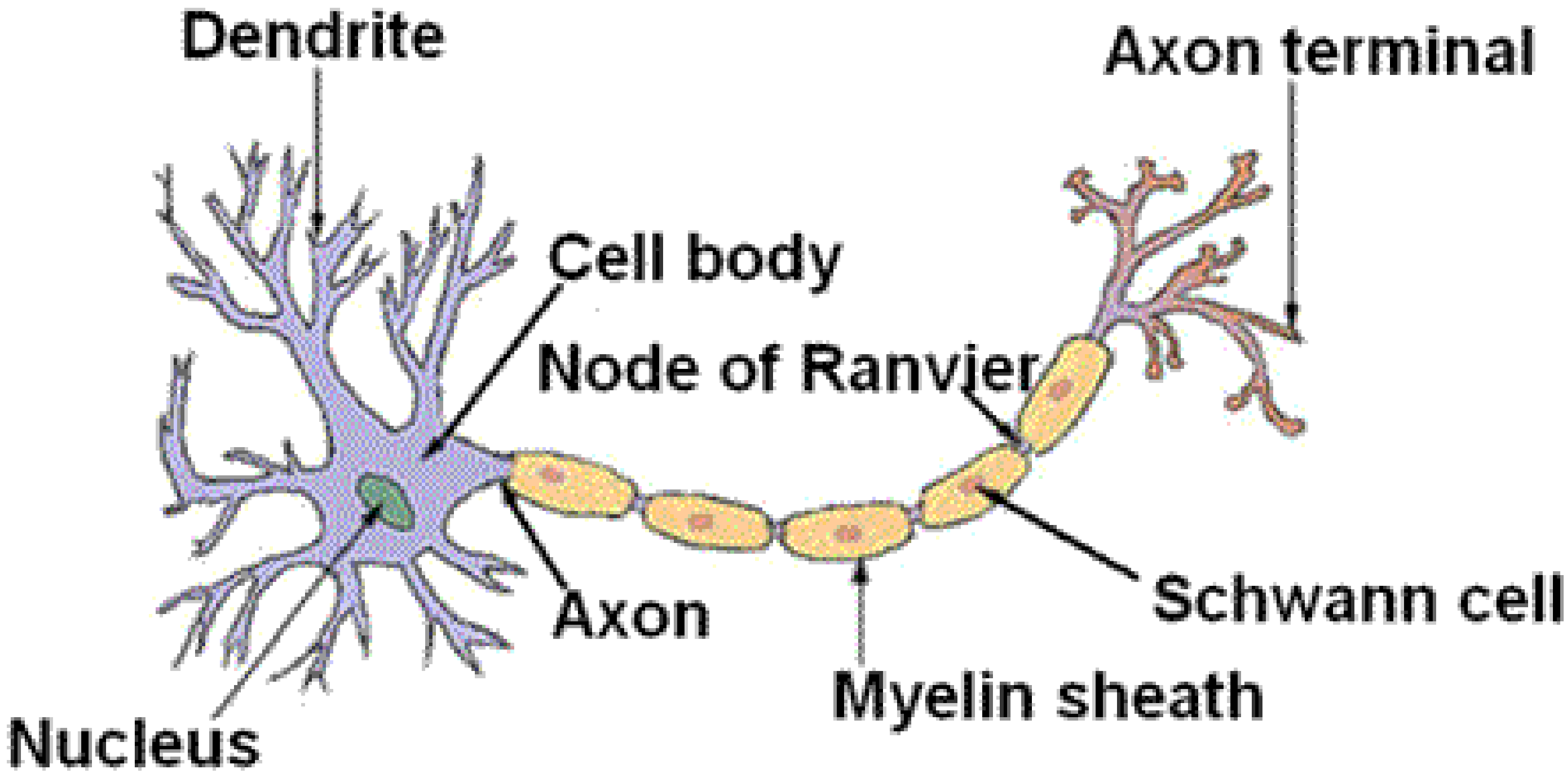
- Artificial neural networks are massively parallel adaptive networks of simple nonlinear computing elements called neurons which are intended to abstract and model some of the functionality of the human nervous system in an attempt to partially capture some of its computational strengths.

Neuron Abstraction

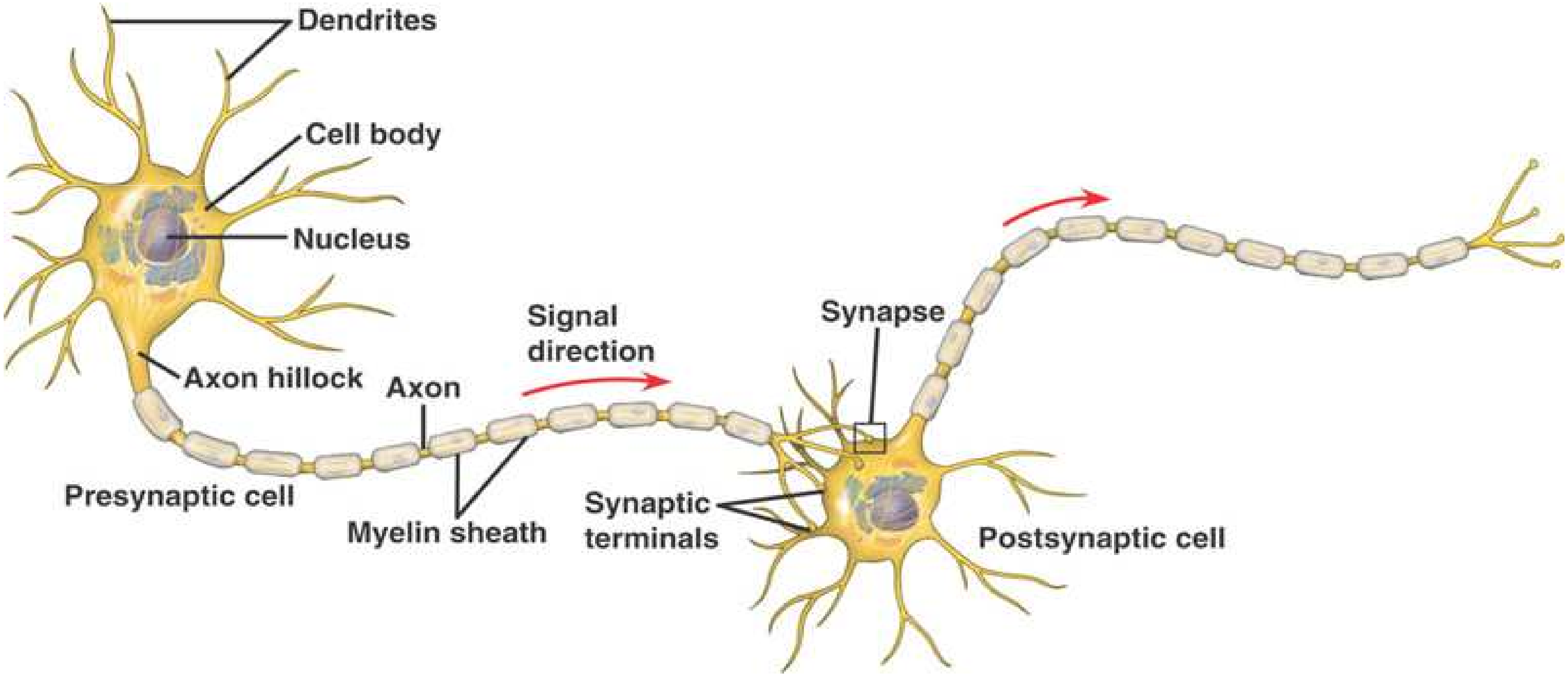
- Neurons transduce signals—electrical to chemical, and from chemical back again to electrical.
- Each synapseis associated with synaptic efficacy
  - It is the efficiency with which a signal is transmitted from the presynaptic to postsynaptic neuron

s

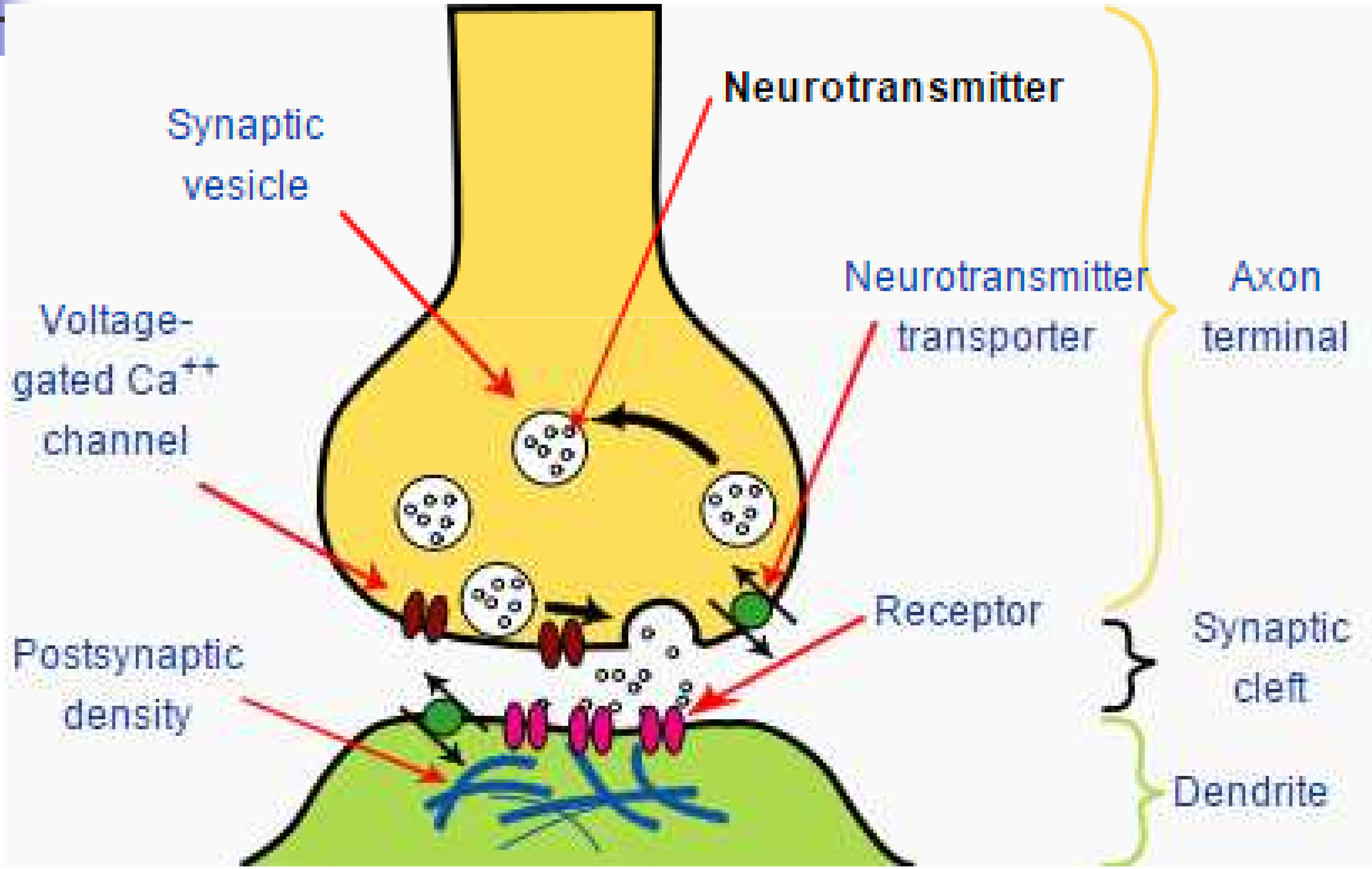
Biological Neuron



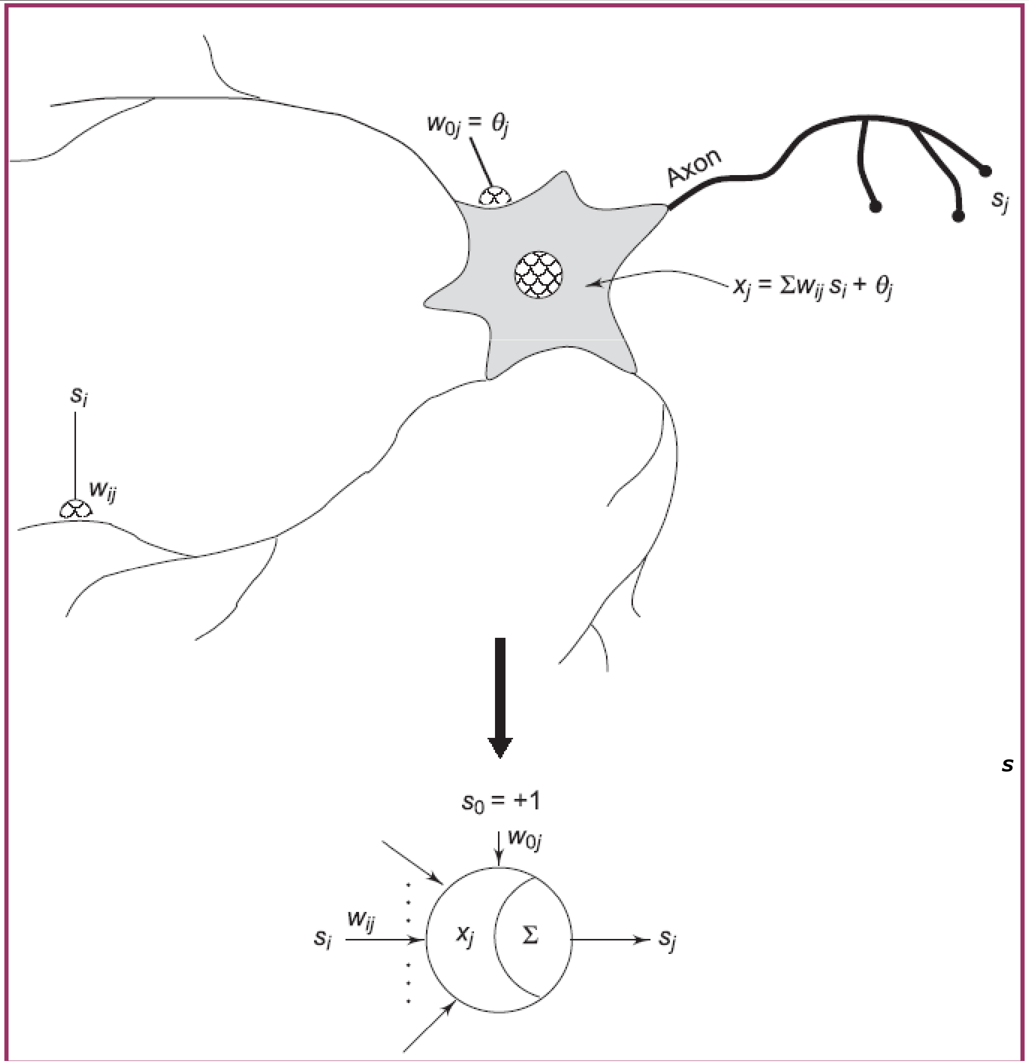
Biological Neuron



Biological Neuron

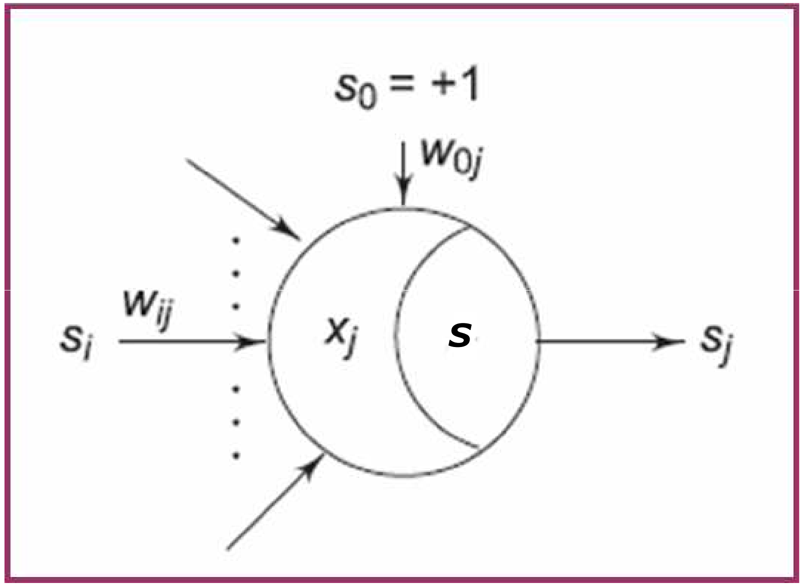


Neuron Abstraction



Neuron Abstraction: Activations

- The j^{th} artificial neuron that receives input signals s_i, from possibly n different sources
- An internal activation x_j which is a linear weighted aggregation of the impinging signals, modified by an internal threshold, θ_j



$$x_j = \sum_{i=1}^{n} w_{ij} s_i + \theta_j$$

Activations Measure Similarities

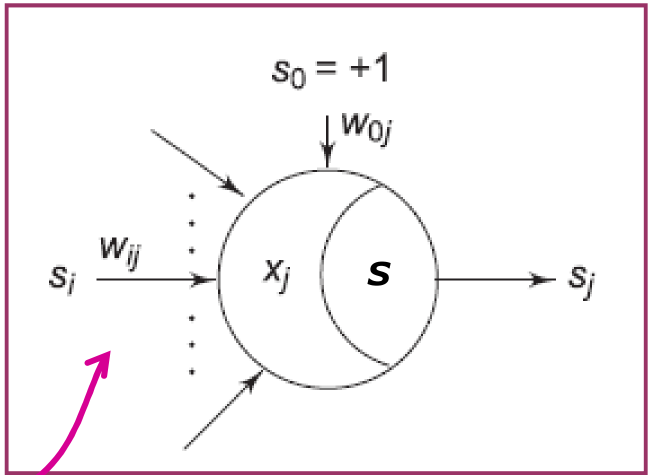
- The activation x_j is simply the binner product of the impinging signal vector $S = (s_0, \ldots, s_n)^T$, with the neuronal weight vector $W_j = (w_{0j}, \ldots, w_{nj})^T$

$$\boxed{x_j = S^T W_j = \sum_{i=0}^{n} w_{ij} s_i}$$

Adaptive Filter

Neuron Abstraction: Weights

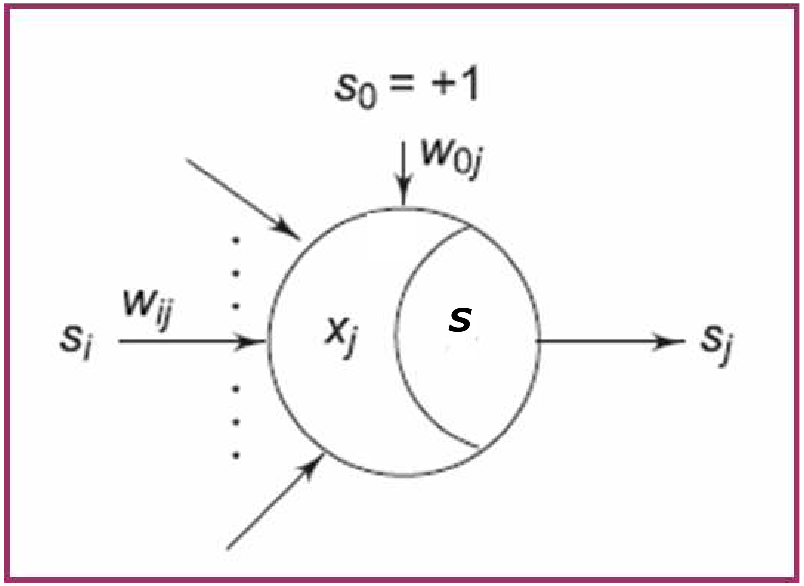
- w_{ij} denotes the weight from neuron *i* to neuron *j* .



$$x_j = \sum_{i=1}^{n} w_{ij} s_i + \theta_j$$

Neuron Abstraction: Weights

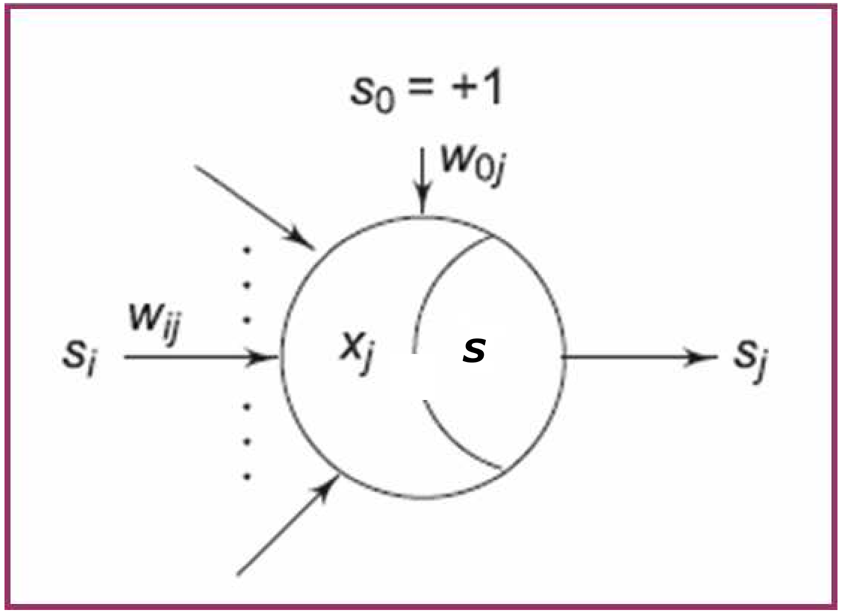
- The j th artificial neuron that connection weights w_{ij} model the synaptic efficacies of various interneuron synapses.



$$x_j = \sum_{i=1}^{n} w_{ij} s_i + \theta_j$$

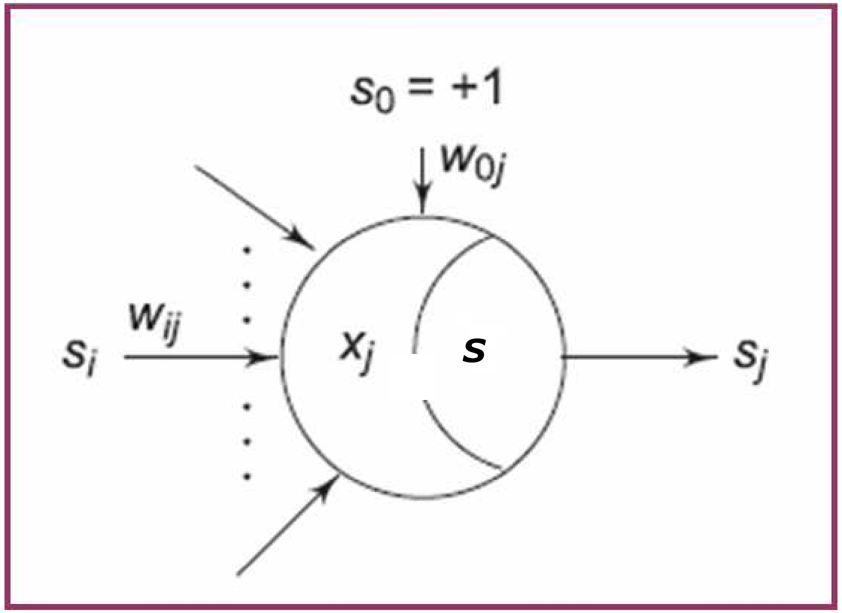
Neuron Abstraction: Signal Function

- The activation of the neuron is subsequently transformed through a *signal function* $S(\cdot)$
- Generates the output signal $s_j = S(x_j)$ of the neuron.



Neuron Abstraction: Signal Function

- A signal function may typically be
  - Binary threshold
  - Linear threshold
  - Sigmoidal
  - Gaussian
  - Probabilistic.



Neuron Signal Functions: Binary Threshold Signal Function

- Net positive activations translate to a +1 signal value
- Net negative activations translate to a 0 signal value.

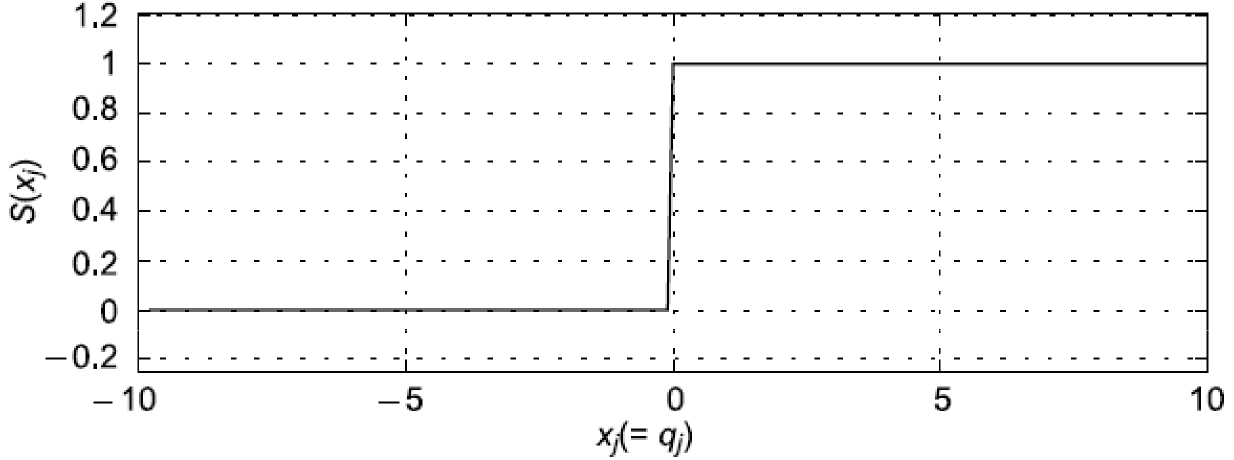
$$\mathcal{S}(x_j) = \begin{cases} 1 & x_j \geq 0 \\ 0 & x_j < 0 \end{cases}$$

Neuron Signal Functions: Binary Threshold Signal Function

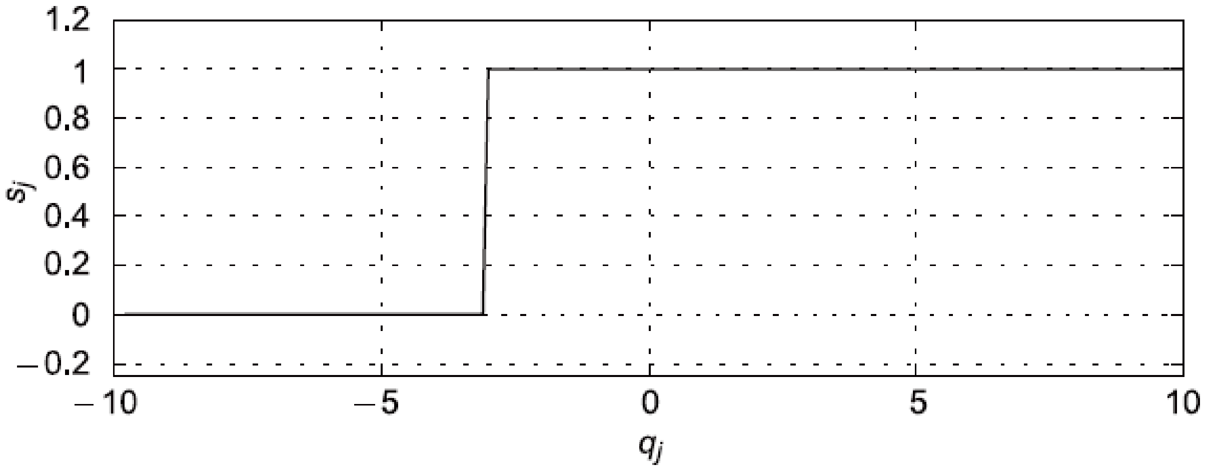
- The threshold logic neuron is a two state machine
  - $s_j = S(x_j) \in \{0, 1\}$

$$S(x_j) = \begin{cases} 1 & x_j \geq 0 \\ 0 & x_j < 0 \end{cases}$$

Neuron Signal Functions: Binary Threshold Signal Function



(a) Binary threshold function: $\theta_j = 0$



(b) Binary threshold function: $\theta_j = +3$

Threshold Logic Neuron (TLN) in Discrete Time

- The updated signal value $S(x_j^{k+1})$ at time instant $k + 1$ is generated from the neuron activation x_i^{k+1} , sampled at time instant $k + 1$.
- The response of the threshold logic neuron as a two-state machine can be extended to the *bipolar* case where the signals are
  - $s_j \in \{-1, 1\}$

$$S(x_j^{k+1}) = \begin{cases} 1 & x_j^{k+1} > 0 \\ S(x_j^k) & x_j^{k+1} = 0 \\ 0 & x_j^{k+1} < 0 \end{cases}$$

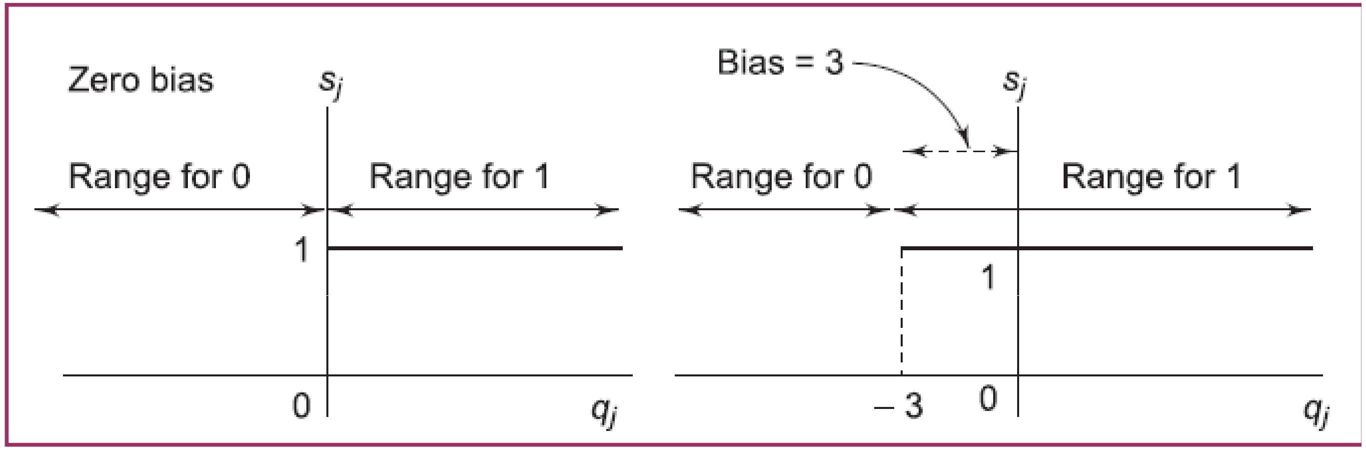
$$S(x_j) = \begin{cases} +1 & x_j > 0 \\ -1 & x_j < 0 \end{cases}$$

Threshold Logic Neuron (TLN) in Discrete Time

- The resulting signal function is then none other than the *signum function*, sign(x) commonly encountered in communication theory.

Interpretation of Threshold

$$x_j = \sum_{i=0}^{n} w_{ij} s_i = \sum_{i=1}^{n} w_{ij} s_i + w_{0j} = q_j + \theta_j$$



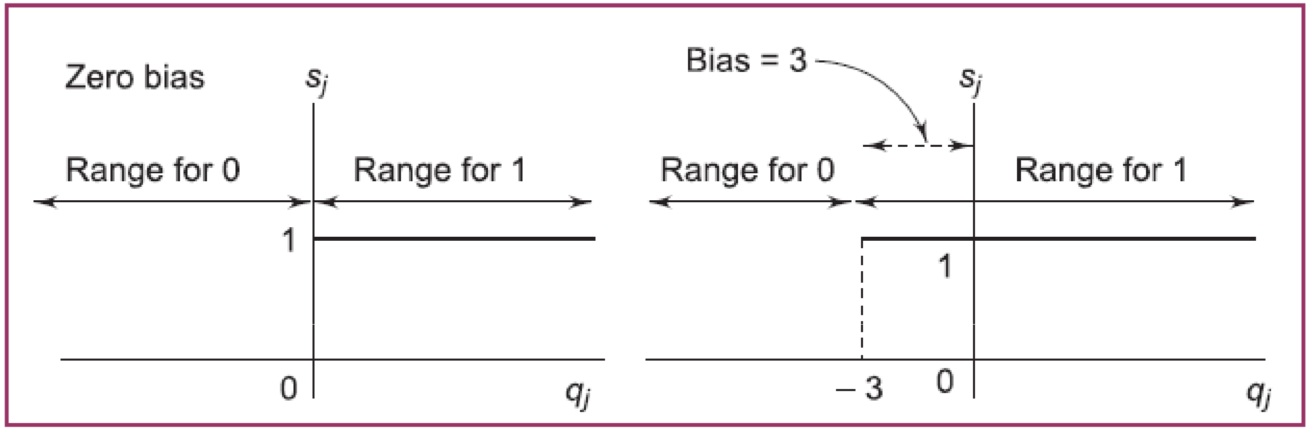
- From the point of view of the net activation x_j
  - The signal is +1 if $x_j = q_j + \theta_j \geq 0$, or $q_j \geq -\theta_j$;
  - And is 0 if $q_j < -\theta_j$.

Interpretation of Threshold

$$x_j = \sum_{i=0}^{n} w_{ij} s_i$$

$$= \sum_{i=1}^{n} w_{ij} s_i + w_{0j}$$

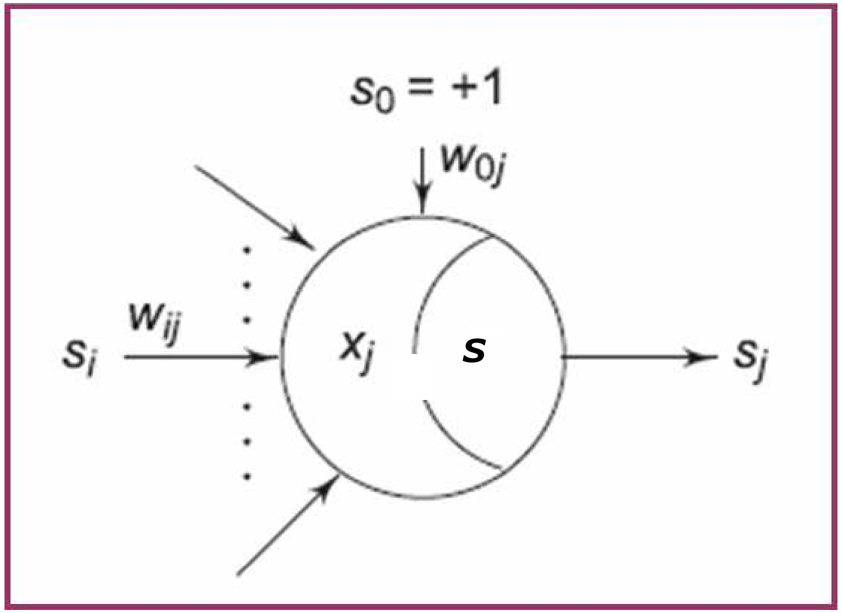
$$= q_j + \theta_j$$



- The neuron thus "compares" the net external input q_j
  - If q_j is greater than the negative threshold, it fires +1, otherwise it fires 0.

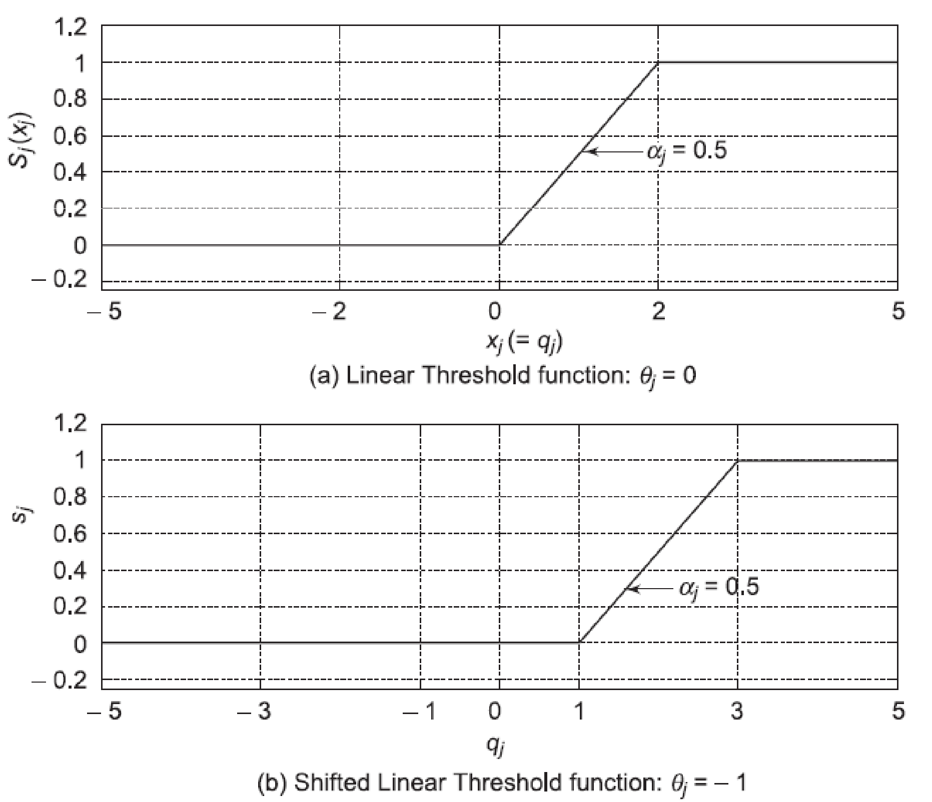
Neuron Abstraction: Signal Function

- A signal function may typically be
  - Binary threshold
  - **Linear threshold**
  - Sigmoidal
  - Gaussian
  - Probabilistic



Linear Threshold Signal Function

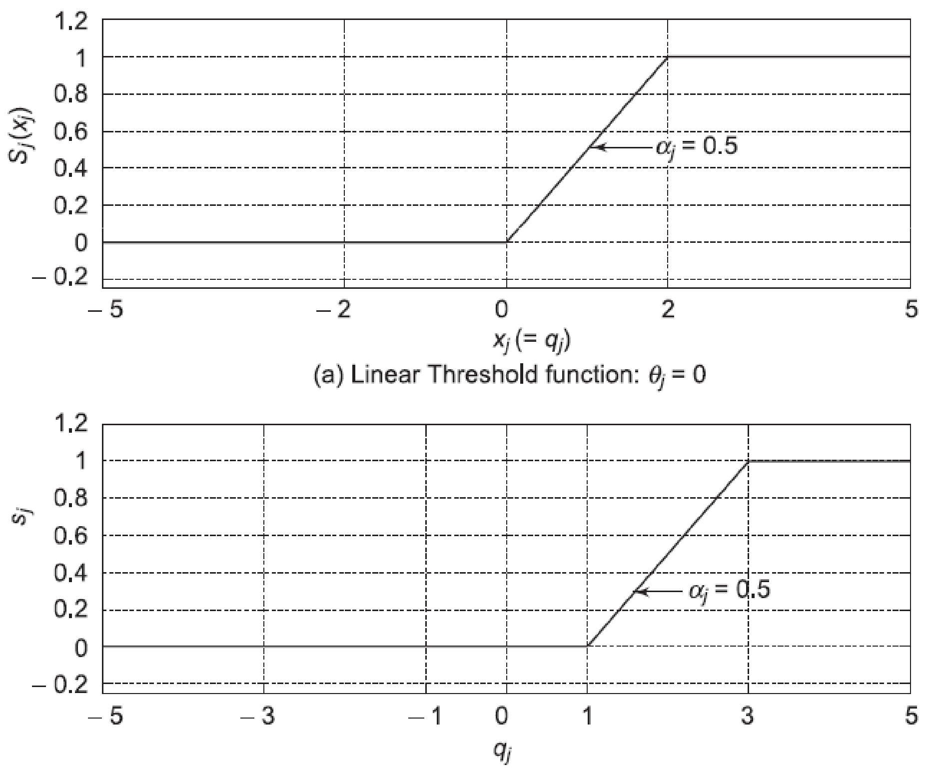
$$S_j(x_j) = \begin{cases} 0 & x_j \le 0 \\ \alpha_j\, x_j & 0 < x_j < x_m \\ 1 & x_j \ge x_m \end{cases}$$



- $\alpha_j = 1/x_m$ is the slope parameter of the function
- Figure plotted for $x_m = 2$ and $\alpha_j = 0.5$. .

Linear Threshold Signal Function

$$S_j(x_j) = \begin{cases} 0 & x_j \leq 0 \\ \alpha_j\, x_j & 0 < x_j < x_m \\ 1 & x_j \geq x_m \end{cases}$$



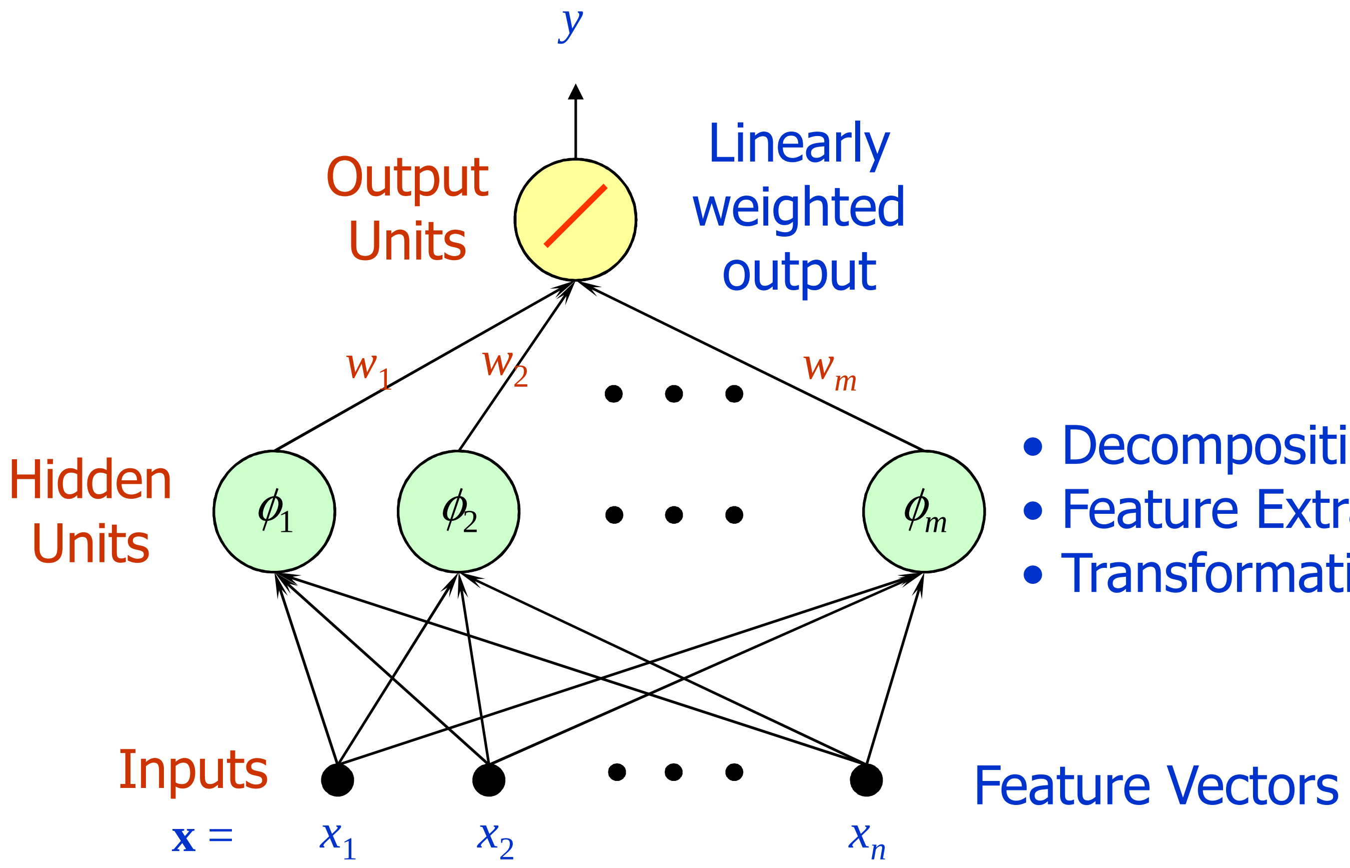
(a) Linear Threshold function: $\theta_j = 0$

(b) Shifted Linear Threshold function: $\theta_j = -1$

- $S_j\,(x_j\,) = \max(0, \min(\alpha_j x_j\,, 1))$
- We assume that neurons within a network are homogeneous.

Linear Models

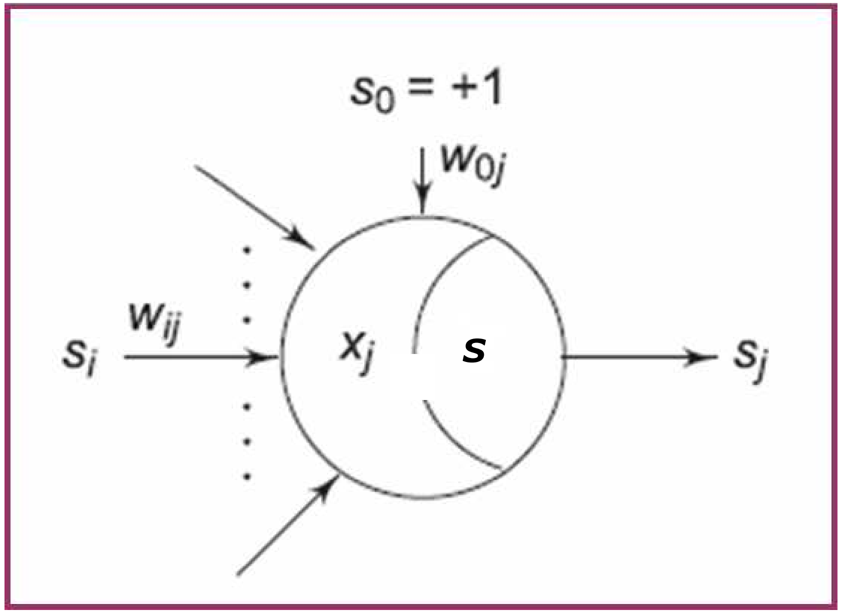
$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$



- Decomposition
- Feature Extraction
- Transformation

Neuron Abstraction: Signal Function

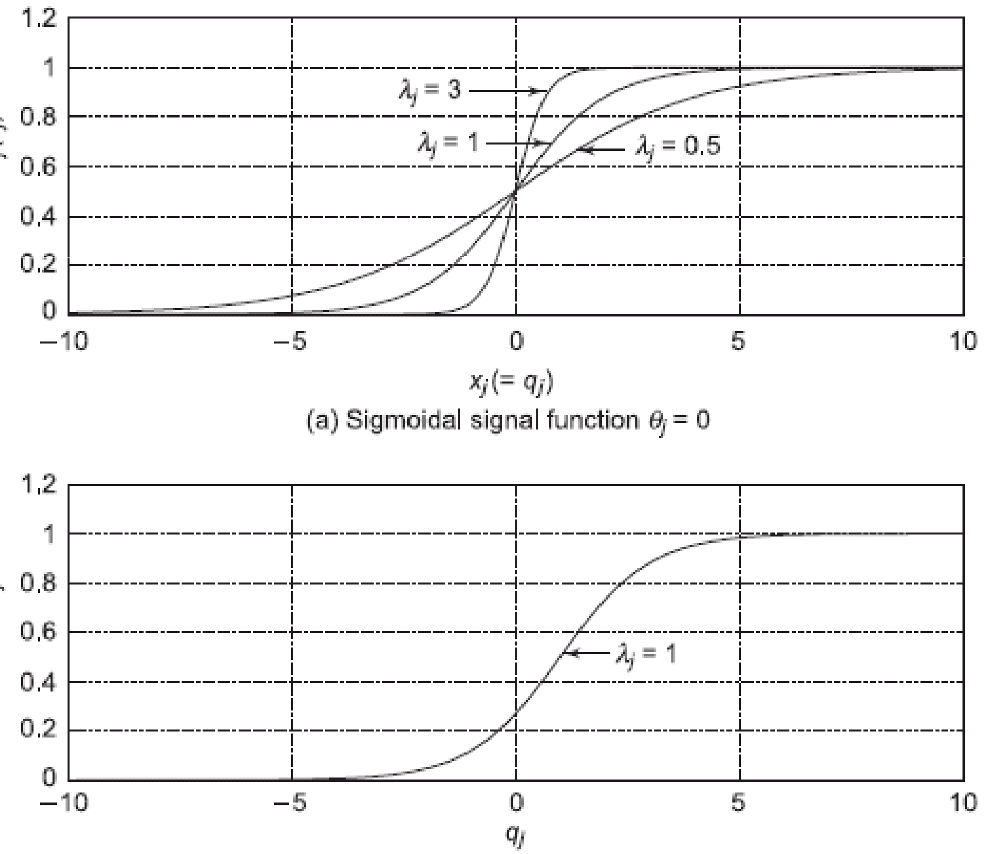
- A signal function may typically be
  - Binary threshold
  - Linear threshold
  - **Sigmoidal**
  - Gaussian
  - Probabilistic



Sigmoidal Signal Function

$$\mathcal{S}_j(x_j) = \frac{1}{1 + e^{-\lambda_j x_j}}$$

- λ_j is a gain scale factor
- In the limit, as $\lambda_j \rightarrow \infty$ the smooth logistic function approaches the non-smooth binary threshold function.



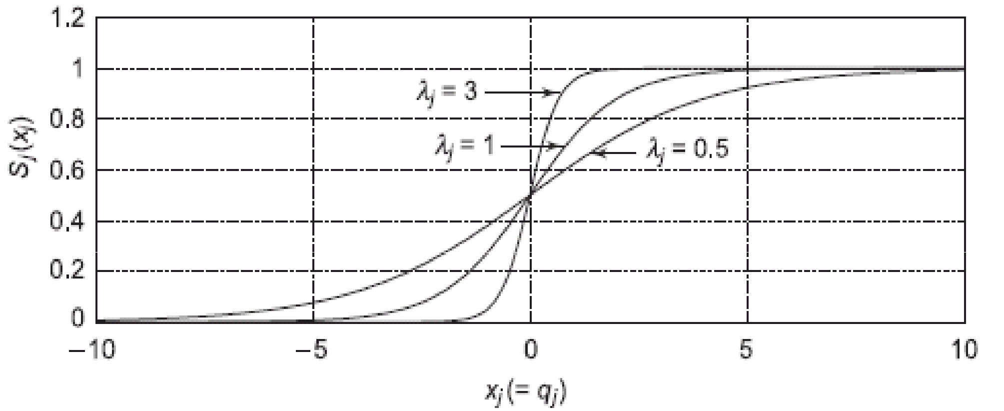
(a) Sigmoidal signal function $\theta_j = 0$

(b) Shifted sigmoidal signal function $\theta_j = -1$

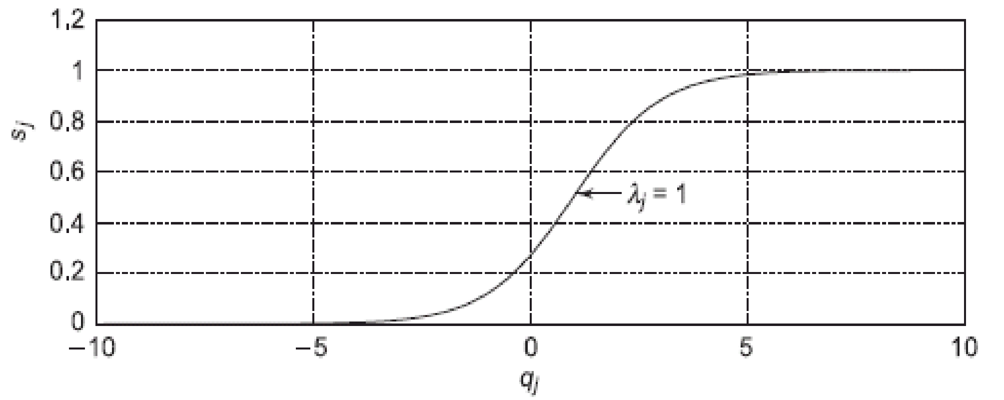
Sigmoidal Signal Function

$$S_j(x_j) = \frac{1}{1 + e^{-\lambda_j x_j}}$$

- The sigmoidal signal function has some very useful mathematical properties.
- It is
  - monotonic
  - continuous
  - bounded



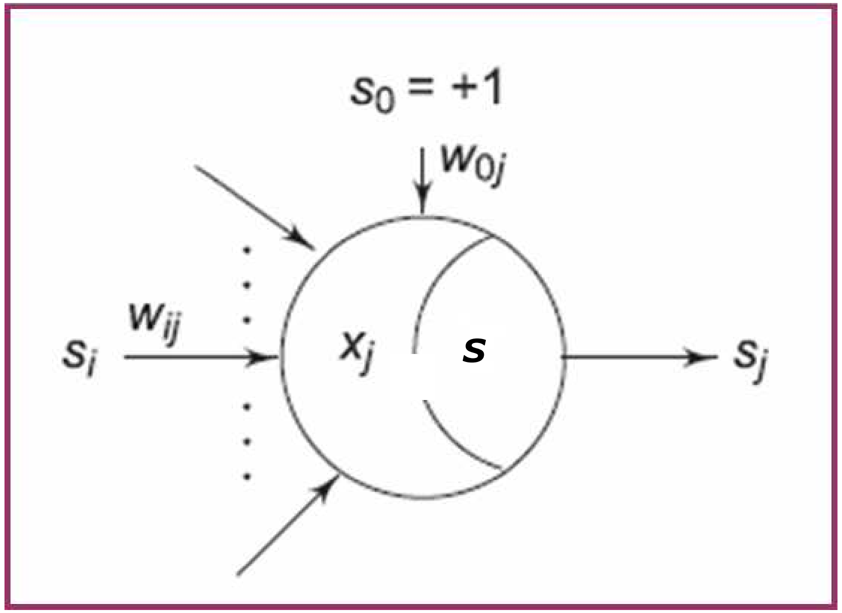
(a) Sigmoidal signal function $\theta_j = 0$



(b) Shifted sigmoidal signal function $\theta_j = -1$

Neuron Abstraction: Signal Function

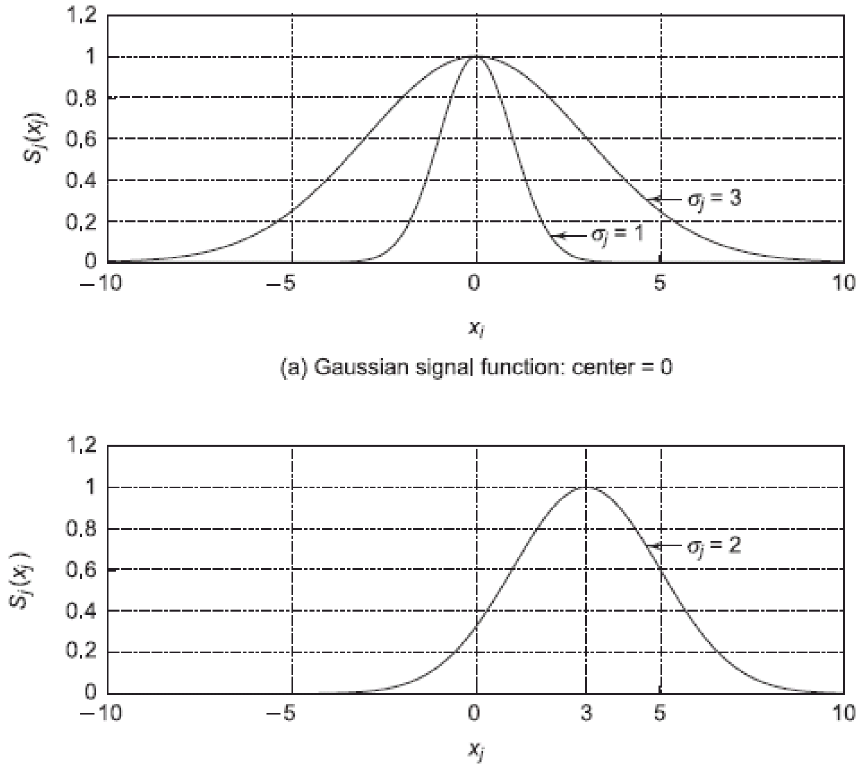
- A signal function may typically be
  - Binary threshold
  - Linear threshold
  - Sigmoidal
  - **Gaussian**
  - Probabilistic



Gaussian Signal Function

$$\mathcal{S}_j(x_j) = \exp\left(-\frac{(x_j - c_j)^2}{2\sigma_j{}^2}\right)$$

- σ_j is the Gaussian spread factor and c_j is the center.
- Varying the spread makes the function sharper or more diffuse.



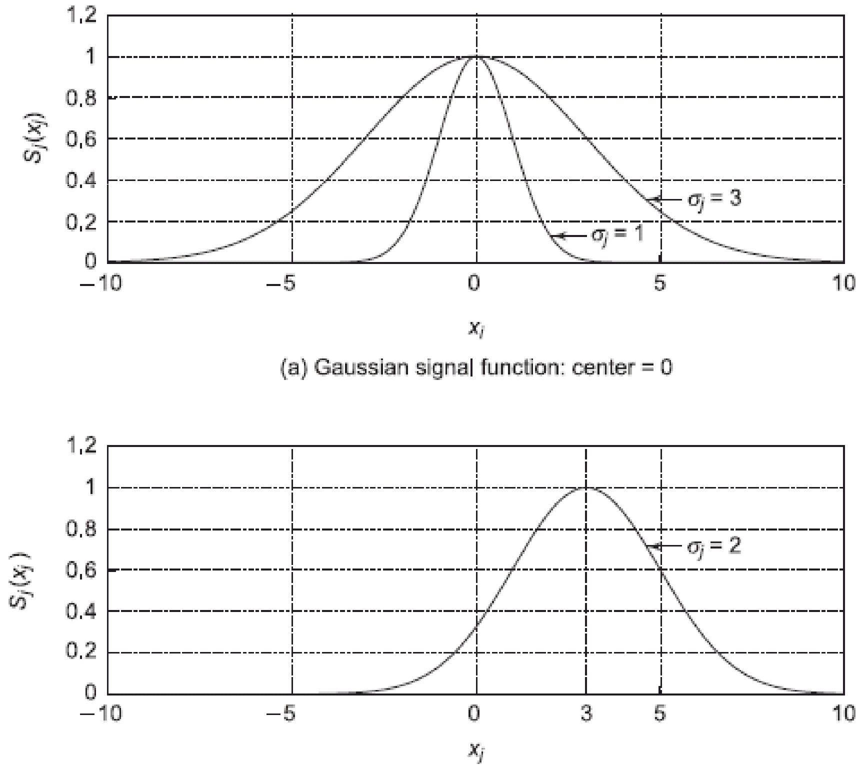
(a) Gaussian signal function: center = 0

(b) Gaussian signal function: center = 3

Gaussian Signal Function

$$\mathcal{S}_j(x_j) = \exp\left(-\frac{(x_j - c_j)^2}{2\sigma_j^2}\right)$$

- Changing the center shifts the function to the right or left along the activation axis
- This function is an example of a *non-monotonic* signal function

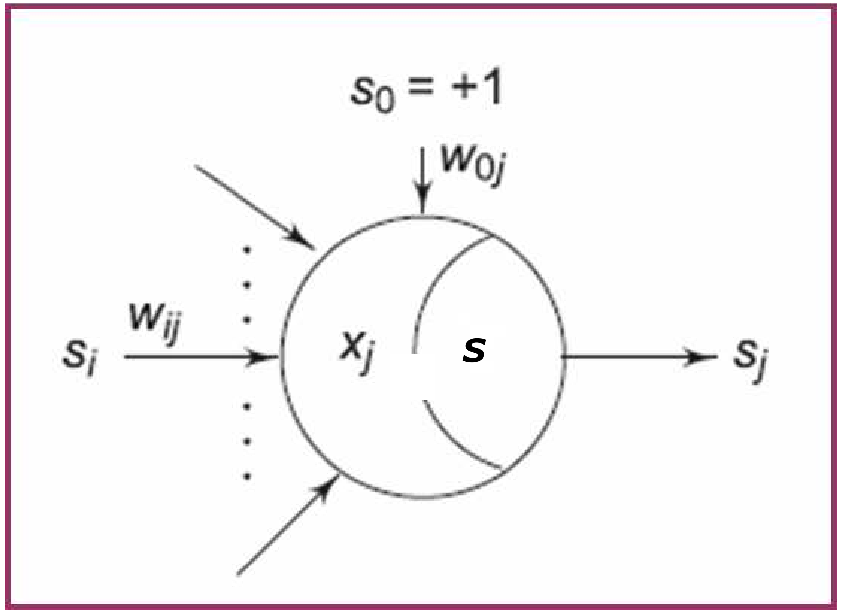


(a) Gaussian signal function: center = 0

(b) Gaussian signal function: center = 3

Neuron Abstraction: Signal Function

- A signal function may typically be
  - Binary threshold
  - Linear threshold
  - Sigmoidal
  - Gaussian
  - **Probabilistic**



Stochastic Neurons

- The signal is assumed to be two state
  - $s_j \in \{0, 1\}$ or $\{-1, 1\}$
- Neuron switches into these states depending upon a *probabilistic function of its activation*, $P(x_j)$.

$$\boxed{P(x_j) = \frac{1}{1 + e^{-x_j/T}}}$$

Summary of Signal Functions

Name	*Function*	*Characteristics*
Binary threshold	$\mathcal{S}(x_j) = \begin{cases} 1 & x_j \geq 0 \\ 0 & x_j < 0 \end{cases}$	Non-differentiable, step-like, $s_j \in \{0, 1\}$
Bipolar threshold	$\mathcal{S}(x_j) = \begin{cases} 1 & x_j \geq 0 \\ -1 & x_j < 0 \end{cases}$	Non-differentiable, step-like, $s_j \in \{-1, 1\}$
Linear	$\mathcal{S}_j(x_j) = \alpha_j x_j$	Differentiable, unbounded, $s_j \in (-\infty, \infty)$
Linear threshold	$\mathcal{S}_j(x_j) = \begin{cases} 0 & x_j \leq 0 \\ \alpha_j\, x_j & 0 < x_j < x_m \\ 1 & x_j \geq x_m \end{cases}$	Differentiable, piece-wise linear, $s_j \in [0, 1]$
Sigmoid	$\mathcal{S}_j(x_j) = \dfrac{1}{1 + e^{-\lambda_j x_j}}$	Differentiable, monotonic, smooth, $s_j \in (0, 1)$
Hyperbolic tangent	$\mathcal{S}_j(x_j) = \tanh(\lambda_j x_j)$	Differentiable, monotonic, smooth, $s_j \in (-1, 1)$
Gaussian	$e^{-(x_j - c_j)^2/2\sigma_j^2}$	Differentiable, non-monotonic, smooth, $s_j \in (0, 1)$
Stochastic	$\mathcal{S}_j(x_j) = \begin{cases} +1 & \text{with probability } P(x_j) \\ -1 & \text{with probability } 1 - P(x_j) \end{cases}$	Non-deterministic step-like, $s_j \in \{0, 1\}$ or $\{-1, 1\}$

Eight Components of Neural Networks

- *Neurons*. These can be of three types:
  - Input: receive external stimuli
  - Hidden: compute intermediate functions
  - Output: generate outputs from the network
- *Activation state vector*. This is a vector of the activation level x_i of individual neurons in the neural network,
  - $X = (x_1, \ldots, x_n)^T \in \mathrm{R}^{\mathrm{n}}$.

Eight Components of Neural Networks

- *Signal function.* A function that generates the output signal of the neuron based on its activation.
- *Pattern of connectivity.* This essentially determines the inter-neuron connection architecture or the graph of the network. Connections which model the inter-neuron synaptic efficacies, can be
  - excitatory (+)
  - inhibitory (−)
  - absent (0).

Eight Components of Neural Networks

- *Activity aggregation rule.* A way of aggregating activity at a neuron, and is usually computed as an inner product of the input vector and the neuron fan-in weight vector.
- *Activation rule.* A function that determines the new activation level of a neuron on the basis of its current activation and its external inputs.

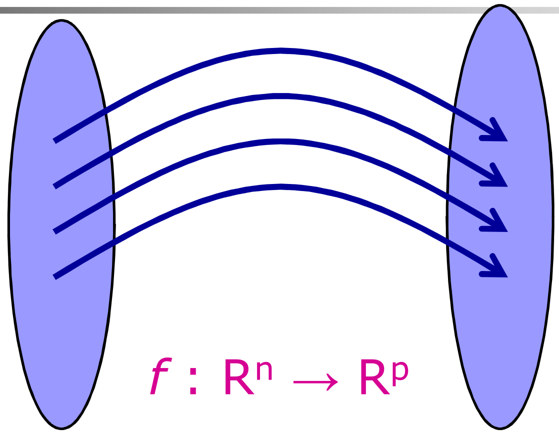
Eight Components of Neural Networks

- *Learning rule.* Provides a means of modifying connection strengths based both on external stimuli and network performance with an aim to improve the latter.
- *Environment.* The environments within which neural networks can operate could be
  - deterministic (noiseless) or
  - stochastic (noisy).

Architectures: Feedforward and Feedback

- Local groups of neurons can be connected in either,
  - A *feedforward* architecture, in which the network has no loops, or
  - A *feedback* (recurrent) architecture, in which loops occur in the network because of feedback connections.

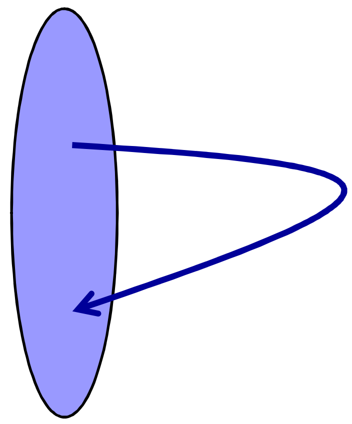
Neural Networks Generate Mappings



- Multilayered networks that associate vectors from one space to vectors of another space are called *heteroassociators*.
  - Map or associate two different patterns with one another—one as input and the other as output. Mathematically we write, $f : R^n \rightarrow R^p$.

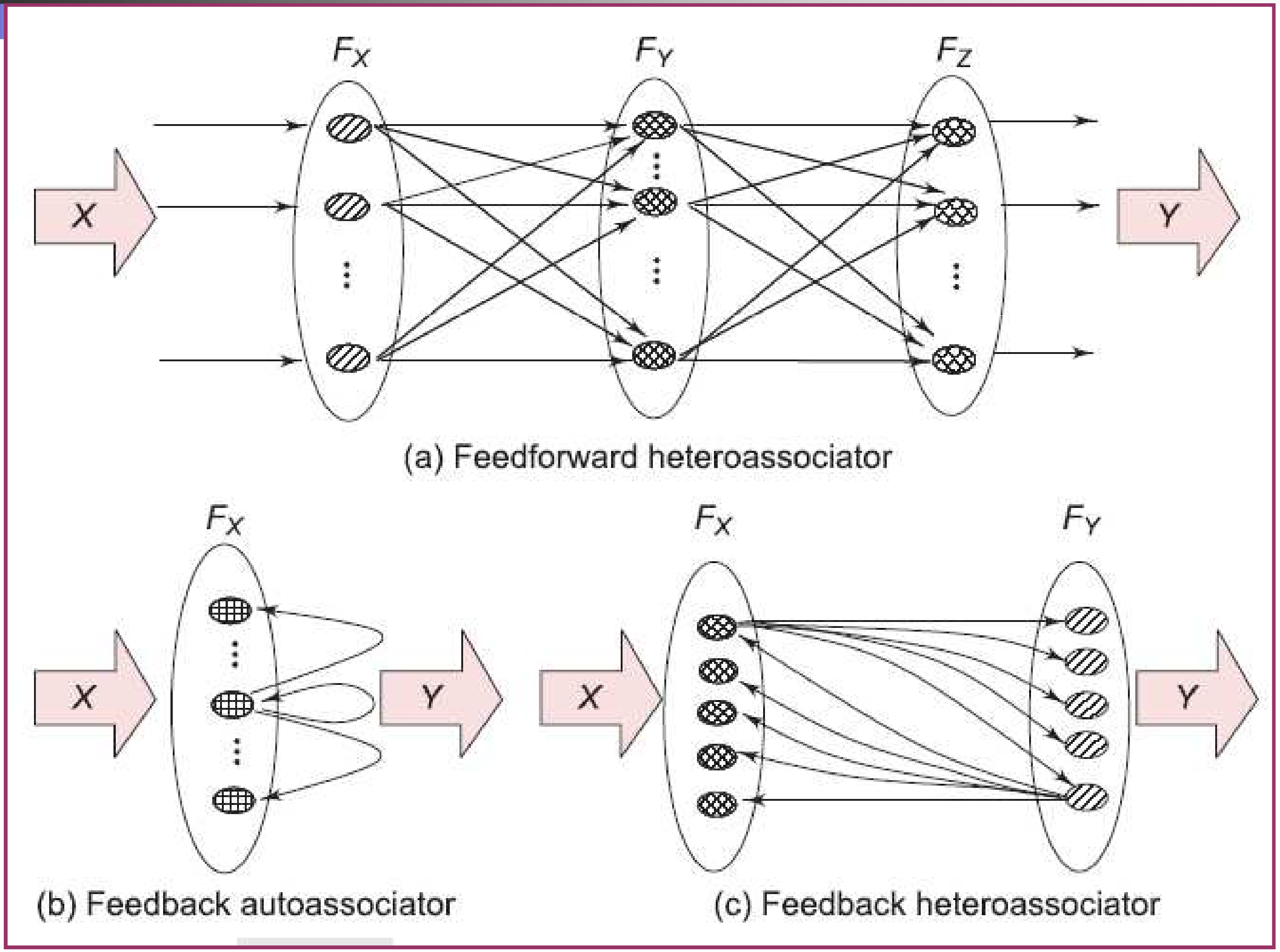
Neural Networks Generate Mappings

- When neurons in a single field connect back onto themselves the resulting network is called an *autoassociator* since it associates a single pattern in R^n with itself.



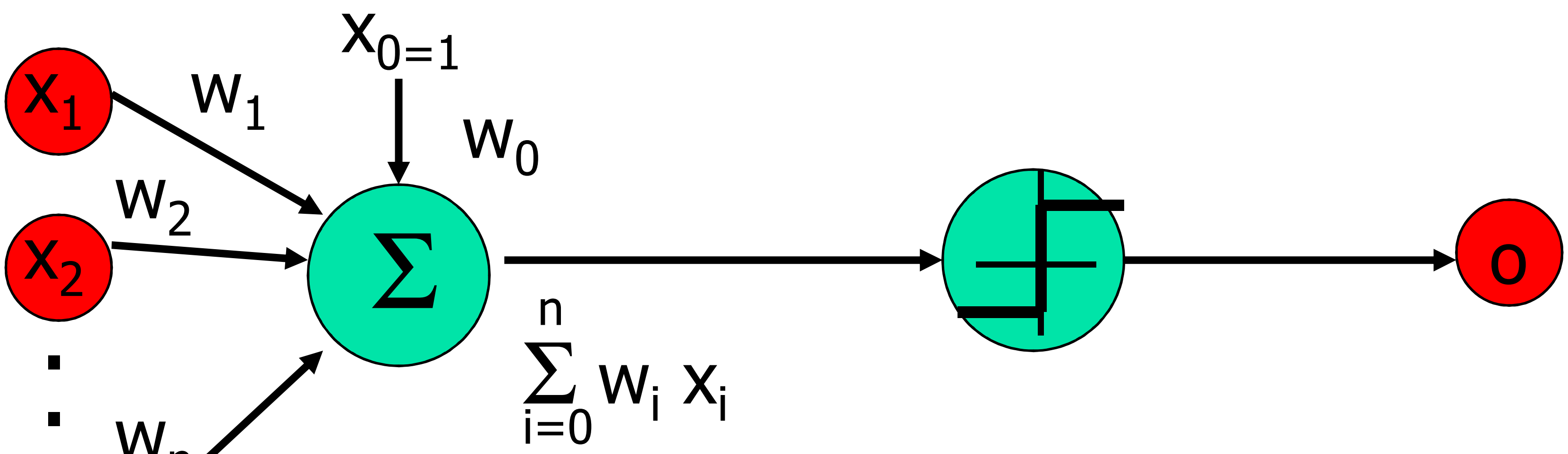
$f : R^n \rightarrow R^n$

Architectures: Feedforward and Feedback



Perceptron

- Linear treshold unit (LTU)



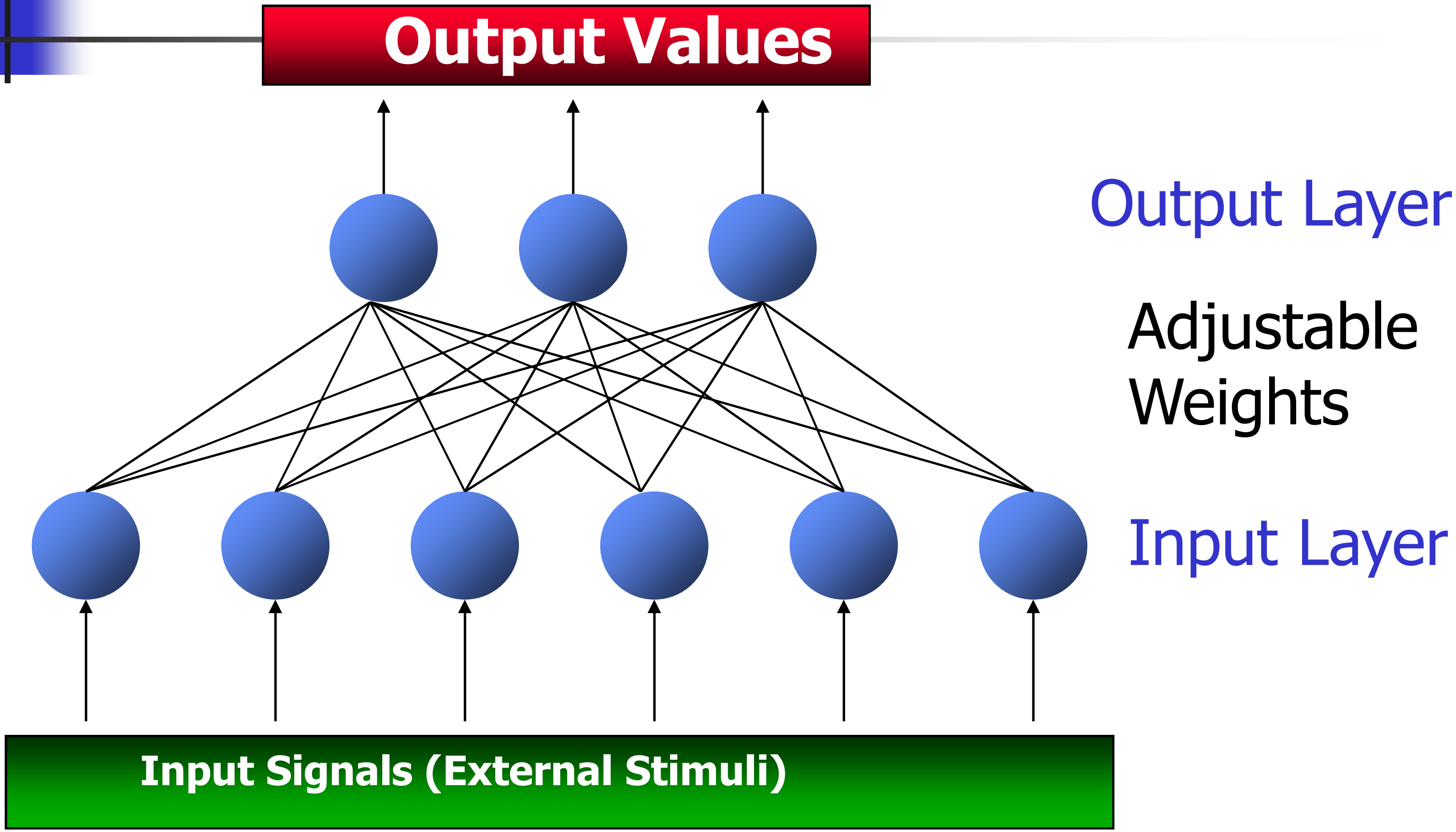
$$o(x_i) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron Learning Rule

$w_i = w_i + \Delta w_i$
$\Delta w_i = \eta\ (t - o)\ x_i$
$t=c(x)$ is the target value
o is the perceptron output
η Is a small constant (e.g. 0.1) called *learning rate*

- If the output is correct ($t=o$) the weights w_i are not changed
- If the output is incorrect ($t \neq o$) the weights w_i are changed such that the output of the perceptron for the new weights is *closer* to t.
- The algorithm converges to the correct classification
    - if the training data is linearly separable
    - and η is sufficiently small

Multilayer Perceptron (MLP)

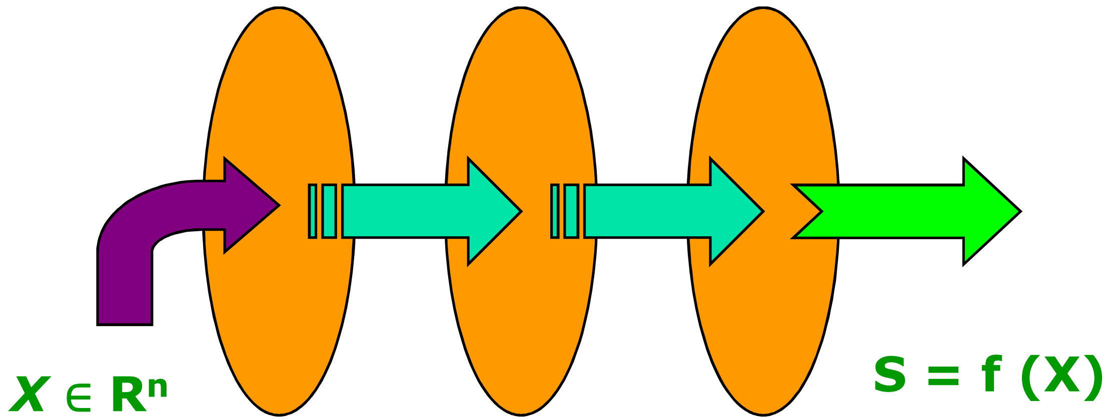


Types of Layers

- The input layer.
  - Introduces input values into the network.
  - No activation function or other processing.
- The hidden layer(s).
  - Perform classification of features
  - Two hidden layers are sufficient to solve any problem
  - Features imply more layers may be better
- The output layer.
  - Functionally just like the hidden layers
  - Outputs are passed on to the world outside the neural network.

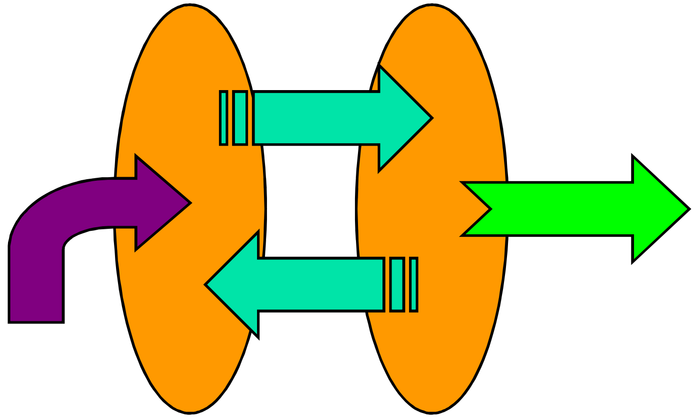
Feedforward vs Feedback: *Multilayer Perceptrons*

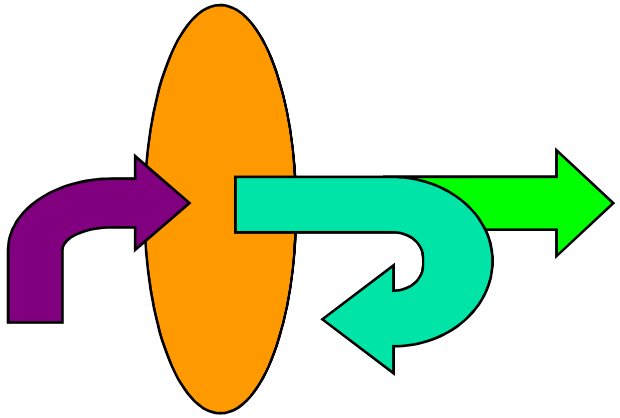
- Organized into different layers
- Unidirectional connections
- memory-less: output depends only on the present input
- Find widespread applications in pattern classification.



Feedforward vs Feedback: Recurrent Neural Networks

- *Non-linear dynamical systems*
- New state of the network is a function of the current input and the present state of the network
- Capable of performing powerful tasks such as
  - pattern completion
  - topological feature mapping
  - pattern recognition





Training Algorithms

- Adjust neural network weights to map inputs to outputs.
- Use a set of sample patterns where the desired output (given the inputs presented) is known.
- The purpose is to learn to generalize
  - Recognize features which are common to good and bad exemplars

Back-Propagation

- A training procedure which allows multi-layer feedforward Neural Networks to be trained;
- Can theoretically perform "any" input-output mapping;
- Can learn to solve linearly inseparable problems.
- In 1969 a method for learning in multi-layer network, Backpropagation, was invented by Bryson and Ho.
- The Backpropagation algorithm is a sensible approach for dividing the contribution of each weight.
- Works basically the same as perceptrons

Backpropagation Network training

- 1. Initialize network with random weights
- 2. For all training cases (called examples):
  - **a.** Present training inputs to network and calculate output
  - **b.** For all layers (starting with output layer, back to input layer):
    - i. Compare network output with correct output (error function)
    - ii. Adapt weights in current layer

This is what you want

Backpropagation Algorithm – Main Idea – error in hidden layers

The ideas of the algorithm can be summarized as follows :

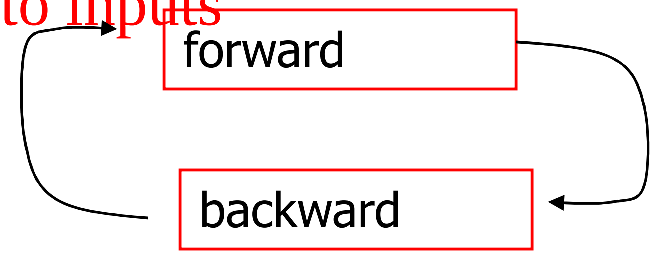
1. Computes the **error term for the output units** using the observed error.

2. From output layer, repeat
   - propagating the error term back to the previous layer and
   - updating the weights between the two layers

   until the earliest hidden layer is reached.

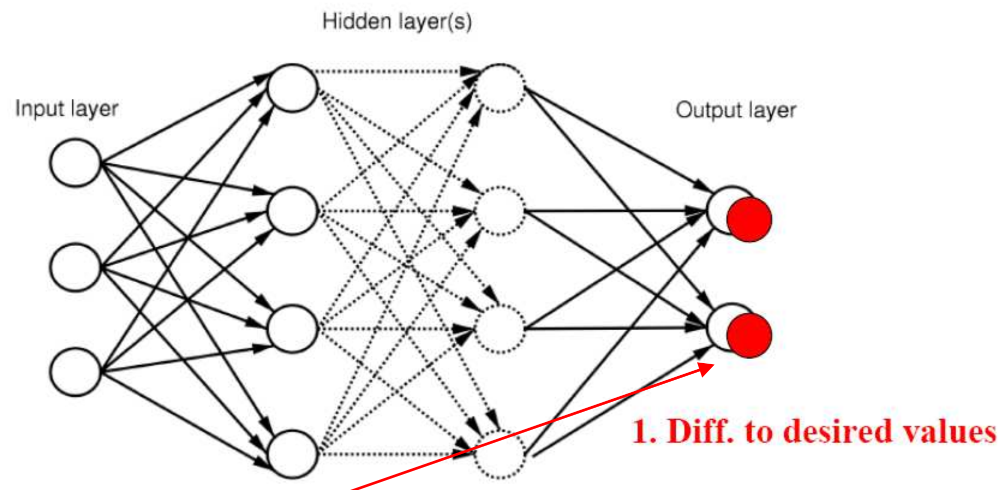
Backpropagation Learning Details

- Method for learning weights in feed-forward (FF) nets
- Can't use Perceptron Learning Rule
  - no teacher values are possible for hidden units
- Use gradient descent to minimize the error
  - propagate deltas to adjust for errors backward from outputs to hidden layers to inputs



Visualization of Backpropagation learning

Backpropagation Learning



Backpropagation Learning

$$E_{\text{out } i} = d_{\text{out } i} - \text{out}_i$$

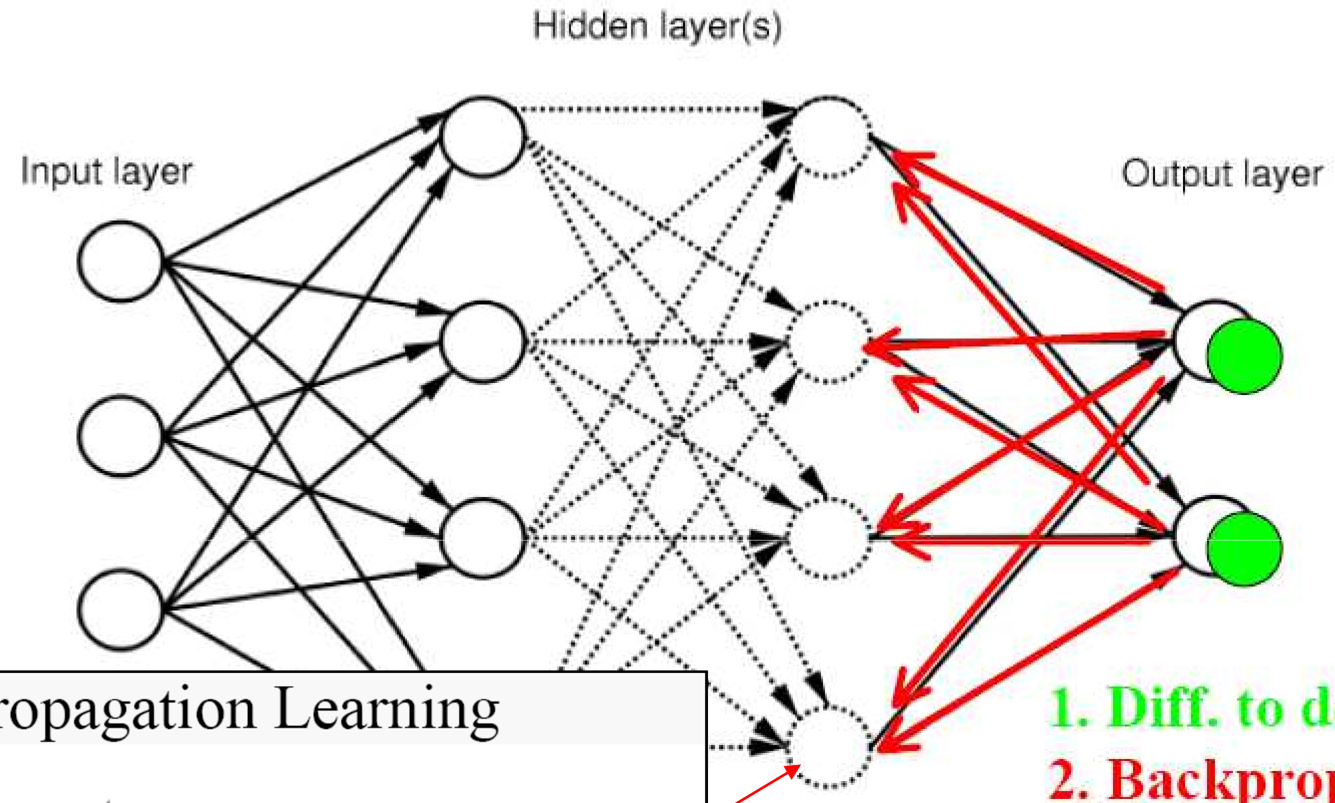
$$E_{\text{total}} = \sum_{i=0}^{\text{num}(n_{\text{out}})} E_{\text{out } i}^2$$

$$E_{\text{hid } i} = \sum_{k=1}^{\text{num}(n_{\text{out}})} E_{\text{out } k} \cdot w_{\text{out } i,k}$$

Backprop output layer

$$\text{diff}_{\text{hid } i} = E_{\text{hid } i} \cdot (1 - o(n_{\text{hid } i})) \cdot o(n_{\text{hid } i})$$

Backpropagation Learning



1. Diff. to desired values

2. Backprop output layer

Backpropagation Learning

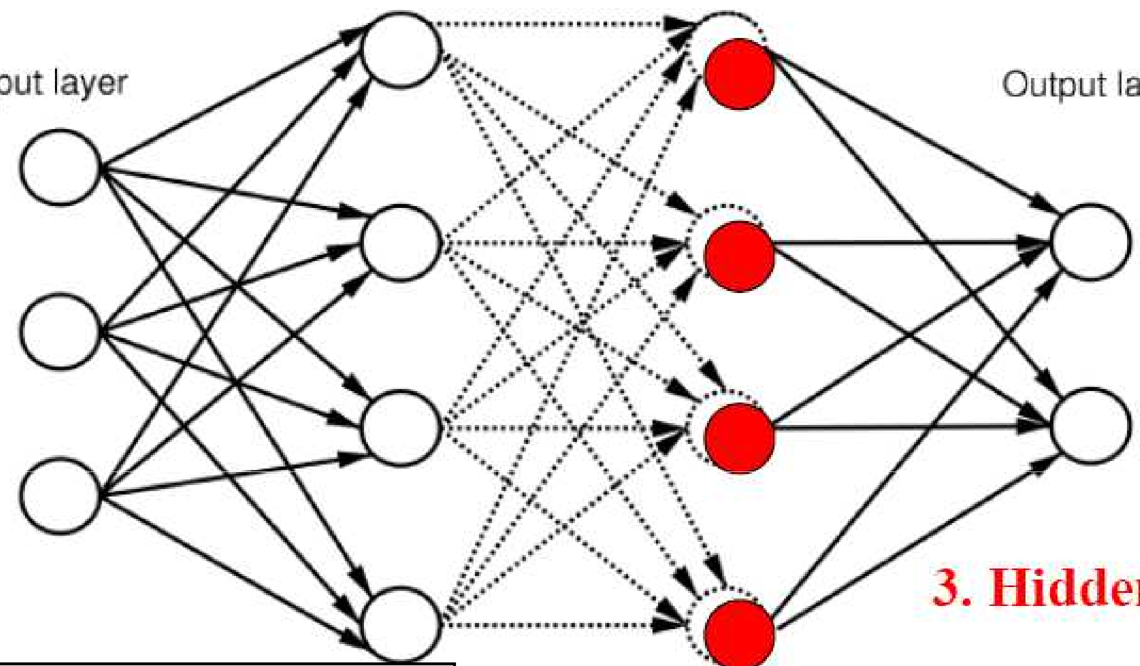
$$E_{out\ i} = d_{out\ i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\ i}^2$$

$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

$$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$$

Backpropagation Learning



3. Hidden error values

Backpropagation Learning

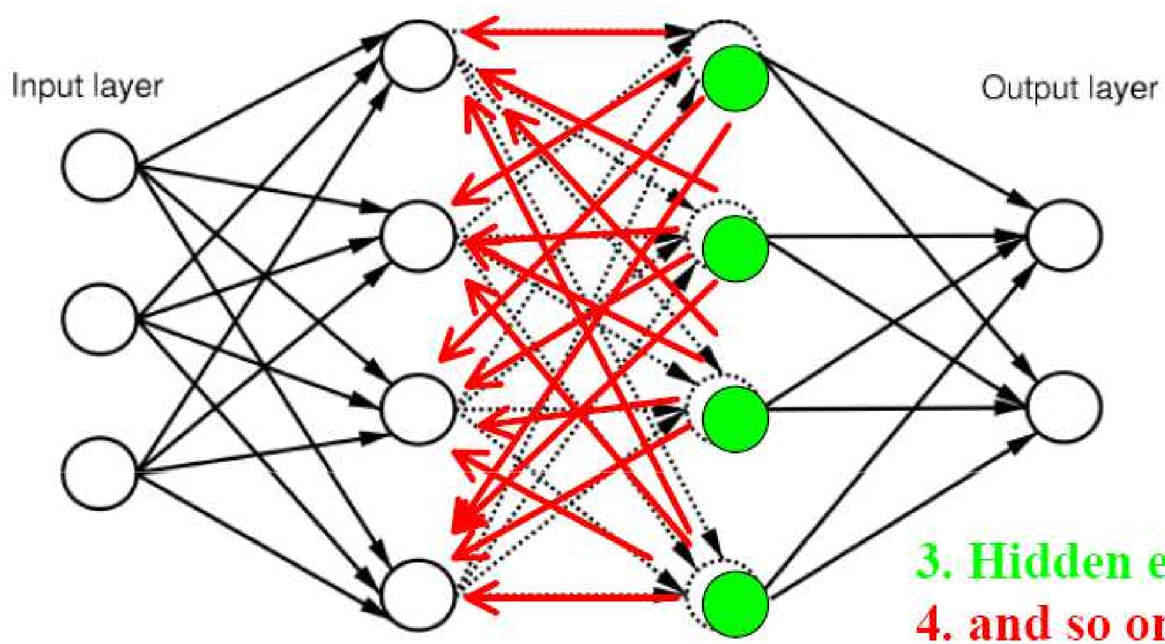
$$E_{out\,i} = d_{out\,i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\,i}^2$$

$$E_{hid\,i} = \sum_{k=1}^{num(n_{out})} E_{out\,k} \cdot w_{out\,i,k}$$

$$diff_{hid\,i} = E_{hid\,i} \cdot (1 - o(n_{hid\,i})) \cdot o(n_{hid\,i})$$

Backpropagation Learning



3. Hidden error values

4. and so on …

Backpropagation Learning

$$E_{out\,i} = d_{out\,i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\,i}^2$$

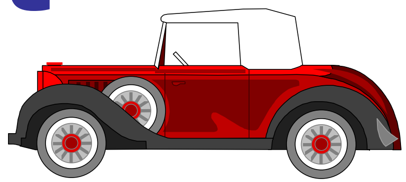
$$E_{hid\,i} = \sum_{k=1}^{num(n_{out})} E_{out\,k} \cdot w_{out\,i,k}$$

$$diff_{hid\,i} = E_{hid\,i} \cdot (1 - o(n_{hid\,i})) \cdot o(n_{hid\,i})$$

Applications

- The properties of neural networks define where they are useful.
  - Can learn complex mappings from inputs to outputs, based solely on samples
  - Difficult to analyse: firm predictions about neural network behaviour difficult;
  - Require limited understanding from trainer, who can be guided by heuristics.

Engine management

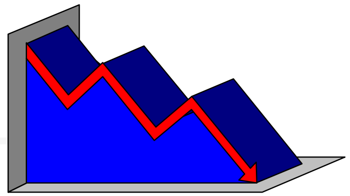


- The behaviour of a car engine is influenced by a large number of parameters
  - temperature at various points
  - fuel/air mixture
  - lubricant viscosity.
- Major companies have used neural networks to dynamically tune an engine depending on current settings.

Signature recognition

- Each person's signature is different.
- There are structural similarities which are difficult to quantify.

Stock market prediction



- “Technical trading” refers to trading based solely on known statistical parameters; e.g. previous price
- Neural networks have been used to attempt to predict changes in prices.
- Difficult to assess success since companies using these techniques are reluctant to disclose information.

Summary

1. Neural network is a computational model that simulate some properties of the human brain.
2. The connections and nature of units determine the behavior of a neural network.
3. Perceptrons are feed-forward networks that can only represent linearly separable functions.
4. Given enough units, any function can be represented by Multi-layer feed-forward networks.
5. Backpropagation learning works on multi-layer feed-forward networks.
6. Neural Networks are widely used in developing artificial learning systems.