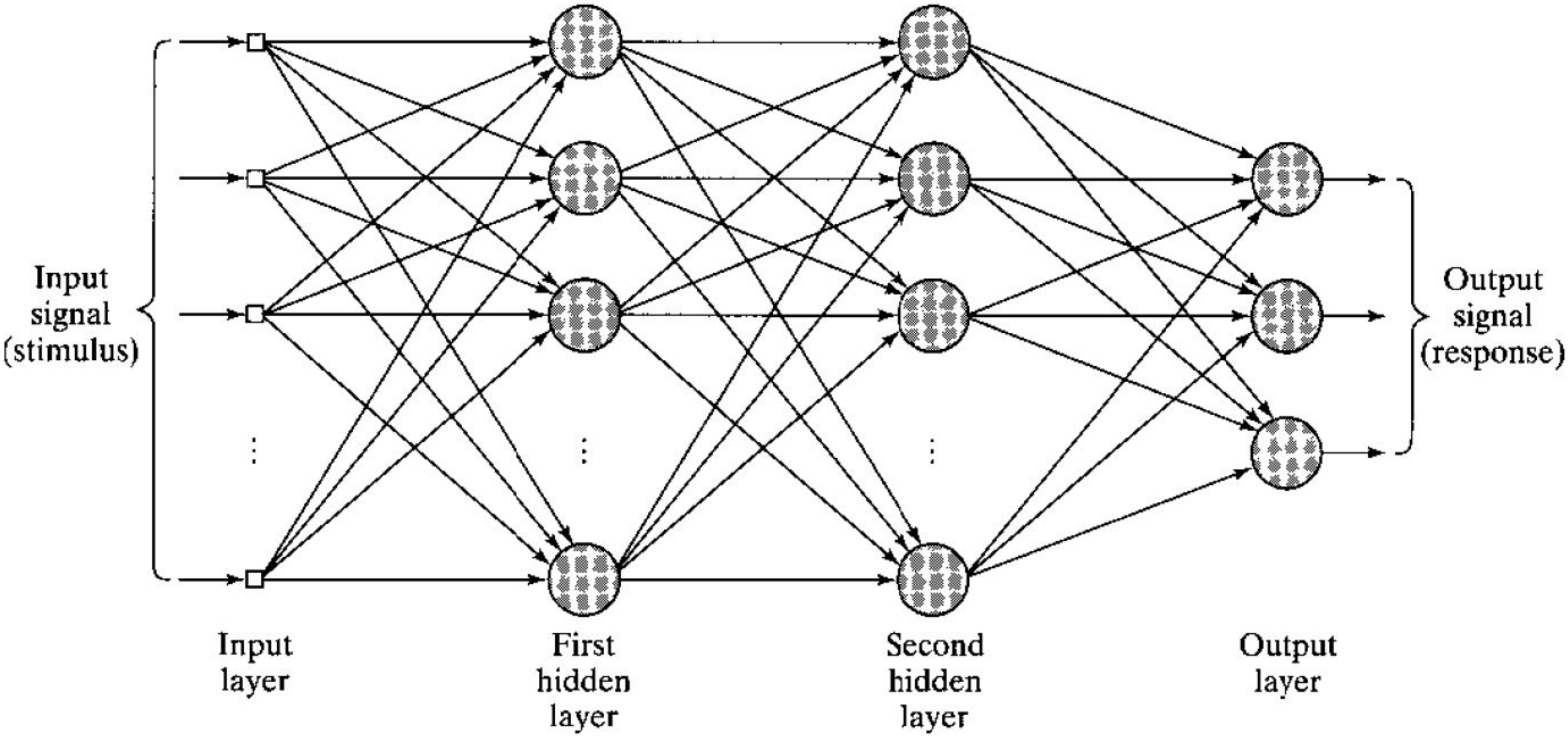
Multilayer Perceptron

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Multilayer Perceptron



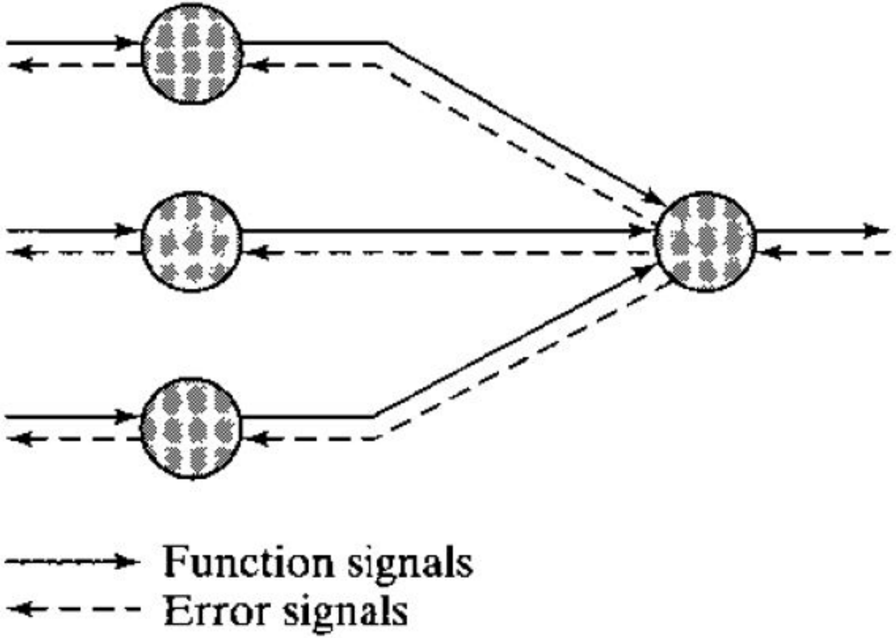
Basic Features

1. Non-linear activation function that is differentiable
2. Hidden Layers
   1. Not part of the input or the output of the network.
3. High degree of connectivity: Fully Connected
   1. The extent of the connections is determined by synaptic weights.

- Theoretical Analysis is difficult
- Learning process is harder to visualize

Phases of MLP

1. Forward Phase: Input/Function Signals
2. Backward Phase: Error Signals



Computations of Hidden/Output Neuron

1. The computation of the function signal appearing at the output of each neuron
2. The computation of an estimate of the gradient vector which is needed for the backward pass.

Hidden Neuron Functions:

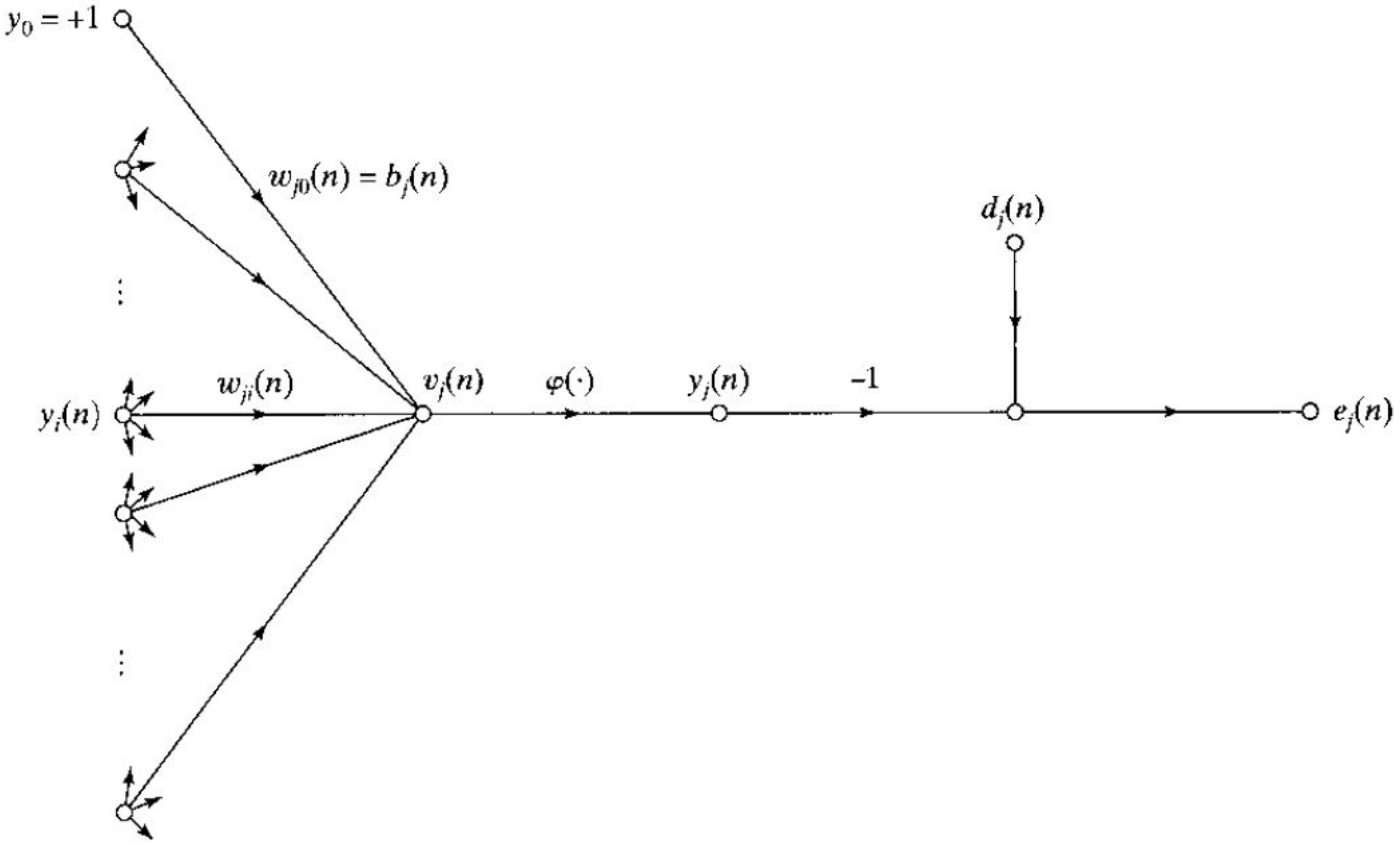
- Act as *feature detectors*
- They gradually discover the salient features that characterize the training data
  - Nonlinear transformation: input data space -> feature space

Credit-Assignment Problem

The problem of assigning *credit* or *blame* for overall outcomes to each of the internal decisions made by the hidden computational units of the distributed learning system, recognizing that those decisions are responsible for the overall outcomes in the first place.

- *Back Propagation Algorithm (BPA)*

Back Propagation Algorithm: Signal Flow



Notations

- Error signal
  - $e_j(n) = d_j(n) - y_j(n)$
- MSE or the instantaneous error energy
  - ع$(n) = 1/2\sum_{j=C} e^2_j(n)$
    - *C: neurons in the output layer*

BPA

- Induced local field $v_j(n)$

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

- Function signal $y_j(n)$

$$y_j(n) = \phi_j(v_j(n))$$

- According to the gradient descent algorithm
    - Weight correction $\Delta w_{ji}(n) \propto \partial \varepsilon(n)/\partial w_{ji}(n)$

$$\boxed{\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}}$$

BPA

- $\partial\mathcal{E}(n)/\partial w_{ji}(n) = \partial\mathcal{E}(n)/\partial e_j(n) * \partial e_j(n)/\partial y_j(n) * \partial y_j(n)/\partial v_j(n) *\partial v_j(n)/\partial w_{ji}(n)$
- $\partial\mathcal{E}(n)/\partial e_j(n) =$
  - $e_j(n)$
- $\partial e_j(n)/\partial y_j(n) =$
  - -1
- $\partial y_j(n)/\partial v_j(n) =$
  - $\varphi'_j(v_j(n))$
- $\partial v_j(n)/\partial w_{ji}(n) =$
  - $y_i(n)$
- $\boldsymbol{\partial\mathcal{E}(n)/\partial w_{ji}(n) = ?}$

BPA

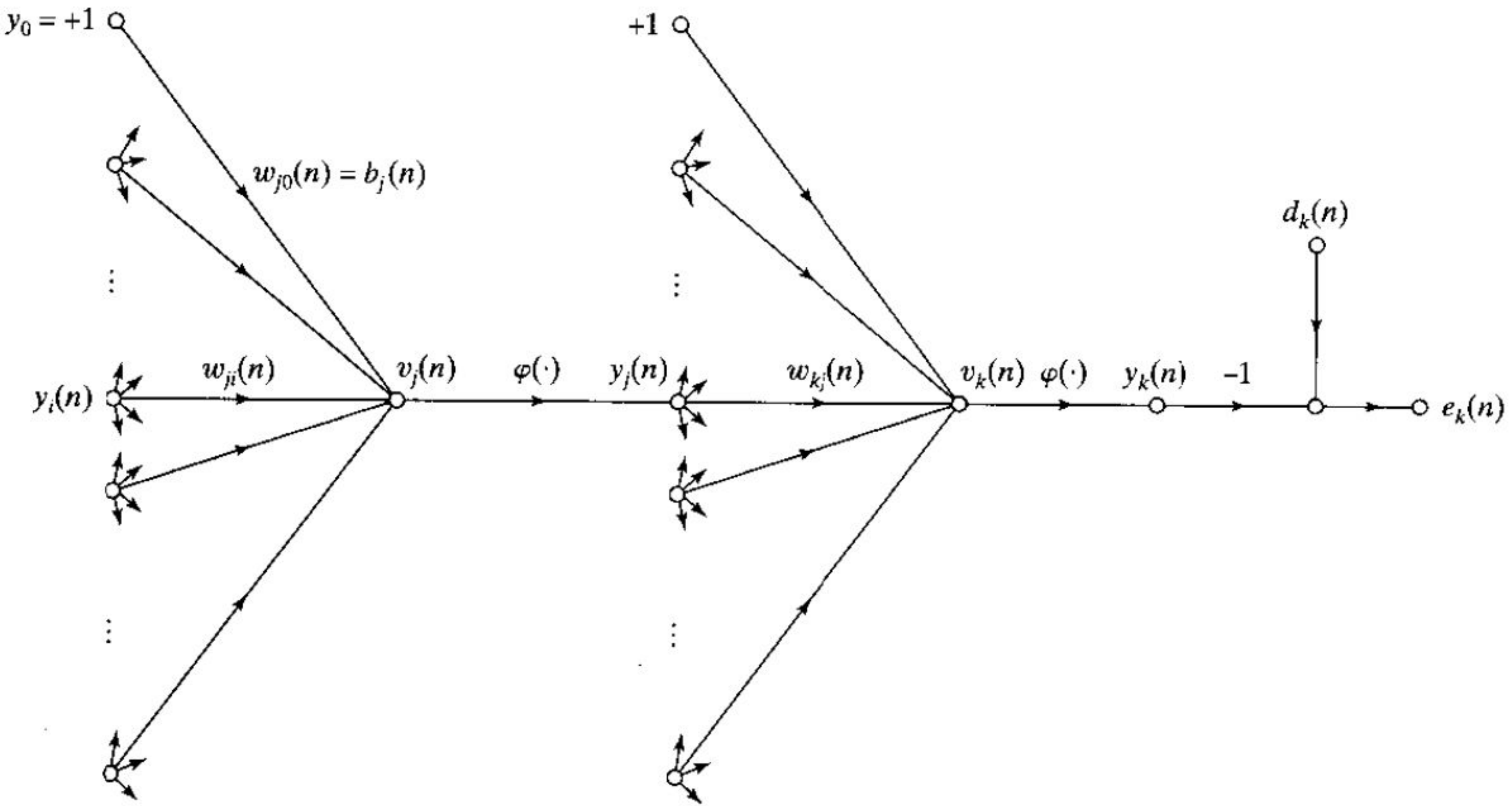
- $\partial \varepsilon(n)/\partial w_{ji}(n) = \partial \varepsilon(n)/\partial e_j(n) * \partial e_j(n)/\partial y_j(n) * \partial y_j(n)/\partial v_j(n) * \partial v_j(n)/\partial w_{ji}(n)$
- $\partial \varepsilon(n)/\partial e_j(n) =$
  - $e_j(n)$
- $\partial e_j(n)/\partial y_j(n) =$
  - -1
- $\partial y_j(n)/\partial v_j(n) =$
  - $\varphi'_j(v_j(n))$
- $\partial v_j(n)/\partial w_{ji}(n) =$
  - $y_i(n)$
- $\boldsymbol{\partial \varepsilon(n)/\partial w_{ji}(n) = -e_j(n)\varphi'_j(v_j(n))y_i(n)}$

BPA

- *Weight correction $\Delta w_{ji}(n)$ is defined by the delta rule*
  - $\Delta w_{ji}(n) = -\eta \partial \mathcal{E}(n) / \partial w_{ji}(n)$
- $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$
  - *Local gradient* $\delta_j(n) = -\partial \mathcal{E}(n) / \partial v_j(n)$

    $= -\partial \mathcal{E}(n) / \partial e_j(n) * \partial e_j(n) / \partial y_i(n) * \partial y_i(n) / \partial v_j(n)$
  - $\boldsymbol{\delta_j(n) = e_j(n) \varphi'_j(v_j(n))}$
- Case 1: Neuron j is an output node
  - $e_j(n) = d_j(n) - y_j(n)$
    - $\boldsymbol{\delta_j(n) = (d_j(n) - y_j(n)) \varphi'_j(v_j(n))}$
- Case 1: Neuron j is a hidden node
  - *$\delta_j(n)$ is recursively determined in terms of the local gradients of all neurons to which neuron j is directly connected*

Back Propagation Algorithm: Signal Flow



Local gradient: Hidden Neuron

- *Local gradient* $\delta_j(n) = -\partial\varepsilon(n)/\partial v_j(n)$

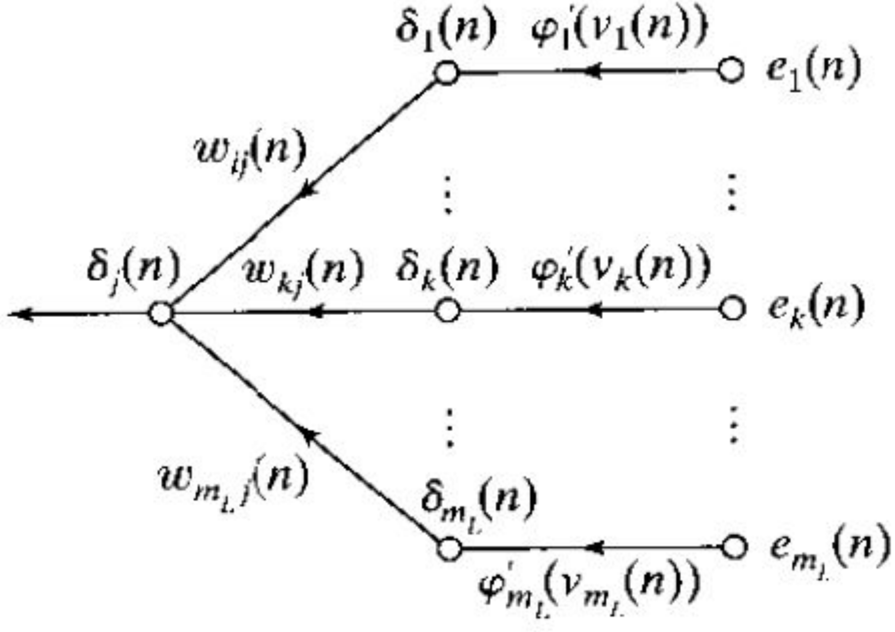
  $= -\partial\varepsilon(n)/\partial y_j(n) * \partial y_j(n)/\partial v_j(n)$

  $= -\partial\varepsilon(n)/\partial y_j(n)\varphi'_j(v_j(n))$
- $\varepsilon(n) = 1/2\sum_{k=C}e^2{}_k(n)$
- $\partial\varepsilon(n)/\partial y_j(n) = \sum_k e_k(n)\partial e_k(n)/\partial y_j(n)$
- *Applying chain rule of partial derivation*
  - $\partial\varepsilon(n)/\partial y_j(n) = \sum_k e_k(n)\partial e_k(n)/\partial v_k(n) * \partial v_k(n)/\partial y_j(n)$
- *From figure:*
  - $e_k(n) = d_k(n) - y_k(n)$
  - $e_k(n) = d_k(n) - \varphi_k(v_k(n))$
- $\partial e_k(n)/\partial v_k(n) =$
  - $- \varphi'_k(v_k(n))$

Local gradient: Hidden Neuron

- $v_k(n) = \sum_m w_{kj}(n) y_j(n)$
  - *m: total number of inputs applied to neuron k*
- $\partial v_k(n)/\partial y_j(n) = w_{kj}(n)$
- $\partial \mathcal{E}(n)/\partial y_j(n) = \sum_k e_k(n) \partial e_k(n)/\partial y_j(n)$
  - $- \sum_k e_k(n) \varphi'_k(v_k(n)) w_{kj}(n)$
  - $- \sum_k \delta_k(n) w_{kj}(n)$
- *Finally,* $\boldsymbol{\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)}$

Error Propagation



- Delta rule
  - $\Delta w_{ji}(n) = \eta \delta_j y_i$
- Output Neuron
  - $\delta_j(n) = (d_j(n) - y_j(n))\varphi'_j(v_j(n))$
- Hidden Neuron
  - $\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$

Two Passes of Computation: Round Trip

- Forward Pass:
  - $y_j(n) = \varphi_j(v_j(n))$
  - $v_k(n) = \sum_m w_{ji}(n) y_i(n)$
    - *m: total number of inputs applied to neuron j*
    - *$w_{ji}(n)$: Synaptic weight connecting neuron i to j*

- *Backward Pass:*
  - $\Delta w_{ji}(n) = \eta \delta_j y_i$
    - *Output Neuron:*
      - $\delta_j(n) = (d_j(n) - y_j(n))\varphi'_j(v_j(n))$
    - *Hidden Neuron:*
      - $\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$

Activation Function-I

- Sigmoidal Function

$$\phi_j(v_j(n)) = \frac{1}{1+exp(-av_j(n))}, a > 0; -\infty < v_j(n) < \infty$$

- Differentiation:

$$\varphi_j'(v_j(n)) = \frac{a\exp(-av_j(n))}{[1 + \exp(-av_j(n))]^2}$$

- $y_j(n) = \varphi_j(v_j(n))$

$$\varphi_j'(v_j(n)) = ay_j(n)[1 - y_j(n)]$$

Local Gradient: Sigmoidal Function

- $\Delta w_{ji}(n) = \eta \delta_j y_i$
- *Output Neuron:*
  - $\delta_j(n) = (d_j(n) - y_j(n))\varphi'_j(v_j(n))$
  - $y_j(n) = o_j(n)$
    - $\boldsymbol{\delta_j(n) = a[d_j(n) - o_j(n)]o_j(n)[1\text{-}o_j(n)]}$

- *Hidden Neuron:*
  - $\delta_j(n) = \varphi'_j(v_j(n))\sum_k \delta_k(n) w_{kj}(n)$
    - $\boldsymbol{\delta_j(n) =\ ?}$

- *Maximum and Minimum of* $\varphi'_j(v_j(n))$*?*
  - *Synaptic weights are changed most where the function signals are in their mid-range.*

Example:XOR Problem

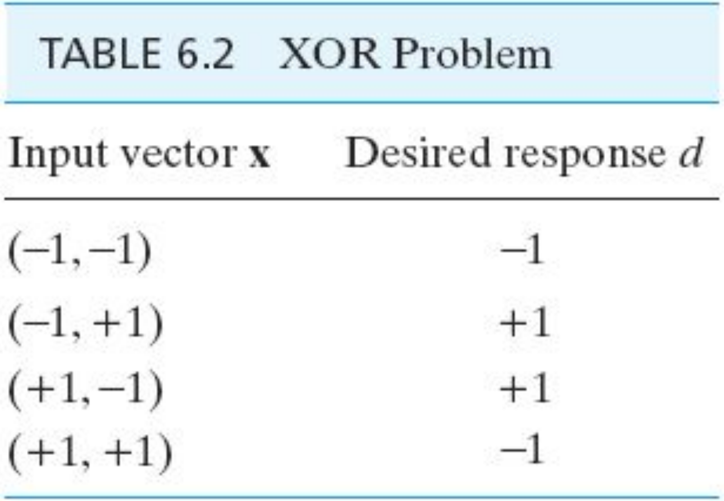
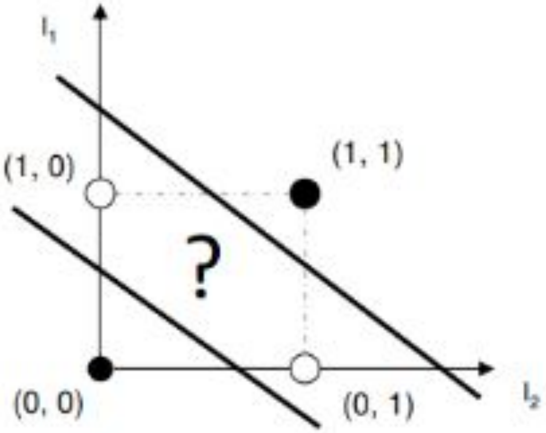
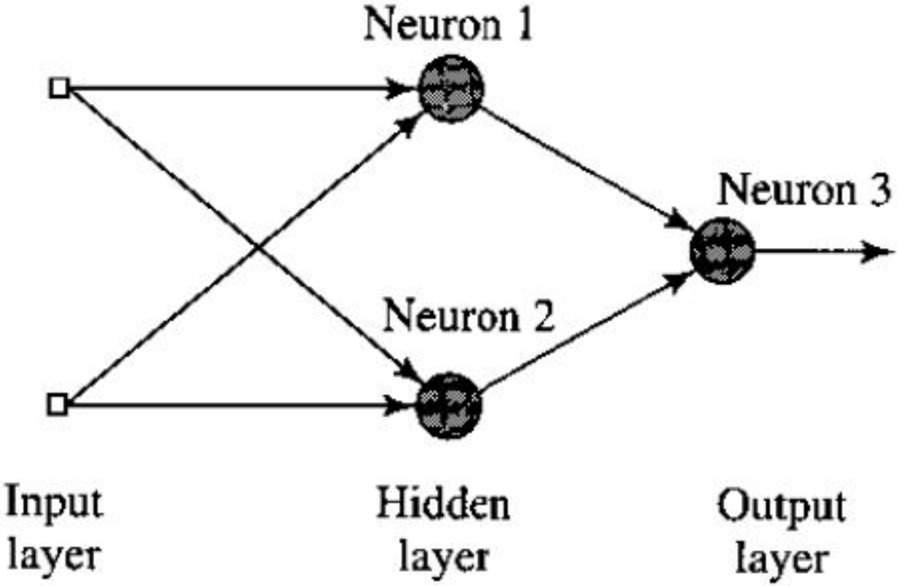


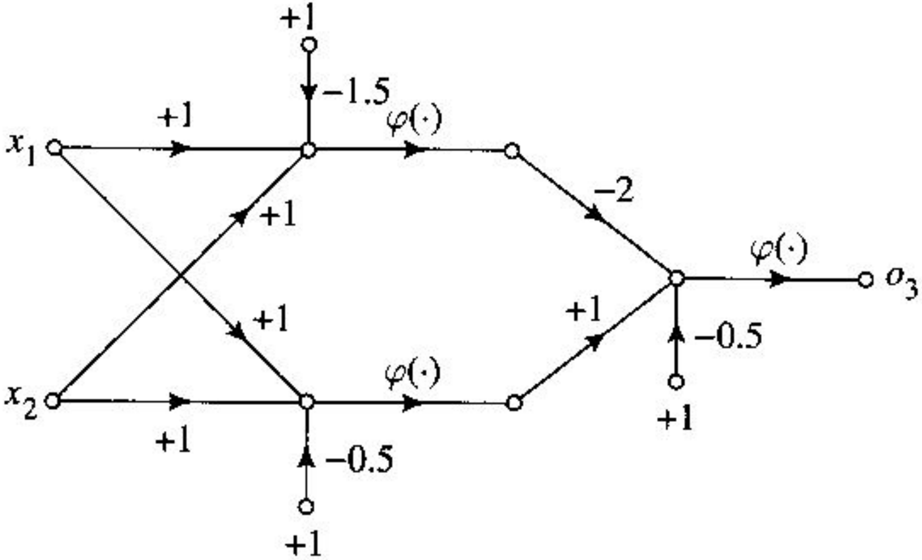
TABLE 6.2 XOR Problem

Input vector **x**	Desired response *d*
(–1, –1)	–1
(–1, +1)	+1
(+1, –1)	+1
(+1, +1)	–1

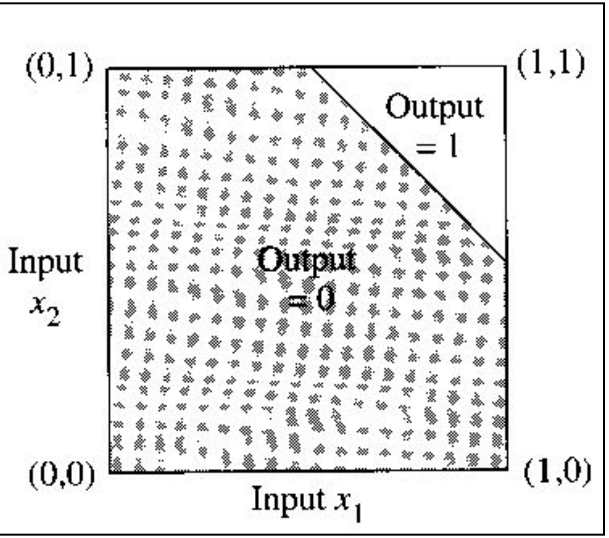


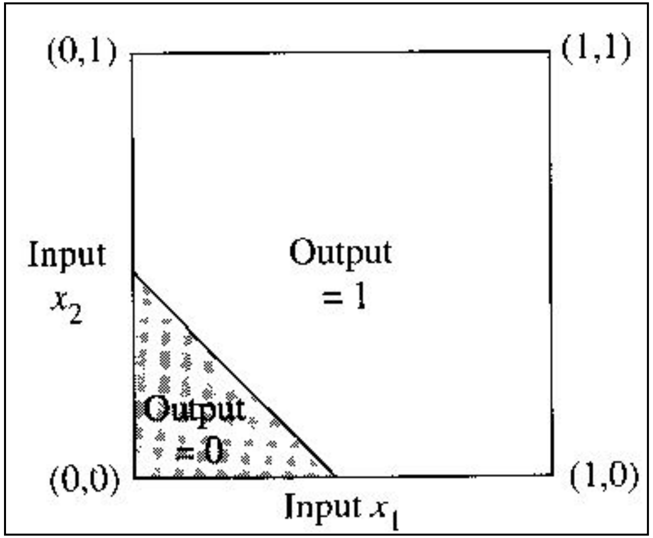
Example:XOR Problem

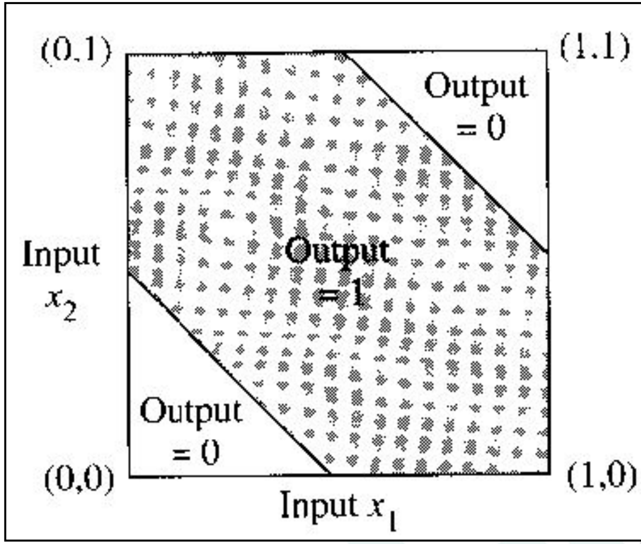




XOR Problem: Decision Boundary







Practical Considerations

- How are the weights initialised?
- How is the learning rate chosen?
- How many hidden layers and how many neurons?
- Which activation function ?
- How to preprocess the data ?
- How many examples in the training data set?

Initial Weights

- We generally start off all the weights with small random values.
    - Gaussian distribution around zero with standard deviation σ.

- If synaptic weights are assigned large initial values neurons are driven into saturation.

- If synaptic weights are assigned small initial values algorithms operate around the origin.

Learning Rate

- Values between 0.1 and 0.9 have been used in many applications.
- Smaller learning rate -> slower rate of learning
    - Smaller changes in the synaptic weights
        - Smoother trajectory in the weight space
- Larger learning rate -> speed up the rate of learning
    - Larger changes in the synaptic weights
        - Unstable (oscillatory) network
- There is no necessity to keep the learning rate fixed throughout the learning process: $\eta(t) = \eta(1)/t$
- Generalized Delta Rule: $\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_j(n)$
        - *α: Momentum constant*

Number of Layers and Neurons

- The number of layers and of neurons depend on the specific task. In practice this issue is solved by trial and error.

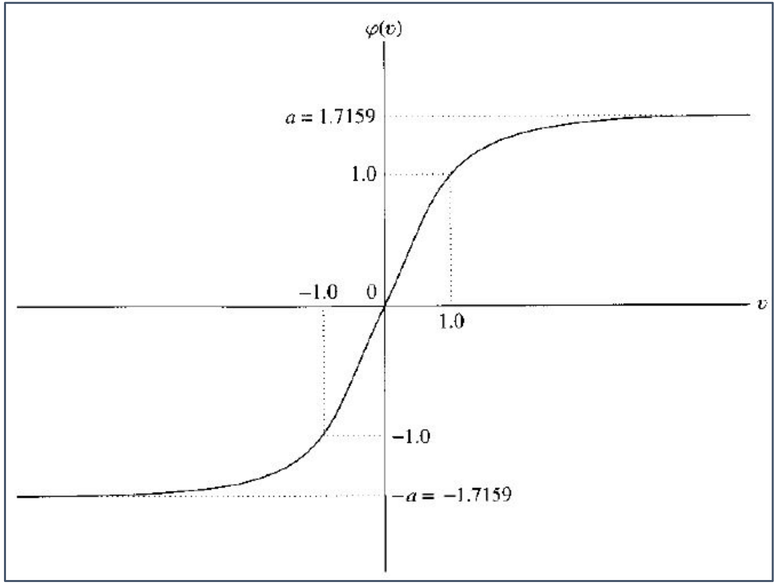
- Two types of adaptive algorithms can be used:
    - start from a large network and successively remove some neurons and links until network performance degrades.
    - begin with a small network and introduce new neurons until performance is satisfactory.

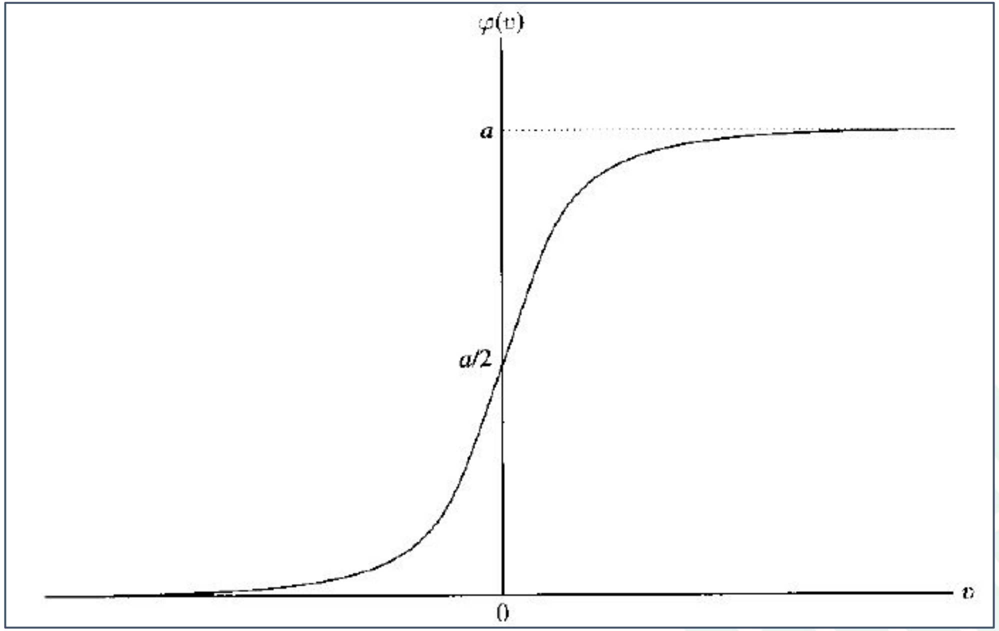
Maximizing Information Content

- Maximization of information content: every training example presented to the backpropagation algorithm must maximize the information content.
  - The use of an example that results in the largest training error.
  - The use of an example that is radically different from all those previously used.

Activation Function

- Network learns faster with antisymmetric functions when compared to non symmetric functions.
    - $\varphi(-v) = -\varphi(v)$
        - $\varphi(v) = a\tanh(bv)$



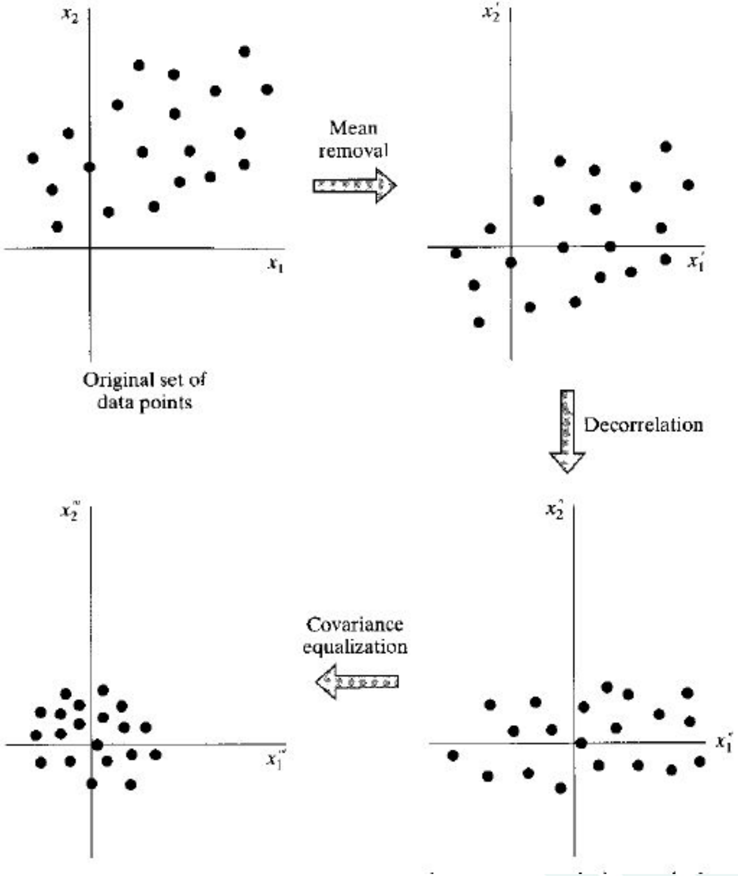


Target Values

- Target values: target values must be chosen within the range of the activation function.
- For the antisymmetric activation function it is necessary to design ε
  - For a+:
    - $d_j = a - \varepsilon$
  - For –a:
    - $d_j = - a + \varepsilon$
- If a=1.7159 we can set ε =0.7159 then d=±1
- We must use targets of +1 and –1 rather than 0 and 1.

Normalization

- Each input variable should be preprocessed so that its mean value, averaged over the entire training sample is zero.
- The input variables in the training data should be uncorrelated.
- The decorrelated input variables should be scaled so that their covariances are approximately equal.



References

1. Chapter 6, 6.3-6.5, Neural Networks: A Comprehensive Foundation (2nd Edition) 2nd Edition by Simon Haykin
2. https://www.youtube.com/watch?v=nz3NYD73H6E&list=PL3EA65335EAC29EE8&index=19
3. https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec13.pdf
4. https://people.eecs.berkeley.edu/~jordan/kernels/0521813972c03_p47-84.pdf
5. http://www.cs.bham.ac.uk/~jxb/NN/l6.pdf
6. https://web.stanford.edu/group/pdplab/originalpdphandbook/Chapter%205.pdf
7. http://francky.me/aifaq/FAQ-comp.ai.neural-net.pdf
8. http://www.iro.umontreal.ca/~pift6266/A06/refs/YannNipsTutorial.pdf