Multilayer Perceptron

Sanjay Singh[⋆†]

[⋆]Department of Information and Communication Technology
Manipal Institute of Technology, Manipal University
Manipal-576104, INDIA
sanjay.singh@manipal.edu

[†]Centre for Artificial and Machine Intelligence (CAMI)
Manipal University, Manipal-576104, INDIA

February 28, 2019

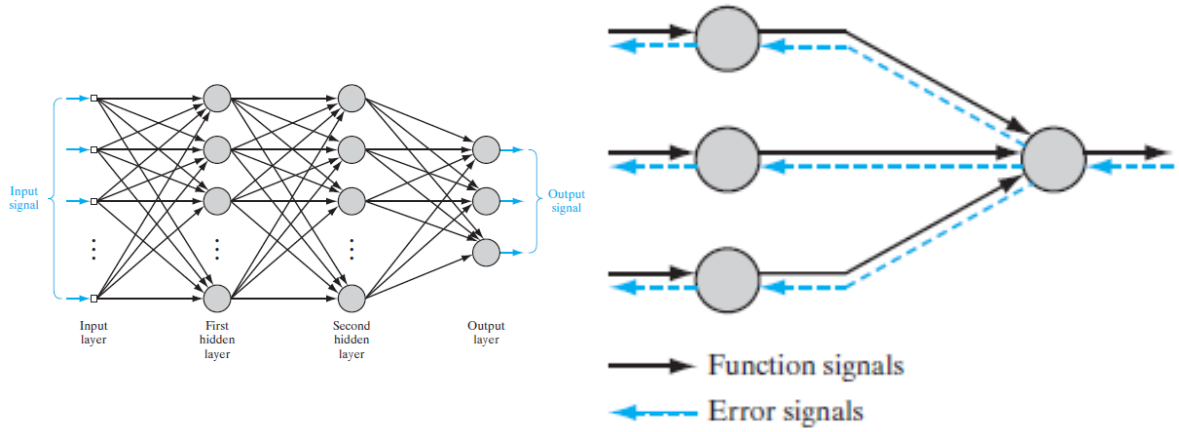
Characteristics of Multilayer Perceptron

A Multilayer Perceptron (MLP) has three distinctive characteristics:

1. Each model neuron has a smooth nonlinear activation function, typically a logistic function

$$y_j = \frac{1}{1 + exp(-v_j)}$$

2. Network contains one or more hidden layers that are not part of the input or output of the network
3. Network exhibits a high degree of connectivity, determined by the synapses of the network



- Multilayer Perceptron (MLP) consist of inputs, hidden, and output layer
- Two kind of signals in MLP:
  - Function Signals
  - Error Signals

Notation

- Indices i, j and k refers to different neurons in network
- Iteration n, nth training pattern presented to network
- $\mathcal{E}(n)$:instantaneous error energy (sum of error squares) at iteration n
- $e_j(n)$: error signal at the output of neuron j for iteration n
- $d_j(n)$: desired response for neuron j
- $y_j(n)$: functional signal at the output of neuron j at iteration n
- $w_{ji}(n)$: synaptic weight connecting the output of neuron i to the input of neuron j at iteration n
- $v_j(n)$: local induced field of neuron j at iteration n
- $\varphi_j(.)$: activation function associated with neuron j
- b_j: bias applied to neuron j
- $x_i(n)$: ith element of the input vector (pattern)
- $o_k(n)$: kth element of the output vector (pattern)
- η: learning rate
- m_l: number of nodes in layer l of MLP, and $l = 0, 1, 2, \ldots, L$,

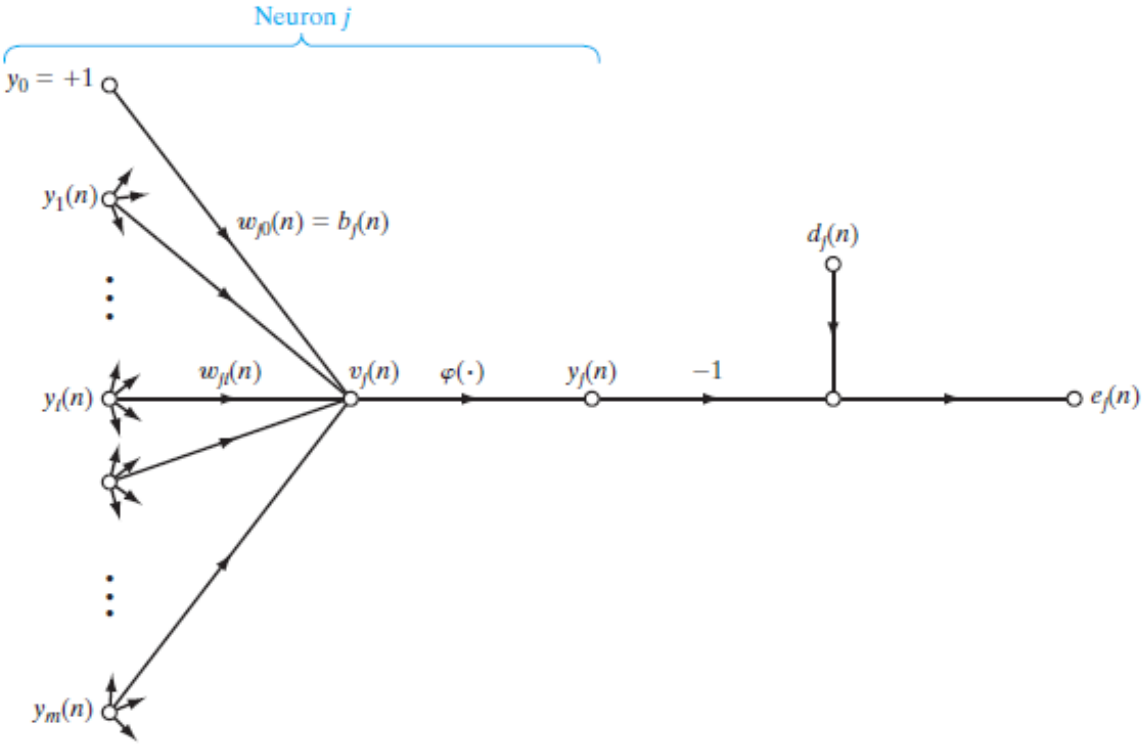


Figure 1: SFG with details of output neuron

- Error signal at out of neuron j

$$e_j(n) = d_j(n) - y_j(n)$$

- Instantaneous value of error energy for neuron j, $\frac{1}{2}e_j^2(n)$
- Instantaneous value of total energy

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

, where C is the set of all neurons in the output layer

- Let N: total number of patterns in training set
- Average squared error energy

$$\mathcal{E}_{av} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{E}(n)$$

- $\mathcal{E}(n)$ and $\mathcal{E}_{av}$ are function of all the free parameters

- For a given training set $\mathcal{T}$, $\mathcal{E}_{av}$ represents the cost function as a measure of learning performance
- Objective of learning process is to adjust free parameters to minimize $\mathcal{E}_{av}$
- We'll use an approximation similar to the derivation of LMS algorithm
- We'll use sequential method of training in which the weights are updated pattern-by-pattern basis until one *epoch*
- Adjustment to the weights are made in accordance with the respective errors computed for each pattern presented to the network

- Consider Fig.1, the induced local field $v_j(n)$ produced at neuron j is given by

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{1}$$

- Function signal at the output of neuron j is

$$y_j(n) = \varphi_j(v_j(n)) \tag{2}$$

- Back-propagation algorithm applies a correction $\Delta w_{ji}(n)$ to the synaptic weight $w_{ji}(n)$ which is proportional to $\partial \mathcal{E} / \partial w_{ji}(n)$
- As per the chain rule

$$\frac{\partial \mathcal{E}}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{3}$$

- Substituting for $\frac{\partial \mathcal{E}}{\partial e_j(n)}, \frac{\partial e_j(n)}{\partial y_j(n)}, \frac{\partial y_j(n)}{\partial v_j(n)}$, and $\frac{\partial v_j(n)}{\partial w_{ji}(n)}$ in eq(3) we get

$$\frac{\partial \mathcal{E}}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{4}$$

- As per the delta rule

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}}{\partial w_{ji}(n)} \tag{5}$$

- Using eq(3) in eq(4) yields

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{6}$$

where the local gradient $\delta_j(n)$ is defined by

$$\begin{aligned} \delta_j(n) &= -\frac{\partial \mathcal{E}}{\partial v_j(n)} \\ &= -\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_i(n)} \frac{\partial y_i(n)}{\partial v_j(n)} \\ &= e_j(n)\varphi_j'(v_j(n)) \end{aligned} \tag{7}$$

BP-Hidden Neuron

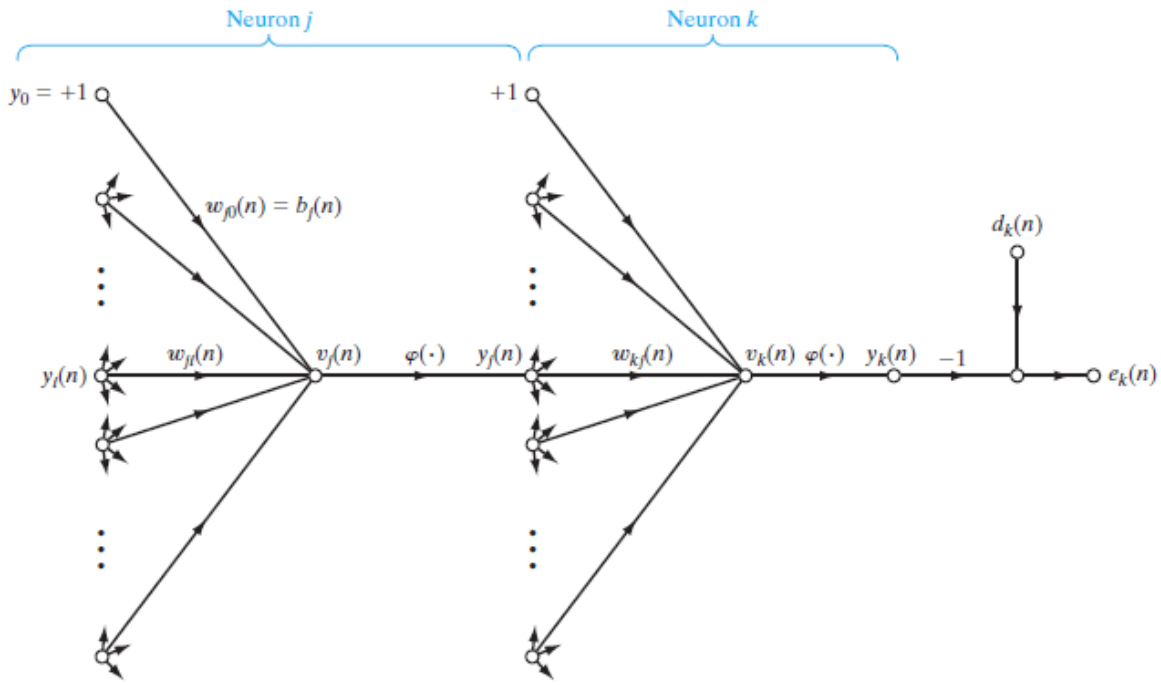


Figure 2: Signal-flow graph highlighting the details of output neuron k connected to hidden neuron j.

- Local gradient for the hidden neuron j

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \varphi_j'(v_j(n)) \tag{8}$$

- Since $\mathcal{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$ k is an output node, on differentiating we get

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} \tag{9}$$

- Using chain rule for $\partial e_k(n)/\partial y_j(n)$, we can rewrite eq(9) as

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \tag{10}$$

- Since $e_k(n) = d_k(n) - y_k(n) = d_k - \varphi_k(v_k(n))$, hence $\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n))$
- Also, the local induced field for neuron k,

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n)$$

- Differentiating $v_k(n)$ w.r.t $y_j(n)$ yields

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}$$

- Now, we get the desired partial derivative

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = -\sum_k e_k(n) \varphi_k'(v_k(n)) w_{kj} = -\sum_k \delta_k(n) w_{kj}(n) \tag{11}$$

- Finally, using eq(11) in eq(8), we get

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \quad j \text{ is a hidden neuron} \tag{12}$$

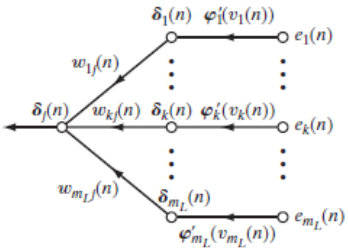


Figure 3: Signal-flow graph of a part of the adjoint system pertaining to backpropagation of error signals.

- $\Delta w_{ji}(n) = \eta \times \delta_j(n) \times y_i(n)$
  where

$$\delta_j(n) = \begin{cases} e_j(n)\varphi_j'(v_j(n)) & j \text{ is an output neuron} \\ \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) & j \text{ is a hidden neuron} \end{cases} \tag{13}$$

Activation Function

MLP uses smooth nonlinearity

1. Logistic Function-sigmoidal nonlinearity in its general form is defined by

$$\varphi_j(v_j(n)) = \frac{1}{1 + exp(-av_j(n))} \quad a > 0 \wedge -\infty < v_j(n) < \infty$$

, differentiating $\varphi_j(v_j(n))$ yields

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

, which can be written as

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

- With $y_j(n) = \varphi_j(v_j(n))$, we may write $\varphi_j'(v_j(n))$ as

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

- For a neuron j at the output layer, $y_j(n) = o_j(n)$, the local gradient for neuron j is given by

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) = a[d_j(n) - o_j(n)]o_j(n)[1 - o_j(n)]$$

- For an arbitrary hidden neuron j, the local gradient is defined as

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n)w_{kj}(n) = ay_j(n)[1-y_j(n)] \sum_k \delta_k(n)w_{kj}(n)$$

- $\varphi_j'(v_j(n))$ attains maximum value at $y_j(n) = 0.5$, and its minimum value at $y_j(n) = 0$ and $y_j(n) = 1$
- For a sigmoid activation function the synaptic weights are changed most for those neurons in the network where the function signals are in their midrange

2 Hyperbolic tangent function-another form of sigmoidal nonlinearity is defined as

$$\varphi_j(v_j(n)) = a\tanh(bv_j(n)), \quad (a,b) > 0$$

- Derivative of $\varphi_j(v_j(n))$ w.r.t $v_j(n)$ is given by

$$\begin{aligned}\varphi_j'(v_j(n)) &= ab\ sech^2(bv_j(n)) \\ &= ab(1 - \tanh^2(bv_j(n))) \\ &= ab(1 - \tanh(bv_j(n)))(1 + \tanh(bv_j(n))) \\ &= \frac{b}{a}[a - y_j(n)][a + y_j(n)]\end{aligned}$$

- For neurons at output layer

$$\begin{aligned}\delta_j(n) &= e_j(n)\varphi_j'(v_j(n)) \\ &= \frac{b}{a}[d_j(n) - y_j(n)][a - o_j(n)][a + o_j(n)]\end{aligned}$$

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n)w_{kj}(n)$$

- $$= \frac{b}{a}[a - y_j(n)][a + y_j(n)] \sum_k \delta_k(n)w_{kj}(n)$$

Rate of Learning

- BPA provides an "approximation" to the trajectory in weight space computed by the method of steepest descent
- For small η trajectory will be smooth but learning will be slow
- For large η, large change in synaptic weights assume such a form that the network may become unstable
- A simple method of increasing the rate of learning yet avoiding the danger of instability is to modify the delta rule

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(i)$$

by including a momentum term

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

where α is a positive number called momentum constant

- $\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$
-
$$\begin{aligned}\Delta w_{ji}(2) &= \alpha \Delta w_{ji}(1) + \eta \delta_j(2) y_i(2) \\ &= \alpha[\alpha \Delta w_{ji}(0) + \eta \delta_j(1) y_i(1)] + \eta \delta_j(2) y_i(2) \\ &= \alpha^2 \Delta w_{ji}(0) + \alpha \eta \delta_j(1) y_i(1) + \eta \delta_j(2) y_i(2) \\ &= \alpha^2[\alpha \Delta w_{ji}(-1) + \eta \delta_j(0) y_i(0)] + \alpha \eta \delta_j(1) y_i(1) + \eta \delta_j(2) y_i(2) \\ &= \alpha^2 \eta \delta_j(0) y_i(0) + \alpha \eta \delta_j(1) y_i(1) + \eta \delta_j(2) y_i(2)\end{aligned}$$
- Above equation can be written as

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{N} \alpha^{n-t} \delta_j(t) y_i(t) \qquad (14)$$

- Since, $\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n)$ and
  $\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\varphi_j'(n)$
- We can write $\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = -\delta_j(n)y_i(n)$ or $\delta_j(n)y_i(n) = -\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- Eq(14) can be written as

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^{N} \alpha^{n-t} \frac{\partial \mathcal{E}(t)}{\partial w_{ji}(t)} \tag{15}$$

From eq(15) following we can make following observations:

- The weight vector is the sum of exponentially weighted time series. For the time series to be convergent, the momentum constant must be restricted to $0 \leq |\alpha| \leq 1$
- When $\partial \mathcal{E}(t)/\partial w_{ji}(t)$ has same sign on consecutive iteration, the weighted update is accelerated (speed up downhill)
- When $\partial \mathcal{E}(t)/\partial w_{ji}(t)$ has opposite sign on consecutive iteration, the sum $\Delta w_{ji}(n)$ shrinks in magnitude, so weight $w_{ji}(n)$ is adjusted by a small amount. Inclusion of momentum in BPA has a stabilizing effect in the directions that oscillate in sign

Sequential and Batch Modes of Training

- One complete presentation of the entire training set during the learning process is called an *epoch*
- Learning process is maintained on an epoch-by-epoch basis until the synaptic weights and bias levels of the network stabilize
- It is a good practice to randomize the order of presentation of training examples from one epoch to the next
- Randomization tends to make the search in weight space stochastic over the learning cycles, thus avoiding the possibility of limit cycles in the evolution of synaptic weight vectors

- Sequential mode:
  - Update rule applied after each input-output presentation
  - Order of presentation should be randomized
  - Benefits: less storage, stochastic search through weight space helps avoid local minima
  - Disadvantage: hard to establish theoretical convergence condition
  - Preferred mode when the training data are redundant i.e., data set contains several copies of exactly the same pattern
- Batch mode:
  - Update rule applied after all input-output pairs are seen
  - Benefits: accurate estimate of the gradient, convergence to local minimum is guaranteed under simple conditions

- In batch mode, for a particular epoch, the cost function is defined as the average squared error and given by

$$\mathcal{E}_{av} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n)$$

- The adjustment applied to synaptic weight $w_{ji}(n)$ is defined by the delta rule

$$\begin{aligned} \Delta w_{ji} &= -\eta \frac{\partial \mathcal{E}_{av}}{\partial w_{ji}} \\ &= -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j(n)}{\partial w_{ji}} \end{aligned}$$

Stopping Criteria

- BPA cannot be shown to converge, and therefore there is no well-defined stopping criteria
- Let $w*$ denote a minimum, be it local or global
- Necessary condition for $w*$ to be minimum is that $g(w)|_{w=w*} = 0$
- *The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small threshold*
- Drawback
  - for successful trials, learning time may be long
  - it requires computation of gradient vector $g(w)$

Stopping Criteria

- We can use the fact that the cost function or error measure $\mathcal{E}_{av}(w)$ is stationary at the point $w = w*$
- We may state different criterion for convergence
  The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small
- Rate of change in $\mathcal{E}_{av}(w)$ is considered to be small enough if it lies in the range of 0.1 to 1% per epoch

Summary of Back-Propagation Algorithm

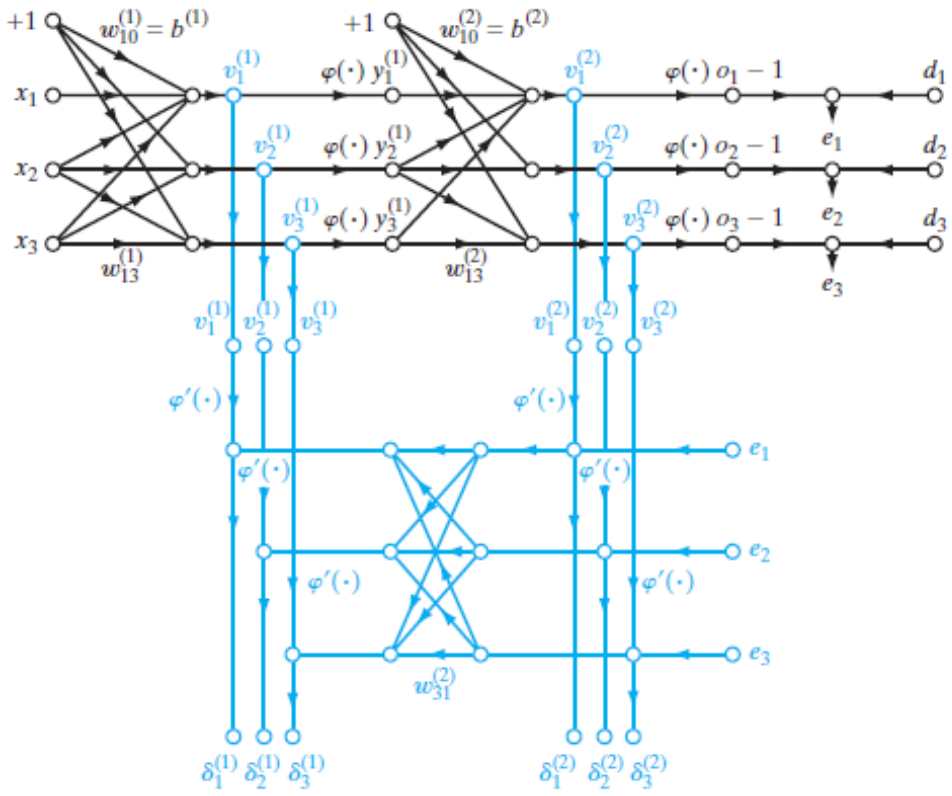


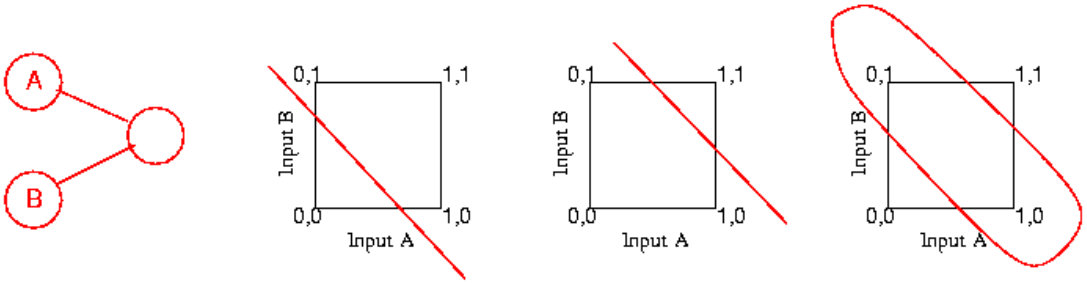
Figure 4: Signal-flow graph summary of back-propagation learning.

$$\Delta w_{ji}^{(l)}(n) = \eta \delta_j^{(l)}(n) y_i^{(l-1)}(n)$$

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\varphi_j'(v_j^{(L)}(n)) & \text{neuron j in output} \\ \varphi_j'(v_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n) & \text{neuron j in hidden layer} \end{cases}$$

XOR Problem

- It is a symbolic nonlinear classification problem
- Any neural network meant for non-linear classification must solve XOR problem



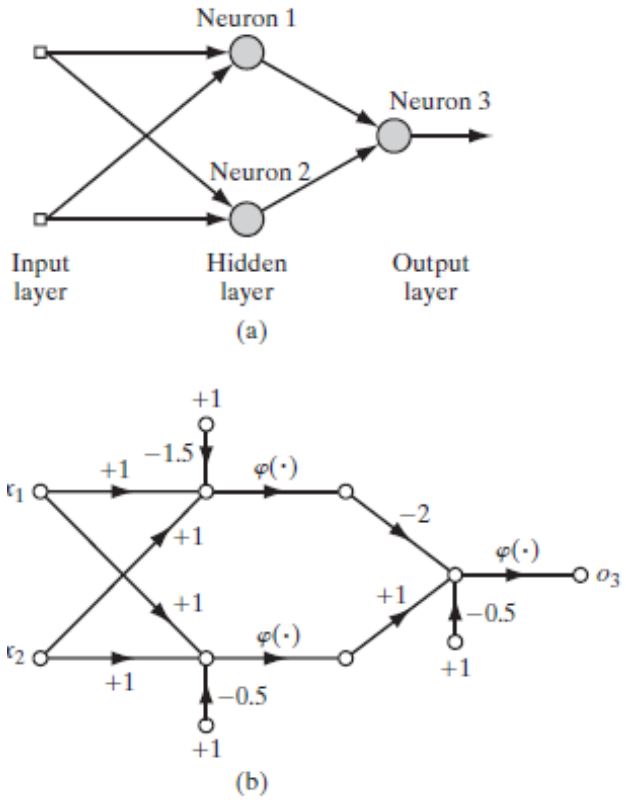
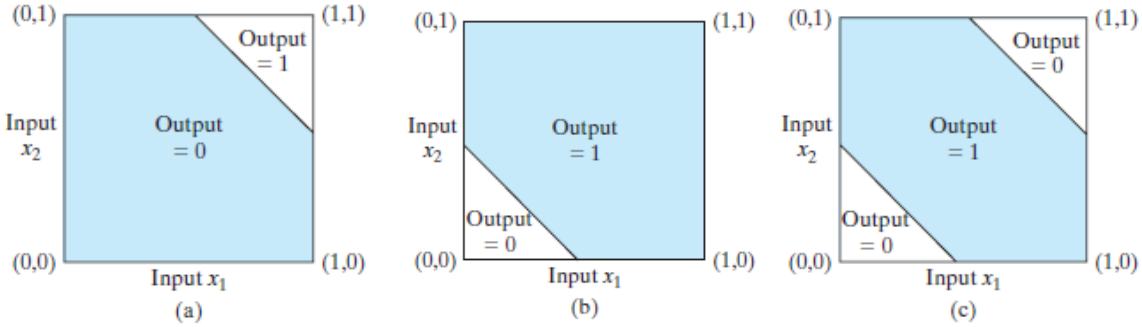


Figure 5: (a) Architectural graph of network solving the XOR problem. (b) Signal-flow graph of the network.



(a) Decision boundary formed by hidden neuron 1. (b) Decision boundary formed by hidden neuron 2. (c) Decision boundary constructed by the complete network.

Heuristics for Making BPA Perform Better

- Sequential vs batch mode: sequential mode of BP learning is computationally faster than the batch mode, especially when the training data is large and redundant
- Maximizing information content-every training examples presented to BPA should be chosen such that information content is largest possible for the task at hand
  - Use of an example that results in the largest training error
  - Use of an example that is radically different from all those previously used
- Activation function-MLP trained with BPA learns faster when antisymmetric form of sigmoid function is used instead of nonsymmetric. Popular example of antisymmetric sigmoid is hyperbolic tangent defined as $\varphi(v) = a\tanh(bv)$

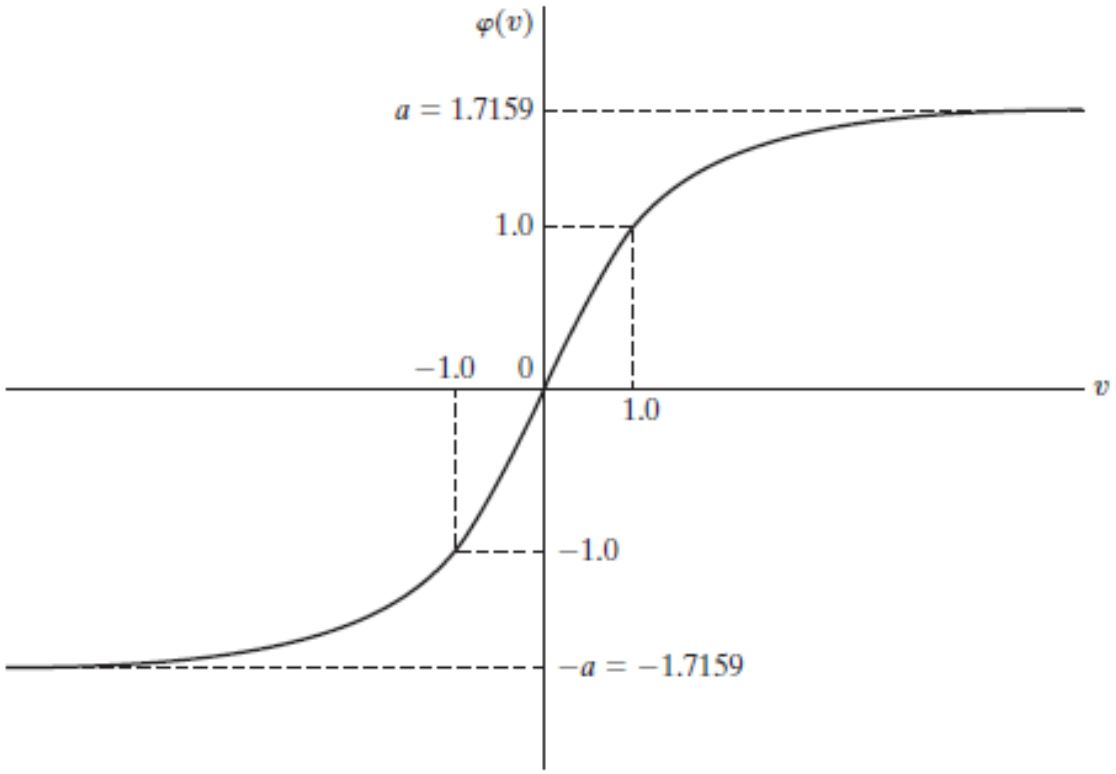


Figure 6: Graph of the hyperbolic tangent function $\varphi(v) = a\tanh(bv)$ for $a = 1.7159$ and $b = 2/3$.The recommended target values are +1 and -1.

- Target values-it must be chosen within the range of the sigmoid activation function. Desired response d_j in the output layer should be offset by some amount ϵ away from the limiting values of sigmoid function
  - For limiting value of $+a$, we get $d_j = a - \epsilon$
  - For limiting value of $-a$, we get $d_j = -a + \epsilon$, where ϵ is an appropriate constant. Here $\epsilon = 0.7159$
- Normalizing the inputs-each input variable should be preprocessed to that its mean value is close to zero or small as compared to its standard deviation. To accelerate BPA, input normalization must include the following measures:
  - Input variable contained in the training set should be uncorrelated (done using PCA)
  - Decorrelated input variable should be scaled so that their covariances are approximately equal, thereby ensuring that different synaptic weights learn at approximately the same speed

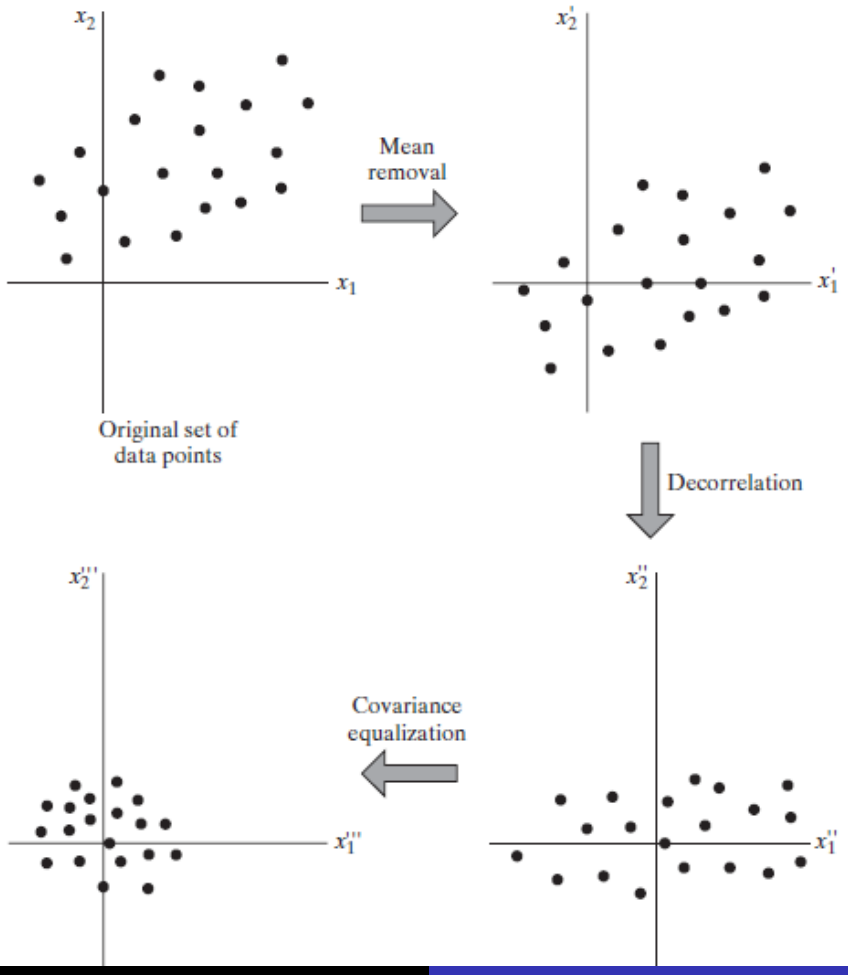


Figure 7: Illustrating the operation of mean removal, decorrelation, and covariance

- Initialization-a good choice of initial values of synaptic weights can be of tremendous help.
- What is a good choice?
- Let a MLP without any bias and hyperbolic tangent function for its activation function
- $v_j = \sum_{i=1}^{m} w_{ji} y_i$
- Assume that input applied to neurons has zero mean and unit variance, i.e., $\mu_y = E[y_i] = 0 \quad \forall i$ and $\sigma_y^2 = E[(y_i - \mu_i)^2] = E[y_i^2] = 1 \quad \forall i$
- Assume that inputs are uncorrelated

$$E[y_i y_k] = \begin{cases} 1 & k = i \\ 0 & k \neq i \end{cases}$$

and

- Synaptic weights are drawn from a uniformly distributed set of numbers with zero mean

$$\mu_w = E[w_{ji}] = 0 \quad \forall (j,i)\ pairs$$

and variance

$$\mu_w^2 = E[(w_{ji} - \mu_w)^2] = E[w_{ji}^2] \quad \forall (j,i)\ pairs$$

- Expressing the mean and variance of local induced field as

$$\mu_v = E[v_j] = E\left[\sum_{i=1}^{m} w_{ji} y_i\right] = \sum_{i=1}^{m} E[w_{ji}] E[y_i] = 0$$

and

$$\begin{aligned} \sigma_v^2 &= E[(v_j - \mu_v)^2] = E[v_j^2] \\ &= E\left[\sum_{i=1}^{m}\sum_{k=1}^{m} w_{ji} w_{jk} y_i y_k\right] \\ &= \sum_{i=1}^{m}\sum_{k=1}^{m} E[w_{ji} w_{jk}] E[y_i y_k] \\ &= \sum_{i=1}^{m} E[w_{ji}^2] \\ &= m\sigma_w^2 \end{aligned}$$

- A good strategy for initializing the synaptic weights so that the standard deviation of local induced field of a neuron lies in the transition area between the linear and saturated parts of the sigmoid activation function
- For hyperbolic sigmoid this objective is satisfied by setting $\sigma_v = 1$ in $\sigma_v^2 = m\sigma_w^2$
- $\sigma_w = m^{-1/2}$ i.e., for the uniform distribution, from which the synaptic weights are selected , to have zero mean and a variance of equal to the reciprocal of the number of synaptic connections of a neuron
- Learning from hints-process of learning may be generalized to include learning from hints, which is achieved by allowing prior information that we may have about the function $f(.)$ to be included in the learning process
- Learning rates-η should be a smaller value in the last layers than in the front layers. Neurons with many inputs should have smaller value of η than neurons with few inputs so as to maintain a similar learning time for all neurons in the network