Chapter - 2

Learning Algorithms
Learning Algorithms

2.1 Introduction

Learning is the most primary attribute of an ANN. An ANN learns to improve its performance from its environment. Learning is a process by which the free parameters of a neural network are adapted through a continuous process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place [1-4]. The taxonomy of the learning process is given in Fig.2.1.

![Learning Process]

Some of the common learning algorithms have been outlined in the following section.

2.2 Outline of some learning algorithms

Some algorithms are outlined in the following sections.

2.2.1 Error-correction Learning

Let \( y_{a,k}(n) \) be the desired response for neuron \( k \) at time \( n \). Let the corresponding simulated or estimated response be \( \hat{y}_{a,k}(n) \). Let the Input be \( x \). The error signal \( e_k(n) \) is given by
\[ e_k(n) = y_{a_k}(n) - y_{a_{-k}}(n) \]  \hspace{1cm} (2.1)

To minimize the error signal \( e_k(n) \), a cost function \( J \) is the mean-square-error criterion and is defined by

\[ J = E \left[ \frac{1}{2} \sum_k e_k^2(n) \right] \]  \hspace{1cm} (2.2)

Where \( E \) is the statistical expectation operator and the sum is over all neurons in the output layer. In order to minimize \( J \), w.r.t. the synaptic weights of the network yields the delta rule or error-correction learning rule. The change in weight for the \( n \)th weight \( w_k(n) \) at time \( n \) is given \([5]\) by

\[ \Delta w_k(n) = \mu e_k(n)x_j(n) \]  \hspace{1cm} (2.3)

where \( \mu \) is the learning rate parameter.

### 2.2.2 Supervised learning

The key element of a supervised learning is presence of an external teacher. The teacher has the knowledge of the environment through an input-output example. The network and the teacher both are exposed to the environment. The network which is unknown to the environment has its own parameters. The teacher supplies the desired response to the network during training. After some times called iterations, the network parameters begin to adapt to the environment. After more successive iterations the network also gets fully converged with the environment. Now the teacher is removed from the training. The network still continues to predict the environment as the parameters are now trained.

There are mainly two important algorithms associated with the supervised learning which are commonly used for training of a NN. These two algorithms are:

1. Least Mean Square Algorithm
2. Back Propagation Algorithm

These two algorithms have been mostly used in this thesis and therefore these have been described later in the chapter.
2.2.3 Hebbian Learning

According to a neuropsychologist Hebb[6]

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased.

On the basis of above statement, following two rules can be deduced [7]
1. If two neurons on either side of a synapse are activated simultaneously (synchronously), then the strength of that synapse is selectively increased.
2. If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.

Such a synapse is called a Hebbian synapse.

2.2.4 Competitive Learning

In this type of learning, the output neurons of a neural network compete among themselves for being the one to be active (fired). It is thus unlike the Hebbian learning where several output neurons may be active simultaneously, here only one output neuron can be active at any time.

2.2.5 Reinforced Learning

The reinforced learning is the on-line learning of an input-output mapping through a process of trial and error designed to maximize a scalar performance index called a reinforcement signal.

2.2.6 Unsupervised Learning

In an unsupervised learning there is no external teacher like the one in case of supervised learning mentioned above. The learning process is thus not overseen by such an external supervisor. To implement the unsupervised learning, we may use a competitive learning rule. For example, we may use a neural network that consists of two layers, namely an input layer and a competitive layer. The input layer receives the available data. The competitive layer consists of neurons that compete with each other for the “opportunity” to respond to features contained in the input.
data. In its simplest form the network operates in a “winner-takes-all” strategy. In such a strategy the neuron with the greatest total input “wins” the competition and turns on; all the other neurons then switch off.

2.3 Least-Mean-Square (LMS) Algorithm

The LMS algorithm was originally designed by Widrow and Hoff [5]. The machine used to simulate the LMS algorithm was an adaline machine. The application of LMS was extended to adaptive equalization of telephone channels for high speed data transmission [8-9], adaptive antennas for the suppression of interference signals originating from unknown directions [10].

Consider P sensors as shown in fig. 2.2. Subsequent inputs are say \( x_1, x_2, \ldots, x_P \) and the corresponding weights are \( w_1, w_2, \ldots, w_P \).

![Spatial filter diagram](image)

**Fig. 2.2 Spatial filter**

Let \( \hat{y}_a \) be the summed output given by

\[
\hat{y}_a = \sum_{k=1}^{p} w_k x_k 
\]  

...(2.4)

Let \( y_a \) be the desired output and \( e \) be the error then

\[
e = y_a - \hat{y}_a
\]  

...(2.5)

The cost function \( J \) is given by
\[ J = \frac{1}{2} E\left[ e^2 \right] \]  

Substituting Eq.(2.4) and (2.5) in (2.6), and then differentiating with respect to \( w_k \), yields

\[ \nabla_{w_k} J = -r_{y,x}(k) + \sum_{j=1}^{p} w_j r_x(j,k) \]  

For optimization \( \nabla_{w_k} J = 0, \quad k=1,2,...p \)  

Thus

\[ \sum_{j=1}^{p} w_j r_x(j,k) = r_{y,x}(k) \]  

\[ k=1,2,...p \]

Eqn.(2.9) is called Wiener-Hopf equation, where

- \( r_{y,x}(k) \) is the cross-correlation function = \( E[y_n x_k] \),
- \( r_x(j,k) \) is the autocorrelation function = \( E[x_j x_k] \)

Let \( w_k(n) \) be the weight of the spatial filter at nth iteration, then Eq.(2.7) can be written as

\[ \nabla_{w_k} J(n) = -r_{y,x}(k) + \sum_{j=1}^{p} w_j(n) r_x(j,k) \]  

According to the method of steepest descent,

\[ \Delta w_k(n) = -\mu \nabla w_k J(n) \]  

for \( k=1,2,...p \)

where the new term \( \mu \) is called the learning rate parameter.

The weight value \( w_k(n) \) can be updated after nth iteration to new value as follows

\[ w_k(n+1) = w_k(n) + \Delta w_k(n) \]

\[ = w_k(n) - \mu \nabla w_k J(n) \]  

\[ k=1,2,...p \]

substituting Eq.(2.10) in (2.12) we get

\[ w_k(n+1) = w_k(n) + \mu [r_{y,x}(k) - \sum_{j=1}^{p} w_j(n) r_x(j,k)] \]  

This is called steepest descent method.
The LMS algorithm is based on the use of the instantaneous estimates of the autocorrelation function \( r_x(j,k) \) and cross correlation function \( r_{xy}(k) \). These estimates are given by

\[
\hat{r}_x(j,k;n) = x_j(n)x_k(n) \quad \ldots(2.14)
\]

and

\[
\hat{r}_{xy}(k;n) = x_k(n)y_p(n) \quad \ldots(2.15)
\]

Substituting these values in Eq.(2.13) we get

\[
w_k(n+1) = w_k(n) + \mu \left[ y_a(n) - y^{\hat{a}}(n) \right] x_k(n) \quad \ldots(2.16)
\]

for \( k=1,2,...,p \)

Eq.(2.16) is known as the LMS algorithm.

**Learning by LMS Algorithms**

The learning of weights by LMS algorithm can be performed in two ways:

**Pattern Wise Learning**

**Sequential Learning**

2.3.1 **Pattern Wise Learning**

Consider a network as shown in Fig. 2.3. Let there be \( N \) input elements in a pattern. These inputs are \( x_0, x_1, ..., x_{N-1} \). There are \( N \) outputs corresponding to \( N \) inputs with each input \( x_i \) corresponds to output \( y_{a_i} \). Let the set of input in a pattern be denoted by \( x \). Similarly the set of outputs be denoted by vector \( y_a \). The connecting weights or the synaptic weights are denoted in a manner such that \( w_{ji} \) means the weight connecting i-th input node to j-th output node. The estimated output is denoted by \( y^{\hat{a}} \), corresponding to the i-th input-output pair. The set of estimated output is denoted by \( y^{\hat{a}} \). The error between actual or desired output \( y_{a_i} \) and the estimated output \( y^{\hat{a}} \) is \( e_i \) for the i-th input-output pair.
Fig. 2.3 An LMS Algorithm applied to a single layer Network

**Step wise algorithm**
Before the pattern matching learning is performed, a set of input patterns and corresponding set of desired patterns are first generated.

1. Let there be \( P \) patterns and each pattern consists of \( N \) inputs.
2. Initialize all connecting weights, \( W_{ji} \) to zero.
3. Compute the sum which is the estimated output \( y_{a,j}^{\wedge}(n) \) for the \( j \)-th output node as follows
   \[
   y_{a,j}^{\wedge}(n) = \sum_{i=0}^{N-1} x_i W_{ji}(n)
   \]
   for \( j = 0, 1, 2, ..., N-1 \) for all the output nodes. \( n \) refers to the \( n \)-th experiment or iteration.
4. Calculate the error \( \epsilon_i(n) \) for the \( i \)-th point as given below
   \[
   \epsilon_i(n) = y_{a,j}^{\wedge}(n) - y_{a,j}^*(n)
   \]
5. Compute change in weight for each of the connection for the current pattern of the \( n \)-th iteration as follows
   \[
   \Delta_{\mu} w_{ji}(n) = \mu \epsilon_i x_i(n) + \alpha \Delta_{\mu} w_{ji}(n-1)
   \]
where $\mu$ and $\alpha$ are called the learning rate parameter and momentum term respectively. $\Delta_p w_{ji}(n)$ denotes the change in weight corresponding to the $p$-th pattern at $n$-th iteration. Similarly $\Delta_p w_{ji}(n-1)$ denotes the change in weight for the $p$-th pattern at the $(n-1)$-th iteration.

(6) Calculate Mean Square Error (MSE) as follows
\[ \text{MSE}_p(n) = 0.5 e^2(n) \]

Where $e(n)$ is the average error given by
\[ e(n) = \frac{1}{N} \sum_{i=0}^{N-1} e_i(n) \]

(7) Apply all the $P$ patterns by repeating the steps from 3 to 6. Compute $\Delta_p w_{ji}(n)$ and $\text{MSE}_p$ for all $p = 0, 1, \ldots, P-1$. This constitutes one iteration or experiment.

(8) Now at the end of the $n$-th iteration, compute average MSE using
\[ \text{MSE}_{av}(n) = \frac{1}{NP} \sum_{p=0}^{P-1} \text{MSE}_p(n) \]

(9) Now compute MSE in deciBels (dB) for the $n$-th iteration using
\[ \text{MSE}_{dB}(n) = 10 \log_{10} \left( \frac{\text{MSE}_{av}(n)}{\text{MSE}_{av}(0)} \right) \]

where $\text{MSE}_{av}(0)$ means average MSE for the first i.e. 0-th iteration.

(10) Compute average change in weight by following equation
\[ \Delta w_{ji}(n) = \frac{1}{P} \sum_{p=0}^{P-1} \Delta_p w_{ji}(n) \]

(11) Update the weights for the $(n+1)$-th iteration as follows
\[ w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n) \]

(12) Use the new weights for the $(n+1)$-th iteration. Repeat the algorithm. Note the MSE in deciBels after each iteration. If $\mu$ and $\alpha$ are judiciously chosen then the value of MSE falls rapidly on increasing the number of iterations. After some iterations, the MSE level remains almost constant for subsequent iterations.

(13) The learning is discontinued at this stage.

(14) The steady state weights are used in the model for testing with new data.

(15) The estimated and the desired results are then compared.

This completes the pattern wise learning using LMS technique.
2.3.2 Sequential Learning

Consider the same network as shown in Fig. 2.3. Let there are N input elements to N-input points to the network. These inputs are \( x_0, x_1, \ldots, x_{N-1} \). There are N outputs corresponding to N inputs with each input \( x_i \) corresponds to output \( y_{a-i} \). Let the set of input be denoted by \( x \). Similarly the set of outputs be denoted by vector \( y_a \).

The connecting weights or the synaptic weights are denoted in a manner such that \( w_{ji} \) means the weight connecting i-th input node to j-th output node. The estimated output is denoted by \( \hat{y}_{a-i} \), corresponding to the i\(^{th}\) input-output pair. The set of estimated output is denoted by \( \hat{y}_a \). The error between actual or desired output \( y_{a-i} \) and the estimated output \( \hat{y}_{a-i} \) is \( e_i \) for the i\(^{th}\) input-output pair.

Step wise algorithm

1. Initialize all connecting weights, \( w_{ji} \) to zero.

2. Compute the sum which is the estimated output \( \hat{y}_{a-j}(n) \) for the j\(^{th}\) output node as follows

\[
\hat{y}_{a-j}(n) = \sum_{i=0}^{N-1} x_i w_{ji}(n)
\]

for \( j=0,1,2,\ldots,N-1 \) for all the output nodes, \( n \) refers to the n\(^{th}\) experiment or iteration.

3. Calculate the error \( e_i(n) \) for the i\(^{th}\) point as given below

\[
e_i(n) = y_{a-i}(n) - \hat{y}_{a-i}(n)
\]

4. Compute change in weight for each of the connection for the n\(^{th}\) iteration as follows

\[
\Delta w_{ji}(n) = \mu e_j x_i(n) + \alpha \Delta w_{ji}(n-1)
\]

where \( \mu \) and \( \alpha \) are called the learning rate parameter and momentum term respectively. \( \Delta w_{ji}(n) \) denotes the change in weight at n\(^{th}\) iteration. Similarly \( \Delta w_{ji}(n-1) \) denotes the change in weight for the (n-1)\(^{th}\) iteration.

5. Calculate Mean Square Error (MSE) as follows
\[ \text{MSE}(n) = 0.5e^2(n) \]

Where \( e(n) \) is the average error given by
\[
e(n) = \frac{1}{N} \sum_{i=0}^{N-1} e_i(n)
\]

(6) Now compute MSE in deciBels (dB) for the \( n \)-th iteration using
\[
\text{MSE}_{dB}(n) = 10 \log_{10} \left( \frac{\text{MSE}(n)}{\text{MSE}(0)} \right)
\]

where MSE(0) means MSE for the first i.e. 0th iteration.

(7) Update the weights for the \((n+1)\)-th iteration as follows
\[
w_{\mu}(n+1) = w_{\mu}(n) + \Delta w_{\mu}(n)
\]

(8) Use the new weights for the \((n+1)\)-th iteration. Repeat the algorithm. Note the MSE in deciBels after each iteration. If \( \mu \) and \( \alpha \) are suitably selected then the value of MSE falls rapidly initially on increasing the number of iterations. After some iterations, the MSE level remains almost constant for subsequent iterations.

(9) The learning is discontinued at this stage.

(10) The steady state weights are used in the model for testing with new data.

(11) The estimated and the desired results are then compared.

This completes the sequential learning using LMS technique.

2.4 Algorithms used for the training of the Neural Network

The training of the neural network depends upon the type of the network used. For multilayer neural network, Back Propagation (BP) algorithm is used. Further for some applications, we have used single layer ANN in which the learning is slightly different. In both ANN structures two models of training are used:

- Pattern matching learning
- Sequential learning

2.4.1 Pattern matching learning

Consider a single layer network as shown in Fig. 2.4. Let there be \( N \) input elements in a pattern. These inputs are \( x_0, x_1, ..., x_{N-1} \). There are \( N \) outputs corresponding to \( N \) inputs with each input \( x_i \) corresponds to output \( y_{a,i} \). Let the set of input in a
pattern be denoted by \( x \). Similarly the set of outputs be denoted by vector \( y_a \). The connecting weights or the synaptic weights are denoted in a manner such that \( w_{ji} \) means the weight connecting \( i \)-th input node to \( j \)-th output node. The estimated output is denoted by \( \hat{y}_{a,i} \), corresponding to the \( i \)-th input-output pair. The set of estimated output is denoted by \( \hat{y}_a \). The error between actual or desired output \( y_{a,i} \) and the estimated output \( \hat{y}_{a,i} \) is \( e_i \) for the \( i \)-th input-output pair. The bias weights are \( w_{b0}, w_{b1}, \ldots, w_{b(N-1)} \) which are applied to each neuron of the output layer. As there are \( N \) neurons at the output layer, there are \( N \) bias weights connected to these \( N \) neurons. \( f(.) \) denotes a sigmoid function which serves as a nonlinear function.

\[ f(x) = \frac{1}{1 + e^{-x}} \]

Fig. 2.4 A learning algorithm for a single layer ANN

**Step wise algorithm**

Before the pattern matching learning is performed, a set of input patterns and corresponding set of desired patterns are first generated.

1. Let there be \( P \) patterns and each pattern consists of \( N \) inputs.
2. Initialize all connecting weights, \( w_{ji} \) to some random values between +0.5 to -0.5.
3. Compute the sum for the \( j \)-th output node as follows

\[ y_{r,j}(n) = \sum_{i=0}^{N-1} w_{ji}(n) + w_{b_j}(n) \]
for j=0,1,2,...,N-1 for all the output nodes, n refers to the n-th experiment or iteration.

(4) The estimated output $y_{a,j}(n)$ for the j-th output node is given by

$$y_{a,j}(n) = f(y_{e,j}(n))$$

where $y_{e,j}(n)$ is the net input at j-th node and f(.) is a nonlinear function such as sigmoid which is defined as

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

where x is the net input to be passed to the function f.

(5) Calculate the error $e_j(n)$ for the i-th point as given below

$$e_j(n) = y_{a,j}(n) - y_{a,j}(n)$$

(6) Compute change in weight for each of the connection for the current pattern of the n-th iteration as follows

$$\Delta_p w_{ji}(n) = \mu f_j'(y_{e,j}) x_i(n) + \alpha \Delta_p w_{ji}(n-1)$$

where $f_j'(y_{e,j}) = e_j(n)(1 - y_{e,j}(n))$

$\mu$ and $\alpha$ are learning rate parameter and momentum term respectively. $\Delta_p w_{ji}(n)$ denotes the change in weight corresponding to the p-th pattern at the n-th iteration. Similarly $\Delta_p w_{ji}(n-1)$ denotes the change in weight for the p-th pattern at the (n-1)-th iteration.

(7) Compute change in bias weight for each of the connection for the p-th pattern of the n-th iteration as follows

$$\Delta_p w_{b,j}(n) = \mu f_j'(y_{e,j})$$

where $f_j'(y_{e,j}) = e_j(n)(1 - y_{e,j}(n))$

(8) Calculate Mean Square Error (MSE) as follows

$$MSE_p(n) = 0.5e^2(n)$$

Where e(n) is the average error, given by $e(n) = \frac{1}{N} \sum_{i=0}^{N-1} e_i(n)$

(9) Apply all the P patterns by repeating the steps from 3 to 6. Compute $\Delta_p w_{ji}(n)$ and $MSE_p(n)$ for all p = 0,1,...,P-1. This constitutes one iteration or experiment.

(10) Now at the end of n-th iteration, compute average MSE using
MSE_{av}(n) = \frac{1}{N_p} \sum_{p=0}^{P-1} MSE_p(n)

(11) Now compute MSE in decibels (dB) for the n-th iteration using

\[ MSE_{av}(n) = 10 \log_{10} \left( \frac{MSE_{av}(n)}{MSE_{av}(0)} \right) \]

where MSE_{av}(0) means average MSE for the first i.e. 0-th iteration.

(12) Compute average change in weights by following equation

\[ \Delta w_{ji}(n) = \frac{1}{P} \sum_{p=0}^{P-1} \Delta w_{ji}(n) \]
\[ \Delta w_{ij}(n) = \frac{1}{P} \sum_{p=0}^{P-1} \Delta w_{ij}(n) \]

(13) Update the weights for the (n+1)-th iteration as follows

\[ w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n) \]
\[ w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n) \]

(14) Use the new weights for the (n+1)-th iteration. Repeat the algorithm with a new set of input. Note the MSE in decibels after each iteration. On the proper values of \( \mu \) and \( \alpha \), the value of MSE falls rapidly in the beginning on increasing the number of iterations. After some iterations, the MSE level remains almost constant for subsequent iterations.

(15) The learning is discontinued at this stage.

(16) The steady state weights are used in the model for testing with new data.

(17) The estimated and the desired results are then compared.

This completes the pattern wise learning technique.

### 2.4.2 Sequential Learning

Consider the same single layer network as shown in Fig. 2.4. Let there be \( N \) input elements to \( N \)-input points of the network. These inputs are \( x_0, x_1, \ldots, x_{N-1} \). There are \( N \) outputs corresponding to \( N \) -inputs with each input \( x_i \) corresponds to output \( y_{a,i} \). Let the set of input in a pattern be denoted by \( x_a \). Similarly the set of outputs be denoted by vector \( y_a \). The connecting weights or the synaptic weights are denoted in
a manner such that \( w_{ji} \) means the weight connecting \( i \)-th input node to \( j \)-th output node. The estimated output is denoted by \( \hat{y}_{a,j} \), corresponding to the \( i \)-th input-output pair. The set of estimated output is denoted by \( \hat{y}_a \). The error between actual or desired output \( y_{a,j} \) and the estimated output \( \hat{y}_{a,j} \) is \( e_i \) for the \( i \)-th input-output pair.

The bias weights are \( w_{b0}, w_{b1}, \ldots, w_{b(N-1)} \) which are applied to each neuron of the output layer. As there are \( N \) neurons at the output layer, there are \( N \) bias weights attached to these \( N \) neurons. \( f(.) \) denotes a sigmoid function.

**Step wise algorithm**

1. Initialize all connecting weights, \( w_{ji} \) to some random values between +0.5 to -0.5.
2. Compute the sum for the \( j \)-th output node as follows
   \[
   y_{e,j}(n) = \sum_{i=0}^{N-1} x_i w_{ji}(n) + w_{bj}(n)
   \]
   for \( j=0,1,2,\ldots,N-1 \) for all the output nodes, \( n \) refers to the \( n \)-th experiment or iteration.
3. The estimated output \( \hat{y}_{a,j}(n) \) for the \( j \)-th output node is given by
   \[
   \hat{y}_{a,j}(n) = f(y_{e,j}(n))
   \]
   where \( f \) is a nonlinear function called sigmoid, defined in Section 2.4.1.
4. Calculate the error \( e_i(n) \) for the \( i \)-th point as follows
   \[
   e_i(n) = y_{a,j}(n) - \hat{y}_{a,j}(n)
   \]
5. Compute change in weight for each of the connection for the \( n \)-th iteration as follows
   \[
   \Delta w_{ji}(n) = \mu f_j'(y_{e,j}) x_i(n) + \alpha \Delta w_{ji}(n-1)
   \]
   where \( f_j'(y_{e,j}) = e_j(n)(1 - \hat{y}_{a,j}^2(n)) \)
   \( \mu \) and \( \alpha \) are the called learning rate parameter and momentum terms respectively.
   \( \Delta w_{ji}(n) \) denotes the change in weight corresponding to the \( n \)-th iteration. Similarly \( \Delta w_{ji}(n-1) \) denotes the change in weight for the \((n-1)\)-th iteration.
(6) Compute change in bias weight for each of the connection for the n-th iteration as follows
\[ \Delta w_{b,j}(n) = \mu f'_j(y_{e,j}) \]
where \( f'_j(y_{e,j}) = e_j(n)(1 - y^2_{e,j}(n)) \)

(7) Calculate Mean Square Error (MSE) as follows
\[ \text{MSE}(n) = 0.5e^2(n) \]
where \( e(n) \) is the average error, given by
\[ e(n) = \frac{1}{N} \sum_{i=0}^{N-1} e_i(n) \]

(8) Now compute MSE in decibels (dB) for the n-th iteration using
\[ \text{MSE}_{\text{dB}}(n) = 10 \log_{10}(\text{MSE}(n) / \text{MSE}(0)) \]
where \( \text{MSE}(0) \) means MSE for the first i.e. 0-th iteration.

(9) Update the weights for the (n+1)-th iteration as follows
\[ \begin{align*}
    w_{\mu}(n+1) &= w_{\mu}(n) + \Delta w_{\mu}(n) \\
    w_{b_j}(n+1) &= w_{b_j}(n) + \Delta w_{b_j}(n)
\end{align*} \]

(10) Use the new weights for the (n+1)-th iteration. Repeat the algorithm with a new set of input elements. Note the MSE in decibels after each iteration. If \( \mu \) and \( \alpha \) are judiciously chosen then the value of MSE falls rapidly on increasing the number of iterations. After some iterations, the MSE level remains almost constant for subsequent iterations.

(11) The learning is discontinued at this stage.

(12) The steady state weights are used in the model for testing with new data.

(13) The estimated and the desired results are then compared.
This completes the sequential learning technique.

2.5 Special Cases for the single layer network learning algorithms

2.5.1 Learning of NN with single layer and single output neuron

The neural learning algorithms for a single layer network for both pattern wise and sequential outlined in the previous section can be applied for a N input-output set. In some practical problems only a single neuron is required at the output.
In that case there will be N-input points but only one output node. Similarly for some applications only single input is required with a single output in such cases $N=1$. The same algorithms can be applied with slightly changing the value of $N$.

2.5.2 Learning of with single layer NN with functional expansion

In many practical problems of DSP, the single input is not capable to train the network. If we increase the number of layers, the complexity of the algorithm increases to a great extent. By taking more layers, a complex problem can be solved but at the cost of more complexity, cost and prolonged training time. But it has been thought that the same complex problem could be solved by a single layer single neuron NN by introducing sufficient nonlinearity to the input by functional expansion. According to Pao[11], in a single layer ANN, the single input can be functionally expanded to many elements. It increases the dimensionality of the network without increasing the number of layers. Such networks are known as Single layer ANN with functional expansions. The functional expansion of an input $x$ can be trigonometric such as $\sin \pi x$, $\sin 2\pi x$, ..., $\cos \pi x$, $\cos 2\pi x$, ..., or power series such as $x$, $x^2$, ..., or product terms such as $\sin \pi x \cos \pi x$, $\sin x \cos \pi x$, ..., $x \sin x$, $x^2 \cos x$, ..., and so on.

Fig.2.5 shows a single layer neural network with functional expansion.

Learning Algorithm

The learning algorithm for the ANN with functional expansion is identical to the one described in the previous section. The learning may be pattern matching or sequential type depending upon the nature of problem.

2.5.3 Pattern wise learning

Consider the single layer neural network with functional expansion scheme shown in Fig. 2.5. There is one input and a single output neuron in this network. The patterns of input and corresponding desired responses are first generated for training...
the NN. The original input \( x \) undergoes functional expansion as \( f_0(x), f_1(x), f_2(x), ..., f_{N-1}(x) \). Let there be \( N \) input for every iteration including the functional expansions and \( x \) itself. As already mentioned these functional expansions can be trigonometric, powerseries or product type. For simplicity, let us denote the input element as \( X_i \) for the \( i \)-th expansion. The connecting weight \( w_i(n) \) refers to the weight connecting \( i \)-th input node to the output at the \( n \)-th iteration. The single bias weight connected to the output neuron is \( w_b(n) \) for the \( n \)-th iteration.

![A single layer ANN with functional expansions](image)

**Fig.2.5 A single layer ANN with functional expansions**

**Step wise Algorithm**

1. Initialize connecting weights \( w_i(0) \) to random values between -0.5 to +0.5 for the first iteration.

2. Compute the sum \( y_e(n) \), for the \( i \)-th input node at the \( n \)-th iteration

\[
y_e(n) = \sum_{i=0}^{N-1} w_i(n) X_i + w_b(n)
\]

where \( X_i = \{f_0(x), \text{for } i=0,1,2,..N-1\} \)

3. Compute the estimated output \( y_a(n) \) as follows

\[
y_a(n) = f(y_e(n))
\]

where \( f(.) \) is the nonlinear function such as a sigmoid defined in Section 2.4.1.
(4) Compute error \( e(n) = y_d(n) - y_a(n) \)

where \( y_d(n) \) is the desired output for the present pattern.

(5) Compute \( \text{MSE}_p(n) \) for the \( p \)-th pattern of the \( n \)-th iteration using following equation

\[
\text{MSE}_p(n) = 0.5e^2(n)
\]

(6) Compute change in weight \( \Delta_p w_i(n) \) for the \( i \)-th connecting weight for \( p \)-th pattern,

\[
\Delta_p w_i(n) = \mu e(n)(1 - y_a^2(n))X_i + \alpha \Delta_p w_i(n - 1)
\]

and the change in bias weight \( \Delta_p w_b(n) \) for the \( p \)-th pattern at the \( n \)-th iteration

\[
\Delta_p w_b(n) = \mu e(n)(1 - y_a^2(n))
\]

where \( \mu \) and \( \alpha \) are the learning rate parameter and the momentum term respectively.

(6) Repeat steps 3 to 6 for all patterns \( P \) in the \( n \)-th iteration. This completes one iteration.

(7) Compute average MSE by following equation

\[
\text{MSE}_{av}(n) = \frac{1}{P} \sum_{p=0}^{P-1} \text{MSE}_p(n)
\]

(8) Compute MSE in decibels

\[
\text{MSE}_{db}(n) = 10 \log_{10}(\frac{\text{MSE}_{av}(n)}{\text{MSE}_{av}(0)})
\]

where \( \text{MSE}_{av}(0) \) refers to the average MSE for the first iteration (\( n=0 \)).

(9) Compute average change in weights \( \Delta w_i(n) \) and \( \Delta w_b(n) \)

\[
\Delta w_i(n) = \frac{1}{P} \sum_{p=0}^{P-1} \Delta_p w_i(n)
\]

\[
\Delta w_b(n) = \frac{1}{P} \sum_{p=0}^{P-1} \Delta_p w_b(n)
\]

(10) Update the weights for the \( (n+1) \)-th iteration as follows

\[
w_i(n + 1) = w_i(n) + \Delta w_i(n)
\]

\[
w_b(n + 1) = w_b(n) + \Delta w_b(n)
\]
Apply the new weights to the algorithm for the \((n+1)\)-th iteration. Note the MSE in deciBels after each iteration. The values of \(\mu\) and \(\alpha\) are suitable selected so that MSE falls progressively as we increase the iterations. After some iterations, the MSE does not fall further.

Under this condition the NN is said to be trained and the learning is discontinued at this stage.

The steady state weights are stored and will be used in the model for testing with new data.

The estimated and the desired results are then compared. This completes the pattern wise learning for a single layer ANN with functional expansion.

**2.5.4 Sequential Learning for Single layer ANN with functional Expansion**

**Step wise Algorithm**

Consider the same single layer network of Fig. 2.5 with single input and a single output neuron. The input \(x\) after functional expansion expands to \(N\) values \(X_i\) for \(i = 0, 1, 2, \ldots, N-1\). For every iteration the new value of \(x\) is chosen during training. The other notations are same as used in the algorithm for pattern wise learning.

1. Initialize connecting weights \(w_i(0)\) to random values between -0.5 to +0.5 for the first iteration.

2. Compute the sum \(y_e(n)\), for the \(i\)-th input node at the \(n\)-th iteration

\[
y_e(n) = \sum_{i=0}^{N-1} w_i(n)X_i + w_b(n)
\]

where \(X_i\) is \(i\)-th expansion of input \(x\), for \(i=0,1,2,\ldots,N-1\).

3. Compute the estimated output \(\hat{y}_a(n)\) as follows

\[
\hat{y}_a(n) = f(y_e(n))
\]

where \(f(.)\) is the nonlinear function such as a sigmoid defined earlier.

4. Compute error \(e(n) = y_d(n) - \hat{y}_a(n)\)

where \(y_d(n)\) is the desired output for the present pattern.
(5) Compute change in weight $\Delta w_i(n)$ for the $i$-th connecting weight for the $n$-th iteration

$$
\Delta w_i(n) = \mu e(n)(1 - y_i^2(n))X_i + \alpha \Delta w_i(n-1)
$$

and the change in bias weight $\Delta w_b(n)$ for the $n$-th iteration

$$
\Delta w_b(n) = \mu e(n)(1 - y_i^2(n))
$$

where $\mu$ and $\alpha$ are the learning rate parameter and the momentum term respectively.

(6) Compute MSE for the $n$-th iteration

$$
MSE(n) = 0.5e^2(n)
$$

(7) Compute MSE in decibels

$$
MSE_{\text{dB}}(n) = 10 \log_{10} \left( \frac{MSE(n)}{MSE(0)} \right)
$$

where $MSE_{\text{dB}}(0)$ refers to the MSE for the first iteration ($n=0$).

(8) Update the weights for the $(n+1)$-th iteration as follows

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

$$
w_b(n+1) = w_b(n) + \Delta w_b(n)
$$

(9) Apply the new weights to the algorithm for the $(n+1)$-th iteration. Note the MSE in decibels after each iteration. The values of $\mu$ and $\alpha$ are judiciously chosen so that MSE progressively is decreased. In the initial stage of the algorithm the MSE falls rapidly with increase in the number of iterations. After some iterations, the MSE reaches to almost a constant value and does not decrease further.

(10) Network is now trained and the learning is discontinued at this stage.

(11) The steady state weights are stored and will be used in the model for testing with new data.

(12) The estimated and the desired results are then compared during simulation.

This completes the pattern wise learning for a single layer ANN with functional expansion.

2.6 Training of a multilayer ANN

The multilayer NN consists of a set of sensory nodes (input nodes) also known as input layer, one or more hidden layers consisting of computing nodes and an
output layer also consisting of computation nodes. The multilayer NN are applied successfully to solve some difficult and diverse problems by supervised method of learning. The algorithm used for the training of a multilayer NN is known as a Back Propagation (BP) algorithm [12-17].

Fig. 2.6 represents a multilayer ANN with three layers. The notations used in the algorithm are as follows:

\[ w^m = \text{synaptic weight vector of a neuron at layer } m. \]

\[ w^m_b = \text{bias weight vector for a neuron at layer } m. \]

\[ y^m_e = \text{Vector of net internal activity levels of neurons at layer } m. \]

\[ y^m = \text{Vector of functional signals of neurons in layer } m. \]

![Fig. 2.6 A Three-Layer ANN](image)

The value of \( m \) may be 0, 1, ..., \( M \) where \( M \) is the output layer and \( m=0 \) is the input layer.

The learning algorithm has been discussed for three layers i.e. \( M=3 \).

Learning of network using Back Propagation Algorithm

2.6.1 Pattern matching learning

Step wise Algorithm
(1) Initialize all the synaptic and bias weights to random values between -0.5 to +0.5.

(2) Present the network with a set of training patterns with known input and desired output.

(3) Let the input vector applied be \( x(n) \) at the n-th iteration and the corresponding output vector be \( y_a(n) \).

(4) Compute the net activity level for the neuron \( j \) in the layer \( m \) as follows

\[
y_{j}^m(n) = \sum_{i=0}^{N} w_{ij}^m(n)y_{i}^{m-1}(n)
\]

where \( y_{i}^{m-1}(n) \) is the function signal of neuron \( i \) in the layer \( m-1 \) at iteration \( n \) and \( w_{ij}^m(n) \) is the connecting weight from \( i \)-th neuron of layer \( m-1 \) to \( j \)-th neuron of \( m \)-th layer. For \( i=0 \), \( y_{0}^{m-1}(n) = 1 \) and \( w_{j0}^m(n) = w_{j0}^m(n) \), where \( w_{j0}^m(n) \) is the bias weight applied to neuron \( j \) in the \( m \)-th layer.

(5) The output of neuron \( j \) is given by a sigmoid function

\[
y_{j}^m(n) = \frac{1 - e^{-o_{i}^{m-1}(n)}}{1 + e^{-o_{i}^{m-1}(n)}}
\]

for the neuron in the first hidden layer (i.e. \( m=1 \)),

\[
y_{j}^0(n) = x_{j}(n)
\]

where \( x_{j}(n) \) is the \( j \)-th element of the input vector \( x(n) \)

for the neuron \( j \) in the output layer i.e. \( m=M \), set

\[
y_{j}^M(n) = y_{a}(n)
\]

(6) Compute the error signal

\[
e_{j}(n) = y_{a}(n) - y_{aj}(n)
\]

(7) Compute the local gradient of the network by the backward propagation

\[
\delta_{j}^M(n) = e_{j}^M(n)\dot{y}_{j}(n)[1 - \dot{y}_{j}(n)], \text{ for the neuron } j \text{ in output layer } m=M.
\]

and

\[
\delta_{j}^m(n) = y_{j}^m(n)[1 - y_{j}^m(n)]\sum_{k} \delta_{k}^{m+1}(n)w_{kj}^{m+1}(n), \text{ for the neuron } j \text{ in the hidden layer } m.
\]
(8) Calculate the change in synaptic weights of the network according to the generalized delta rule for all patterns in n-th iteration

$$\Delta_p w^m_j(n) = \mu \delta^m_j(n) y^m_i(n) + \alpha [\Delta_p w^m_j(n-1)]$$

where $\Delta_p w^m_j(n-1)$ is the change in weight of the p-th pattern at the (n-1)-th iteration and

$\mu$ is the learning rate parameter and $\alpha$ is the momentum term.

(9) Calculate the MSEp for the p-th pattern for all the N neurons in the output layer ($m=M$) as follows

$$MSE_p(n) = \frac{1}{2} \sum_{j=0}^{N-1} e^j_p(n)$$

(10) Apply all the patterns P to the network and repeat the algorithm for each of the pattern applied in the n-th iteration n.

(11) Calculate average change in weights as follows

$$\Delta w^m_j(n) = \frac{1}{P} \sum_{p=0}^{P-1} \Delta_p w^m_j(n)$$

(12) Calculate average MSEav(n) using following equation

$$MSE_{av}(n) = \frac{1}{NP} \sum_{p=0}^{P-1} MSE_p(n)$$

(13) Compute MSE in deciBels for each iteration

$$MSE_{dB}(n) = 10 \log_{10} \left( \frac{MSE_{av}(n)}{MSE_{av}(0)} \right)$$

where MSEav(0) is the average MSE for the iteration n=0.

(14) Update the weights by using following equation

$$w^m_j(n+1) = w^m_j(n) + \Delta w^m_j(n)$$

(15) Repeat the algorithm for next iteration n+1 with the new weights and the complete set of P patterns again.

(16) Note MSE in deciBels after each iteration. Adjust the values of $\mu$ and $\alpha$ for best possible convergence.

(17) MSE falls progressively with subsequent iterations. After some iterations, the MSE reaches to a steady state. Now on subsequent iterations MSE remains almost constant and does not fall.
(18) Discontinue the learning process and note down the weights of the steady state.
(19) Test the network with some new set of data for simulation of the algorithm.

This completes the training of a multilayer ANN using back Propagation algorithm.

2.6.2 Sequential learning by BP algorithm

The sequential learning of the multilayer ANN of Fig. 2.6 resembles to the pattern wise learning. Necessary steps involved in sequential learning of multilayer ANN are given in the following algorithm.

Step wise Algorithm

(1) Follow the steps 1-7 as given in the pattern wise learning of BP algorithm in Section 2.5.1 with one change in step 2 that there is no pregenerated pattern in sequential learning. For every iteration a new set of training data consisting of an input and desired output will be supplied to the NN.

(2) Calculate the change in synaptic weights of the network according to the generalized delta rule in n-th iteration

\[ \Delta w_{ji}(n) = \mu \delta_j(n)y_i^{\text{prev}}(n) + \alpha[\Delta w_{ji}(n-1)] \]

where \( \Delta w_{ji}(n-1) \) is the change in weight at the (n-1)-th iteration and \( \mu \) is the learning rate parameter and \( \alpha \) is the momentum term.

(3) Calculate the MSE for all the neurons in the output layer (m=M) as follows

\[ \text{MSE}_i(n) = \frac{1}{2} \sum_{j=0}^{N} e_i^j(n) \text{ for } i=1,2,...,N \text{ in the output layer} \]

(4) Calculate \( \text{MSE}_{av}(n) \) using following equation

\[ \text{MSE}_{av}(n) = \frac{1}{N} \sum_{i=1}^{N} \text{MSE}_i(n) \]

(5) Compute MSE in decibels for each iteration

\[ \text{MSE}_{\text{db}}(n) = 10 \log_{10}(\text{MSE}_{av}(n) / \text{MSE}_{av}(0)) \]

where \( \text{MSE}_{av}(0) \) is the average MSE for the iteration \( n=0 \).
(6) Update the weights by using following equation

\[ w_j^m(n + 1) = w_j^m(n) + \Delta w_j^m(n) \]

(7) Repeat the algorithm with new set of weights for the \((n+1)\)-th iteration. Chose suitable value of \(\mu\) and \(\alpha\) for best possible convergence.

(8) MSE falls progressively with increase in iterations. After some iterations, the MSE does not decrease further on subsequent iterations. This is called the steady state.

(9) At this stage the NN is said to have been learnt. Discontinue the learning process and note down the weights of the steady state.

(10) Test the network with some new set of data for simulation of the algorithm. This completes the sequential learning of a multilayer ANN using back Propagation algorithm.

2.7 Some Commonly Used Terms

The algorithms defined and derived in the previous sections have been applied throughout the thesis. Along with the application of these algorithms some important terms that have also been associated with the literature quite often, are outlined below.

2.7.1 Sigmoid Function

When back propagation is applied, the computed sum along with the bias parameters are input to the neuron of the next layer (the next layer may also be an output layer), the final output called the predicted or the estimated or the simulated output is computed by passing this net input through a nonlinear function called activation function or a sigmoid function. The function is denoted by \(f(.)\) and is defined as follows:

\[ f(x) = \tanh(x/2) = \frac{1 - e^{-x}}{1 + e^{-x}} \]

where \(x\) is the net input to the neuron also called net internal activity level of that neuron. According to this nonlinearity, the amplitude of the output lies between 0 to 1.
2.7.2 Rate of Learning and momentum

The rate of learning parameter $\mu$ has been used in all the algorithms. Its value is chosen by the network user. The value is fixed on error and trial basis. Usually this value lies between 0.01 to 0.99. The learning parameter $\mu$ reflects the rate of learning of the network. If $\mu$ is very large, it can lead to instability in the network and unsatisfactory learning. On the other hand if $\mu$ is too small, it can lead to excessively slow learning. A simple method of increasing the rate of learning and yet avoiding the danger of instability is to include a new term called momentum in the weight updation delta rule. This term is indicated by symbol $\alpha$ and its value lies between 0.01 to 0.99. The momentum term is also selected on the error and trial basis like $\mu$. 

---

39
References


