from:
N.J., Smelter, & P.B., Baltes (Eds.) (2001).
Encyclopedia of the Social and Behavioral Sciences.
London: Elsevier Science.

Article Title: ***Linear Algebra for Neural Networks***

By: ***Hervé Abdi***

Author Address: Hervé Abdi, School of Human Development, MS: Gr.4.1, The University of Texas at Dallas, Richardson, TX 750833–0688, USA

Phone: 972 883 2065, ***fax:*** 972 883 2491 ***Date:*** June 1, 2001

E-mail: herve@utdallas.edu

Abstract

Neural networks are quantitative models which learn to associate input and output patterns adaptively with the use of learning algorithms. We expose four main concepts from linear algebra which are essential for analyzing these models: 1) the projection of a vector, 2) the eigen and singular value decomposition, 3) the gradient vector and Hessian matrix of a vector function, and 4) the Taylor expansion of a vector function. We illustrate these concepts by the analysis of the Hebbian and Widrow-Hoff rules and some basic neural network architectures (i.e., the linear autoassociator, the linear heteroassociator, and the error backpropagation network). We show also that neural networks are equivalent to iterative versions of standard statistical and optimization models such as multiple regression analysis and principal component analysis.

1 Introduction

Linear algebra is particularly well suited to analyze the class of neural networks called *associators*. These quantitative models learn to associate input and out-

put patterns adaptively with the use of learning algorithms. When the set of input patterns is different from the set of output patterns, the models are called *heteroassociators*. When input patterns and output patterns are the same, the models are called *autoassociators*. Associators consist of layers of elementary units called neurons. Information flows from the first layer (the input layer), to the last one (the output layer). Some architectures may include intermediate layers (the hidden layers). Typically neurons from a given layer are connected to the neurons of another layer. Linear algebra operations describe the transformations of the information as it flows from one layer to another one.

2 Prerequisite notions and notations

Vectors are represented by lower case boldface letters (*e.g.,* $\boldsymbol{x}$), matrices by upper case boldface letters (*e.g.,* $\boldsymbol{X}$). The following elementary notions are supposed to be known (see Abdi, Valentin & Edelman, 1999, for a neural network approach): the transpose operation, (*e.g.,* $\boldsymbol{x}^{\mathsf{T}}$), the norm of a vector (*e.g.,* $\|\boldsymbol{x}\|$), the scalar product (*e.g.,* $\boldsymbol{x}^{\mathsf{T}}\boldsymbol{w}$) and the product of two matrices (*e.g.,* $\boldsymbol{AB}$). We will use also the elementwise or Hadamar product (*e.g.,* $\boldsymbol{A} \circledast \boldsymbol{B}$).

3 Projection

3.1 Cosine between two vectors

The *cosine* of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is the cosine of the angle made by the origin of the space and the points defined by the coordinates of the vectors. Thus,

$$\mathsf{cos}(\boldsymbol{x},\boldsymbol{y}) = \frac{\boldsymbol{x}^{\mathsf{T}}\boldsymbol{y}}{\|\boldsymbol{x}\|\,\|\boldsymbol{y}\|} \ . \tag{1}$$

The cosine reflects the *similarity* between vectors. When two vectors are proportional to each other (*i.e.,* when they have the same direction), their cosine is equal to 1; when they are orthogonal to each other, their cosine is equal to 0.

3.2 Distance between vectors

Among the large family of *distances* between vectors, the most popular by far is the *Euclidean* distance. It is closely related to the cosine between vectors and is defined as

$$\begin{aligned} \mathsf{d}^2(\boldsymbol{x},\boldsymbol{y}) &= (\boldsymbol{x}-\boldsymbol{y})^{\mathsf{T}}(\boldsymbol{x}-\boldsymbol{y}) = \boldsymbol{x}^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{y}^{\mathsf{T}}\boldsymbol{y} - 2\boldsymbol{x}^{\mathsf{T}}\boldsymbol{y} \\ &= \|\boldsymbol{x}\| + \|\boldsymbol{y}\| - 2\left[\mathsf{cos}(\boldsymbol{x},\boldsymbol{y}) \times \|\boldsymbol{x}\| \times \|\boldsymbol{y}\|\right] = \sum (x_i - y_i)^2 \ . \end{aligned} \tag{2}$$

3.3 Projection of one vector onto another vector

The (orthogonal) projection of vector $\boldsymbol{x}$ on vector $\boldsymbol{w}$ is defined as

$$\mathsf{proj}_{\boldsymbol{w}}\boldsymbol{x} = \frac{\boldsymbol{x}^\mathsf{T}\boldsymbol{w}}{\boldsymbol{w}^\mathsf{T}\boldsymbol{w}}\boldsymbol{w} = \mathsf{cos}(\boldsymbol{x},\boldsymbol{w}) \times \frac{\|\boldsymbol{x}\|}{\|\boldsymbol{w}\|}\boldsymbol{w} \ . \tag{3}$$

The norm of $\mathsf{proj}_{\boldsymbol{w}}\boldsymbol{x}$ is its distance to the origin of the space. It is equal to

$$\|\mathsf{proj}_{\boldsymbol{w}}\boldsymbol{x}\| = \frac{|\boldsymbol{x}^\mathsf{T}\boldsymbol{w}|}{\|\boldsymbol{w}\|} = |\mathsf{cos}(\boldsymbol{x},\boldsymbol{y})| \times \|\boldsymbol{x}\| \ . \tag{4}$$

3.4 Illustration: Hebbian and Widrow-Hoff learning rules

A neural network consists of cells connected to other cells via modifiable weighted *connections* called *synapses*. Consider a network made of I input cells and only one output cell. The information is transmitted via the synapses, from a set of external input cells to the output cell which gives a response corresponding to its state of activation. If the input pattern and the set of synaptic weights are given by I-dimensional vectors denoted $\boldsymbol{x}$, and $\boldsymbol{w}$, the activation of the output cell is obtained as

$$a = \boldsymbol{x}^\mathsf{T}\boldsymbol{w} \ . \tag{5}$$

So the activation is proportional to the norm of the projection of the input vector onto the weight vector. The *response* or *output* of the cell is denoted o. For a *linear cell*, it is proportional to the activation (for convenience, assume that the proportionality constant is equal to 1). *Linear heteroassociators* and *autoassociators* are made of linear cells. In general, the output of a cell is a *function* (often, but not necessarily, continuous), called the *transfer function*, of its activation

$$o = f\,(a) \ . \tag{6}$$

For example, in *backpropagation networks*, the (nonlinear) transfer function is usually the logistic function

$$o = f\,(a) = \mathrm{logist}\,\boldsymbol{w}^\mathsf{T}\boldsymbol{x} = \frac{1}{1+\exp\{-a\}} \ . \tag{7}$$

Often, a neural network is designed to associate, to a given input, a specific response called the target, denoted t. *Learning* is equivalent to defining a rule that specifies how to add a small quantity to each synaptic weight at each learning iteration. Learning makes the output of the network closer to the target.

Learning rules come in two main flavors: *supervised* (*e.g.,* Widrow-Hoff) which take into account the error or distance between the response of the neuron and the target and *unsupervised* (*e.g.,* Hebbian) which do not require such "feedback." The Hebbian learning rule modifies the weight vector at iteration $n+1$ as

$$\boldsymbol{w}_{[n+1]} = \boldsymbol{w}_{[n]} + \eta t\boldsymbol{x} \ , \tag{8}$$

where η is a small positive constant called the *learning constant*. So, a Hebbian learning iteration moves the weight vector in the direction of the input vector by an amount proportional to the target.

The Widrow-Hoff learning rule uses the error and the derivative of the transfer function (f') to compute the correction as

$$\boldsymbol{w}_{[n+1]} = \boldsymbol{w}_{[n]} + \eta f'(a)(t-o)\boldsymbol{x} \ . \tag{9}$$

So, a Widrow-Hoff learning iteration moves the weight vector in the direction of the input vector by an amount proportional to the error.

For networks with several cells (say J) in the output layer, the pattern of activation, output, and target become J-dimensional vectors (denoted $\boldsymbol{a}$, $\boldsymbol{o}$, and $\boldsymbol{t}$, respectively), and the synaptic weights are stored in an $I \times J$ matrix $\boldsymbol{W}$. The learning equations become

$$\begin{aligned} \boldsymbol{W}_{[n+1]} &= \boldsymbol{W}_{[n]} + \eta \boldsymbol{x}\boldsymbol{t}^{\mathsf{T}} \text{ (Hebbian) and} \\ \boldsymbol{W}_{[n+1]} &= \boldsymbol{W}_{[n]} + \eta (f'(\boldsymbol{a}) \circledast \boldsymbol{x})(\boldsymbol{t}-\boldsymbol{o})^{\mathsf{T}} \text{ (Widrow-Hoff)} \ , \end{aligned} \tag{10}$$

(the derivative of the transfer function is applied elementwise).

In general, several (say K) input/target associations are to be learned. Then the set of input patterns is stored in an $I \times K$ matrix denoted $\boldsymbol{X}$, the activation and target patterns respectively are stored in $J \times K$ matrices denoted $\boldsymbol{A}$ and $\boldsymbol{T}$ respectively. The activation and learning iterations can be computed for all the patterns at once (this is called *batch* mode). The output matrix is computed as

$$\boldsymbol{O} = f(\boldsymbol{A}) = f\left(\boldsymbol{W}\boldsymbol{X}^{\mathsf{T}}\right) \tag{11}$$

(f is applied elementwise). The learning equations become

$$\boldsymbol{W}_{[n+1]} = \boldsymbol{W}_{[n]} + \eta \boldsymbol{X}\boldsymbol{T}^{\mathsf{T}} \text{ (Hebb) and} \tag{12}$$

$$\boldsymbol{W}_{[n+1]} = \boldsymbol{W}_{[n]} + \eta (f'(\boldsymbol{A}) \circledast \boldsymbol{X})(\boldsymbol{T}-\boldsymbol{O})^{\mathsf{T}} \text{(Widrow-Hoff)} \ . \tag{13}$$

4 Eigenvalues, eigenvectors, and the singular value decomposition

Eigenvectors of a given square matrix $\boldsymbol{W}$ (resulting from its *eigendecomposition*) are vectors invariant under multiplication by $\boldsymbol{W}$. The eigendecomposition is best defined for a subclass of matrices called *positive semi-definite* matrices. A matrix $\boldsymbol{X}$ is positive semi-definite if there exists another matrix $\boldsymbol{Y}$ such that $\boldsymbol{X} = \boldsymbol{Y}\boldsymbol{Y}^{\mathsf{T}}$. This is the case for most matrices used in neural networks and so we will consider only this case here.

Formally, a (nonzero) vector $\boldsymbol{u}$ is an eigenvector of a square matrix $\boldsymbol{W}$ if

$$\lambda \boldsymbol{u} = \boldsymbol{W}\boldsymbol{u} \ . \tag{14}$$

The scalar λ is the *eigenvalue* associated with $\boldsymbol{u}$. So, $\boldsymbol{u}$ is an eigenvector of $\boldsymbol{W}$if its direction is invariant under multiplication by $\boldsymbol{W}$ (only its length is changed

if $\lambda \neq 1$). There are, in general, several eigenvectors for a given matrix (at most as many as the dimension of $\boldsymbol{W}$). They are in general ordered by decreasing order of their eigenvalue. So, the first eigenvector, $\boldsymbol{u}_1$ has the largest eigenvalue λ_1. The number of eigenvectors with a non-zero eigenvalue is the *rank* of the matrix.

The eigenvalues of positive semi-definite matrices are always positive or zero (a matrix with strictly positive eigenvalues, is *positive definite*). Also, any two eigenvectors with different eigenvalues are orthogonal, i.e.

$$\boldsymbol{u}_\ell^\mathsf{T} \boldsymbol{u}_{\ell'} = 0 \qquad \forall \quad \ell \neq \ell' \ . \tag{15}$$

In addition, the set of eigenvectors of a matrix constitutes an orthogonal basis for its rows and columns. This is expressed by defining two matrices: the eigenvector matrix $\boldsymbol{U}$, and the diagonal matrix of the eigenvalues: $\boldsymbol{\Lambda}$. The eigendecomposition of $\boldsymbol{W}$ (with rank L) is

$$\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\mathsf{T} = \sum_\ell^L \lambda_\ell \boldsymbol{u}_\ell \boldsymbol{u}_\ell^\mathsf{T}, \text{ or equivalently:} \quad \boldsymbol{\Lambda} = \boldsymbol{U}^\mathsf{T}\boldsymbol{W}\boldsymbol{U} \ . \tag{16}$$

The *singular value decomposition* (SVD) generalizes the eigendecomposition to rectangular matrices. If $\boldsymbol{X}$ is an $I \times K$ matrix, its SVD is defined as

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{V}^\mathsf{T} \text{ with } \boldsymbol{U}^\mathsf{T}\boldsymbol{U} = \boldsymbol{V}^\mathsf{T}\boldsymbol{V} = \boldsymbol{I} \text{ and } \boldsymbol{\Delta} \text{ being a diagonal matrix} \ . \tag{17}$$

($\boldsymbol{I}$ being the *identity* matrix). The diagonal elements of $\boldsymbol{\Delta}$ are real positive numbers called the *singular values* of $\boldsymbol{X}$. The matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are the left and right matrices of singular vectors (which are also eigenvectors, see below). The SVD is closely related to the eigendecomposition because $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{\Delta}$ can be obtained from the eigendecomposition of matrices $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ and $\boldsymbol{X}\boldsymbol{X}^\mathsf{T}$ as

$$\boldsymbol{X}\boldsymbol{X}^\mathsf{T} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\mathsf{T}, \quad \boldsymbol{X}^\mathsf{T}\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T}, \text{ and } \boldsymbol{\Delta} = \boldsymbol{\Lambda}^{\frac{1}{2}} \tag{18}$$

(note that $\boldsymbol{X}^\mathsf{T}\boldsymbol{X}$ and $\boldsymbol{X}\boldsymbol{X}^\mathsf{T}$ have the same eigenvalues).

The eigen- and singular value decompositions are used in most fields of applied mathematics including statistics, image processing, mechanics, and dynamical systems. For neural networks, they are essential for studying the dynamics of linear autoassociators and heteroassociators.

4.1 Iterative processes

A linear heteroassociator using the Widrow-Hoff rule, learning modifies only the eigenvalues of the weight matrix. Specifically, if the patterns to be learned are stored in an $I \times K$ matrix $\boldsymbol{X}$, with singular value decomposition as $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Delta}\boldsymbol{V}^\mathsf{T}$, then the learning equation (see Equation 12) becomes [because for a linear heteroassociator, $\boldsymbol{O} = \boldsymbol{A}$, and $f'(\boldsymbol{A}) = \boldsymbol{I}$]

$$\boldsymbol{W}_{[n+1]} = \boldsymbol{W}_{[n]} + \eta \boldsymbol{X}(\boldsymbol{T} - \boldsymbol{A})^\mathsf{T} = \boldsymbol{U}\left\{\boldsymbol{\Delta}^{-1}\left[\boldsymbol{I} - \left(\boldsymbol{I} - \eta\boldsymbol{\Delta}^2\right)^{n+1}\right]\right\}\boldsymbol{V}^\mathsf{T}\boldsymbol{T}^\mathsf{T} \ , \tag{19}$$

(see Abdi, 1994, p.54*ff.*).

The Hebbian weight matrix corresponds to the first iteration of the algorithm, i.e.,

$$\boldsymbol{W}_{[1]} = \boldsymbol{U}\left\{\boldsymbol{\Delta}^{-1}\left[\boldsymbol{I} - \left(\boldsymbol{I} - \eta\boldsymbol{\Delta}^{2}\right)^{1}\right]\right\}\boldsymbol{V}^{\mathsf{T}}\boldsymbol{T}^{\mathsf{T}} = \boldsymbol{U}\left\{\eta\boldsymbol{\Delta}\right\}\boldsymbol{V}^{\mathsf{T}}\boldsymbol{T}^{\mathsf{T}} = \eta\boldsymbol{X}\boldsymbol{T}^{\mathsf{T}} \ . \quad (20)$$

Equation 19 characterizes the values of η that allow the iterative process to converge. Denoting by $\delta_{\max}$ the largest singular value of $\boldsymbol{X}$, if η is such that

$$0 < \eta < 2\delta_{\max}^{-2} \quad (21)$$

then it can be shown that (see Abdi, 1994)

$$\lim_{n\to\infty}\left(\boldsymbol{I} - \eta\boldsymbol{\Delta}^{2}\right)^{n} = \boldsymbol{0}, \text{ and } \lim_{n\to\infty}\boldsymbol{W}_{[n]} = \boldsymbol{U}\boldsymbol{\Delta}^{-1}\boldsymbol{V}^{\mathsf{T}}\boldsymbol{T}^{\mathsf{T}} = \boldsymbol{X}^{+}\boldsymbol{T} \ . \quad (22)$$

The matrix $\boldsymbol{X}^{+} = \boldsymbol{U}\boldsymbol{\Delta}^{-1}\boldsymbol{V}^{\mathsf{T}}$ is the *pseudo-inverse* of $\boldsymbol{X}$. It gives a least-square optimal solution for the association between the input and the target. Thus, the linear heteroassociator is equivalent to *linear multiple regression*. If η is outside the interval defined by Equation 21, then both the singular values and the elements of the weight matrix will grow at each iteration. In practice, because neural networks are simulated by digital computers, the weight matrix will eventually reach the limits of the precision of the machine.

When the target vectors are the same as the input vectors (*i.e.,* when each input is associated to itself), the linear heteroassociator becomes a linear autoassociator. The previous approach shows that, now, the Hebbian weight matrix is the cross-product matrix

$$\boldsymbol{W}_{[1]} = \boldsymbol{X}\boldsymbol{X}^{\mathsf{T}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{\mathsf{T}} \ . \quad (23)$$

With Widrow-Hoff learning, when convergence is reached, all nonzero eigenvalues of the weight matrix are equal to 1. The weight matrix is then said to be *spherical*; it is equal to

$$\boldsymbol{W}_{[\infty]} = \boldsymbol{U}\boldsymbol{U}^{\mathsf{T}} \ . \quad (24)$$

Because the statistical technique of *principal component analysis* (PCA) computes the eigendecomposition of a cross-product matrix similar to $\boldsymbol{W}$, the linear autoassociator is considered as the neural network equivalent of PCA.

5 Optimization, Derivative and Matrices

Neural networks are often used to optimize a function of the synaptic weights. The *differentiation* of a function is the major concept for exploring *optimization* problems and, for neural networks, it involves differentiating vectors or matrix functions. In this context, we need to consider the transfer function as being a function of the weight vector. This is expressed by rewriting Equation 6 as

$$o = f\left(\boldsymbol{w}\right) \ . \quad (25)$$

The derivative of $f(\boldsymbol{w})$ with respect to the $I \times 1$ vector $\boldsymbol{w}$ is denoted by $\boldsymbol{\nabla_{f(w)}}$. It is also called the *gradient* of f, i.e.,

$$\boldsymbol{\nabla_{f(w)}} = \frac{\partial f}{\partial \boldsymbol{w}} = \left[\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_i}, \ldots, \frac{\partial f}{\partial w_I}\right]^{\mathsf{T}} . \tag{26}$$

For example, the derivative of the output of a linear neuron is

$$\frac{\partial f}{\partial \boldsymbol{w}} = \left[\frac{\partial \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}}{\partial w_1}, \ldots, \frac{\partial \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}}{\partial w_i}, \ldots, \frac{\partial \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}}{\partial w_I}\right]^{\mathsf{T}} = \left[x_1, \ldots, x_i, \ldots, x_I\right]^{\mathsf{T}} = \boldsymbol{x} . \tag{27}$$

When a function is twice differentiable, the second order derivatives are stored in a matrix called the *Hessian* matrix of the function. It is often denoted by $\boldsymbol{H}$ or $\boldsymbol{\nabla_f^2}$ and is formally defined as

$$\boldsymbol{H} = \boldsymbol{\nabla_f^2} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 w_2} & \cdots & \frac{\partial^2 f}{\partial w_1 w_I} \\ \frac{\partial^2 f}{\partial w_2 w_1} & \frac{\partial^2 f}{\partial w_2^2} & \cdots & \frac{\partial^2 f}{\partial w_2 w_I} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_I w_1} & \frac{\partial^2 f}{\partial w_I w_2} & \cdots & \frac{\partial^2 f}{\partial w_I^2} \end{bmatrix} . \tag{28}$$

5.1 Conditions for minimum

A standard problem is to show that a given learning rule finds an optimum solution in the sense that a function of the weight vector (or matrix) called the *error function* reaches its minimum value when learning has converged. Often, the error function is defined as the sum of the squared error over all patterns.

When the gradient of the error function can be evaluated, a necessary condition for optimality (*i.e.,* either minimum or maximum) is to find a weight vector $\widetilde{\boldsymbol{w}}$ such that

$$\boldsymbol{\nabla_{f(\tilde{w})}} = \boldsymbol{0} . \tag{29}$$

This condition is also sufficient provided $\boldsymbol{H}$ is positive definite (cf. Haykin, 1999).

5.2 Taylor expansion

The Taylor expansion is the standard technique used to obtain a linear or a quadratic approximation of a function of one variable. Recall that the Taylor expansion of a continuous function $f(x)$ is

$$\begin{aligned} f(x) &= f(a) + (x-a)\frac{f^{'}(a)}{1!} + (x-a)^2\frac{f^{''}(a)}{2!} + \ldots (x-a)^n\frac{f^{[n]}(a)}{n!} + \ldots \\ &= f(a) + (x-a)\frac{f^{'}(a)}{1!} + (x-a)^2\frac{f^{''}(a)}{2!} + \mathcal{R}_2 . \end{aligned} \tag{30}$$

(where $\mathcal{R}_2$ represents all the terms of higher order than 2, and a is a "convenient" value at which to evaluate f).

This technique can be extended to matrix and vector functions. It involves the notion of gradient and Hessian. Now a vector function $f(\boldsymbol{x})$ is expressed as:

$$f(\boldsymbol{x}) = f(\boldsymbol{a}) + f(\boldsymbol{x}-\boldsymbol{a})^{\mathsf{T}}\boldsymbol{\nabla}_{\boldsymbol{f(a)}} + f(\boldsymbol{x}-\boldsymbol{a})^{\mathsf{T}}\boldsymbol{\nabla}^{\boldsymbol{2}}_{\boldsymbol{f(a)}} f(\boldsymbol{x}-\boldsymbol{a}) + \mathcal{R}_2 \ . \quad (31)$$

5.3 Iterative minimization

A learning rule can be shown to converge to an optimum if it diminishes the value of the error function at each iteration. When the gradient of the error function can be evaluated, the *gradient* technique (or *steepest descent*) adjusts the weight vector by moving it in the direction opposite to the gradient of the error function. Formally, the correction for the $(n+1)$-th iteration is

$$\boldsymbol{w}_{[n+1]} = \boldsymbol{w}_{[n]} + \boldsymbol{\Delta} = \boldsymbol{w}_{[n]} - \eta\boldsymbol{\nabla}_{\boldsymbol{f(w)}} \quad (32)$$

(where $\boldsymbol{\nabla}_{\boldsymbol{f(w)}}$ is computed for $\boldsymbol{w}_{[n]}$).

As an example, let us show that for a linear heteroassociator, the Widrow-Hoff learning rule minimizes iteratively the squared error between target and output. The error function is

$$e^2 = (t-o)^2 = t^2 + o^2 - 2to = t^2 + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{w}\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} - 2t\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} \ . \quad (33)$$

The gradient of the error function is

$$\frac{\partial e}{\partial \boldsymbol{w}} = 2(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x})\boldsymbol{x} - 2t\boldsymbol{x} = -2(t - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x})\boldsymbol{x} \ . \quad (34)$$

The weight vector is corrected by moving it in the opposite direction of the gradient. This is obtained by adding a small vector denoted $\boldsymbol{\Delta}_{\boldsymbol{w}}$ opposite to the gradient. This gives the following correction for iteration $n+1$:

$$\boldsymbol{w}_{[n+1]} = \boldsymbol{w}_{[n]} + \boldsymbol{\Delta}_{\boldsymbol{w}} = \boldsymbol{w}_{[n]} - \eta\frac{\partial e}{\partial \boldsymbol{w}} = \boldsymbol{w}_{[n]} + \eta(t - \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x})\boldsymbol{x} = \boldsymbol{w}_{[n]} + \eta(t-o)\boldsymbol{x} \ . \quad (35)$$

This gives the rule defined by Equation 9.

The gradient method works because the gradient of $\boldsymbol{w}_{[n]}$ is a first order Taylor approximation of the gradient of the optimal weight vector $\widetilde{\boldsymbol{w}}$. It is a favorite technique in neural networks because the popular error backpropagation is a gradient technique.

Newton's method is a second order Taylor approximation, it uses the inverse of the Hessian of $\boldsymbol{w}$ (supposing it exists). It gives a better numerical approximation but necessitates more computation. Here the correction for iteration $n+1$ is

$$\boldsymbol{w}_{[n+1]} = \boldsymbol{w}_{[n]} + \boldsymbol{\Delta} = \boldsymbol{w}_{[n]} - (\boldsymbol{H}^{-1})(\boldsymbol{\nabla}_{\boldsymbol{f(w)}}) \quad (36)$$

(where $\boldsymbol{\nabla}_{\boldsymbol{f(w)}}$ is computed for $\boldsymbol{w}_{[n]}$).

6 Useful References

Linear algebra at the level of this presentation is available in the following recent books: Abdi *et al.* (1999), Bishop (1995) Ellacot and Bose (1996), Haggan, Demuth, and Beale (1996), Haykin (1999), Reed and Marks (1999), Ripley (1996), and Rojas (1996).

See also: Artificial neural networks: neurocomputation; Backpropagation; Hebb, Donald Olding (1904–1985); Statistical pattern recognition.

References

[1] ABDI, H. (1994a) *Les réseaux de neurones*. Grenoble, France: PUG.

[2] ABDI, H., VALENTIN, D., & EDELMAN, B. (1999) *Neural networks*. Thousand Oak, CA: Sage.

[3] BISHOP, C.M. (1995) *Neural network for pattern recognition*. Oxford, UK: Oxford University Press.

[4] ELLACOTT, S., & BOSE, D. (1996) *Neural networks: Deterministic methods of analysis*. London: ITC.

[5] HAGAN, M. T., DEMUTH, H. B., & BEALE, M. (1996) *Neural networks design*. Boston: PWS.

[6] HAYKIN, S. (1999) *Neural networks: A comprehensive foundation* (2nd ed). New York: Prentice Hall.

[7] REED, R.D., MARKS R.J. (1999) *Neural smithing*. Cambridge, MA: MIT press.

[8] RIPLEY, B.D. (1996) *Pattern recognition and neural networks*. Cambridge, MA: Cambridge University Press.

[9] ROJAS, R. (1996) *Neural networks*. New York: Springer-Verlag.