

8 Self-Organizing Neural Networks

8.1 Supervised and Unsupervised Learning

Learning algorithms which were considered for a single perceptron, linear adaline, and multilayer perceptron belong to the class of **supervised learning** algorithms.

In this case the training data is divided into input signals, $\mathbf{x}(n)$, and target signals, $\mathbf{d}(n)$. A typical learning algorithm is driven by error signals $\varepsilon(n)$ which are the differences between the actual network output, $\mathbf{y}(n)$, and the desire (or target) output for a given input.

For a pattern learning, we can express the weight update in the following general form

$$\Delta \mathbf{w}(n) = \mathcal{L}(\mathbf{w}(n), \mathbf{x}(n), \varepsilon(n))$$

where \mathcal{L} represents a learning algorithm.

If we say that a neural network can describe a model of data, then a multilayer perceptron describes the data in a form of a hypersurface which approximates a functional relationship between $\mathbf{x}(n)$, and $\mathbf{d}(n)$.

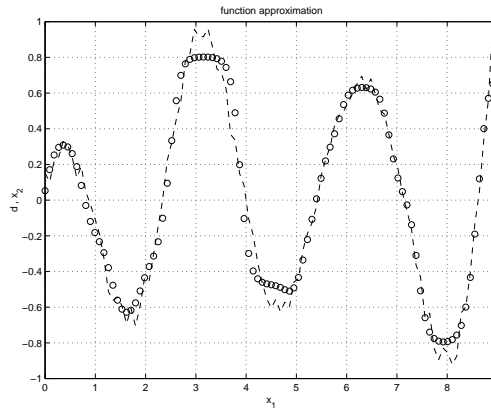


Figure 8–1: Functional relationship between data points

Self-organizing neural networks — Unsupervised Learning

Following the terminology used in [Hay99], we say that a self-organising neural network, which employs the principle of unsupervised learning, discovers **characteristic features** in input data without using a target or desired output (an external teacher).

Information about the characteristic features of input data is created during the learning process and stored in the synaptic weights. Output signals describe relationship between the current input signals and the weight vectors.

Two basic groups of unsupervised learning algorithms and related self-organizing neural networks, namely:

- (Generalised) Hebbian Learning
- Competitive Learning

can be distinguished by the type of characteristic features that they “discover” from the input data, namely, “shape” of data and clusters of points.

Generalised Hebbian Learning extracts from data a set of **principal directions** along which data is organised in a p -dimensional space.

Each direction is represented by a relevant weight vector. The number of those principal directions is, at most, equal to the dimensionality of the input space p .

In an illustrative example presented in Figure 8–2 the two-dimensional data is organised along two principal directions, w_1 and w_2 .

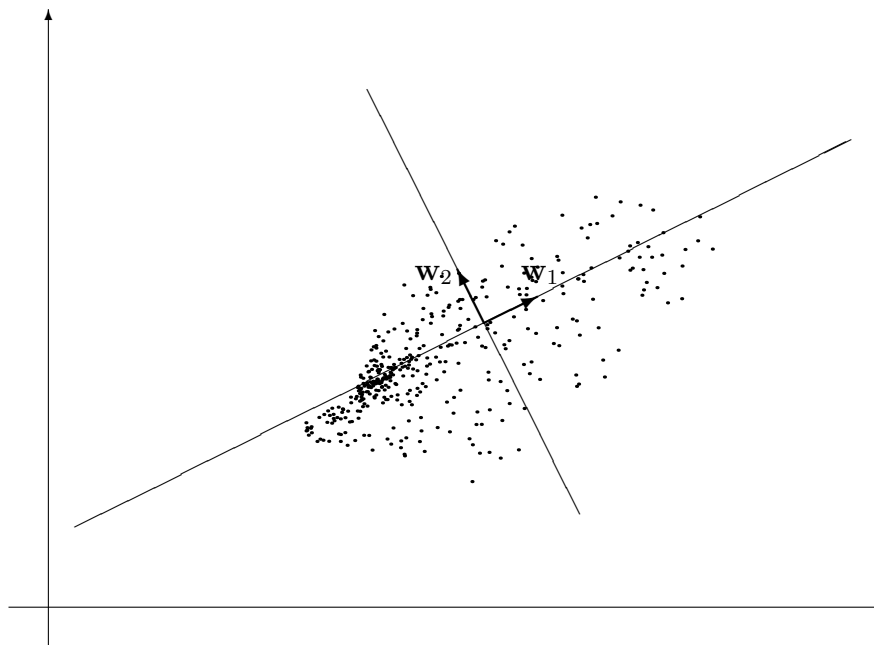


Figure 8–2: A 2-D pattern with principal directions

Competitive Learning extracts from data a set of **centers of data clusters**.

Each center point is stored as a weight vector. It is obvious that the number of clusters is independent of dimensionality of the input space.

In Figure 8–3 two-dimensional data is organised in three clusters, their centres represented by three weight vectors.

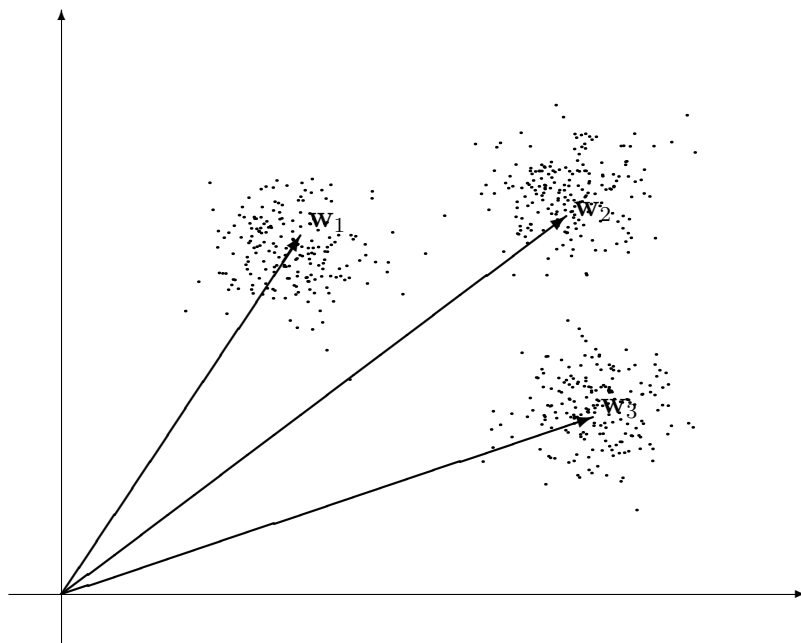


Figure 8–3: A 2-D pattern with three clusters of data

An important extension of a basic competitive learning is known as **feature maps**. A feature map is obtained by adding some form of topological organization to neurons.

Generalized Hebbian Learning

8.2 Basic structure of Hebbian learning neural networks

The basic structure of a neural network employing Hebbian learning algorithms consists of a single layer decoding part and a learning part as illustrated in Figure 8–4.

The decoding layer contains a single weight matrix W which is $m \times p$. Each row weight vector is associated with one neuron.

The decoding layer is often linear, which further simplifies the network structure. The complexity of the network comes from the structure of the learning law employed.

Note the absence of the target value in the encoding/learning part (un-supervised learning).

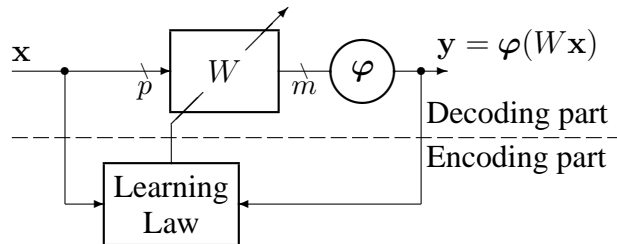


Figure 8–4: A block-diagram of a basic Hebbian learning neural network

The basic idea behind a Hebbian learning law is to make the update of a synaptic weight proportional to both input and output signals, that is,

$$w_{ji}(n+1) = f(w_{ji}(n), y_j(n), x_i(n)) = w_{ji}(n) + \eta y_j(n) x_i(n) \quad (8.1)$$

These two signals, y_j and x_i are locally available at the ji synapse, therefore, this type of a learning law is termed as a **local learning law**.

The concept of the **local learning** law is illustrated in Figure 8–5.

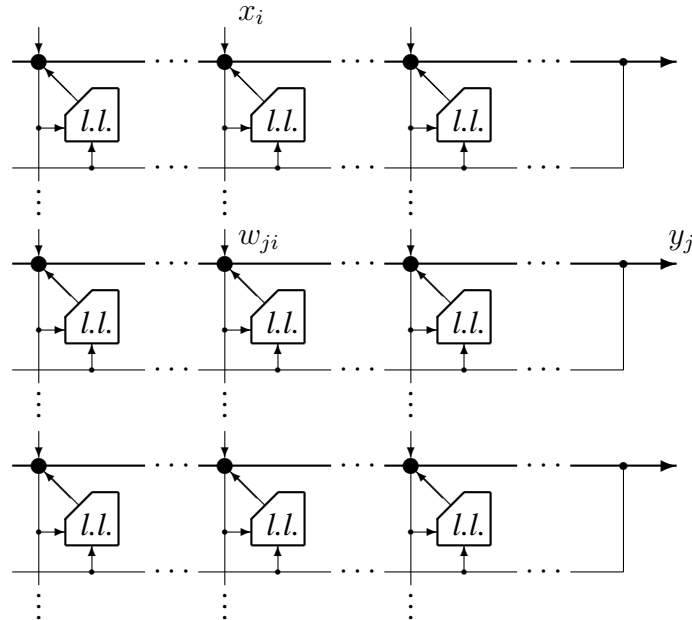


Figure 8–5: A neural network with a local learning law circuitry (*l.l.*)

It may be observed that the neuron output signal, y_j , is available locally at the synapse through the additional feedback connection. This feedback is essential to the learning process.

The **basic Hebbian learning** law in the form as in eqn (8.1) cannot be used because it is fundamentally **unstable**, that is, weights reveal an unlimited grow during the learning process.

It is relatively simple to show that if a stable solution to the learning law (8.1) exists it must be zero.

Assuming for simplicity linear neurons and re-writing eqn (8.1) in a matrix form gives:

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \eta \mathbf{y}(n) \mathbf{x}^T(n) = \mathbf{W}(n) + \eta \mathbf{W}(n) \mathbf{x}(n) \mathbf{x}^T(n)$$

Application of the expectation operator, $E[\cdot]$, to both sides yields

$$E[\mathbf{W}] = E[\mathbf{W}] + \eta E[\mathbf{W}] E[\mathbf{x} \mathbf{x}^T]$$

Hence, the steady-state value of the weight matrix, $\bar{\mathbf{W}} = E[\mathbf{W}]$, must satisfy the following equation

$$\bar{\mathbf{W}} R = 0$$

where

$$R = E[\mathbf{x} \mathbf{x}^T] \approx \frac{1}{N} X X^T$$

is the input **correlation matrix**.

The input correlation matrix is non-singular, therefore, the only possible steady-state value of the weight matrix is $\bar{\mathbf{W}} = 0$.

More detailed consideration can be found in [Hay99].

8.3 Stable Hebbian learning

In order to stabilize a Hebbian learning law two basic steps are required

- Assuming that \mathbf{x} is a p -dimensional random vector representing the input data, it is required that its **mean** to be zero:

$$E[\mathbf{x}] = 0$$

In practical calculations with MATLAB, when input vectors are collected in the $p \times N$ input matrix X , the non-zero mean is removed from the input data in the following way:

$$\mathbf{mX} = \text{mean}(X, 2) ; \quad X = X - \mathbf{mX}(\text{ones}(1, N)) ;$$

Note that if the mean is zero **correlation** matrix is identical to **covariance** matrix.

- The basic Hebbian learning law is to be modified in such a way that the magnitude of weight vectors should tend to unity.

Assuming for simplicity a single neuron network, it can be achieved by the following normalization:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta y \mathbf{x}^T ; \quad \mathbf{w} \leftarrow \mathbf{w} / \|\mathbf{w}\| \quad (8.2)$$

where the vector magnitude is calculated in an usual way as:

$$\|\mathbf{w}\| = \sqrt{\sum_{i=1}^p w_i^2}, \quad \text{also} \quad \|\mathbf{w}\|^2 = \mathbf{w} \mathbf{w}^T$$

Normalization as in eqn (8.2) is computationally relatively complex, therefore, we can use the following simplification based on the Taylor series expansion.

$$\begin{aligned} \frac{1}{\|\mathbf{w} + \eta y \mathbf{x}^T\|} &= \\ &= \frac{1}{\sqrt{(\mathbf{w} + \eta y \mathbf{x}^T)(\mathbf{w} + \eta y \mathbf{x}^T)^T}} = \frac{1}{\sqrt{\mathbf{w}\mathbf{w}^T + 2\eta y \mathbf{w}\mathbf{x} + \eta^2 y^2 \mathbf{x}^T \mathbf{x}}} \end{aligned}$$

If we assume that the previous weight vector was normalised, that is,

$$\mathbf{w}\mathbf{w}^T = \|\mathbf{w}\|^2 = 1$$

and that $\eta \ll 1$ is small so that η^2 is negligible, then we can further write:

$$\begin{aligned} \frac{\mathbf{w} + \eta y \mathbf{x}^T}{\|\mathbf{w} + \eta y \mathbf{x}^T\|} &\approx \frac{\mathbf{w} + \eta y \mathbf{x}^T}{\sqrt{1 + 2\eta y \mathbf{w}\mathbf{x}}} \approx (\mathbf{w} + \eta y \mathbf{x}^T)(1 - \eta y \mathbf{w}\mathbf{x}) \\ &\approx \mathbf{w} + \eta y \mathbf{x}^T - \eta y^2 \mathbf{w} - \eta^2 y^2 \mathbf{x}^T \approx \mathbf{w} + \eta y (\mathbf{x}^T - y \mathbf{w}) \end{aligned}$$

It can be proved that if we update weights according to the last equation, that is,

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta y(n)(\mathbf{x}^T(n) - y(n)\mathbf{w}(n)) \quad (8.3)$$

then the magnitude of the weight vector will be close to unity, $\|\mathbf{w}\| \rightarrow 1$.

This is the form of the weight update used in the generalised Hebbian learning.

8.4 A single neuron case — the Oja's rule

The unsupervised learning algorithm described in eqn (8.3) for a single neuron case is known as the Oja's rule, and can be written in the following form:

$$\Delta \mathbf{w} = \eta y (\mathbf{x}^T - y \mathbf{w}) = \eta y \tilde{\mathbf{x}}^T, \quad y = \mathbf{w} \mathbf{x} = \mathbf{x}^T \mathbf{w}^T \quad (8.4)$$

where the augmented input vector is:

$$\tilde{\mathbf{x}} = \mathbf{x} - y \mathbf{w}^T \quad (8.5)$$

The negative term brings in the required stabilization of the learning law. To show this we calculate the projection of the update vector, $\Delta \mathbf{w}$ onto the current weight vector, \mathbf{w} :

$$\Delta \mathbf{w} \mathbf{w}^T = \mathbf{x}^T \mathbf{w}^T - y \mathbf{w} \mathbf{w}^T = y(1 - \|\mathbf{w}\|^2)$$

Therefore, if the current weight vector is not on the unit circle, the update vector will bring the next weight vector closer to the the unit circle.

The Figure 8–6 illustrate the internal structure of the Oja's rule (8.4).

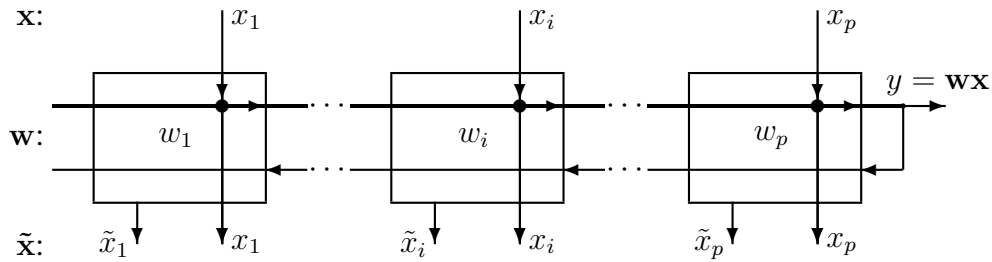


Figure 8–6: Structure of the Oja's rule neural network

Each synapse aggregates the dendritic signal, and, during learning, generates the augmented input signal, and updates its weight.

Extraction of the first principal direction

It is now possible to show that applying the learning law of eqn (8.4), the weight vector \mathbf{w} converges to the **eigenvector** \mathbf{q}_1 of the input **correlation matrix** R associated with the largest **eigenvalue** λ_1 of R .

The direction of the eigenvector \mathbf{q}_1 is referred to as the first **principal direction**.

The sketch of the proof of the necessary convergence condition is as follows (for a rigorous treatment the reader is referred to [Hay99]).

Substitution of eqn (8.5) in eqn (8.4) yields:

$$\Delta \mathbf{w} = \eta(\mathbf{w}\mathbf{x}\mathbf{x}^T - \mathbf{w}\mathbf{x}\mathbf{x}^T\mathbf{w}^T\mathbf{w})$$

Applying the statistical expectation operator to both sides gives:

$$E[\Delta \mathbf{w}] = \eta(\mathbf{w}E[\mathbf{x}\mathbf{x}^T] - \mathbf{w}E[\mathbf{x}\mathbf{x}^T]\mathbf{w}^T\mathbf{w}) \quad (8.6)$$

The terms in eqn (8.6) can be estimated as follows. If we assume that the weight vector converges to a steady-state value, then the expectation of the weight update vectors is zero, that is, $E[\Delta \mathbf{w}] = 0$. The expectation of outer products of input vectors is the covariance (correlation) matrix:

$$R = E[\mathbf{x}\mathbf{x}^T] \approx \frac{1}{N}X X^T$$

Now, eqn (8.6) can be written as

$$\mathbf{w}R = (\mathbf{w}R\mathbf{w}^T)\mathbf{w} \quad (8.7)$$

If we denote the scalar:

$$\lambda_1 = \mathbf{w}R\mathbf{w}^T \quad (8.8)$$

Using definition (8.8) we can finally re-write eqn (8.7) in the following form

$$\mathbf{w}R = \lambda_1 \mathbf{w} \quad (8.9)$$

Alternatively, if we assume that

$$\mathbf{w}(n) \rightarrow \pm \mathbf{q}_1^T \text{ as } n \rightarrow \infty \quad (8.10)$$

we have:

$$R\mathbf{q}_1 = \lambda_1 \mathbf{q}_1 \quad \text{where} \quad \lambda_1 = \mathbf{q}_1^T R \mathbf{q}_1 \quad (8.11)$$

Eqn (8.11) specifies a pair: an eigenvector \mathbf{q}_1 and related eigenvalue λ_1 which are characteristic values of a matrix (R in this case) such that, if R acts on \mathbf{q}_1 it modifies only the magnitude of the eigenvector.

It can be shown that a “well behaving” $p \times p$ matrix has exactly p eigenvector-eigenvalue pairs.

It can also be shown that λ_1 defined in eqn (8.8) or (8.11) and obtained using the Oja’s rule is the largest eigenvalue of the input correlation matrix, R .

As it is stated in eqn (8.10) the weight vector converges to the eigenvector \mathbf{q}_1 , but the orientation of these two vectors does not have to be the same.

Therefore, comparing the weight vector with the eigenvector when monitoring the convergence process, it is better to use the projection rather than the difference, that is:

$$|\mathbf{w} \cdot \mathbf{q}_1| \rightarrow 1 \quad \text{whereas} \quad \|\mathbf{w} - \mathbf{q}_1\| \rightarrow 0 \text{ or } +2 \quad (8.12)$$

Condition (8.12) can be used to conveniently monitor the convergence process.

8.5 Self-Organizing Principal Component Analysis

The single neuron structure can be extended into a p -neuron network, the weight vector associated with subsequent neurons extracting the subsequent eigenvectors of the input correlation matrix.

Such a network performs the **Principal Component Analysis** also known as the Karhunen-Loève transform. The objective of this analysis (transform) is to extract all principal directions characterising input data.

The learning law involved is known as the Sanger's rule or Generalized Hebbian Algorithm (GHA).

The idea behind the generalization of the Oja's rule neural network is to use in the learning part of neurons the **augmented input vectors** as specified by eqn (8.5).

The weight update in the Sanger's rule, that is, the Generalized Hebbian Algorithm can be described in the following way:

$$\begin{aligned}
 y_1 &= \mathbf{w}_1 \mathbf{x}, \quad \tilde{\mathbf{x}}_1 = \mathbf{x} - y_1 \mathbf{w}_1^T, & \Delta \mathbf{w}_1 &= \eta y_1 \tilde{\mathbf{x}}_1^T \\
 y_2 &= \mathbf{w}_2 \mathbf{x}, \quad \tilde{\mathbf{x}}_2 = \tilde{\mathbf{x}}_1 - y_2 \mathbf{w}_2^T, & \Delta \mathbf{w}_2 &= \eta y_2 \tilde{\mathbf{x}}_2^T \\
 &\dots & & \\
 y_j &= \mathbf{w}_j \mathbf{x}, \quad \tilde{\mathbf{x}}_j = \tilde{\mathbf{x}}_{j-1} - y_j \mathbf{w}_j^T, & \Delta \mathbf{w}_j &= \eta y_j \tilde{\mathbf{x}}_j^T \\
 &\dots & &
 \end{aligned} \tag{8.13}$$

The above equations are illustrated in Figure 8–7.

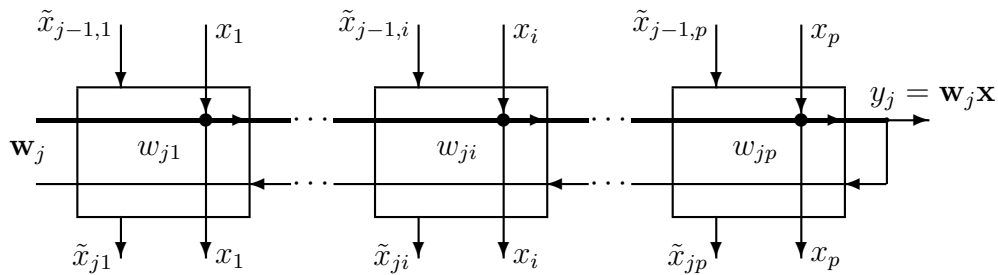


Figure 8–7: Structure of the j th neuron of the GHA neural network

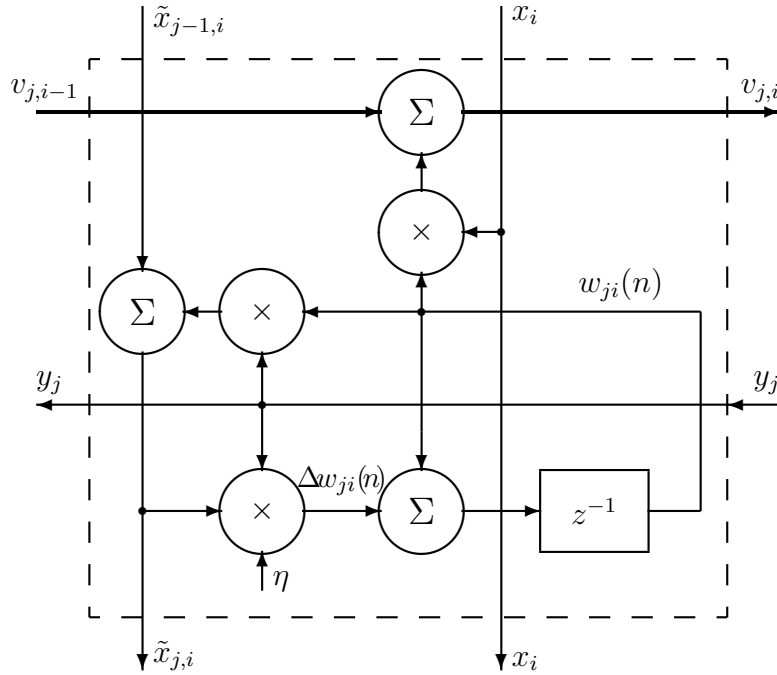


Figure 8–8: A synapse implementing the Generalised Hebbian Learning

Detailed structure of a single synapse implementing the generalised Hebbian learning is presented in Figure 8–8. The signals in the synapse are specified as follows:

- the dendritic activation signal

$$v_{j,i} = v_{j,i-1} + w_{ji} \cdot x_i, \quad (v_{j,0} = 0, \quad y_j = v_{j,p})$$

- the augmented input signal

$$\tilde{x}_{ji} = \tilde{x}_{j-1,i} - w_{ji} \cdot y_j$$

- the synaptic weight update (learning law)

$$\Delta w_{ji}(n) = \eta \cdot y_j(n) \cdot \tilde{x}_{ji}(n), \quad w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n)$$

In order to re-write the **Generalised Hebbian Learning** algorithm in a matrix form, let us first note that from eqn (8.13), we can write the augmented input signal vectors in the following form:

$$\begin{aligned}\tilde{\mathbf{x}}_j^T &= \mathbf{x}^T - [y_1 \dots y_j] \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_j \end{bmatrix} = \mathbf{x}^T - [y_1 \dots y_j \ 0 \dots 0] W \\ &= \mathbf{x}^T - \tilde{\mathbf{y}}_j^T W\end{aligned}$$

where

$$\tilde{\mathbf{y}}_j^T = [y_1 \dots y_j \ 0 \dots 0]$$

is the output vector \mathbf{y} in which the last $m - j$ components are set to zero. Subsequently, the weight update for the j neuron specified in eqn (8.13) can be re-written in the following form

$$\Delta \mathbf{w}_j = \eta (y_j \mathbf{x}^T - y_j \tilde{\mathbf{y}}_j^T W)$$

Finally, the pattern update of the whole weight matrix in the GHA algorithm can be expressed in the following compact form:

$$\mathbf{y} = W \mathbf{x}, \quad \Delta W = \eta (\mathbf{y} \mathbf{x}^T - \text{tril}(\mathbf{y} \mathbf{y}^T) W) \quad (8.14)$$

where $\text{tril}(\cdot)$ denotes the lower-triangular matrix with elements above the main diagonal set to zero.

The sketch of the prove that the network weight vectors converge to the eigenvalues of the input correlation matrix is as follows. Taking the statistical expectation operator on both sides of the weight update equation (8.14), and assuming that in the steady-state the updates are zeros, we have

$$0 = E[W \mathbf{x} \mathbf{x}^T] - E[\text{tril}(W \mathbf{x} \mathbf{x}^T W^T) W]$$

This can be re-written as

$$WR = \text{tril}(WRW^T)W$$

or

$$\begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_p \end{bmatrix} R = \text{tril}\left(\begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_p \end{bmatrix} R \begin{bmatrix} \mathbf{w}_1^T & \dots & \mathbf{w}_p^T \end{bmatrix}\right) \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_p \end{bmatrix}$$

The lower-triangular matrix in the right-hand side is of the form:

$$\Lambda = \text{tril}(WRW^T) = \begin{bmatrix} \lambda_1 & & \\ \{\lambda_{ji}\} & \ddots & \mathbf{0} \\ & & \lambda_p \end{bmatrix}, \quad \text{where} \quad \lambda_{ji} = \mathbf{w}_j R \mathbf{w}_i^T$$

Therefore we finally have

$$\begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_p \end{bmatrix} R = \begin{bmatrix} \lambda_1 & & \\ \{\lambda_{ji}\} & \ddots & \mathbf{0} \\ & & \lambda_p \end{bmatrix} \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_p \end{bmatrix} \quad (8.15)$$

or

$$WR = \Lambda W \quad (8.16)$$

The first row of eqn (8.15), namely,

$$\mathbf{w}_1 R = \lambda_1 \mathbf{w}_1, \quad \text{where} \quad \lambda_1 = \mathbf{w}_1 R \mathbf{w}_1^T$$

is exactly the same as eqn (8.11) from the Oja's network, the weight vector of the first neuron, \mathbf{w}_1 , converging to the eigenvector of the input correlation matrix associated with the largest eigenvalue, λ_1 .

In order to show that the other weight vectors converge to subsequent eigenvectors it is enough to show that the off-diagonal coefficients $\lambda_{ji} = \mathbf{w}_j R \mathbf{w}_i^T$ converge to zero due to orthogonality of the eigenvectors.

8.6 Example of image compression using GHA

An image to be compressed is a $rr \times cc$ sub-image from 'gatlin':

```
load gatlin
rr = 120 ; cc = 180 ; %numbers of rows and columns of Img
Img = X((1:rr)+20, (1:cc)+20) ;
figure(1), image(Img) , colormap(map)
```

The image is divided into $r \times c$ blocks, each n th block being converted into a $p = r \times c$ component vector $\mathbf{x}(n)$. These vectors are stored in a $p \times N$ matrix X :

```
r = 4 ; c = 4 ; p = r*c ;
X = blkM2vc(Img, [r c]) ;
[p N] = size(X) ; % X is 16 by 1350 = 120 by 180
```

The next step is to remove the mean from X , that is, to normalize it. The pattern matrix X is now ready to be used in the learning algorithm:

```
Xm = mean(X')' ;
X = X - Xm(:, ones(1, N)) ;
X = X/max(max(abs(X))) ;
```

The internal learning loop goes through all patterns from X (one epoch) updating weights in a 'pattern mode'. In order to monitor the convergence process the length of the weight vectors, lw , is calculated. It is expected that when the weights converge to respective eigenvectors their lengths will be unity. This condition is checked in the outer loop.

```

m = 4 ; % number of neurons
W = 0.6*(rand(m, p)-0.5) ; % weight initialisation
lw = sum((W.^2)') ; % length of weight vectors
W2 = zeros(N, m) ; % changes to the length of weight vectors
figure(2)
eta = 2e-2 ; % learning gain
er = .05 ; % the length convergence error
ep = 0 ; % number of training epochs
while (sum( abs(1-lw) < er ) < m) & (ep < 16)
    [rs rn] = sort(rand(1, N)) ;
    for n = 1:N
        x = X(:, rn(n)) ; % randomised selection of patterns
        y = W*x ;
        dW = eta*(y*x' - tril(y*y')*W) ;
        W = W + dW ;
        lw = sum((W.^2)') ; % length of weight vectors
        W2(n, :) = lw ;
    end
    plot(W2),
ep = ep+1 ;
    if ep == 1
        plot(W2), axis([0 1400 0 1.1])
        title(['lengths of weight vectors during training'])
        xlabel('pattern number')
    end
    grid, drawnow
end

plot(W2), axis([0 1400 0.5 1.1])
title(['lengths of weight vectors after ', num2str(ep), 'epochs'])
xlabel('pattern number (the last epoch)'), grid, drawnow

```

It the above example, there are $m = 4$ neurons, that is, the neural network is trained for only $m = 4$ out of $p = 16$ ‘principal directions’, specified by the eigenvectors of the input correlation matrix.

After one pass through the training data only the first weight vector representing the most significant eigenvector has converged to achieve the unity length as demonstrated in Figure 8–9.

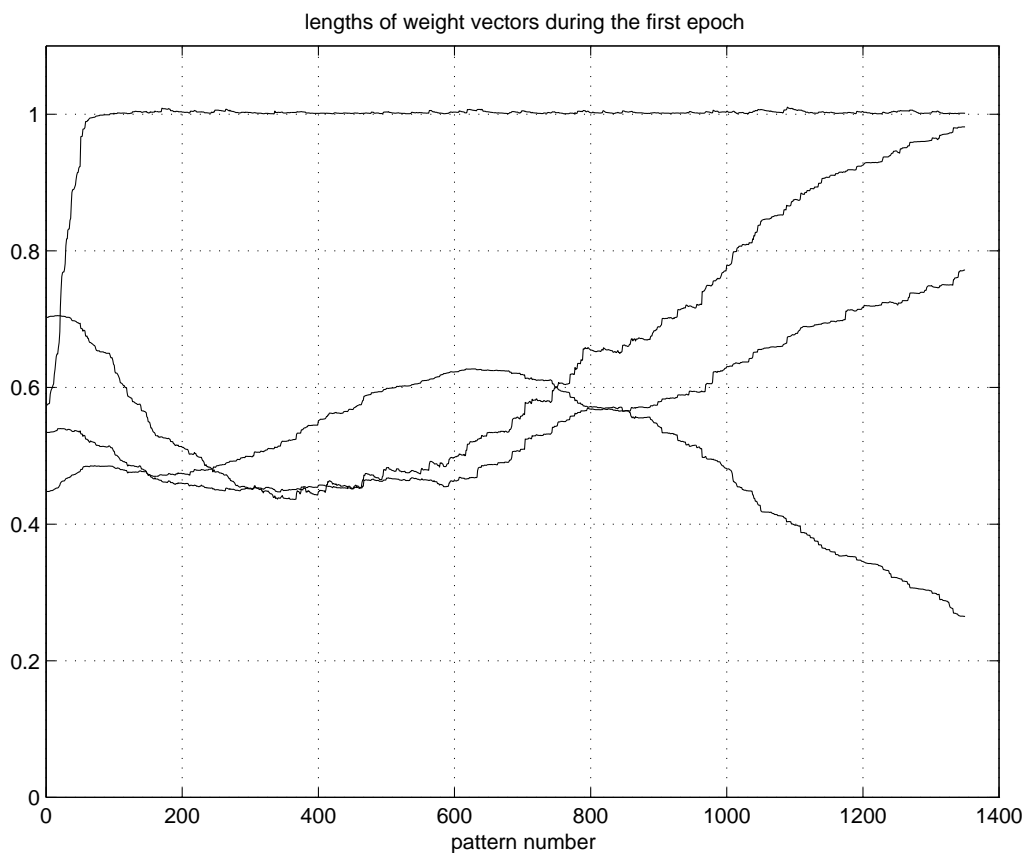


Figure 8–9: The length of the weight vectors representing the m most significant principal directions.

After $ep = 7$ epochs, all $m = 4$ weight vectors have attained the unity length within the error specified by er as presented in Figure 8–10.

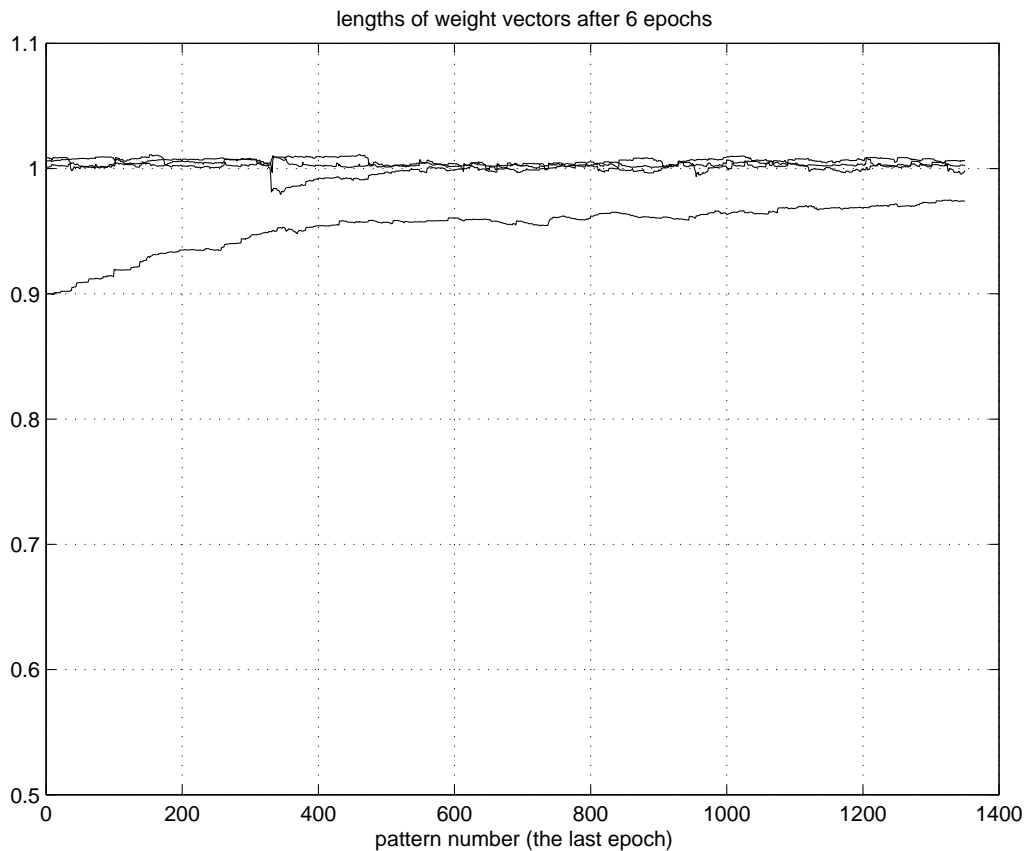


Figure 8-10: The length of the weight vectors representing the m most significant principal directions.

It is possible to observe that during training the weight vectors converge in a serial way, one by one, starting from the weight vector representing the most significant principal direction.

Next we will check that the weight vectors are really equal to the eigenvectors of the input correlation matrix, $R = X \cdot X'$. A number of aspects need to be taken into account in this comparison.

First, the `eig` function arranges the eigenvectors as the column vectors whereas the weights are row vectors.

Secondly, the eigenvalues must be sorted in the descending order and the respective eigenvectors must be re-arranged accordingly.

Thirdly, the orientation of the eigenvectors and the weight vectors might be opposite. This can be done as follows:

```
R = (X*X')/N ; % the autocorrelation matrix
[V D] = eig(R) ;
dd = diag(D)
d = flipud(dd(p-m+1:p, :)) ;
VV = fliplr( V(:, p-m+1:p)) ;
WV = W*VV
```

Now, \mathbf{d} contains m largest eigenvalues, and \mathbf{VV} is a $p \times m$ matrix of respective eigenvectors of the input correlation matrix.

In order to verify that the weight matrix has converged to m principal directions we calculate all possible inner products between all m weight vectors and m eigenvectors. The $m \times m$ matrix \mathbf{WV} stores these products. The diagonal terms of this matrix should ideally be equal to ± 1 , whereas the off-diagonal terms should be zero. In our example, we have;

```
WV = W*VV = 1.0004    0.0308   -0.0029    0.0164
              0.0059    0.9972   -0.0465   -0.0316
              0.0039   -0.0199   -1.0029   -0.0001
              -0.0022    0.0353    0.0100   -0.9860
```

which looks as a sensible approximation.

The next test is to check that the variance of the output signals is equal to the eigenvalues of the input correlation matrix.

```
yy = W*X ; % the output signals m by N
vy = var(yy')' ;
[ d    vy ]
 2.2875    2.2913
 0.1575    0.1571
 0.1138    0.1147
 0.0424    0.0415
```

Indeed, the approximation seems to be satisfactory.

In order to obtain the compressed image, the matrix of output vectors $Y = yy$ is transformed first into a reconstructed input matrix $\hat{X} = Xr$

$$\hat{X} = W' \cdot Y \quad (8.17)$$

The reconstructed input matrix can now be re-arranged into $r \times c$ image blocks. This can be done in the following way:

```
Xr = W'*yy ;
Imr = vc2blkM(Xr, r, rr) ;
Imr = round(mc*Imr/max(max(Imr))) ;
figure(3), image(Imr), colormap(gray(mc))
```

The original and compressed images are shown in Figure 8–11.

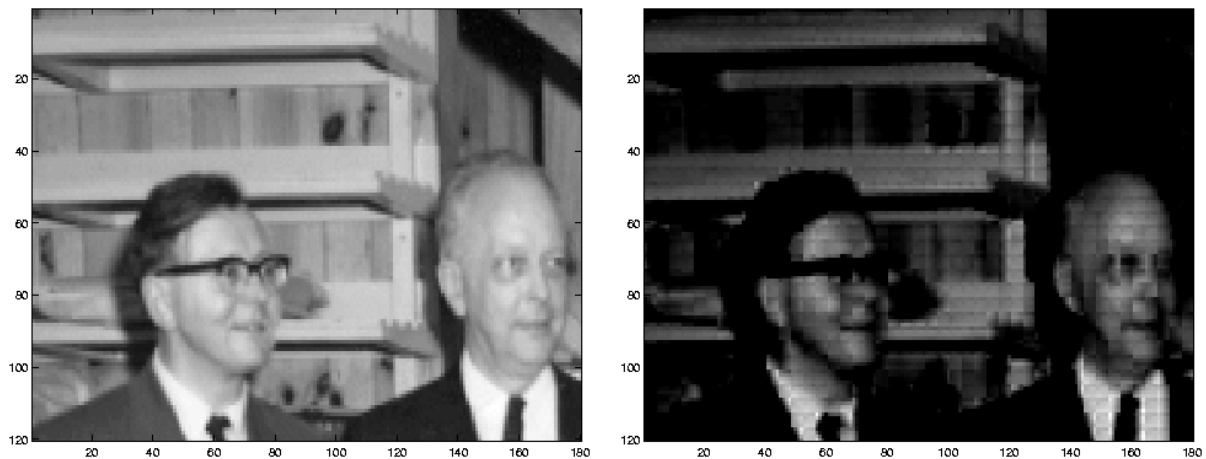


Figure 8–11: The original and compressed images.

References

- [DB98] H. Demuth and M. Beale. *Neural Network TOOLBOX User's Guide. For use with MATLAB*. The MathWorks Inc., 1998.
- [FS91] J.A. Freeman and D.M. Skapura. *Neural Networks. Algorithms, Applications, and Programming Techniques*. Addison-Wesley, 1991. ISBN 0-201-51376-5.
- [Has95] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. The MIT Press, 1995. ISBN 0-262-08239-X.
- [Hay99] Simon Haykin. *Neural Networks – a Comprehensive Foundation*. Prentice Hall, New Jersey, 2nd edition, 1999. ISBN 0-13-273350-1.
- [HDB96] Martin T. Hagan, H Demuth, and M. Beale. *Neural Network Design*. PWS Publishing, 1996.
- [HKP91] Hertz, Krogh, and Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991. ISBN 0-201-51560-1.
- [Kos92] Bart Kosko. *Neural Networks for Signal Processing*. Prentice-Hall, 1992. ISBN 0-13-617390-X.
- [Sar98] W.S. Sarle, editor. *Neural Network FAQ*. Newsgroup: comp.ai.neural-nets, 1998. URL: <ftp://ftp.sas.com/pub/neural/FAQ.html>.