8 Unsupervised Learning

8.1 Linear neuron model: (Hebbian Learning)

$$y = \sum_{i=1}^{n} w_i x_i \tag{8.1.1}$$

In vector form:

$$y = w \bullet x \tag{8.1.2}$$

⍰

Apply Hebbian learning,

$$\Delta w_i = \eta y x_i \tag{8.1.3}$$

$$\Delta w = \eta y x \tag{8.1.4}$$

$$= \eta (w^T x) x = \eta x (x^T w) = \eta (x x^T) w \tag{8.1.5}$$

For multiple patterns the learning rule is,

$$\Delta w = \eta \sum_{p} (x(p) x(p)^T) w \tag{8.1.6}$$

$$R = \sum_{p} x(p) x(p)^T \tag{8.1.7}$$

R is the autocorrelation matrix of training data. It is:

- symmetric,
- positive semi-definite

Proof:

1) Symmetric:

$$R_{ij} = \sum_p x_i(p) x_j(p) = R_{ji}$$

2) Positive semi-definite:
   Consider the quadratic form associated with R,

$$\frac{1}{2} u^T R u \ ,$$

where u is a non-zero real vector.

$$\frac{1}{2} u^T R u = \frac{1}{2} u^T (\sum_p x(p) x(p)^T) u$$

$$= \frac{1}{2} (\sum_p (u^T x(p))(x(p)^T u))$$

$$= \frac{1}{2} \sum_p (u^T x(p))^2 \geq 0$$

$$\Delta w = \eta R w \qquad (8.1.8)$$

In differential equation form,

$$\dot{w} = R w \qquad (8.1.9)$$

⇨ Maximize $\frac{1}{2} w^T R w$

Therefore, hebbian learning of a linear neuron

⇨ Maximizing the quadratic form, E, of R

$$E(w) = \frac{1}{2} w^T R w \qquad (\ 8.1.10)$$

$$= \frac{1}{2}(\sum_{p}(w^{T}x(p))(x(p)^{T}w))$$

$$= \frac{1}{2}\sum_{p}(w^{T}x(p))^{2} = \frac{1}{2}\sum_{p}(y(p))^{2}$$

(8.1.11)

Therefore, Hebbian learning of a linear neuron

- ⇨ Maximizing the quadratic form of R (= $\frac{1}{2}w^{T}Rw$)
- ⇨ Maximizing the average squared output of the neuron.
- ⇨ We also note that if the data is 'zero mean' (E[x] = 0), Hebbian learning also maximizes output variance.Since the neuron is linear, for zero-mean input data, mean squared value of the output equals output variance. i.e.,
  $E[y^{2}] = \sigma_{y}^{2}$

But if R is positive semi-definite, E (w) does not have a maximum. Therefore, E must be constrained. A simple constraint is to make *w* a unit norm vector. The unit norm constraint can be added as a cost to E(w), yielding the new E' as follows:

$$E(w) = \frac{1}{2}w^{T}Rw - \frac{1}{2}\lambda(\|w\|^{2} - 1)$$

(8.1.12)

Here, λ is the Lagragian multiplier.

Calculating the gradient,

$$\nabla_{w}E(w) = Rw - \lambda w = 0 \text{ ,}$$

Or,

$$Rw = \lambda w$$

(8.1.12)

which is an eigenvalue equation.

Therefore, when trained by Hebbian learning, the weight vector of a linear neuron converges to the eigenvectors of the autocorrelation matrix, R.

But since a symmetric real matrix of size‘n Xn’ has n eigenvectors, it is not clear which of them w tends to.

We will show that w tends to the eigenvector corresponding to the highest eigenvalue.

Proof:
Let Q be a orthogonal, diagonalizingmatrix Q such that,

$$Q^T RQ = \Lambda$$

Where

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_i & 0 \\ 0 & 0 & \lambda_n \end{bmatrix}$$

Q = [q_1|…|q_i|…|q_n]

Where q_i are the eigenvectors of R.

Now consider the linear transformation,

w = Qx,

and express E(w) in terms of x, as follows,

$$E(w) = \frac{1}{2}(Qx)^T RQx \tag{8.1.13}$$

$$= \frac{1}{2}x^T Q^T RQx = \frac{1}{2}x^T \Lambda x = \frac{1}{2}\sum_{i=1}^{n}\lambda_i x_i^2 \tag{8.1.14}$$

Since Q is also a rotational transformation, maximum of the new function E(x) is the same as the maximum of the older function E(w). Let us consider the maximum of E'(x),

Let the eigenvalues of R be ordered such that,

$$\lambda_1 \geq \ldots \geq \lambda_i \geq \ldots \lambda_n \tag{8.1.15}$$

$$E(x) = \frac{1}{2}\sum_{i=1}^{n} \lambda_i x_i^2$$

We now impose the unit norm constraint on x as follows,

$$= \frac{1}{2}\sum_{i=1}^{n} (\lambda_i x_i^2)$$

$$= \frac{1}{2}(\lambda_1 (1 - \sum_{i \neq 1} x_i^2) + \sum_{i \neq 1}^{n} \lambda_i x_i^2) \tag{8.1.16}$$

The maximum of the above function can be found by solving the following differential equations,

$$\frac{dx_i}{dt} = (\lambda_i - \lambda_1) x_i \tag{8.1.17}$$

There are (n-1) such equations corresponding to (n-1) components, x_i i = 2,…n.

Since λ_1 is the largest eigenvalue, in all the above differential equations, x_i➔ 0, i = 2,…n.

Since ||x|| = 1, the only remaining component x_1= 1. Therefore the maximum of E(x) occurs when,

x = [1 0 0 …0].

Since w = Qx, we have, w = q_1.

Thus the weight vector of the linear neuron of eqn. (2) converges to the eigenvector corresponding to the highest eigenvalue of E, when trained by Hebbian learning.

8.2 Oja's Rule:

Under the action of Hebbian learning, weight vector of a linear neuron converged to the first eigenvector of R only when the weights are normalized as ||w||=1.

But such a condition is artificial and not part of the Hebbian mechanism which is biologically motivated. Therefore, Oja (1982) proposed a modification of Hebbian mechanism in which the weight vector is automatically normalized without explicitly an explicit step like,

w➔ w/||w||

The weight update according to Oja (1982) is as follows:

$$\Delta w_i = \eta y(x_i - yw_i) \tag{8.2.1}$$

In vector form, the update rule can be written as,

$$\Delta w = \eta y(x - yw) \tag{8.2.2}$$

Let us prove that the above rule does the following:

- Maximizes $\frac{1}{2}w^T Rw$
- ||w|| = 1

Consider the average update in w for the entire data set S, when the weight vector converges.

$$E[\Delta w] = \eta E[y(x - yw)] = 0$$

$$= \eta E[(yx - y^2 w)]$$

$$= \eta E[((w^T x)x - (w^T x)^2 w)]$$

$$= \eta[(E(xx^T)w - E(w^T(xx^T)w)w)]$$

$$= \eta[(Rw - (w^T Rw)w)] = 0$$

The last equation is the eigenvalue equation in R.

$Rw - \lambda w = 0$

Thus w is an eigenvector of R.

where $\lambda = w^T Rw$

or, $\lambda = w^T (\lambda w)$

⇨ ||w||=1.

Like Hebbian learning, an advantage of Oja's rule is that it is local: update for the i'th component,w_i, of the weight vector, w, is dependent on quantities that are locally available at the presynaptic or postsynaptic ends of the synapse that is represented by w_i,.

Example:

Long term potentiation in hippocampal neurons of brain

8.3 Principal Component Analysis and Hebbian Learning:

Before we proceed to prove an interesting result relating Hebbian learning and principal component analysis (PCA) we state a result from linear algebra.

Spectral Theorem: If R is a real symmetric matrix, and Q is an orthogonal, diagonalizingmatrix such that,

$$Q^T RQ = \Lambda \tag{8.3.1}$$

Where

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_i & 0 \\ 0 & 0 & \lambda_n \end{bmatrix} Q = [q1| \dots |qi| \dots |qn]$$

Then

$$R = \sum_{i=1}^{n} \lambda_i q_i q_i^T \tag{8.3.2}$$

Proof:

Since,

$$Q^T RQ = \Lambda$$

$$R \;=\; Q\Lambda Q^T$$

$$= \sum_{i=1}^{n} \lambda_i q_i q_i^T$$

To derive the last result, we used the following,

$$q_i q_j^T = \delta(i, j) \tag{8.3.3}$$

Where δ(i,j) is the Kronecker delta, defined as,

$$\delta(i, j) = 1, \quad if\,(i = j)$$

$$= 1, \quad otherwise$$

We have shown earlier that, Hebbian learning of a linear neuron

- ⇨ Maximizing the quadratic form of R (= $\frac{1}{2} w^T Rw$)
- ⇨ Maximizing the average squared output of the neuron.
- ⇨ We also note that if the data is ‘zero mean’ (E[x] = 0), Hebbian learning also maximizes output variance.Since the neuron is linear, for zero-mean input data, mean squared value of the output equals output variance. i.e.,
  $E[y^2] = \sigma_y^2$

We now show that Hebbian learning can used for data compression. Using this mechanism, a vector, x, of dimension, n, can transformed into another vector, y, of dimension, m, (m <n), such that x can be reconstructed from y, with minimum reconstruction error.

To enable such compression, we assume, for the moment, the following:

We have seen that Hebbian learning extracts the eigenvector corresponding to the largest eigenvalue of the autocorrelation matrix, R. But we assume that it is possible to extract all the eigenvectors of R, by some sort of an extension of Hebbian learning. We will describe such an extension in the next section. But for now we assume such an extension, and describe how data compression can be achieved by Hebbian learning.

Let x, be a data point drawn from a data set S. R is the autocorrelation matrix associated with S. Assume that the data is zero-mean (E[x]=0). Q is a matrix constructed out of the eigenvectors of R as follows,

$$Q = [q_1|\ldots|q_i|\ldots|q_n] \tag{8.3.4}$$

Consider the following linear transformation,

$$y = Q^T x \tag{8.3.5}$$

Note that the components of y, are the projections of x onto the first m eigenvectors of R.

$$y_i = q_i^T x \tag{8.3.6}$$

Let us calculate the variance of y_i, which we will use shortly.

Since the data set, S, is zero-mean,

E[x] = 0. Therefore, from linearity of eqn. (21) above, we have E[y_i]=0. Therefore,

$$\sigma_{y_i}^2 = E[y_i^2] = E[(q_i^T x)(q_i^T x)]$$
$$= q_i^T E[(x^T x)]q_i \tag{8.3.7}$$

$$= q_i^T R q_i = q_i^T \lambda_i q_i = \lambda_i \tag{8.3.8}$$

Thus the variance of the i’th component,^{y_i} , is the corresponding eigenvalue.

‘x’ can be reconstructed from y, by simply inverting the transformation of eqn. (20).

$$x = Qy \tag{8.3.9}$$

In the last equation, x can be expressed as a weighted sum eigenvectors as,

$$x = \sum_{i=1}^{n} q_i y_i \tag{8.3.10}$$

Eigenvectors are ordered such that the corresponding eigenvalues are in the descending order.

$$\lambda_1 \geq \ldots \geq \lambda_i \geq \ldots \lambda_n \tag{8.3.11}$$

Now consider a reconstruction of x, denoted by $\tilde{x}$, produced by taking only a partial summation of the expression in eqn. (26) above.

$$\hat{x} = \sum_{i=1}^{m} q_i y_i \tag{8.3.12}$$

Reconstruction error,

$$e = x - \tilde{x} = \sum_{i=1}^{n} q_i y_i - \sum_{i=1}^{m} q_i y_i = \sum_{i=m+1}^{n} q_i y_i \tag{8.3.13}$$

Root Mean Square (RMS) value of the reconstruction error is,

$$E[e^T e] = E[(\sum_{i=m+1}^{n} q_i y_i)^T (\sum_{i=m+1}^{n} q_i y_i)] \tag{8.3.14}$$

$$= E[(\sum_{i=m+1}^{n} q_i^T q_i y_i^2)] = E[(\sum_{i=m+1}^{n} y_i^2)] =$$

$$= \sum_{i=m+1}^{n} E[y_i^2] = \sum_{i=m+1}^{n} \lambda_i \tag{8.3.15}$$

Thus the reconstruction error is the sum of lower (corresponding to larger i) eigenvalues of R. As m increases, and approaches n, error reduces since there are fewer terms in the expansion of eqn. (31). But error decreases also because the eigenvalues are sorted and lower eigenvalues are smaller in magnitude than higher ones (larger i).

Typically, in high-dimensional real-world data, there are a small number of large-valued eigenvalues, and a large number of small-valued eigenvalues. If ‘m’ is chosen such that the larger eigenvalues are included, x can be expressed in a compressed form, y, and reconstructed again, $\hat{x}$, with minimal loss.

Example:

1. Take random 100 dim data. Show the distribution of eigenvalues.

Taken a random 100 dim data distributed between 0 and 1. The Eigen value is distributed as Fig. 8.3.1

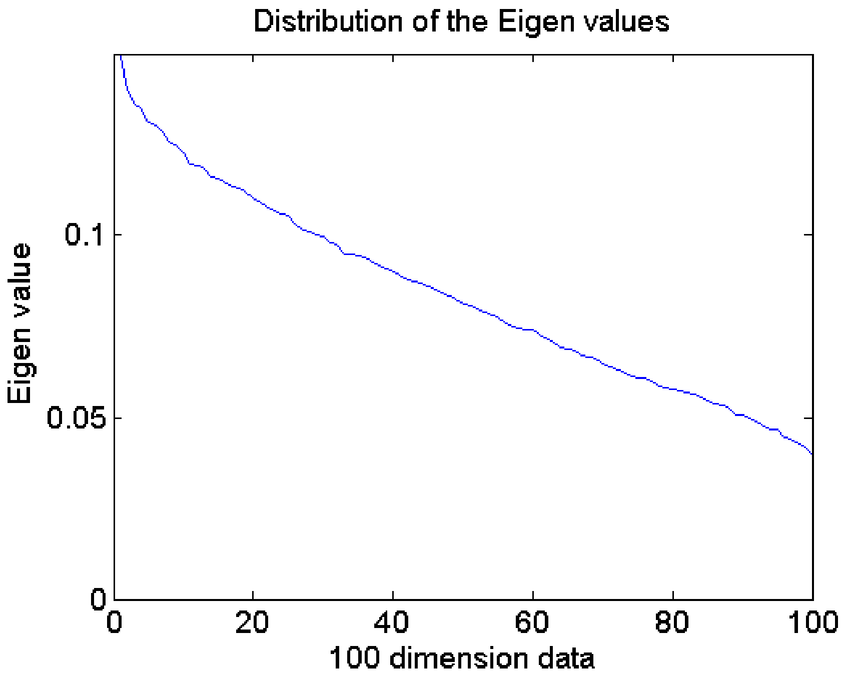

Figure 8.3.1: The Eigen value distribution.

Code:

```
%consider an artificial data set of 100 variables (e.g., genes) and 10
samples
data=rand(100,1000);

% remove the mean variable-wise (row-wise)
data=data-repmat(mean(data,2),1,size(data,2));
```

```
% calculate eigenvectors (loadings) W, and eigenvalues of the covariance
matrix
[W, EvalueMatrix] = eig(cov(data'));
Evalues = diag(EvalueMatrix);

% order by largest eigenvalue
Evalues = Evalues(end:-1:1);


%Plotting
figure(1);set(gca,'FontSize',14)
plot(Evalues);
ylim([0 .15]);
ylabel('Eigen value');
xlabel('100 dimension data');
title('Distribution of the Eigen values');
```

8.4 Sanger’s rule:

We have seen that Hebb’s rule gives the weight vector that is the first eigenvector of the autocorrelation matrix R. Oja’s rule also essentially provided the same result, with the distinction that it achieved normalization naturally. ***It would be desirable to extend these results to the case of m-principal components***. Two such extensions are available – one due to Sanger (1989) and the other due again to Oja (1989).

Since M-principal components need to be discovered in this case, we have a network with ‘m’ output neurons. The neurons are linear as before. The output of the i’th output neuron can therefore be expressed as,

$$y_i = \sum_{j=1}^{m} w_{ij} x_j \tag{8.4.1}$$

The above equation in matrix form becomes,

$$y = w_i^T x = x^T w_i \tag{8.4.2}$$

The weights are trained by the following rule (Sanger, 1989):

$$\Delta w_{ij} = \eta y_i (x_i - \sum_{k=1}^{i} w_{kj} y_k) \tag{8.4.3}$$

8.5 Linsker’s model:

Ralf Linsker proposed a model of the visual system that consists of a multilayered network trained by Hebbian learning. The layers in the network are two-dimensional, analogous to the sheets of neurons in various stages of the real visual system. Neurons are all layers are linear. Each neuron in a given layer receives inputs from a local neighborhood in the previous layers, a feature that is also inspired the connectivity patterns in the visual system. Since all the neurons are linear, the entire network has linear input-output relationship. Therefore, it may not be capable of representing rich input-output characteristics, and does not enjoy the universal approximation property of the MLPs. *But the training is done is stages, one layer at a time. Due to such layer-wise training*, weights in each weight stage evolve differently, producing interesting response patterns like orientation sensitivity, exhibited by neurons in the primary visual cortex.

Consider the response, y, of a neuron in one of the layers of Linsker's model.

$$y = a + \sum_{j=1}^{K} w_j V_j \tag{8.5.1}$$

V_j could be the input pattern, x_j, or the response of a neuron in the previous layer.

A variation of Hebb's rule is used to train the weights, w_i:

$$\Delta w_i = \eta(V_i y + bV_i + cy + d) \tag{8.5.2}$$

The first term on the right-hand side (the product $V_i y$) is the product term that appears in the original Hebb's rule. The remaining terms are linear and constant terms that appear in Linsker's variation.

The parameters b, c and d can be tuned appropriately so that neurons in various layers display various response properties. The weights are prevented from blowing up by clipping them as follows,

$$w_- < w_i < w_+ \tag{8.5.3}$$

To compute the final values to which the weights converge, let us consider the average change in weights, which must be zero at convergence.

$$0 = E[\Delta w_i] = \eta(E[V_i y] + bE[V_i] + cE[y] + d) \tag{8.5.4}$$

Let $V_i = \overline{V} + v_i$, where $\overline{V} = E[V_i]$and v_i is the deviation from the mean. Then,

$$E[\Delta w_i] = \eta(E[(\overline{V} + v_i)(a + \sum_j^K w_j(\overline{V} + v_j))] + b\overline{V} + cE[(a + \sum_j^K w_j(\overline{V} + v_j))] + d) \tag{8.5.5}$$

$$= \eta(aE[(\overline{V} + v_i) + E[\sum_j^K w_j(\overline{V} + v_j)(\overline{V} + v_i)] + b\overline{V} + ac + cE[\sum_j^K w_j(\overline{V} + v_j)] + d) \tag{8.5.6}$$

$$= \eta(a\overline{V} + \sum_j^K w_j\overline{V}^2 + E[\sum_j^K w_j v_j v_i] + b\overline{V} + ac + cE[\sum_j^K w_j(\overline{V} + v_j)] + d) \tag{8.5.7}$$

$$= \eta(a\overline{V} + \sum_j^K w_j\overline{V}^2 + \sum_j^K w_j C_{ij} + b\overline{V} + ac + c\sum_j^K w_j\overline{V} + d) \tag{8.5.8}$$

Where Cij is the the (i,j) element of the covariance matrix E[$v_i v_j$].

Terms in the last equation can be regrouped as,

$$= \eta(\sum_j^K w_j\overline{V}^2 + \sum_j^K w_j C_{ij} + a\overline{V} + b\overline{V} + ac + d + c\sum_j^K w_j\overline{V} + d) \tag{8.5.9}$$

$$= \eta((\sum_j^K w_j C_{ij}) + (a\overline{V} + b\overline{V} + ac + d) + (\overline{V}^2 + c\overline{V})(\sum_j^K w_j)) \tag{8.5.10}$$

$$= \eta((\sum_j^K w_j C_{ij}) + \lambda(\mu - \sum_j^K w_j)) \tag{8.5.11}$$

where λ and μ are functions of the constants a, b, c, d and $\overline{V}$.

The above weight dynamics can be interpreted as gradient descent over a cost function:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \tag{8.5.12}$$

$$E = -\frac{1}{2} w^T C w + \frac{\lambda}{2}(\mu - \sum_j w_j)^2 \tag{8.5.13}$$

The first term is similar to standard Hebbian learning that maximizes quadratic form associated with the autocorrelation matrix of the input data. The second term is a Lagrangian multiplier that imposes the condition

$$\sum_j w_j = \mu \tag{8.5.14}$$

on the weights.

Training results:

Linsker's model consist of 7 two-dimensional layers of neurons, A to G, where each layer projects to the subsequent layer in a 'pyramidal' fashion (Fig. 8.5.1). Training is done in a sequential fashion. Weights of a stage are first trained to saturation, before the next stage is trained. Note that all training in this model is unsupervised governed by eqns. (8.5.5-8.5.11). There is no attempt to map the input to some pre-specified target output.

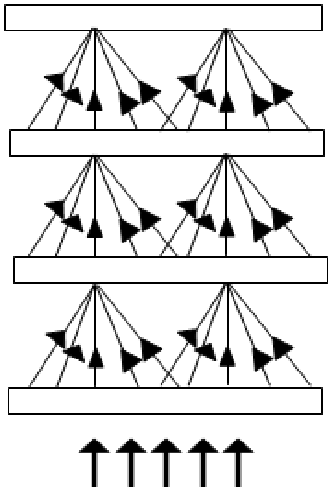

Figure 8.5.1: The Linsker's model (Layers A to G)

Independent random noise is given as input to the first layer, A. therefore the autocorrelation matrix is an identity matrix. It was found that for a range of parameters all the weight saturated to w_+. Since all the weights are the same, the response of Layer B is simply a local average, or smoothed version of the image presented to layer A. Therefore, neural activation in layer B turned out to have high local correlation. But beyond a small radius of high correlation, layer B neurons had low correlation.

Due to this reason, neurons in layer C developed center-surround kind of receptive fields. They responded strongly either to a bright dot with a dark background, or to a black dot with a white background. This trend continued to all the way to layer F where neurons had center-surround receptive fields. (Fig. 8.5.2)

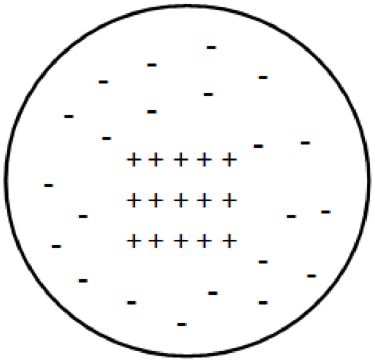

Figure 8.5.2: The Center-Surround receptive field in lower layers

But in Layer G the parameter values were changed. Therefore, trained produced a variety of weight patterns, many of them were asymmetric. Some neurons had receptive fields with alternating bands of positive and negative weights. Such cells had orientation sensitivity. Some other cells had a central positive region surrounding by several islands of negative regions. (Fig. 8.5.3)

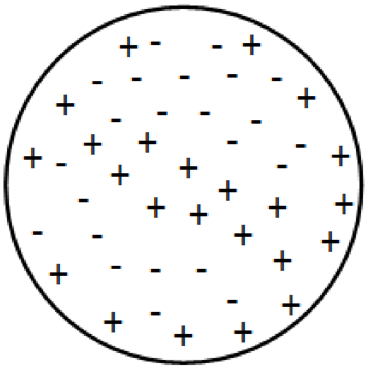

Figure 8.5.3: The orientation sensitivity in higher layers

Thus Linsker’s model shows that in a multilayer model, trained by simple Hebbian mechanism, neurons in the lower layers had center-surround kind of receptive fields, while those of higher layers had orientation sensitivity.(Fig. 8.5.1)