Linear Hebbian learning and PCA

Bruno A. Olshausen

May 11, 1998

Abstract

This handout describes linear Hebbian learning and its relation to principal components analysis (PCA). Hebbian learning constitutes a biologically plausible form of synaptic modification because it depends only upon the correlation between pre- and post-synaptic activity. Understanding the functions that can be performed by networks of Hebbian neurons is thus an important step in gaining an understanding of the effects of activity-dependent synaptic modification in the brain. This material is also described at greater length in chapter 8 of Herz, Krogh & Palmer.

In the early 1960's, Horace Barlow postulated his theory of *redundancy reduction*, which states that a useful goal of sensory coding is to transform the input in such manner that reduces the redundancy[1] due to complex statistical dependencies among elements of the input stream. The usefulness of redundancy reduction can be understood by considering the process of image formation, which occurs by light reflecting off of independent entities (i.e., objects) in the world and being focussed onto an array of photoreceptors in the retina. The activities of the photoreceptors themselves do not form a particularly useful signal to the organism because the structure present in the world is not made explicit, but rather is embedded in the form of complex statistical dependencies, or redundancies, among photoreceptor activities. A reasonable goal of the visual system, then, is to extract these statistical dependencies so that images may be explained in terms of a collection of independent events. Such a representation may then recover an explicit representation of the underlying independent entities that gave rise to the image, which would no doubt be useful to the survival of the organism.

In recent years, a substantial body of work has shown that the response properties of neurons at early stages of the visual system can be accounted for in terms of a strategy for reducing the redundancy in natural images. These successes provide encouragement that further aspects of cortical processing may be understood using

[1] A confusion that often arises from the term "redundancy reduction" is that it would seem to contradict the conventional wisdom that the brain contains redundant circuitry to deal with noise and physical damage. It is important however to distinguish between the form of redundancy that is present within the raw input stream (which reflects structure in the external world), and redundancy that is introduced by the nervous system through schemes such as population coding (e.g., as in the motor system). It is the former notion of redundancy that we refer to here.

this principle. Here, we shall consider the simplest form of redundancy that may occur in an input stream—i.e., linear pairwise correlations—and we shall show that a network of linear neurons that modifies its weights according to a Hebbian learning rule will reduce this form of redundancy by performing *principal components analysis*, or PCA.

PCA

Let us consider an input stream $\mathbf{x}$ that has linear pairwise correlations among its elements. That is,

$$c_{ij} = \langle x_i \, x_j \rangle \neq 0 \,. \tag{1}$$

The brackets $\langle \cdot \rangle$ mean "average over many input examples." Throughout this document we shall assume that all variables have zero mean ($\langle x_i \rangle = 0$), so $c_{ij} \neq 0$ implies that there are statistical dependencies among the inputs x_i. If x_i and x_j were statistically independent, then c_{ij} would equal zero since in this case $\langle x_i \, x_j \rangle = \langle x_i \rangle \langle x_j \rangle = 0$. (Note however that the converse is note true—i.e., $c_{ij} = 0$ does not imply that x_i and x_j are statistically independent. Statistical independence is a stronger condition for which we require $P(x_i, x_j) = P(x_i)\, P(x_j)$).

The goal of principal components analysis is to transform the input $\mathbf{x}$ to a new representation in which the variables are pairwise decorrelated. That is,

$$y_i = \mathbf{e}_i \cdot \mathbf{x} \tag{2}$$

where $\langle y_i \, y_j \rangle = 0 \; \forall_{i \neq j}$. Thus, the redundancy due to linear pairwise correlations is eliminated. By definition, in PCA the vectors $\mathbf{e}_i$ are orthonormal, meaning that $\mathbf{e}_i \cdot \mathbf{e}_j = 0 \; \forall_{i \neq j}$ and $|\mathbf{e}_i| = 1 \; \forall_i$. Also, the $\mathbf{e}_i$ are ordered according to the variance on the y_i such that $\langle y_1^2 \rangle > \langle y_2^2 \rangle > \ldots > \langle y_n^2 \rangle$. The process of PCA may be pictured in geometric terms, as shown in Figure 1.

PCA is a useful tool for analyzing structure in data, but it is important to realize its limitations. One limitation of PCA is that it takes into account only 2nd-order statistics among input variables (i.e., correlations among pairs of inputs). In many real-life situations, though, there will be higher-order statistical dependencies among the variables that are also important to consider (e.g., $\langle x_i x_j x_k \rangle$), and PCA is blind to these forms of structure. Another limitation of PCA is that the vectors $\mathbf{e}_i$ are forced to be orthogonal, and in many cases there is no *a priori* reason for thinking this is appropriate. An example of an input distribution with higher-order, non-orthogonal structure, and the way that PCA deals with this, is shown in Figure 1*b*.

Despite the limitations of PCA, it still can be of use to us in finding the effective dimensionality of an input space (i.e., the principal axes that account for most of the variance), as well as finding independent components when the data have Gaussian structure. It is also a useful building block for the development of more advanced techniques, so it is a good thing to know about in general.

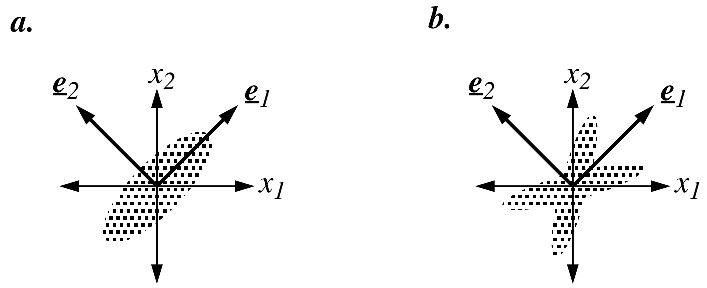

Figure 1: A geometric interpretation of PCA. The vectors $\mathbf{e}_i$ constitute the "principal axes" of the input distribution. The first principal component, $\mathbf{e}_1$, captures the most amount of variance in the input distribution, and $\mathbf{e}_2$ captures what is left after subtracting out the component along $\mathbf{e}_1$. The distribution with respect to $\mathbf{e}_1$ and $\mathbf{e}_2$ is decorrelated (because there is no diagonal structure in this new reference frame). Note that an underlying assumption of PCA is that the data have Gaussian structure—i.e., that they fall in a distribution shaped like a football, as in *a*. If the data are not Gaussian distributed, as in *b*, then PCA is not an appropriate strategy for revealing the structure of the input ensemble.

Linear Hebbian learning

Hebb's rule[2] states that the synaptic weight between two neurons should be increased proportional to the correlation between the pre-synaptic and post-synaptic activities. Thus, for a linear neuron,

$$y = \sum_i w_i x_i \,, \tag{3}$$

each weight w_i should be increased proportional to the correlation between y and x_i, or

$$\dot{w}_i = \langle y\, x_i \rangle \,. \tag{4}$$

Now, since y depends on all the inputs, one can see intuitively from equation 4 that the evolution of w_i will depend on the correlation between x_i and all the other inputs. Let's write this out explicitly, substituting the expression for y in equation 3 into equation 4:

$$\begin{aligned} \dot{w}_i &= \langle \sum_j w_j x_j\, x_i \rangle \\ &= \sum_j w_j \langle x_j\, x_i \rangle \,. \end{aligned} \tag{5}$$

The last step of moving w_j outside of the ensemble average may be done since the w_i are changing over a much slower time-scale than the x_i. In vector notion, then, we

[2]Named for Donald Hebb, who first described the idea in his influential book, *The Organization of Behavior*, in 1949

have

$$\dot{\mathbf{w}} = \mathbf{C}\mathbf{w} \tag{6}$$

where $\mathbf{C}$ is the matrix with elements $c_{ij} = \langle x_i \, x_j \rangle$. Equation 6 states that the growth of $\mathbf{w}$ depends solely on the input covariance matrix, $\mathbf{C}$. In other words, the evolution of $\mathbf{w}$ is governed by the linear pairwise statistics of the input ensemble. But how exactly?

To get a better handle on how $\mathbf{w}$ evolves, let us examine a simple one-dimensional system of the form

$$\dot{w} = c\, w\,. \tag{7}$$

This is just a linear, first-order differential equation. The solution is

$$w(t) = w(0)\, e^{c\,t} \tag{8}$$

where $w(0)$ is the initial weight state at time zero. Thus, if c is positive then w will grow exponentially. If c is negative, then w will decay exponentially. How fast w grows or decays is set by the constant c.

Now let us examine a slightly more complex system consisting of two weights, w_1 and w_2:

$$\begin{aligned} \dot{w}_1 &= c_{11}w_1 + c_{12}w_2 \\ \dot{w}_2 &= c_{21}w_1 + c_{22}w_2\,. \end{aligned} \tag{9}$$

This is just a two-dimensional version of equation 6, written out explicitly in terms of the vector and matrix components. It is difficult to see a simple solution here because the evolution of w_1 depends on w_2, and likewise the evolution of w_2 depends on w_1. In other words, the two variables are coupled together. If we could transform the w_i to a new coordinate system in which the variables were de-coupled, then the behavior would be simple to analyze, as in equation 7. So, let's do that.

An aside: Eigenvectors and eigenvalues

It turns out that if $\mathbf{C}$ is symmetric (which it is in this case since $c_{ij} = \langle x_i x_j \rangle = \langle x_j x_i \rangle = c_{ji}$), then we can re-write it in the form

$$\mathbf{C} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^T\,. \tag{10}$$

Here, $\mathbf{E}$ is an orthonormal matrix with columns $\mathbf{e}_i$,

$$\mathbf{E} = \begin{bmatrix} | & | & & | \\ \mathbf{e}_1 & \mathbf{e}_2 & \cdots & \mathbf{e}_n \\ | & | & & | \end{bmatrix}, \tag{11}$$

where $\mathbf{e}_i \perp \mathbf{e}_i \; \forall_{i \neq j}$ and $|\mathbf{e}_i| = 1 \; \forall_i$; $\mathbf{E}^T$ is the *transpose* of $\mathbf{E}$, which is just $\mathbf{E}$ tipped on its side ($(\mathbf{E}^T)_{ij} = (\mathbf{E})_{ji}$); and $\mathbf{\Lambda}$ is a diagonal matrix with non-zero terms only along the diagonal

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \lambda_n \end{bmatrix}. \tag{12}$$

In geometric terms, $\mathbf{E}$ is a *rotation* matrix, $\mathbf{\Lambda}$ is a *scaling* matrix, and $\mathbf{E}^T$ is simply another rotation matrix that rotates in the opposite direction of $\mathbf{E}$. Thus, the act of multiplying a point or vector by the matrix $\mathbf{C}$ can thus be thought of as first rotating to another coordinate frame (multiplying by $\mathbf{E}^T$), then scaling each axis according to λ_i within this new coordinate frame (multiplying by $\mathbf{\Lambda}$), and then rotating back to the original coordinate frame (multiplying by $\mathbf{E}$). This process is pictured in Figure 2.

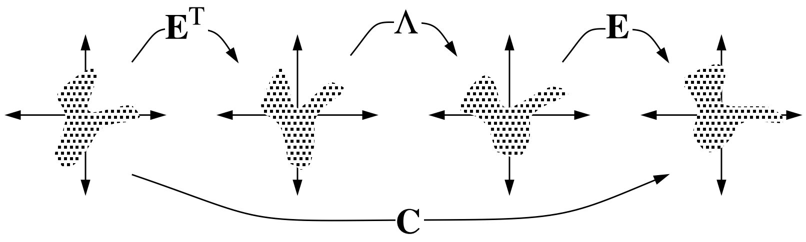

Figure 2: The act of multiplying a point or vector by $\mathbf{C}$ may be broken into three separate geometric operations—rotation, scaling, and counter-rotation—which is illustrated here for a collection of points distributed according to a hypothetical distribution. Multiplying by $\mathbf{E}^T$ rotates counter-clockwise in this case by about 30°. Multiplying by $\mathbf{\Lambda}$ scales the distribution by dilating along the horizontal axis ($\lambda_1 > 1$) and contracting along the vertical axis ($\lambda_2 < 1$). Multiplying by $\mathbf{E}$ simply applies the opposite rotation of $\mathbf{E}^T$.

The vectors $\mathbf{e}_i$, which form the columns of $\mathbf{E}$, are called the *eigenvectors* of the matrix $\mathbf{C}$, and the λ_i are termed the *eigenvalues*. The eigenvectors have the special property that if you multiply the matrix $\mathbf{C}$ by an eigenvector, then you get the same vector back but simply scaled by its eigenvalue:

$$\mathbf{C}\mathbf{e}_i = \lambda_i \mathbf{e}_i \, . \tag{13}$$

Thus, the term "eigen," which in German means "self" or "characteristic." The eigenvectors of $\mathbf{C}$ also constitute the principal components, because they will decorrelate the input distribution (you will show this).

Now we are in a position to understand the dynamics of our simple two-dimensional system in equation 9. Let us transform $\mathbf{w}$ to a new coordinate system by multiplying by $\mathbf{E}^T$. That is, we shall work with a new vector,

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \mathbf{E}^T \mathbf{w} \, , \tag{14}$$

where v_1 is the projection of $\mathbf{w}$ along the first eigenvector, $\mathbf{e}_1$, and v_2 is the projection of $\mathbf{w}$ along the second eigenvector, $\mathbf{e}_2$. Thus, if we pre-multiply both sides of equation 6 by $\mathbf{E}^T$, we get

$$\dot{\mathbf{v}} = \mathbf{\Lambda} \mathbf{v} \tag{15}$$

Now, since $\mathbf{\Lambda}$ is diagonal, it is easy to see how $\mathbf{v}$ evolves:

$$\begin{aligned} \dot{v}_1 &= \lambda_1 v_1 \\ \dot{v}_2 &= \lambda_2 v_2 \,. \end{aligned} \tag{16}$$

Thus, we have as the solution for v_1 and v_2:

$$\begin{aligned} v_1(t) &= v_1(0)\, e^{\lambda_1 t} \\ v_2(t) &= v_2(0)\, e^{\lambda_2 t} \,. \end{aligned} \tag{17}$$

Because λ_1 and λ_2 represent exponential growth rates, then even a slight imbalance between the two will result in one rapidly outpacing the other. Thus, if $\lambda_1 > \lambda_2$, then $\mathbf{v}$ will grow in the direction $[1, 0]$, which in terms of our original reference frame is just in the direction of $\mathbf{e}_1$ (Fig. 3). Otherwise, if $\lambda_2 > \lambda_1$, then $\mathbf{v}$ will grow in the direction $[0, 1]$, which in terms of our original reference frame is in the direction of $\mathbf{e}_2$.

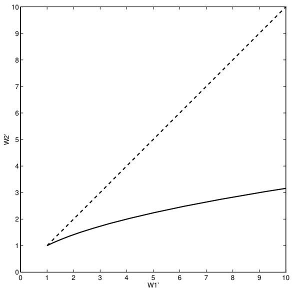

Figure 3: The evolution of $\mathbf{v}$. In this case, $\lambda_1 = 2$ and $\lambda_2 = 1$, so the growth of v_1 rapidly outpaces v_2. Since the v_1 axis is simply the same as the vector $\mathbf{e}_1$ in the original coordinate system, then $\mathbf{w}$ will grow in the direction of $\mathbf{e}_1$. The dotted line shows the expected evolution of $\mathbf{v}$ if v_1 and v_2 were to grow at equal rates.

Constraining the growth of w: Oja's rule

So far we have shown that a linear neuron that updates its weights according to a simple Hebbian rule, $\dot{w}_i \propto \langle y\, x_i \rangle$, will grow its weight vector along the direction of the eigenvector of the input covariance matrix, $\mathbf{C}$, with maximum eigenvalue (or the first principal component). As it stands, though, $\mathbf{w}$ will grow without bound (i.e., $|\mathbf{w}| \to \infty$), which is not feasible for a physically realizable system. We can constrain the growth of $\mathbf{w}$ using a modified form of Hebb's rule, termed Oja's rule[3]:

$$\dot{\mathbf{w}} = \langle y\, (\mathbf{x} - y\mathbf{w}) \rangle \,. \tag{18}$$

Let's get a better feel for what this rule does by re-writing it in two terms:

$$\dot{\mathbf{w}} = \langle y\, \mathbf{x} \rangle - \langle y^2 \rangle \mathbf{w} \,. \tag{19}$$

[3]Named for Erkki Oja (of the Helsinki University of Technology in Espoo, Finland), who invented the rule.

The first term on the right side is just the same as Hebb's rule—each w_i increases proportional to the correlation between y and x_i. The second term may be interpreted essentially as providing a subtractive weight decay to prevent $\mathbf{w}$ from growing without bound. The equilibrium solution for $\mathbf{w}$ is reached when $\dot{\mathbf{w}} = 0$, or when the first term equals the second. Since the first term $\langle y\,\mathbf{x}\rangle$ is just equal to $\mathbf{Cw}$ (as we showed in the beginning, in eq. 6), then at equilibrium we have

$$\mathbf{Cw} = \langle y^2\rangle \mathbf{w}\,. \tag{20}$$

Thus, by definition (eq. 13), $\mathbf{w}$ will be an eigenvector of $\mathbf{C}$ at equilibrium. Furthermore, we can show that $|\mathbf{w}| = 1$, since

$$\begin{aligned}\langle y^2\rangle &= \mathbf{w}^T\mathbf{Cw} \\ &= \mathbf{w}^T\langle y^2\rangle\mathbf{w} \\ &= \langle y^2\rangle|\mathbf{w}|^2.\end{aligned} \tag{21}$$

Showing that $\mathbf{w}$ is the eigenvector with maximum eigenvalue is a little more involved (see Hertz, Krogh, & Palmer), but can be seen intuitively from equation 18, since $\dot{\mathbf{w}}$ is just the negative of the gradient of $\langle|\mathbf{x} - y\mathbf{w}|^2\rangle$ with respect to $\mathbf{w}$. In other words, $\mathbf{w}$ is attempting to move in a direction that captures the most amount of variance in the input distribution, which is the property of the first principal component.

The network implementation of Oja's rule takes the output, y, and sends it back through the weights $\mathbf{w}$ to form a prediction of the input state $\hat{\mathbf{x}}$. The prediction $\hat{\mathbf{x}}$ is subtracted from the input $\mathbf{x}$, and then $\mathbf{w}$ is updated according to a Hebb rule between y and the residual input signal, $\mathbf{x} - \hat{\mathbf{x}}$.

Learning multiple eigenvectors: Sanger's rule

It would be nice to be able to learn more than just the first principal component. If we had a system of m neurons, we would like to properly coordinate them so as to learn the first m principal components. For this, we can use Sanger's rule[4]:

$$\dot{\mathbf{w}}_i = \langle y_i\,(\mathbf{x} - \sum_{j\le i} y_j\mathbf{w}_j)\rangle\,. \tag{22}$$

Sanger's rule may be seen as an extension of Oja's rule. First of all, we can see that for $i = 1$, Sanger's rule is precisely equivalent to Oja's one-unit rule, since the summation in equation 22 is over $j \le 1$. Thus, $\mathbf{w}_1$ will converge to the first principal component as before, and its evolution will not be affected by the evolution of $\mathbf{w}_2$ through $\mathbf{w}_m$. Now for $\mathbf{w}_2$, let's pretend that $\mathbf{w}_1$ has already converged to the first eigenvector $\mathbf{e}_1$. So the learning rule for $\mathbf{w}_2$ is

$$\dot{\mathbf{w}}_2 = \langle y_2\,(\mathbf{x}_1 - y_2\mathbf{w}_2)\rangle\,, \tag{23}$$

where $\mathbf{x}_1$ is what remains of $\mathbf{x}$ after subtracting out its component along the direction of $\mathbf{e}_1$—i.e., $\mathbf{x}_1 = \mathbf{x} - y_1\mathbf{e}_1$. Thus, the learning rule for $\mathbf{w}_2$ is simply Oja's rule applied

[4]Named for Terry Sanger, who invented it while as an E.E. Master's student at MIT.

to a sub-space that is orthogonal to $\mathbf{e}_1$. $\mathbf{w}_2$ will therefore converge to the eigenvector of this subspace with maximum eigenvalue, which is $\mathbf{e}_2$ within the original space. This procedure is repeated then for $\mathbf{w}_3$ (subtracting out the component along $\mathbf{e}_1$ and $\mathbf{e}_2$), and so on until m components are learned. A full-fledged, formal proof of the convergence of Sanger's rule may be found in Hertz, Krogh, & Palmer.

The network implementation of Sanger's rule is not unlike the network implementation of Oja's rule, except now care must be taken to subtract out the predicted input for each y_i in a progressive manner. The output of each neuron, y_i, is fed back through its weight vector, $\mathbf{w}_i$, and progressively accumulated to form a prediction $\hat{\mathbf{x}}_i = \sum_{j \leq i} y \mathbf{w}_i$. This is subtracted from the input, $\mathbf{x}$, and then $\mathbf{w}_i$ is updated according to a Hebb rule between y_i and the residual, $\mathbf{x} - \hat{\mathbf{x}}_i$. Thus, each $\mathbf{w}_i$ learns from a different input ensemble, with progressively more structure subtracted out as i increases. This ordered progression is a bit cumbersome in neurobiological terms. However, it turns out that there is an extension of Oja's one-unit rule to multiple units that does not require this strict ordering. Sanger's rule is just a bit simpler to understand in terms of why and how it works.

Non-linear Hebbian learning

We were able to make headway in analyzing Hebbian learning with a linear neuron because we could perform the manipulation in equation 5 and thus derive a closed form solution for $\mathbf{w}$. If the neuron has an output non-linearity, though, this won't be so easy. Let us say that the output y is an arbitrary non-linear function of the weighted inputs:

$$y = f(\sum_i w_i x_i)\,. \tag{24}$$

Performing Hebbian learning with such a neuron then gives us

$$\begin{aligned} \dot{w}_i &= \langle y x_i \rangle \\ &= \langle f(\sum_j w_j x_j) x_i \rangle\,. \end{aligned} \tag{25}$$

Now, it is not so easy to pull the w_j outside of the summation and analyze the evolution of $\mathbf{w}$. We could, however, perform a Taylor expansion on f, in which case we would see that the evolution of $\mathbf{w}$ now depends on many higher-order statistics of the input ensemble. Which statistics are learned from, however, depends on the precise form of f. This then begs the question, what form of non-linearity is appropriate for learning the structure of a given input ensemble? This question is addressed by networks that perform competitive learning (or clustering), sparse coding, and independent components analysis (ICA), which we turn to next.