Principal Components Analysis and Unsupervised Hebbian Learning

Robert Jacobs
Department of Brain & Cognitive Sciences
University of Rochester
Rochester, NY 14627, USA

August 8, 2008

Reference: Much of the material in this note is from Haykin, S. (1994). *Neural Networks.* New York: Macmillan.

Here we consider training a single layer neural network (no hidden units) with an unsupervised Hebbian learning rule. It seems sensible that we might want the activation of an output unit to vary as much as possible when given different inputs. After all, if this activation was a constant for all inputs then the activation would not be telling us anything about which input is present. In contrast, if the activation has very different values for different inputs then there is some hope of knowing something about the input based just on the output unit's activation.

So we might want to train the output unit to adapt its weights so as to maximize the variance of its activation. However, one way in which the unit might do this is by making its weights arbitrarily large. This would not be a satisfying solution. We would prefer a solution in which the variance of the output activation is maximized subject to the constraint that the weights are bounded. For example, we might want to constrain the length of the weight vector to be one.

Lagrange optimization is a method for maximizing (or minimizing) a function subject to one or more constraints. At this point, we take a brief detour and give an example of Lagrange optimization. From this example, you should get the general idea of how this method works.

Example: Suppose that we want to find the shortest vector $\mathbf{y} = [y_1 \ y_2]^T$ that lies on the line $2y_1 - y_2 - 5 = 0$. We write down an objective function L in which the first term states the function that we want to minimize, and the second term states the constraint:

$$L = \frac{1}{2}(y_1^2 + y_2^2) + \lambda(2y_1 - y_2 - 5) \tag{1}$$

where λ is called a Lagrange multiplier (it is always placed in front of the constraint). The first term gives the length of the vector (actually its 1/2 times the length of the vector squared); the second term gives the constraint. We now take derivatives, and set these derivatives equal to zero:

$$\frac{\partial L}{\partial y_1} = y_1 + 2\lambda = 0 \tag{2}$$

$$\frac{\partial L}{\partial y_2} = y_2 - \lambda = 0 \tag{3}$$

$$\frac{\partial L}{\partial \lambda} = 2y_1 - y_2 - 5 = 0. \tag{4}$$

Note that we have three equations and three unknown variables. From the first derivative we know that $y_1 = -2\lambda$, and from the second derivative we know that $y_2 = \lambda$. If we plug these values into

the third derivative, we get $-4\lambda - \lambda = 5$ or that $\lambda = -1$. From this, we can now solve for y_1 and y_2; in particular, $y_1 = 2$ and $y_2 = -1$. So the solution to our problem is $\mathbf{y} = [2 \ -1]^T$.

Now let's turn to the problem that we really care about. We want to maximize the variance of the output unit's activation subject to the constraint that the length of the weight vector is equal to one. Without loss of generality, let's make things simpler by assuming that the input variables each have a mean of zero ($E[x_i] = 0$). Because the mapping from inputs to the output is linear ($y = \mathbf{w}^T\mathbf{x} = \mathbf{x}^T\mathbf{w}$), this means that the output activation will also have a mean of zero ($E[y] = 0$). We are now ready to write down the objective function:

$$L = \frac{1}{2}\sum_p y_p^2 + \lambda(1 - \mathbf{w}^T\mathbf{w}) \tag{5}$$

where y_p is the output activation on data pattern p. The first term, $\frac{1}{2}\sum_p y_p^2$, is proportional to the sample variance of the output unit, and the second term, $\lambda(1 - \mathbf{w}^T\mathbf{w})$ gives the constraint that the weight vector should have a length of one. We now do some algebraic manipulations:

$$\begin{aligned} L &= \frac{1}{2}\sum_p (\mathbf{w}^T\mathbf{x}_p)(\mathbf{x}_p^T\mathbf{w}) + \lambda(1 - \mathbf{w}^T\mathbf{w}) && (6) \\ &= \frac{1}{2}\sum_p \mathbf{w}^T(\mathbf{x}_p\mathbf{x}_p^T)\mathbf{w} + \lambda(1 - \mathbf{w}^T\mathbf{w}) && (7) \\ &= \frac{1}{2}\mathbf{w}^T Q\mathbf{w} + \lambda(1 - \mathbf{w}^T\mathbf{w}) && (8) \end{aligned}$$

where $Q = \sum_p \mathbf{x}_p\mathbf{x}_p^T$ is the sample covariance matrix. Now let's take the derivative of L with respect to $\mathbf{w}$, and set this derivative equal to zero:

$$\frac{\partial L}{\partial \mathbf{w}} = Q\mathbf{w} - 2\lambda\mathbf{w} = 0 \tag{9}$$

which means that $Q\mathbf{w} = 2\lambda\mathbf{w}$. In other words, $\mathbf{w}$ is an eigenvector of the covariance matrix Q (the corresponding eigenvalue is 2λ). So if we want to maximize the variance of the output activation (subject to the constraint that the weight vector is of length one), then we should set the weight vector to be an eigenvector of the input covariance matrix (in fact it should be the eigenvector with the largest eigenvalue). As discussed below, this has an interesting relationship to principal component analysis and unsupervised Hebbian learning.

The idea behind principal component analysis (PCA) is that we want to represent the data with respect to a new basis, where the vectors that form this new basis are the eigenvectors of the data covariance matrix. Recall that eigenvectors with large eigenvalues give directions of large variance (again, we are considering the eigenvectors of a covariance matrix), whereas eigenvectors with small eigenvalues give directions of small variance (so the eigenvector with the largest eigenvalue gives the direction in which the input data shows the greatest variance, and the eigenvector with the smallest eigenvalue gives the direction in which the input data shows the least variance). Each eigenvalue gives the value of the variance in the direction of its corresponding eigenvector (thus the

eigenvalues are all positive). Because the data covariance matrix is symmetric, its eigenvectors are orthogonal (as usual, we assume that the eigenvectors are of length one).

Let $\mathbf{x}^{(t)} = [x_1^{(t)}, \ldots, x_n^{(t)}]^T$ be a data item, and $\mathbf{e}_i = [e_1, \ldots, e_n]^T$ be the i^{th} eigenvector (assume that the eigenvectors are ordered such that the eigenvector with the largest eigenvalue is $\mathbf{e}_1$, the eigenvector with the next largest eigenvalue is $\mathbf{e}_2$, and so on). The projection of $\mathbf{x}^{(t)}$ onto $\mathbf{e}_i$ is denoted $y_i^{(t)}$, and is given by the inner product of these two vectors:

$$y_i^{(t)} = [\mathbf{x}^{(t)}]^T \mathbf{e}_i = \mathbf{e}_i^T \mathbf{x}^{(t)}. \tag{10}$$

That is, $y_i^{(t)}$ is the coordinate of $\mathbf{x}^{(t)}$ along the axis given by $\mathbf{e}_i$, and $\mathbf{y}^{(t)} = [y_1^{(t)}, \ldots, y_n^{(t)}]^T$ is the representation of $\mathbf{x}^{(t)}$ with respect to the new basis. Let $E = [\mathbf{e}_1, \ldots, \mathbf{e}_n]$ be the matrix whose i^{th} column is the i^{th} eigenvector. Then we can move back and forth between the two representations of the same data by the linear equations:

$$\mathbf{y}^{(t)} = E^T \mathbf{x}^{(t)}, \tag{11}$$

and

$$\mathbf{x}^{(t)} = E \mathbf{y}^{(t)}. \tag{12}$$

Please make sure that you understand these equations [we have used the fact that for an orthogonal matrix E (a matrix whose columns are orthogonal vectors), its inverse, E^{-1}, is equal to its transpose E^T].

Principal component analysis is useful for at least two reasons. First, as a feature detection mechanism. It is often the case that the eigenvectors give interesting underlying features of the data. PCA is also useful as a dimensionality reduction technique. Suppose that we represent the data using only the first m eigenvectors, where $m < n$ (that is, suppose we represent $\mathbf{x}^{(t)}$ with $\hat{\mathbf{y}}^{(t)} = [y_1^{(t)}, \ldots, y_m^{(t)}]^T$ instead of $\mathbf{y}^{(t)} = [y_1^{(t)}, \ldots, y_n^{(t)}]^T$). Out of all linear transformations that produce an m-dimensional representation, this transformation is optimal in the sense that it preserves the most information about the data (yes, I am being vague here). That is, PCA can be used for optimal linear dimensionality reduction.

In the remainder of this handout, we prove that the use of a particular Hebbian learning rule results in a network that performs PCA. That is, the weight vector of the i^{th} output unit goes to the i^{th} eigenvector $\mathbf{e}_i$ of the data covariance matrix, and the output of this unit $y_i^{(t)}$ is the coordinate of the input $\mathbf{x}^{(t)}$ along the axis given by $\mathbf{e}_i$. For simplicity, we assume that we are dealing with a two-layer linear network with n input units and only one output unit (the result extends to the case with n output units in which the weight vector of each output unit is a different eigenvector, but we will not prove that here). We will also assume that the input is a random variable with mean equal to zero.

The learning rule is:

$$w_i(t+1) = w_i(t) + \epsilon[y(t)x_i(t) - y^2(t)w_i(t)] \tag{13}$$

where $w_i(t)$ is the value of the output unit's i^{th} weight at time t. Note that the first term inside the brackets is the standard Hebbian learning term, and the second term is a type of "weight decay" term that prevents the weight from getting too big. In vector notation, this equation may be re-written:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \epsilon[y(t)\mathbf{x}(t) - y^2(t)\mathbf{w}(t)]. \tag{14}$$

Recall that we are using a linear network so that

$$y(t) = \mathbf{w}^T(t)\mathbf{x}(t) = \mathbf{x}^T(t)\mathbf{w}(t). \tag{15}$$

Using this fact, we re-write the weight update rule as

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \epsilon[\mathbf{x}(t)\mathbf{x}^T(t)\mathbf{w}(t) - \mathbf{w}^T(t)\mathbf{x}(t)\mathbf{x}^T(t)\mathbf{w}(t)\mathbf{w}(t)]. \tag{16}$$

Let $C = E[\mathbf{x}(t)\mathbf{x}^T(t)]$ be the data covariance matrix (we use the fact that the input has a mean equal to zero). The update rule converges when $E[\mathbf{w}(t+1)] = E[\mathbf{w}(t)]$. Denote the expected value of $\mathbf{w}(t)$ at convergence as $\mathbf{w}_o$. Convergence occurs when the expected value of what is inside the brackets in the update rule is equal to zero:

$$\begin{aligned} 0 &= E[\mathbf{x}(t)\mathbf{x}^T(t)]\mathbf{w}_o - \mathbf{w}_o^T E[\mathbf{x}(t)\mathbf{x}^T(t)]\mathbf{w}_o\mathbf{w}_o & (17)\\ &= C\mathbf{w}_o - \mathbf{w}_o^T C\mathbf{w}_o\mathbf{w}_o. & (18) \end{aligned}$$

In other words,

$$C\mathbf{w}_o = \lambda_o \mathbf{w}_o \tag{19}$$

where

$$\lambda_o = \mathbf{w}_o^T C\mathbf{w}_o. \tag{20}$$

That is, $\mathbf{w}_o$ is an eigenvector of the data covariance matrix C with λ_o as its eigenvalue. We know that $\mathbf{w}_o$ has length one because

$$\begin{aligned} \lambda_o &= \mathbf{w}_o^T C\mathbf{w}_o & (21)\\ &= \mathbf{w}_o^T \lambda_o \mathbf{w}_o & (22)\\ &= \lambda_o \parallel \mathbf{w}_o \parallel^2 & (23) \end{aligned}$$

which is only possible if $\parallel \mathbf{w}_o \parallel^2 = 1$. We also know that λ_o is the variance in the direction of $\mathbf{w}_o$ because

$$\begin{aligned} \lambda_o &= \mathbf{w}_o^T C\mathbf{w}_o & (24)\\ &= \mathbf{w}_o^T E[\mathbf{x}(t)\mathbf{x}^T(t)]\mathbf{w}_o & (25)\\ &= E[\{\mathbf{w}^T(t)\mathbf{x}(t)\}\{\mathbf{x}^T(t)\mathbf{w}(t)\}] & (26)\\ &= E[y^2(t)] & (27) \end{aligned}$$

which is the variance of the output (assuming that the output has mean zero which is true given our assumption that the input has mean zero and that the input and output are linearly related).

We still need to prove that $\mathbf{w}_o$ is the particular eigenvector with the largest eigenvalue, i.e. $\mathbf{w}_o = \mathbf{e}_1$. This is trickier to prove so we shall omit it.

This result is due to Oja (1982). The extension to networks with n output units in which the weight vector of each output unit goes to a different eigenvector is due to Sanger (1989). The extension to networks (using Hebbian and anti-Hebbian learning) with n output units in which the n weight vectors span the same space as the first n eigenvectors is due to Földiák (1990).

Földiák, P. (1990) Adaptive network for optimal linear feature extraction. *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*.

Oja, E. (1982) A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15, 267-273.

Sanger, T.D. (1989) Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 12, 459-473.