EM Algorithms for PCA and Sensible PCA

Sam Roweis
California Institute of Technology

CNS-TR-97-02 (April 1997)
DRAFT — please do not cite

Abstract

I present an expectation-maximization (EM) algorithm for principal component analysis (PCA). The algorithm allows a few eigenvectors and eigenvalues to be extracted from large collections of high dimensional data. It is computationally efficient in space and time and does not require computing the sample covariance of the data. It also naturally accommodates missing information. I introduce a new variation of PCA known as *sensible* principal component analysis (SPCA) which defines a proper density model in the data space. Learning for SPCA is also done with an EM algorithm. I include results of simulations showing that, in only a few iterations, these EM algorithms correctly and efficiently find the leading eigenvectors of the covariance of large datasets (having up to half a million datapoints in tens of thousands of dimensions).

1 Why EM for PCA?

Principal component analysis (PCA) is a widely used dimensionality reduction technique in data analysis. Its popularity comes from three important properties. First, it is the *optimal* (in terms of mean squared error) *linear* scheme for compressing a set of high dimensional vectors into a set of lower dimensional vectors and then reconstructing. Second, the model parameters can be computed *directly* from the data – for example by diagonalizing the sample covariance of the data. Third, compression and decompression are easy operations to perform given the model parameters – they require only matrix multiplications.

Despite these attractive features, however, PCA models have several shortcomings. One is that naive methods for finding the principal component directions have trouble with high dimensional data. Consider attempting to diagonalize the sample covariance matrix of n vectors in a space of p dimensions when n and p are several hundred or several thousand. Difficulties can arise both in the form of computational complexity and also data scarcity. On the data scarcity front, we often do not have enough data in high dimensions for the sample covariance to be of full rank and so we must be careful to employ techniques such as the singular value decomposition which do not require full rank matrices. On the complexity front, direct diagonalization of a symmetric matrix thousands of rows in size can be extremely costly since this operation is $O(p^3)$ for $p \times p$ inputs. Fortunately, several techniques exist for reducing this cost when only the first few leading eigenvectors and eigenvalues are required (for example the power method [9] which is only $O(p^2)$). However, even computing the sample covariance itself is very costly, requiring $O(np^2)$ operations.

In general it is best to avoid altogether computing the sample covariance explicitly. Methods such as the *snap-shot* algorithm [8] do this by assuming that the eigenvectors being searched for are linear combinations of the datapoints; their complexity is $O(n^3)$. In this note, I present a version of the expectation-maximization (EM) algorithm [1] for learning the principal components of a dataset. The algorithm does not require computing the sample covariance and has a complexity limited by $O(knp)$ operations where k is the number of leading eigenvectors to be learned. For large n and p I am not aware of any other methods which possess this attractive scaling.

Another shortcoming of standard approaches to PCA is that it is not obvious how to deal properly with missing data. Most of the methods discussed above cannot accommodate missing values and so incomplete points must either be discarded or completed using a variety of ad-hoc interpolation methods. (For example, a standard solution is to replace missing coordinates with the mean of the known values in that coordinate or with a linear regression of the unknown value from the known values.) On the other hand, the EM algorithm for PCA enjoys all the benefits [4] of other EM algorithms in terms of estimating the maximum likelihood values for missing information directly at each iteration.

Finally, the PCA model itself suffers from a critical flaw which is independent of the technique used to compute its parameters: it does not define a proper probability model in the space of inputs. This is because the density is not normalized within the principal subspace. In other words, if we perform PCA on some data and then ask how well *new* data are fit by the model, the only criterion available is the squared distance of the new data from their projections into the principal subspace. A datapoint far away from the training data but nonetheless near the principal subspace will be assigned a high "pseudo-likelihood" or low error (see figure 1). Similarly, it is not possible to generate "fantasy" data from a PCA model. In this note I introduce a new model called *sensible* principal component analysis (SPCA), an obvious modification of PCA which *does* define a proper covariance structure in the data space. Its parameters can also be learned with an EM algorithm, given below.

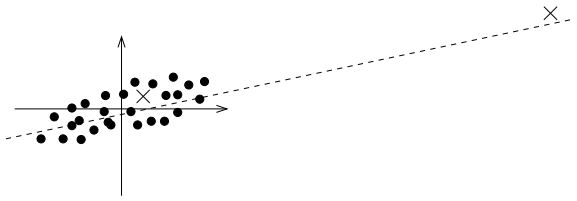

Figure 1: PCA does not define a proper density model in the data space. The only measure of how well new data fits the model is the distance from the principal subspace; thus points far away from the bulk of the data but nonetheless near the principal subspace will have low reconstruction error. In the figure, the two points denoted with X have the same reconstruction error.

2 Whence EM for PCA?

Principal component analysis can be viewed as a limiting case of a particular class of linear-Gaussian models. The goal of such models is to capture the covariance structure of an observed p-dimensional variable $\mathbf{y}$ using fewer than the $p(p+1)/2$ free parameters required in a full covariance matrix. Linear-Gaussian models do this by assuming that $\mathbf{y}$ was produced as a linear transformation of some k-dimensional latent variable $\mathbf{x}$ plus additive Gaussian noise. Denoting the transformation by

the $p \times k$ matrix $\mathbf{C}$, and the (p-dimensional) noise by $\mathbf{v}$ (with covariance matrix $\mathbf{R}$) the generative model can be written[1] as

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{v} \qquad \mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad \mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{R}) \tag{1a}$$

The latent or cause variables $\mathbf{x}$ are assumed to be independent and identically distributed according to a unit variance spherical Gaussian. Since $\mathbf{v}$ are also independent and normal distributed (and assumed independent of $\mathbf{x}$), the model reduces to a single Gaussian model for $\mathbf{y}$ which we can write explicitly:

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{C}\mathbf{C}^T + \mathbf{R}\right) \tag{1b}$$

In order to save parameters over the direct covariance representation in p-space, it is necessary to choose $k < p$ and also to restrict the covariance structure of the Gaussian noise $\mathbf{v}$ by constraining the matrix $\mathbf{R}$. The latter restriction is not merely to save on parameters: the covariance of the observation noise *must* be restricted in some way[2] for the model to capture any interesting or informative projections in the state $\mathbf{x}$. If $\mathbf{R}$ were not restricted, the learning algorithm could simply choose $\mathbf{C} = \mathbf{0}$ and then set $\mathbf{R}$ to be the sample covariance of the data thus trivially achieving the maximum likelihood model by explaining all of the structure in the data as noise. (Remember that since the model has reduced to a single Gaussian distribution for $\mathbf{y}$ we can do no better than having the covariance of our model equal the sample covariance of our data.)

2.1 Inference and learning

There are two central problems of interest when working with the linear-Gaussian models described above. The first problem is that of *state inference* or *compression* which asks: *given* fixed model parameters $\mathbf{C}$ and $\mathbf{R}$, what can be said about the unknown hidden states $\mathbf{x}$ given some observations $\mathbf{y}$? Since the datapoints are independent, we are interested in the posterior probability $P(\mathbf{x}|\mathbf{y})$ over a single hidden state given the corresponding single observation. This can be easily computed by linear matrix projection and the resulting density is itself Gaussian:

$$P(\mathbf{x}|\mathbf{y}) = \frac{P(\mathbf{y}|\mathbf{x})\,P(\mathbf{x})}{P(\mathbf{y})} = \frac{\mathcal{N}(\mathbf{C}\mathbf{x}, \mathbf{R})\,|_{\mathbf{y}}\;\mathcal{N}(\mathbf{0}, \mathbf{I})\,|_{\mathbf{x}}}{\mathcal{N}(\mathbf{0}, \mathbf{C}\mathbf{C}^T + \mathbf{R})\,|_{\mathbf{y}}} \tag{2a}$$

$$P(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\boldsymbol{\beta}\mathbf{y}, I - \boldsymbol{\beta}\mathbf{C})\,|_{\mathbf{x}}\,, \qquad \boldsymbol{\beta} = \mathbf{C}^T(\mathbf{C}\mathbf{C}^T + \mathbf{R})^{-1} \tag{2b}$$

from which we obtain not only the expected value $\boldsymbol{\beta}\mathbf{y}$ of the unknown state but also an estimate of the uncertainty in this value in the form of the covariance $\mathbf{I} - \boldsymbol{\beta}\mathbf{C}$. Computing $\mathbf{y}$ from $\mathbf{x}$ ("reconstruction") is also straightforward: $P(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{C}\mathbf{x}, \mathbf{R})\,|_{\mathbf{y}}$. Finally, computing the likelihood of any datapoint $\mathbf{y}$ is merely an evaluation under (1b).

The second problem is that of *learning*, or *parameter fitting* which consists of identifying the matrices $\mathbf{C}$ and $\mathbf{R}$ that make the model assign the highest likelihood to the observed data. There are a family of *EM algorithms* to do this (which differ slightly for the various cases of restrictions to $\mathbf{R}$ but all follow a similar structure). They use the inference formula above in the **e-step** to estimate the unknown state and then choose $\mathbf{C}$ and (restricted) $\mathbf{R}$ in the **m-step** so as to maximize the expected joint likelihood of the estimated $\mathbf{x}$ and the observed $\mathbf{y}$.

[1] All vectors are column vectors. To denote the transpose of a vector or matrix I use the notation $\mathbf{x}^T$. The determinant of a matrix is denoted by $|\mathbf{A}|$ and matrix inversion by $\mathbf{A}^{-1}$. The zero matrix is $\mathbf{0}$ and the identity matrix is $\mathbf{I}$. The symbol $\sim$ means "distributed according to". A multivariate normal (Gaussian) distribution with mean μ and covariance matrix $\boldsymbol{\Sigma}$ is written as $\mathcal{N}(\mu, \boldsymbol{\Sigma})$. The same Gaussian evaluated at the point $\mathbf{x}$ is denoted $\mathcal{N}(\mu, \boldsymbol{\Sigma})\,|_{\mathbf{x}}$.

[2] For example, if the shape of the noise distribution is restricted to be axis aligned (in other words its covariance matrix is diagonal) the model is known as *factor analysis*.

2.2 Zero noise limit

Principal component analysis is a limiting case of the linear-Gaussian model as the covariance of the noise $\mathbf{v}$ becomes infinitesimally small and equal in all directions. Mathematically, PCA is obtained by taking the limit $\mathbf{R} = \lim_{\epsilon \to 0} \epsilon \mathbf{I}$. This has the effect of making the likelihood of a point $\mathbf{y}$ dominated solely by the squared distance between it and its reconstruction $\mathbf{Cx}$. The directions of the columns of $\mathbf{C}$ which minimize this error are known as the *principal components*. Inference now reduces to[3] simple least squares projection:

$$P(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\boldsymbol{\beta}\mathbf{y}, I - \boldsymbol{\beta}\mathbf{C})\,|_{\mathbf{x}}\,, \qquad \boldsymbol{\beta} = \lim_{\epsilon \to 0} \mathbf{C}^T(\mathbf{C}\mathbf{C}^T + \epsilon\mathbf{I})^{-1} \tag{3a}$$

$$P(\mathbf{x}|\mathbf{y}) = \mathcal{N}\left((\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{y}, \mathbf{0}\right)|_{\mathbf{x}} = \delta(\mathbf{x} - (\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{y}) \tag{3b}$$

Since the noise has become infinitesimal, the posterior over states collapses to a single point and the covariance becomes zero.

3 An EM algorithm for PCA

The key observation of this note is that even though the principal components can be computed explicitly, there is still an EM algorithm for learning in the zero noise limit. It can be easily derived from the standard algorithms (see for example [3, 2]) by replacing the usual **e-step** with the projection above. The algorithm is:

- **e-step**: $\mathbf{X} = (\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{Y}$
- **m-step**: $\mathbf{C}^{new} = \mathbf{Y}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}$

where $\mathbf{Y}$ is a $p \times n$ matrix of all the observed data and $\mathbf{X}$ is a $k \times n$ matrix of the unknown states. The columns of $\mathbf{C}$ will span the space of the first k principal components. To compute the corresponding eigenvectors and eigenvalues explicitly, the data can be projected into this k-dimensional subspace and an ordered orthogonal basis for the covariance in the subspace can be constructed.

The workings of the algorithm are illustrated graphically in Figure 2 below. The figure shows the current model at each iteration; the directions of the solid lines indicate the eigenvector guesses and the lengths of the lines indicate the eigenvalues as computed by projecting the data onto the eigenvector direction.

The intuition behind the algorithm is as follows: guess an orientation for the principal subspace. *Fix* the guessed subspace and project the data $\mathbf{y}$ into it to give the values of the hidden states $\mathbf{x}$. Now *fix* the values of the hidden states and choose the subspace orientation which minimizes the squared reconstruction errors of the datapoints. For the simple two-dimensional example above, I can give a physical analogy. Imagine that we have a rod pinned at the origin which is free to rotate. Pick an orientation for the rod. Holding the rod still, project every datapoint onto the rod, and attach each projected point to its original point with a spring. Now release the rod. Repeat. The direction of the rod represents our guess of the principal component of the dataset. The energy stored in the springs is the reconstruction error we are trying to minimize.

[3] Recall that if $\mathbf{C}$ is $p \times k$ with $p > k$ and is rank k then left multiplication by $\mathbf{C}^T(\mathbf{C}\mathbf{C}^T)^{-1}$ (which appears not to be well defined because $(\mathbf{C}\mathbf{C}^T)$ is not invertible) is *exactly equivalent to* left multiplication by $(\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T$. This is the same as the SVD idea of defining the "inverse" of the diagonal singular value matrix as the inverse of an element unless it is zero in which case it remains zero. The intuition is that even though $\mathbf{C}\mathbf{C}^T$ truly is not invertible, the directions along which it is not invertible are exactly those which $\mathbf{C}^T$ is about to project out.

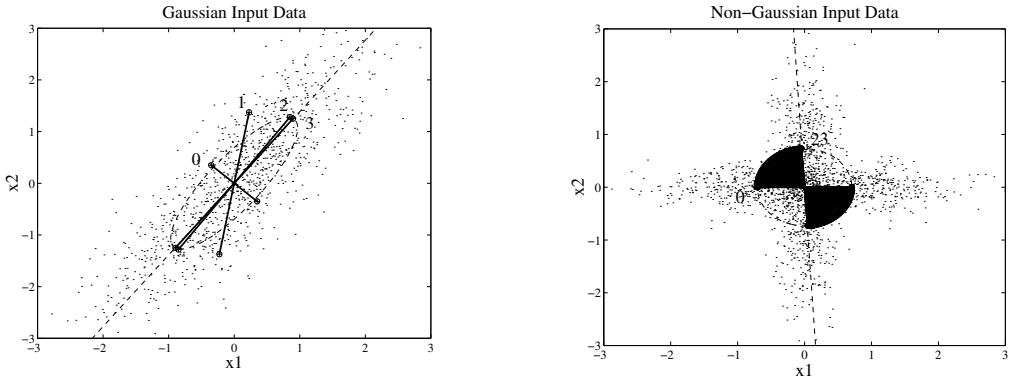

Figure 2: Examples of iterations of the algorithm. The left panel shows the learning of the first principal component of data drawn from a Gaussian distribution, while the right panel shows learning on data from a non-Gaussian distribution. The dashed lines indicate the direction of the leading eigenvector of the sample covariance. The dashed ellipse is the one standard deviation contour of the sample covariance. The progress of the algorithm is indicated by the solid lines whose directions indicate the guess of the eigenvector and whose lengths indicate the guess of the eigenvalue at each iteration. The iterations are numbered; number 0 is the initial condition. Notice that the difficult learning on the right does not get stuck in a local minimum, although it does take more than 20 iterations to converge which is unusual for Gaussian data (see Figure 2).

3.1 Convergence and Complexity

The EM learning algorithm for PCA amounts to an iterative procedure for finding the k leading eigenvectors without explicit computation of the sample covariance. It is attractive because for small k its complexity is limited by $O(knp)$ per iteration and so depends only linearly on *both* the dimensionality of the data and the number of points. Methods that explicitly compute the sample covariance matrix have complexities limited by $O(np^2)$, while methods like the snap-shot method that form linear combinations of the data must compute and diagonalize a matrix of all possible inner products between points and thus are limited by $O(n^2p)$ complexity. The complexity scaling of the algorithm compared to standard methods is shown in Figure 2 below. For these simulations, the number of datapoints n was always fixed to be 10 times the dimensionality p. For each dimensionality, a random covariance matrix $\mathbf{\Sigma}$ was generated[4] and then $10p$ points were drawn from $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$. The number of floating point operations required to find the first principal component was recorded using MATLAB's `flops` function. As expected, the EM algorithm scales more favourably than others in cases where k is small and both p and n are large. If $k \approx p \approx n$ (we want all the eigenvectors) then all methods are $O(p^3)$.

The standard convergence proofs for EM [1] apply to this algorithm as well, so we can be sure that it will always reach a local maximum of likelihood. Furthermore, it is believed that in fact for PCA learning there are *no* stable local maxima other than the global optimum and so the algorithm will always converge to the true principal subspace. (There are certainly other local extrema, however the conjecture is that they are all saddle points or minima of the likelihood surface. A proof for this has not yet been discovered, however in extensive simulations local maxima have never been

[4] First, an axis-aligned covariance is created with the p eigenvalues drawn at random from a uniform distribution in some positive range (which scaled with the dimensionality). Then a single point is drawn from a p-dimensional zero mean spherical Gaussian and the axis with the largest eigenvalue is aligned with this point.

observed.) Another possible concern is that the number of iterations required for convergence may scale with p or n. To investigate this question, I have explicitly computed the leading eigenvector for synthetic datasets (as above, with $n = 10p$) of varying dimension and recorded the number of iterations of the EM algorithm required for the inner product of the eigendirection with the current guess of the algorithm to be 0.999 or greater. Up to 450 dimensions (4500 datapoints), the number of iterations remains roughly constant with a mean of 3.6. The rate of convergence depends on the ratio of the largest eigenvalue to the second largest eigenvalue; the closer the two are in magnitude the slower the convergence will be. In the degenerate limit (which never occurs for real data) of two equal eigenvalues, the principal directions are not well defined.

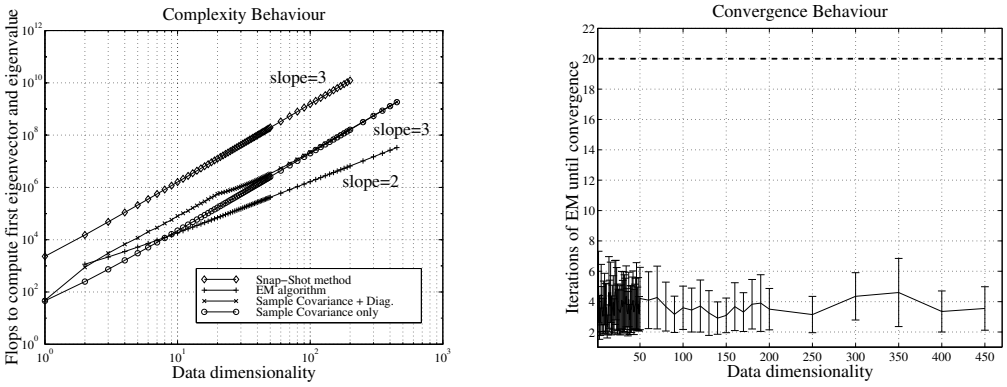

Figure 3: Time complexity and convergence behaviour of the algorithm. In all cases, the number of datapoints n is 10 times the dimensionality p. For the left panel, the number of floating point operations to find the leading eigenvector and eigenvalue were recorded. The EM algorithm was always run for exactly 20 iterations. The cost shown for EM includes the explicit orthogonalization into the eigenbasis after the principal subspace is found. The cost shown for diagonalization of the sample covariance uses the MATLAB functions `cov` and `eigs`. The snap-shot method is shown to indicate scaling only; one would not normally use it when $n > p$. In the right hand panel, convergence was investigated by explicitly computing the leading eigenvector and then running the EM algorithm until the dot product of its guess and the true eigendirection was 0.999 or more. The error bars show $\pm$ one standard deviation across many runs. The dashed line shows the number of iterations used to produce the EM algorithm curve ('+') in the left panel.

3.2 Missing data

In the complete data setting, the values of the projections or hidden states $\mathbf{x}$ are viewed as the "missing information" for EM. During the **e-step** we compute these values by projecting the observed data into the current subspace. This minimizes the model error given the observed data and the model parameters. However, if some of the input points are missing certain coordinate values, we can easily estimate those values in the same fashion. Instead of estimating only $\mathbf{x}$ as the value which minimizes the squared distance between the point and its reconstruction we can generalize the **e-step** to:

- **e-step**: For each (possibly incomplete) point $\mathbf{y}$ find the unique pair of points $\mathbf{x}^*$ and $\mathbf{y}^*$ (such that $\mathbf{x}^*$ lies in the current principal subspace and $\mathbf{y}^*$ lies in the subspace defined by the known coordinates of $\mathbf{y}$) which minimize the norm $\|\mathbf{C}\mathbf{x}^* - \mathbf{y}^*\|$. Set the corresponding column of $\mathbf{X}$ to $\mathbf{x}^*$ and the corresponding column of $\mathbf{Y}$ to $\mathbf{y}^*$.

If $\mathbf{y}$ is complete, then $\mathbf{y}^* = \mathbf{y}$ and $\mathbf{x}^*$ is found exactly as before. If not, then $\mathbf{x}^*$ and $\mathbf{y}^*$ are the solution to a least squares problem and can be found by, for example, QR factorization of a particular constraint matrix. Notice that this method is not restricted to missing coordinates in the data; the unknown degrees of freedom may lie in any directions in the space. For example, we may know the sum of two coordinates of a datapoint but not their individual values. Using this generalized **e-step** I have found the leading principal components for datasets in which *every* point is missing some coordinates. This outperforms replacing each missing coordinate with the mean of known coordinates. I am currently investigating the performance of this algorithm on a very high-dimensional vision problem with missing data.

4 Sensible Principal Component Analysis

If we require $\mathbf{R}$ to be a multiple $\epsilon\mathbf{I}$ of the identity matrix (in other words the covariance ellipsoid of $\mathbf{v}$ is spherical) but do *not* take the limit as $\epsilon \to 0$ then we have a model which I shall call *sensible principal component analysis* or SPCA. The columns of $\mathbf{C}$ are still known as the *principal components* (it can be shown that they are the same as in regular PCA) and we will call the scalar value ϵ on the diagonal of $\mathbf{R}$ the *global noise level*. Note that SPCA uses $1 + pk - k(k-1)/2$ free parameters to model the covariance. Once again, inference is done with equation (2b) and learning by an EM algorithm (given below). Because it has a *finite* noise level ϵ, SPCA defines a proper generative model and probability distribution in the data space:

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{C}\mathbf{C}^T + \epsilon\mathbf{I}\right) \tag{4}$$

which makes it possible to evaluate the actual *likelihood* of new test data under an SPCA model. Furthermore, this likelihood will be much lower for data far from the training set even if they are near the principal subspace, unlike the reconstruction error from PCA (see figure 1).

The EM algorithm for SPCA is:

- **e-step**: $\boldsymbol{\beta} = \mathbf{C}^T(\mathbf{C}\mathbf{C}^T + \epsilon\mathbf{I})^{-1} \qquad \mu_{\mathbf{x}} = \boldsymbol{\beta}\mathbf{Y} \qquad \boldsymbol{\Sigma}_{\mathbf{x}} = n\mathbf{I} - n\boldsymbol{\beta}\mathbf{C} + \mu_{\mathbf{x}}\mu_{\mathbf{x}}^T$
- **m-step**: $\mathbf{C}^{new} = \mathbf{Y}\mu_{\mathbf{x}}^T\boldsymbol{\Sigma}^{-1} \qquad \epsilon^{new} = \mathrm{trace}[\mathbf{Y}\mathbf{Y}^T - \mathbf{C}\mu_{\mathbf{x}}\mathbf{Y}^T]/np$

Two subtle points about complexity are important to notice: First, since $\epsilon\mathbf{I}$ is diagonal, the inversion in the **e-step** can be performed efficiently using the matrix inversion lemma:

$$(\mathbf{C}\mathbf{C}^T + \epsilon\mathbf{I})^{-1} = (\mathbf{I}/\epsilon - \mathbf{C}(\mathbf{I} + \mathbf{C}^T\mathbf{C}/\epsilon)^{-1}\mathbf{C}^T/\epsilon^2) \tag{5}$$

Second, since we are only taking the trace of the matrix in the **m-step**, we do not need to compute the full sample covariance $\mathbf{X}\mathbf{X}^T$ but instead can compute only the variance along each coordinate. These two observations mean that for small k, learning for SPCA also enjoys a complexity limited by $O(knp)$ and not worse.

5 An example with real image data

The EMPCA algorithm was tested on a large database of natural images.[5] Each grayscale image consisted of a 256×256 array of real values between 0 and 1. Each image was represented as a single long column vector (in raster-scan order), making the dimensionality p of the data space equal to 2^{16}. Several hundred thousand images were used, making the number n of datapoints roughly 2^{19}. The first 16 eigenvectors and eigenvalues were extracted from the data. Furthermore, to minimize the memory impact of the algorithm, the online version of the algorithm was used: each datapoint was read in from disk and then discarded before the next point was processed. The entire computation took less than four hours of wall clock time on a 133 MHz original Pentium processor running MATLAB (most of which was dominated by disk access). The interesting point of this experiment is not so much the resulting eigenvectors but rather to show that such a computation *could* be performed on a very large dataset using the EMPCA algorithm in an extremely short time.

The results are shown in the figures 4 and 5. The mean frame is almost uniform, although expanding its dynamic range shows a "horizon" effect in which the upper portion is lighter than the lower portion indicating that on average, images have more light above than below. The first eigenmode is an almost purely DC mode, although again some structure is observed in the expanded dynamic range image. This structure is believed to be the response function of the CCD that was used to digitize the images. The first 10 eigenmodes are shown and correspond roughly to Fourier modes as is expected for data which comes from a translation-invariant distribution.

6 Summary and relationships to previous methods

The methods developed in this paper provide three advantages. They allow *simple and efficient* computation of a few eigenvectors and eigenvalues when working with many datapoints in high dimensions. They permit this computation even in the presence of missing data. On a real vision problem with missing information, I have computed the 10 leading eigenvectors and eigenvalues of 2^{19} points in 2^{16} dimensions in a few hours using MATLAB on a modest workstation. Finally, through a small variation, these methods allow the computation not only of the principal subspace but of a complete Gaussian probabilistic model which allows one to generate data and compute true likelihoods.

The EM algorithm for PCA, derived above using probabilistic arguments, is closely related to two well know sets of algorithms. The first are *power iteration* methods for solving matrix eigenvalue problems. Roughly speaking, these methods iteratively update their eigenvector estimates through repeated multiplication by the matrix to be diagonalized. In the case of PCA, explicitly forming the sample covariance and multiplying by it to perform such power iterations would be disastrous. However since the sample covariance is in fact a sum of outer products of individual vectors, we can multiply by it efficiently without ever computing it. In fact, the EM algorithm is exactly equivalent to performing power iterations for finding $\mathbf{C}$ using this trick. Iterative methods for partial least squares (e.g. the NIPALS algorithm) are doing the same trick for regression. Taking the singular value decomposition (SVD) of the data matrix directly is a related way to find the principal subspace. If Lanczos or Arnoldi methods are used to compute this SVD, the resulting iterations are similar to those of the EM algorithm. The second class of methods are the competitive

[5] Thanks to Dawei Dong for providing this data.

learning methods for finding the principal subspace such as Sanger's and Oja's rules [6, 7, 5]. These methods enjoy the same storage and time complexities as the EM algorithm; however, their update steps reduce but do not minimize the cost and so they typically need more iterations and require a learning rate parameter to be set by hand.

Acknowledgements

I would like to thank John Hopfield and my fellow graduate students for constant and excellent feedback on these ideas. In particular I am grateful to Erik Winfree for discussions about the missing data portion of this work as well as Carlos Brody, Sanjoy Mahajan and Maneesh Sahani for generally useful comments. The work of Zoubin Ghahramani and Geoff Hinton was an important motivation for this study, and their comments on many recent ideas are much appreciated.

References

[1] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society series B*, 39:1–38, 1977.

[2] B. S. Everitt. *An Introduction to Latent Variable Models.* Chapman and Hill, London, 1984.

[3] Zoubin Ghahramani and Geoffrey Hinton. The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1, Dept. of Computer Science, University of Toronto, May 1996 (revised Feb. 1997).

[4] Zoubin Ghahramani and Michael I. Jordan. Supervised learning from incomplete data via an EM approach. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 120–127. Morgan Kaufmann, 1994.

[5] E. Oja. Neural networks, principal components, and subspaces. *International Journal Of Neural Systems*, 1(1):61–68, 1989.

[6] Terence David Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473, 1989.

[7] Terence David Sanger. An optimality principle for unsupervised learning. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 11–19. Morgan Kaufmann, 1989.

[8] L. Sirovich. Turbulence and the dynamics of coherent structures. *Quarterly Applied Mathematics*, 45(3):561–590, 1987.

[9] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Claredon Press, Oxford, England, 1965.

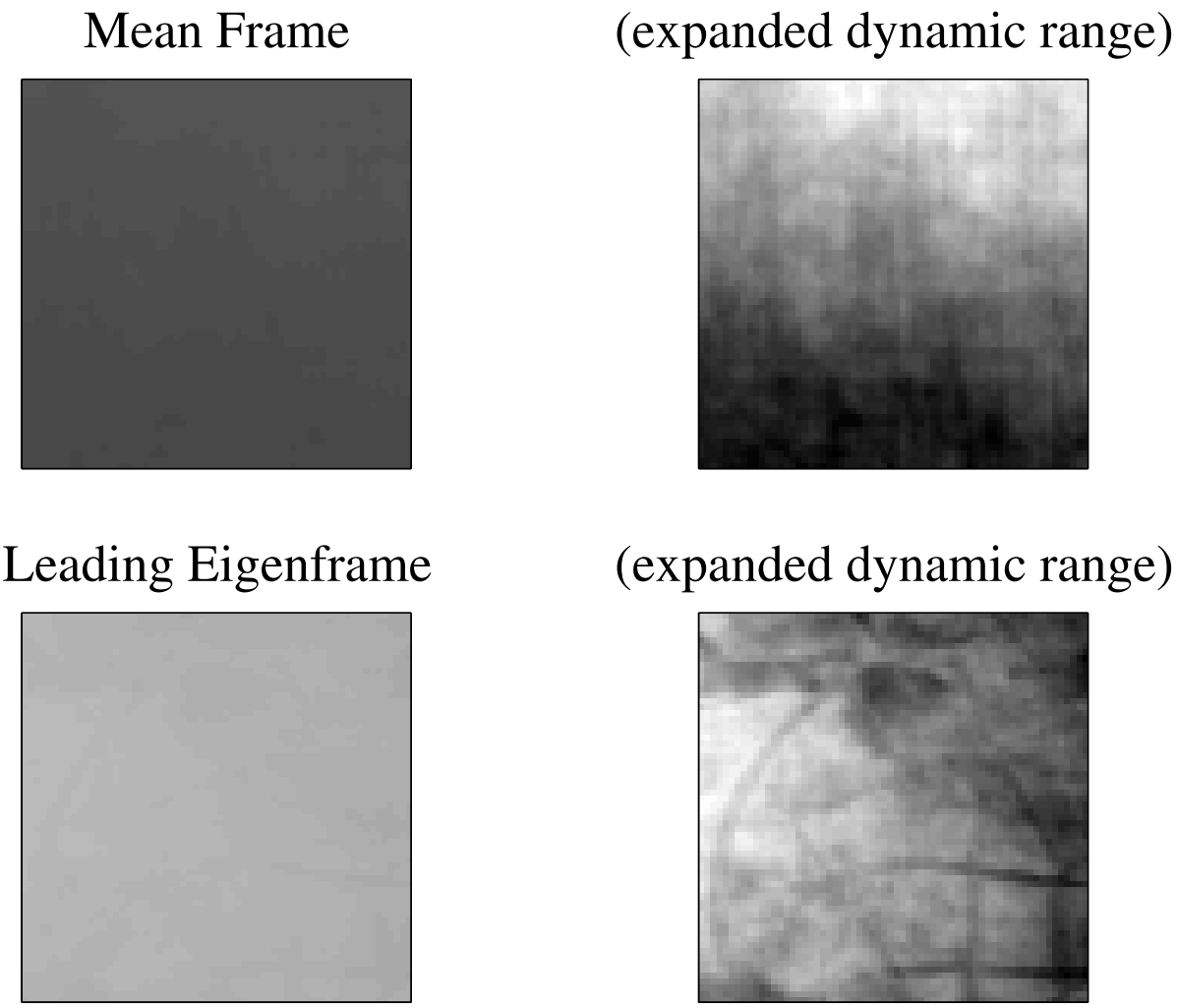

Figure 4: Mean frame and leading eigenmode as computed by the EMPCA algorithm for a large database of roughly 2^{19} natural images. Each image contains 2^{16} pixels. The mean frame is almost uniform, although expanding its dynamic range shows a "horizon" effect in which the upper portion is lighter than the lower portion indicating that on average, images have more light above than below. The first eigenmode is an almost purely DC mode, although again some structure is observed in the expanded dynamic range image. This structure is believed to be the response function of the CCD that was used to digitize the images.

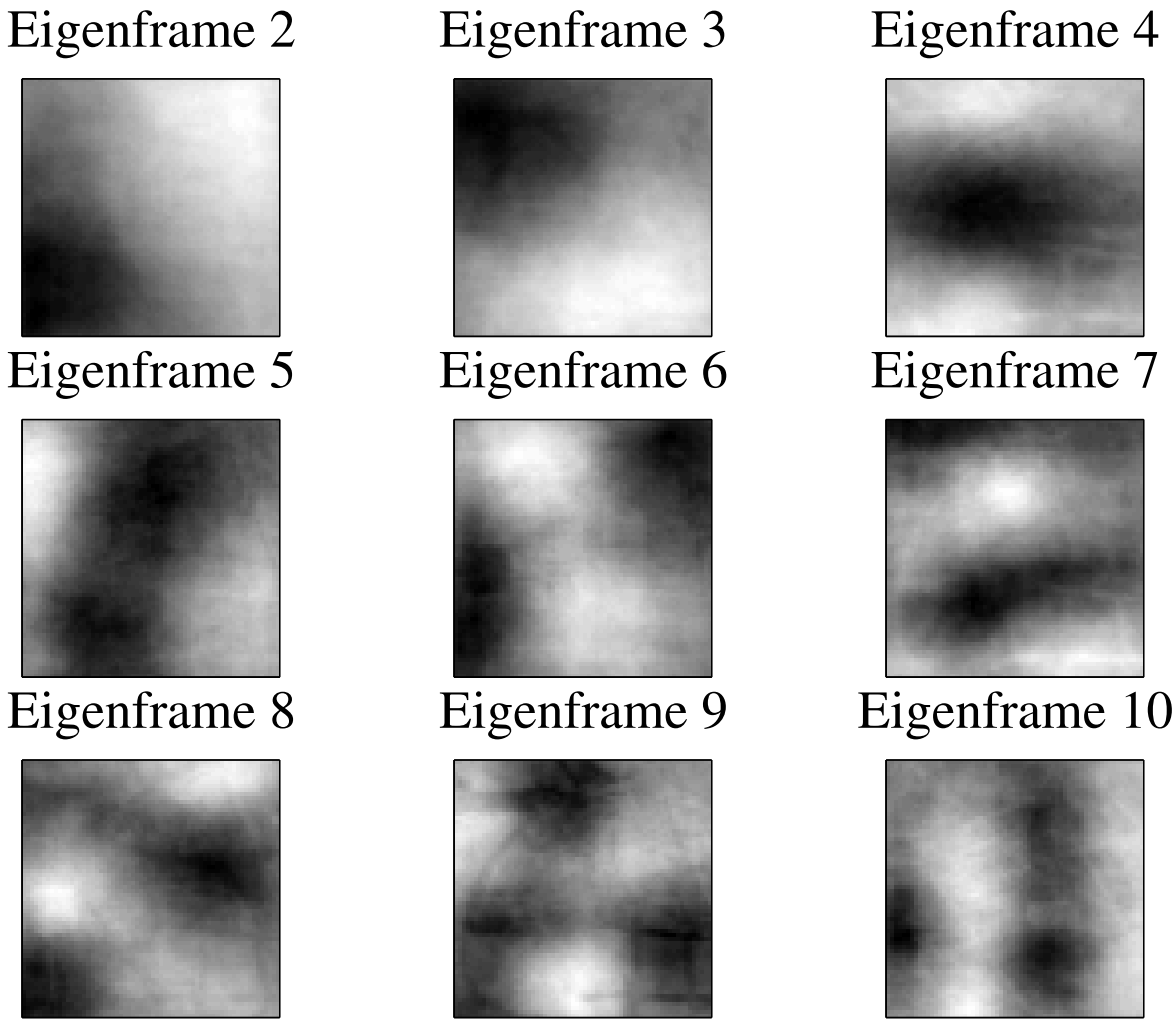

Figure 5: Eigenmodes 2 through 8 of the natural images, corresponding roughly to the Fourier modes. Each image contains 2^{16} pixels. Computing the first 16 eigenvectors for a dataset of roughly 2^{19} images took less than four hours in MATLAB.