Principal Component Analysis for Sparse High-Dimensional Data

Tapani Raiko, Alexander Ilin, and Juha Karhunen

Adaptive Informatics Research Center, Helsinki Univ. of Technology
P.O. Box 5400, FI-02015 TKK, Finland
{Tapani.Raiko, Alexander.Ilin, Juha.Karhunen}@tkk.fi
http://www.cis.hut.fi/projects/bayes/

Abstract. Principal component analysis (PCA) is a widely used technique for data analysis and dimensionality reduction. Eigenvalue decomposition is the standard algorithm for solving PCA, but a number of other algorithms have been proposed. For instance, the EM algorithm is much more efficient in case of high dimensionality and a small number of principal components. We study a case where the data are high-dimensional and a majority of the values are missing. In this case, both of these algorithms prove inadequate. We propose using a gradient descent algorithm inspired by Oja's rule, and speeding it up by an approximate Newton's method. The computational complexity of the proposed method is linear to the number of observed values in the data and to the number of principal components. The experiments with Netflix data confirm that the proposed algorithm is about ten times faster than any of the four comparison methods.

1 Introduction

Principal component analysis (PCA) [1–6] is a classic technique in data analysis. It can be used for compressing higher dimensional data sets to lower dimensional ones for data analysis, visualization, feature extraction, or data compression. PCA can be derived from a number of starting points and optimization criteria [2–4]. The most important of these are minimization of the mean-square error in data compression, finding mutually orthogonal directions in the data having maximal variances, and decorrelation of the data using orthogonal transformations [5].

While standard PCA is a very well-established linear statistical technique based on second-order statistics (covariances), it has recently been extended into various directions and considered from novel viewpoints. For example, various adaptive algorithms for PCA have been considered and reviewed in [4, 6]. Fairly recently, PCA was shown to emerge as a maximum likelihood solution from a probabilistic latent variable model independently by several authors; see [3] for a discussion and references.

In this paper, we study PCA in the case where most of the data values are missing (or unknown). Common algorithms for solving PCA prove to be

inadequate in this case, and we thus propose a new algorithm. The problem of overfitting is also studied and solutions based on regularization and variational Bayesian learning are given.

2 Algorithms for Principal Component Analysis

Principal subspace and components Assume that we have n d-dimensional data vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, which form the $d \times n$ data matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$. The matrix $\mathbf{X}$ is decomposed into

$$\mathbf{X} \approx \mathbf{AS}, \tag{1}$$

where $\mathbf{A}$ is a $d \times c$ matrix, $\mathbf{S}$ is a $c \times n$ matrix and $c \leq d \leq n$. Principal subspace methods [6, 4] find $\mathbf{A}$ and $\mathbf{S}$ such that the reconstruction error

$$C = \|\mathbf{X} - \mathbf{AS}\|_F^2 = \sum_{i=1}^{d} \sum_{j=1}^{n} (x_{ij} - \sum_{k=1}^{c} a_{ik} s_{kj})^2 , \tag{2}$$

is minimized. There F denotes the Frobenius norm, and x_{ij}, a_{ik}, and s_{kj} elements of the matrices $\mathbf{X}$, $\mathbf{A}$, and $\mathbf{S}$, respectively. Typically the row-wise mean is removed from $\mathbf{X}$ as a preprocessing step.

Without any further constraints, there exist infinitely many ways to perform such a decomposition. However, the subspace spanned by the column vectors of the matrix $\mathbf{A}$, called the *principal subspace*, is unique. In PCA, these vectors are mutually orthogonal and have unit length. Further, for each $k = 1, \ldots, c$, the first k vectors form the k dimensional principal subspace. This makes the solution practically unique, see [4, 2, 5] for details.

There are many ways to determine the principal subspace and components [6, 4, 2]. We will discuss three common methods that can be adapted for the case of missing values.

Singular Value Decomposition PCA can be determined by using the singular value decomposition (SVD) [5]

$$\mathbf{X} = \mathbf{U\Sigma V}^T , \tag{3}$$

where $\mathbf{U}$ is a $d \times d$ orthogonal matrix, $\mathbf{V}$ is an $n \times n$ orthogonal matrix and $\mathbf{\Sigma}$ is a $d \times n$ pseudodiagonal matrix (diagonal if $d = n$) with the singular values on the main diagonal [5]. The PCA solution is obtained by selecting the c largest singular values from $\mathbf{\Sigma}$, by forming $\mathbf{A}$ from the corresponding c columns of $\mathbf{U}$, and $\mathbf{S}$ from the corresponding c rows of $\mathbf{\Sigma V}^T$.

Note that PCA can equivalently be defined using the eigendecomposition of the $d \times d$ covariance matrix $\mathbf{C}$ of the column vectors of the data matrix $\mathbf{X}$:

$$\mathbf{C} = \frac{1}{n}\mathbf{XX}^T = \mathbf{UDU}^T , \tag{4}$$

Here, the diagonal matrix $\mathbf{D}$ contains the eigenvalues of $\mathbf{C}$, and the columns of the matrix $\mathbf{U}$ contain the unit-length eigenvectors of $\mathbf{C}$ in the same order [6, 4, 2, 5]. Again, the columns of $\mathbf{U}$ corresponding to the largest eigenvalues are taken as $\mathbf{A}$, and $\mathbf{S}$ is computed as $\mathbf{A}^T\mathbf{X}$. This approach can be more efficient for cases where $d \ll n$, since it avoids the $n \times n$ matrix.

EM Algorithm The EM algorithm for solving PCA [7] iterates updating $\mathbf{A}$ and $\mathbf{S}$ alternately. When either of these matrices is fixed, the other one can be obtained from an ordinary least-squares problem. The algorithm alternates between the updates

$$\mathbf{S} \leftarrow (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{X}\,, \qquad \mathbf{A} \leftarrow \mathbf{X}\mathbf{S}^T(\mathbf{S}\mathbf{S}^T)^{-1}. \tag{5}$$

This iteration is especially efficient when only a few principal components are needed, that is $c \ll d$ [7].

Subspace Learning Algorithm It is also possible to minimize the reconstruction error (2) by any optimization algorithm. Applying the gradient descent algorithm yields rules for simultaneous updates

$$\mathbf{A} \leftarrow \mathbf{A} + \gamma(\mathbf{X} - \mathbf{A}\mathbf{S})\mathbf{S}^T\,, \qquad \mathbf{S} \leftarrow \mathbf{S} + \gamma\mathbf{A}^T(\mathbf{X} - \mathbf{A}\mathbf{S})\,. \tag{6}$$

where $\gamma > 0$ is called the learning rate. Oja-Karhunen learning algorithm [8, 9, 6, 4] is an online learning method that uses the EM formula for computing $\mathbf{S}$ and the gradient for updating $\mathbf{A}$, a single data vector at a time.

A possible speed-up to the subspace learning algorithm is to use the natural gradient [10] for the space of matrices. This yields the update rules

$$\mathbf{A} \leftarrow \mathbf{A} + \gamma(\mathbf{X} - \mathbf{A}\mathbf{S})\mathbf{S}^T\mathbf{A}^T\mathbf{A}\,, \qquad \mathbf{S} \leftarrow \mathbf{S} + \gamma\mathbf{S}\mathbf{S}^T\mathbf{A}^T(\mathbf{X} - \mathbf{A}\mathbf{S})\,. \tag{7}$$

If needed, the end result of subspace analysis can be transformed into the PCA solution, for instance, by computing the eigenvalue decomposition $\mathbf{S}\mathbf{S}^T = \mathbf{U}_\mathbf{S}\mathbf{D}_\mathbf{S}\mathbf{U}_\mathbf{S}^T$ and the singular value decomposition $\mathbf{A}\mathbf{U}_\mathbf{S}\mathbf{D}_\mathbf{S}^{1/2} = \mathbf{U}_\mathbf{A}\mathbf{\Sigma}_\mathbf{A}\mathbf{V}_\mathbf{A}^T$. The transformed $\mathbf{A}$ is formed from the first c columns of $\mathbf{U}_\mathbf{A}$ and the transformed $\mathbf{S}$ from the first c rows of $\mathbf{\Sigma}_\mathbf{A}\mathbf{V}_\mathbf{A}^T\mathbf{D}_\mathbf{S}^{-1/2}\mathbf{U}_\mathbf{S}^T\mathbf{S}$. Note that the required decompositions are computationally lighter than the ones done to the data matrix directly.

3 Principal Component Analysis with Missing Values

Let us consider the same problem when the data matrix has missing entries[1]. In the following there are $N = 9$ observed values and 6 missing values marked with

[1] We make the typical assumption that values are missing at random, that is, the missingness does not depend on the unobserved data. An example where the assumption does not hold is when out-of-scale measurements are marked missing.

a question mark (?):

$$\mathbf{X} = \begin{bmatrix} -1 & +1 & 0 & 0 & ? \\ -1 & +1 & ? & ? & 0 \\ ? & ? & -1 & +1 & ? \end{bmatrix} . \tag{8}$$

We would like to find $\mathbf{A}$ and $\mathbf{S}$ such that $\mathbf{X} \approx \mathbf{AS}$ for the observed data samples. The rest of the product $\mathbf{AS}$ represents the reconstruction of missing values.

Adapting SVD: Imputation Algorithm One can use the SVD approach (4) in order to find an approximate solution to the PCA problem. However, estimating the covariance matrix $\mathbf{C}$ becomes very difficult when there are lots of missing values. If we estimate $\mathbf{C}$ leaving out terms with missing values from the average, we get for the estimate of the covariance matrix

$$\mathbf{C} = \frac{1}{n}\mathbf{X}\mathbf{X}^T = \begin{bmatrix} 0.5 & 1 & 0 \\ 1 & 0.667 & ? \\ 0 & ? & 1 \end{bmatrix} . \tag{9}$$

There are at least two problems. First, the estimated covariance 1 between the first and second components is larger than their estimated variances 0.5 and 0.667. This is clearly wrong, and leads to the situation where the covariance matrix is not positive (semi)definite and some of its eigenvalues are negative. Secondly, the covariance between the second and the third component could not be estimated at all[2].

Another option is to complete the data matrix by iteratively imputing the missing values (see, e.g., [11]). Initially, the missing values can be replaced by zeroes. The covariance matrix of the complete data can be estimated without the problems mentioned above. Now, the product $\mathbf{AS}$ can be used as a better estimate for the missing values, and this process can be iterated until convergence. This approach requires the use of the complete data matrix, and therefore it is computationally very expensive if a large part of the data matrix is missing. The time complexity of computing the sample covariance matrix explicitly is $O(nd^2)$. We will further refer to this approach as the *imputation algorithm*.

Note that after convergence, the missing values do not contribute to the reconstruction error (2). This means that the imputation algorithm leads to the solution which minimizes the reconstruction error of observed values only.

Adapting the EM Algorithm Grung and Manne [12] studied the EM algorithm in the case of missing values. Experiments showed a faster convergence compared to the iterative imputation algorithm. The computational complexity is $O(Nc^2 + nc^3)$ per iteration, where N is the number of observed values, assuming naïve matrix multiplications and inversions but exploiting sparsity. This is quite a bit heavier than EM with complete data, whose complexity is $O(ndc)$ [7] per iteration.

[2] It could be filled by finding a value that maximizes the determinant of the covariance matrix (and thus the entropy of the underlying Gaussian distribution).

Adapting the Subspace Learning Algorithm The subspace learning algorithm works in a straightforward manner also in the presence of missing values. We just take the sum over only those indices i and j for which the data entry x_{ij} (the ijth element of $\mathbf{X}$) is observed, in short $(i,j) \in O$. The cost function is

$$C = \sum_{(i,j)\in O} e_{ij}^2 \,, \qquad \text{with} \qquad e_{ij} = x_{ij} - \sum_{k=1}^{c} a_{ik} s_{kj} \,. \tag{10}$$

and its partial derivatives are

$$\frac{\partial C}{\partial a_{il}} = -2 \sum_{j|(i,j)\in O} e_{ij} s_{lj} \,, \qquad \frac{\partial C}{\partial s_{lj}} = -2 \sum_{i|(i,j)\in O} e_{ij} a_{il} \,. \tag{11}$$

The update rules for gradient descent are

$$\mathbf{A} \leftarrow \mathbf{A} + \gamma \frac{\partial C}{\partial \mathbf{A}} \,, \qquad \mathbf{S} \leftarrow \mathbf{S} + \gamma \frac{\partial C}{\partial \mathbf{S}} \tag{12}$$

and the update rules for natural gradient descent are

$$\mathbf{A} \leftarrow \mathbf{A} + \gamma \frac{\partial C}{\partial \mathbf{A}} \mathbf{A}^T \mathbf{A} \,, \qquad \mathbf{S} \leftarrow \mathbf{S} + \gamma \mathbf{S}\mathbf{S}^T \frac{\partial C}{\partial \mathbf{S}} \,. \tag{13}$$

We propose a novel speed-up to the original simple gradient descent algorithm. In Newton's method for optimization, the gradient is multiplied by the inverse of the Hessian matrix. Newton's method is known to converge fast especially in the vicinity of the optimum, but using the full Hessian is computationally too demanding in truly high-dimensional problems. Here we use only the diagonal part of the Hessian matrix. We also include a control parameter α that allows the learning algorithm to interpolate between the standard gradient descent ($\alpha = 0$) and the diagonal Newton's method ($\alpha = 1$), much like the well known Levenberg-Marquardt algorithm. The learning rules then take the form

$$a_{il} \leftarrow a_{il} - \gamma' \left(\frac{\partial^2 C}{\partial a_{il}^2} \right)^{-\alpha} \frac{\partial C}{\partial a_{il}} = a_{il} + \gamma \frac{\sum_{j|(i,j)\in O} e_{ij} s_{lj}}{\left(\sum_{j|(i,j)\in O} s_{lj}^2 \right)^{\alpha}} \,, \tag{14}$$

$$s_{lj} \leftarrow s_{lj} - \gamma' \left(\frac{\partial^2 C}{\partial s_{lj}^2} \right)^{-\alpha} \frac{\partial C}{\partial s_{lj}} = s_{lj} + \gamma \frac{\sum_{i|(i,j)\in O} e_{ij} a_{il}}{\left(\sum_{i|(i,j)\in O} a_{il}^2 \right)^{\alpha}} \,. \tag{15}$$

The computational complexity is $O(Nc + nc)$ per iteration.

4 Overfitting

A trained PCA model can be used for reconstructing missing values:

$$\hat{x}_{ij} = \sum_{k=1}^{c} a_{ik} s_{kj} \,, \quad (i,j) \notin O \,. \tag{16}$$

Although PCA performs a linear transformation of data, overfitting is a serious problem for large-scale problems with lots of missing values. This happens when the value of the cost function C in Eq. (10) is small for training data, but the quality of prediction (16) is poor for new data. For further details, see [13].

Regularization A popular way to regularize ill-posed problems is penalizing the use of large parameter values by adding a proper penalty term into the cost function; see for example [3]. In our case, one can modify the cost function in Eq. (2) as follows:

$$C_\lambda = \sum_{(i,j)\in O} e_{ij}^2 + \lambda(\|\mathbf{A}\|_F^2 + \|\mathbf{S}\|_F^2)\,. \tag{17}$$

This has the effect that the parameters that do not have significant evidence will decay towards zero.

A more general penalization would use different regularization parameters λ for different parts of $\mathbf{A}$ and $\mathbf{S}$. For example, one can use a λ_k parameter of its own for each of the column vectors $\mathbf{a}_k$ of $\mathbf{A}$ and the row vectors $\mathbf{s}_k$ of $\mathbf{S}$. Note that since the columns of $\mathbf{A}$ can be scaled arbitrarily by rescaling the rows of $\mathbf{S}$ accordingly, one can fix the regularization term for $\mathbf{a}_k$, for instance, to unity.

It is well known that an equivalent optimization problem can be obtained using a probabilistic formulation with (independent) Gaussian priors and a Gaussian noise model:

$$p(x_{ij} \mid \mathbf{A}, \mathbf{S}) = \mathcal{N}\left(x_{ij}; \sum_{k=1}^{c} a_{ik} s_{kj}, v_x\right), \tag{18}$$

$$p(a_{ik}) = \mathcal{N}(a_{ik}; 0, 1)\,, \quad p(s_{kj}) = \mathcal{N}(s_{kj}; 0, v_{sk})\ , \tag{19}$$

where $\mathcal{N}(x; m, v)$ denotes the random variable x having a Gaussian distribution with the mean m and variance v. The regularization parameter $\lambda_k = v_{sk}/v_x$ is the ratio of the prior variances v_{sk} and v_x. Then, the cost function (ignoring constants) is minus logarithm of the posterior for $\mathbf{A}$ and $\mathbf{S}$:

$$C_{\mathrm{BR}} = \sum_{(i,j)\in O} \left(e_{ij}^2/v_x + \ln v_x\right) + \sum_{i=1}^{d}\sum_{k=1}^{c} a_{ik}^2 + \sum_{k=1}^{c}\sum_{j=1}^{n} \left(s_{kj}^2/v_{sk} + \ln v_{sk}\right) \tag{20}$$

An attractive property of the Bayesian formulation is that it provides a natural way to choose the regularization constants. This can be done using the evidence framework (see, e.g., [3]) or simply by minimizing C_{BR} by setting v_x, v_{sk} to the means of e_{ij}^2 and s_{kj}^2 respectively. We will use the latter approach and refer to it as *regularized PCA*.

Note that in case of joint optimization of C_{BR} w.r.t. a_{ik}, s_{kj}, v_{sk}, and v_x, the cost function (20) has a trivial minimum with $s_{kj} = 0$, $v_{sk} \to 0$. We try to avoid this minimum by using an orthogonalized solution provided by unregularized PCA from the learning rules (14) and (15) for initialization. Note also that

setting v_{sk} to small values for some components k is equivalent to removal of irrelevant components from the model. This allows for automatic determination of the proper dimensionality c instead of discrete model comparison (see, e.g., [14]). This justifies using separate v_{sk} in the model in (19).

Variational Bayesian Learning Variational Bayesian (VB) learning provides even stronger tools against overfitting. VB version of PCA by [14] approximates the joint posterior of the unknown quantities using a simple multivariate distribution. Each model parameter is described *a posteriori* using independent Gaussian distributions. The means can then be used as point estimates of the parameters, while the variances give at least a crude estimate of the reliability of these point estimates. The method in [14] does not extend to missing values easily, but the subspace learning algorithm (Section 3) can be extended to VB. The derivation is somewhat lengthy, and it is omitted here together with the variational Bayesian learning rules because of space limitations; see [13] for details. The computational complexity of this method is still $O(Nc + nc)$ per iteration, but the VB version is in practice about 2–3 times slower than the original subspace learning algorithm.

5 Experiments

Collaborative filtering is the task of predicting preferences (or producing personal recommendations) by using other people's preferences. The Netflix problem [15] is such a task. It consists of movie ratings given by $n = 480189$ customers to $d = 17770$ movies. There are $N = 100480507$ ratings from 1 to 5 given, from which 1408395 ratings are reserved for validation (or probing). Note that 98.8% of the values are thus missing. We tried to find $c = 15$ principal components from the data using a number of methods.[3] We subtracted the mean rating for each movie, assuming 22 extra ratings of 3 for each movie as a Dirichlet prior.

Computational Performance In the first set of experiments we compared the computational performance of different algorithms on PCA with missing values. The root mean square (rms) error is measured on the training data, $\mathrm{E}_O = \sqrt{\frac{1}{|O|} \sum_{(i,j) \in O} e_{ij}^2}$. All experiments were run on a dual cpu AMD Opteron SE 2220 using Matlab.

First, we tested the imputation algorithm. The first iteration where the missing values are replaced with zeros, was completed in 17 minutes and led to $E_O = 0.8527$. This iteration was still tolerably fast because the complete data matrix was sparse. After that, it takes about 30 hours per iteration (we computed 500 rows (and columns) of the covariance matrix at a time because the size of the complete data matrix is huge: $d \times n > 8 \cdot 10^9$). After three iterations, E_O was still 0.8513. Also, estimating the covariance matrix based on only

[3] The PCA approach has been considered by other Netflix contestants as well (see, e.g., [16, 17]).

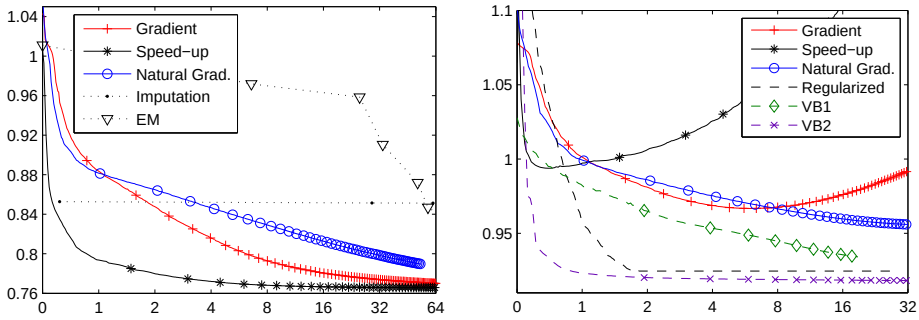

Fig. 1. *Left:* Learning curves for unregularized PCA (Section 3) applied to the Netflix data: Root mean-square error on the training data E_O is plotted against computation time in hours. *Right:* The root mean square error on the validation data E_V from the Netflix problem during runs of several algorithms: basic PCA (Section 3), regularized PCA (Section 4) and VB (Section 4). VB1 fixes variances v_{sk} to large values while VB2 updates all the parameters. The time scales are linear below 1 and logarithmic above 1.

the observed part of the data led to problems as explained in Section 3. The covariance matrix had both missing values and values out of range.

Using the EM algorithm by [12], the E-step (updating $\mathbf{S}$) takes 7 hours and the M-step (updating $\mathbf{A}$) takes 18 hours. (There is some room for optimization since we used a straightforward Matlab implementation.) Each iteration gives a much larger improvement compared to the imputation algorithm, but starting from a random initialization, EM could not reach a good solution in reasonable time.

We also tested the subspace learning algorithm described in Section 3 with and without the proposed speed-up. Each run of the algorithm with different values of the speed-up parameter α was initialized in the same starting point (generated randomly from a normal distribution). The learning rate γ was adapted such that if an update decreased the cost function, γ was multiplied by 1.1. Each time an update would increase the cost, the update was canceled and γ was divided by 2. Figure 1 (left) shows the learning curves for basic gradient descent, natural gradient descent, and the proposed speed-up with the best found parameter value $\alpha = 0.625$. The proposed speed-up gave about a tenfold speed-up compared to the gradient descent algorithm even if each iteration took longer. Natural gradient was slower than the basic gradient. Table 1 gives a summary of the computational complexities.

Overfitting We compared PCA (Section 3), regularized PCA (Section 4) and VB-PCA (Section 4) by computing the rms reconstruction error for the validation set V, that is, testing how the models generalize to new data: $\mathrm{E}_V = \sqrt{\frac{1}{|V|}\sum_{(i,j)\in V} e_{ij}^2}$. We tested VB-PCA by firstly fixing some of the parameter values (this run is marked as VB1 in Fig. 1, see [13] for details) and secondly by

Method	Complexity	Seconds/Iter	Hours to $E_O = 0.85$
Gradient	$O(Nc + nc)$	58	1.9
Speed-up	$O(Nc + nc)$	110	0.22
Natural Grad.	$O(Nc + nc^2)$	75	3.5
Imputation	$O(nd^2)$	110000	$\gg 64$
EM	$O(Nc^2 + nc^3)$	45000	58

Table 1. Summary of the computational performance of different methods on the Netflix problem. Computational complexities (per iteration) assume naïve computation of products and inverses of matrices and ignores the computation of SVD in the imputation algorithm. While the proposed speed-up makes each iteration slower than the basic gradient update, the time to reach the error level 0.85 is greatly diminished.

adapting them (marked as VB2). We initialized regularized PCA and VB1 using normal PCA learned with $\alpha = 0.625$ and orthogonalized $\mathbf{A}$, and VB2 using VB1. The parameter α was set to $2/3$.

Fig. 1 (right) shows the results. The performance of basic PCA starts to degrade during learning, especially using the proposed speed-up. Natural gradient diminishes this phenomenon known as overlearning, but it is even more effective to use regularization. The best results were obtained using VB2: The final validation error E_V was 0.9180 and the training rms error E_O was 0.7826 which is naturally larger than the unregularized $E_O = 0.7657$.

6 Discussion

We studied a number of different methods for PCA with sparse data and it turned out that a simple gradient descent approach worked best due to its minimal computational complexity per iteration. We also gave it a more than tenfold speed-up by using an approximated Newton's method. We found out empirically that setting the parameter $\alpha = 2/3$ seems to work well for our problem. It is left for future work to find out whether this generalizes to other problem settings. There are also many other ways to speed-up the gradient descent algorithm. The natural gradient did not help here, but we expect that the conjugate gradient method would. The modification to the gradient proposed in this paper, could be used together with the conjugate gradient speed-up. This will be another future research topic.

There are also other benefits in solving the PCA problem by gradient descent. Algorithms that minimize an explicit cost function are rather easy to extend. The case of variational Bayesian learning applied to PCA was considered in Section 4, but there are many other extensions of PCA, such as using non-Gaussianity, non-linearity, mixture models, and dynamics.

The developed algorithms can prove useful in many applications such as bioinformatics, speech processing, and meteorology, in which large-scale datasets with missing values are very common. The required computational burden is linearly proportional to the number of measured values. Note also that the proposed techniques provide an analogue of confidence regions showing the reliability of

estimated quantities.

Acknowledgments This work was supported in part by the Academy of Finland under its Centers for Excellence in Research Program, and the IST Program of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views. We would like to thank Antti Honkela for useful comments.

References

1. Pearson, K.: On lines and planes of closest fit to systems of points in space. Philosophical Magazine **2**(6) (1901) 559–572
2. Jolliffe, I.: Principal Component Analysis. Springer-Verlag (1986)
3. Bishop, C.: Pattern Recognition and Machine Learning. Springer-Verlag (2006)
4. Diamantaras, K., Kung, S.: Principal Component Neural Networks - Theory and Application. Wiley (1996)
5. Haykin, S.: Modern Filters. Macmillan (1989)
6. Cichocki, A., Amari, S.: Adaptive Blind Signal and Image Processing - Learning Algorithms and Applications. Wiley (2002)
7. Roweis, S.: EM algorithms for PCA and SPCA. In: Advances in Neural Information Processing Systems 10, Cambridge, MA, MIT Press (1998) 626–632
8. Karhunen, J., Oja, E.: New methods for stochastic approximation of truncated Karhunen-Loeve expansions. In: Proc. 6th Int. Conf. on Pattern Recognition, Springer-Verlag (1982) 550–553
9. Oja, E.: Subspace Methods of Pattern Recognition. Research Studies Press and J. Wiley (1983)
10. Amari, S.: Natural gradient works efficiently in learning. Neural Computation **10**(2) (1998) 251–276
11. Tipping, M., Bishop, C.: Probabilistic principal component analysis. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **61**(3) (1999) 611–622
12. Grung, B., Manne, R.: Missing values in principal components analysis. Chemometrics and Intelligent Laboratory Systems **42**(1) (August 1998) 125–139
13. Raiko, T., Ilin, A., Karhunen, J.: Principal component analysis for large scale problems with lots of missing values. In: Proceedings of the 18th European Conference on Machine Learning (ECML 2007), Springer-Verlag (September 2007) 691–698
14. Bishop, C.: Variational principal components. In: Proc. 9th Int. Conf. on Artificial Neural Networks (ICANN99). (1999) 509–514
15. Netflix: Netflix prize webpage (2007) `http://www.netflixprize.com/`.
16. Funk, S.: Netflix update: Try this at home. Available at `http://sifter.org/~simon/journal/20061211.html` (December 2006)
17. Salakhutdinov, R., Mnih, A., Hinton, G.: Restricted Boltzmann machines for collaborative filtering. In: Proc. Int. Conf. on Machine Learning. (2007)