Fast Dimensionality Reduction and Simple PCA

Matthew Partridge* and Rafael Calvo†
SEDAL
Department of Electrical Engineering
University of Sydney
NSW, 2006, Australia
{mp,rafa}@sedal.usyd.edu.au

December 2, 1997

Abstract

A fast and simple algorithm for approximately calculating the principal components (PCs) of a data set and so reducing its dimensionality is described. This Simple Principal Components Analysis (SPCA) method was used for dimensionality reduction of two high-dimensional image databases, one of handwritten digits and one of handwritten Japanese characters. It was tested and compared with other techniques. On both databases SPCA shows a fast convergence rate compared with other methods and robustness to the reordering of the samples.

KEYWORDS: Principal component analysis, matrix diagonalization, Hebbian learning, image compression.

*All correspondance should be addressed to this author.
†Permanent address: Instituto de Fisica Rosario, Bvd. 27 de Febrero 210 Bis, 2000 Rosario, Argentina.

1 Introduction

High dimensional data analysis is becoming increasingly common as new problems are placing greater demands on computing resources. With high dimensional data, it is difficult to understand the underlying structure: it is difficult to “see the wood for the trees”. Additionally, the storage, transmission and processing of high dimensional data places great demands on systems. Hence, it is desirable to reduce the dimensionality of the data, whilst maintaining as much of its original structure.

Since the beginning of this century, several researchers (for example, [2], [6], [8] and [9]), have developed dimensionality reduction techniques; principal component analysis (PCA) is one of these techniques. In mathematical terms, n correlated random variables are transformed into a set of $d \leqslant n$ uncorrelated variables. These uncorrelated variables are linear combinations of the original variables and can be used to express the data in a reduced form. Data modelling and pattern recognition are better able to work on this reduced form, and the form is efficient for storage and transmission. PCA is also sometimes used as a data visualization technique since high dimensional datasets can be reduced to a low dimension and then plotted.

Due to the computational requirements of standard methods only data with $n \leqslant 100$ dimensions were considered until recently. Furthermore, the standard techniques do not work adaptively and so are often much too slow for real-time applications.

We introduce, in this article, the Simple PCA (SPCA) method that produces approximate solutions without the need for calculating a variance-covariance matrix and then diagonalizing it (as do most traditional methods) and does not depend on learning parameters (as do neural networks). Most importantly, for high dimensional datasets, SPCA is faster than other existing techniques and is easy to implement as a batch or an adaptive technique.

Section 2 briefly describes previous work on batch and adaptive techniques for PCA. Section 3 describes the SPCA algorithm and in Section 4 we show the results of using SPCA on two handwritten characters databases and compare this with other methods. Finally, Section 5 concludes.

2 Dimensionality Reduction Methods

Two types of methods have been used for PCA. Firstly, there are the more conventional *matrix methods*, in which all the data are used to calculate the variance-covariance structure and express it in a matrix. This matrix is then decomposed such that the decomposition reveals more about the principal directions of the variances. In practice this usually means that the matrix is diagonalized using some numerical technique such as singular value decomposition or the Householder-QR technique ([1] and [5]).

We will call the second type *data methods* since they work directly with the data. They might be implemented adaptively so the directions of the PCs are adjusted after a new datum is received, without the need of reusing all the data. This approach is suitable for real-time applications or for very high dimensional problems where the computational expense is an important consideration. Neural networks with Hebbian learning have been proposed for adaptive PCA [4]. However, the performance of these schemes depends heavily on the learning parameters and their proper determination makes them time consuming.

2.1 Matrix Methods

Most multivariate analysis textbooks (for example [1], [3], [7] and [10]) describe matrix methods for performing PCA. The goal is to find the eigenvectors of the covariance matrix. These eigenvectors correspond to the directions of the principal components of the original data, their statistical significance is given by their corresponding eigenvalues. In more detail, these techniques can be structured as:

1. Collect $\mathbf{x}_i$ of an n dimensional data set $\mathbf{X}$, $i = 1, 2, \ldots, m$.

2. Mean correct all the points: calculate the mean $\overline{\mathbf{x}}$ and substract it from each data point $\mathbf{x}_i - \overline{\mathbf{x}}$.

3. Calculate the variance-covariance matrix $\mathbf{C}$. Sometimes the correlation matrix $\mathbf{R}$ is used instead.

$$C_{ij} = (\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})$$

4. Determine eigenvalues and eigenvectors of the matrix. $\mathbf{C}$ is a real symmetric matrix so a positive real number λ and a nonzero vector $\boldsymbol{\alpha}$ can be found such that

$$\mathbf{C}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}$$

   where λ is called an eigenvalue and $\boldsymbol{\alpha}$ is an eigenvector of $\mathbf{C}$. To find a nonzero $\boldsymbol{\alpha}$ the characteristic equation $|\mathbf{C} - \lambda\mathbf{I}| = 0$ must be solved. If $\mathbf{C}$ is a $n \times n$ matrix of full rank, n eigenvalues can be found $\lambda_1, \lambda_2, \ldots, \lambda_n$. Using $(\mathbf{C} - \lambda I)\boldsymbol{\alpha} = 0$ all the corresponding eigenvectors can be found.

5. Sort the eigenvalues (and corresponding eigenvectors) so that $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n$.

6. Select the first $d \leqslant n$ eigenvectors and generate the data set in the new (usually compressed) representation.

2.1.1 Finding Eigenvalues and Eigenvectors

The determination of eigenvalues and eigenvectors in item 4 above can be performed using any diagonalization routine. If the matrix $\mathbf{A} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_n)$ contains the eigenvectors of a symmetric matrix $\mathbf{C}$, then $\mathbf{A}$ is orthogonal, and $\mathbf{C}$ can be decomposed as

$$\mathbf{C} = \mathbf{A}\mathbf{D}\mathbf{A}^T$$

where $\mathbf{D}$ is a diagonal matrix of the eigenvalues. Singular value decomposition (SVD) is widely used for this diagonalisation because of its numerical stability. It will always find all the eigenvalues and eigenvectors, even when only the biggest eigenvalues and corresponding eigenvectors are required. SVD can also be viewed as a more general method used for solving the eigen-problem in non-square matrices.

However, the fastest known technique for finding all the eigenvectors and eigenvalues of a square matrix is to firstly transform the matrix into tridiagonal form (using the Householder transformation) and then to decompose this tridiagonal matrix into $\mathbf{C} = \mathbf{QR}$ form, where $\mathbf{Q}$ is orthogonal and $\mathbf{R}$ is upper triangular (see [5]).

We ultimately want $d \leqslant n$ of the eigenvalues and eigenvectors. If $d \ll n$ then it is wasteful to calculate all n eignevalues and eigenvectors, only then to discard $n - d$ of these. Hotelling's power method [6] is an iterative technique which can be used to find just the largest $d \leqslant n$ of the eigenvalues and eigenvectors. As it finds the eigenvalues and eigenvectors in order, item 5 above is unnecessary. As it is an iterative technique, the time taken depends on the number of iterations required until convergence.

The time complexity of the matrix methods for the different items given above are: item 2: $O(mn)$, item 3: $O(mn^2)$, item 4: $O(n^3)$ for SVD and Householder-QR and $O(dn^2)$ for Hotelling's power method, item 5: $O(n \log n)$, item 6: $O(dn)$. In summary, to calculate the principal components of a dataset using SVD or the Householder-QR technique, the time complexity is $O(mn^2 + n^3)$, and using Hotelling's power method, it is $O(mn^2 + dn^2)$.

2.2 Data Methods

Although only d of the n dimensions are used, we see that the complexity of matrix methods grow at least quadratically with the dimension of the data. This can render the matrix methods impractical when n is large, or when time is at a premium.

For this reason neural network models that perform PCA have recently been developed [4], most of them using Hebbian learning rules. In general these neural networks will contain processing units with forward connections given by matrix $\mathbf{A}$, where the column vectors $\mathbf{a}_i$ are the connections between the inputs $\mathbf{x}_k$ and the output y_i. The i^{th} weight, $\mathbf{a}_i$ is used to approximate the i^{th} eigenvector, $\boldsymbol{\alpha}_i$. The output units have a value given by:

$$y_i = \mathbf{a}_i^T \mathbf{x}_k$$

In some of Hebbian architectures there are also lateral connections between outputs, such that these outputs effect each other (see [4] for a full description).

In the k^{th} iteration, Hebbian learning rules update the weights using:

$$\mathbf{a}_i^{k+1} = \mathbf{a}_i^k + \Phi(y_i, \mathbf{x}_k) \tag{1}$$

where Φ is some function of the inputs and outputs.

Oja's rule is a popular Hebbian technique for finding a principal component, and is representative of many of the approaches which have been studied. Its form for Φ is given by:

$$\Phi(y_i, \mathbf{x}_k) = \beta_k \left(y_i \mathbf{x}_k - y_i y_i \mathbf{a}_i^k \right)$$

where β_k is a learning rate, $y_i \mathbf{x}_k$ is known as the Hebbian term, and $y_i y_i \mathbf{a}_i^k$ is an anti-Hebbian term. The Hebbian term is responsible for driving the weight $\mathbf{a}_i^k$ towards the eigenvector $\boldsymbol{\alpha}_i$, and the anti-Hebbian term is responsible for keeping the weights bounded. All Hebbian learning rules have the Hebbian term; they vary in the learning parameter, the anti-Hebbian term and the lateral connections matrix. It can be shown that under some reasonable conditions, that $\mathbf{a}_i^k \to \boldsymbol{\alpha}_i$ as $k \to \infty$ (see [4]).

The complexity of Hebbian rules is $O(dmn)$ for the determination of d principal components of m n-dimensional data. The constants depend on the time required until convergence, which in turn is dependant largely on the learning rate β_k.

3 The Simple PCA (SPCA) Algorithm

In this section we describe a new algorithm that produces approximate solutions in an efficient way. It is a data oriented method and like in Hebbian learning, the algorithm does not explicitly calculate nor diagonalise the covariance matrix. Also SPCA does not require the tuning of learning parameters (a problem with Hebbian learning) and convergence is obtained with very few iterations.

SPCA is not a Hebbian algorithm, although it does have similarities. It does not necessarily add a Hebbian term (linear function of $y_i \mathbf{x}_k$), instead, SPCA considers other forms of Φ. In particular, we will show here the results for two different functions Φ_1 and Φ_2. As SPCA is intended to be a fast approximator for the principal components, we consider the threshold function:

$$\Phi_1(y_i, \mathbf{x}_k) = \begin{cases} \mathbf{x}_k & \text{if } y_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

When considering mean corrected data, it can be shown that this is equivalent to chosing $\Phi_1^\star$, except that Φ_1 can be implemented faster.

$$\Phi_1^\star(y_i, \mathbf{x}_k) = \begin{cases} +\mathbf{x}_k & \text{if } y_i \geq 0 \\ -\mathbf{x}_k & \text{otherwise} \end{cases}$$

We also consider a second function, Φ_2 which gives the Hebbian term.

$$\Phi_2(y_i, \mathbf{x}_k) = y_i \mathbf{x}_k$$

Each principal component direction found is normalised, in order to avoid divergence, so we do not use the anti-Hebbian term. Either of these algorithms may be implemented in an iterative or a batch mode. Initially $\mathbf{a}^0$ can be set to any vector. In a batch mode, the sum of $\Phi(y_i, \mathbf{x}_j)$ over all samples $\mathbf{x}_j$, is calculated where y_i is calculated relative to the old estimate $\mathbf{a}^k$. (For Φ_1 this is equivalent to summing those vectors positively correlated with $\mathbf{a}^k$.) This sum is then normalised to give the new estimate (equation 2).

$$\mathbf{a}^{k+1} = \frac{\sum_j \Phi(y_i, \mathbf{x}_j)}{\left\| \sum_j \Phi(y_i, \mathbf{x}_j) \right\|} \; ; \qquad y_i = \left(\mathbf{a}^k\right)^T \mathbf{x}_j \tag{2}$$

In an iterative mode, updating is performed according to equation 1, and the estimate is normalised once every pass. Unfortunately with Φ_2, the terms $\mathbf{a}^k$ and y can diverge during the pass, if unconstrained. Hence Φ_2 is modified for the iterative rule to include a normalising constant:

$$\Phi_2^{\text{iter.}}(y_i, \mathbf{x}_k) = \frac{1}{\|\mathbf{a}^k\|} y_i \mathbf{x}_k$$

So far the method of finding one principal component according to SPCA has been described. In order to find the next principal component, the effect of the first principal component is removed from the dataset, so as to avoid being found again. The removal of a component from the dataset, is called "deflation". This is performed by subtracting the component of each datum which is parallel to the principal component from that datum, leaving only an orthogonal component. That is, datum $\mathbf{x}_i$, is deflated by the unit principal component direction $\mathbf{a}$, according to:

$$\mathbf{x}_i' = \mathbf{x}_i - (\mathbf{a}^T \mathbf{x}_i)\mathbf{a}$$

The batch SPCA_1 algorithm is geometrically described in Figures 1 to 3.

The algorithmic complexity of the SPCA algorithms is $O(dmn)$ which is equal to the complexity of Hebbian learning algorithms, however each iteration of SPCA is faster and we experimentally show that the time taken until convergence is faster.

4 Experiments and Results

The goal of PCA is to explain as much variance as possible with the smallest number of variables (PCs). All the methods which we examined have similar compression rates, that is when they have converged they all explain approximately the same variance using the same number of principal components.

Our intention with these experiments is twofold, first to show that SPCA is a fast method for producing the principal components. This is important since in both databases a small number of variables can be used to explain most of the variance (*ie*

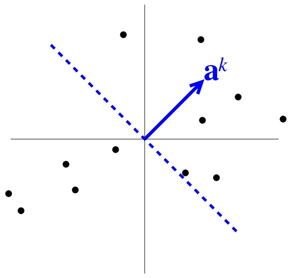

Figure 1: Data points (dots) with an approximate PC direction (an arrow), and its orthogonal plane (a dashed line).

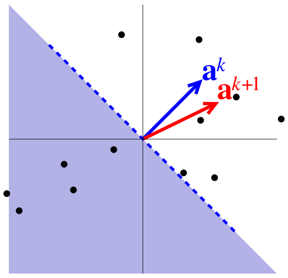

Figure 2: A better approximation to the PC direction is given by the normalised sum of the points "above" the plane.

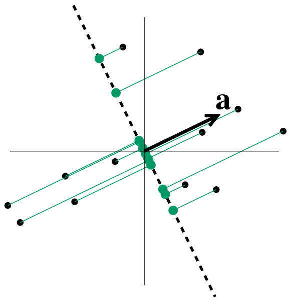

Figure 3: The data is deflated onto the plane orthogonal to the PC found. SPCA is then repeated on the deflated data.

there is a high compression rate). Secondly, that SPCA's performance is robust, (*ie*. the errors due to reordering the data are small).

For the sake of succinctness we only show here the results on two datasets of optical characters; several other datasets ($14 \leqslant n \leqslant 10\ 304$) in different domains have been experimented with and similar results have been obtained. These results are given in terms of the accumulated explained variance (EV) of d principal components.

$$EV_d = \frac{\sum_{i=1}^{d} \lambda_i}{Tr(\mathbf{C})} = \frac{\sum_{i=1}^{d} \lambda_i}{\sum_{i=1}^{n} \lambda_j}$$

This shows the accumulated relevance of the principal components which are found, relative to the total variance of the dataset.

4.1 Datasets

The handwritten digits dataset compiled by AT&T [11] consists of 11716 samples in 256-dimensional greyscale images (16×16) classified in 10 classes, one for each digit: most are real handwritten; some are artificially generated. Most of the classes contain more than 1000 samples. Examples of these images are given in Figure 4.

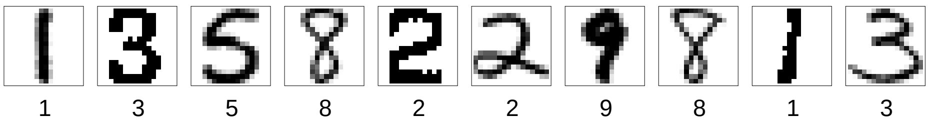

Figure 4: Sample images of the handwritten digits database.

The second database studied was based on a set of handwritten Japanese Kanji characters represented in 32×32 greyscale images (1024 dimensions) formed by averaging 2×2 regions in the original 64×64 binary images of the characters. Only a small subset of the original database was used for these experiments: we used 1920 items of 1024-dimensional data from 12 classes. There were 160 images of each of the 12 classes used. Examples of these images are given in Figure 5.

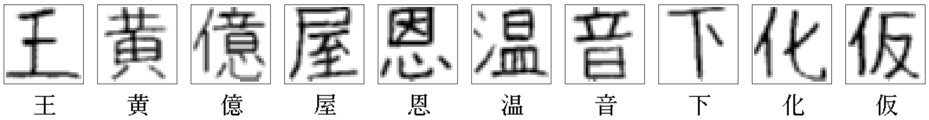

Figure 5: Sample images of the handwritten Japanese database.

4.2 Methods

The matrix methods which we examined were singular value decomposition (Numerical Recipes' implementation [5]), Householder-QR (Numerical Recipes' implementation [5]) and Hotelling's power method (from [4]). In the diagrams which follow, these are given the labels "SVD", "Householder-QR" and "Power" respectively.

The data oriented methods which we examined were Oja's single-unit rule, SPCA_1 and SPCA_2. For Oja's rule, in order to calculate more than one principal component direction, the deflation procedure (given above) was used, as opposed to Oja's M-unit rule or Sanger's rule (see [4]). This was done in order to examine the effects of the SPCA algorithm in isolation, though no significant departures from the multi-unit rules were noted. We experiented with two versions of Oja's single-unit rule, one with a fixed learning rate β, and one with a learning rate which is optimally ([4]) adaptively changed according to

$$\beta_k = \frac{\beta_0}{\sigma_k}$$

where σ_k is roughly equal to the variance of the output, as defined in [4]. Oja's single-unit rule with the fixed learning rate is denoted by "Oja"; with the adaptive learning rate, by "A-Oja" in the figures which follow.

In all the following experiments involving SPCA we applied one iterative pass through the data using Φ_1, this was followed by between zero and ten batch iterations using Φ_1 for SPCA_1 and Φ_2 for SPCA_2.

4.3 Results

In Figures 6 and 7 we have plotted the explained variance of 10 components as a percentage for the SPCA methods for different numbers of iterations. The dashed lines in these figures represent the maximum obtainable explained variance—this explained variance was obtained using SVD. The performance of SPCA_1 is denoted with a "1"; SPCA_2 with a "2". We note that with both datasets the methods behave in similar ways for all the classes, they all tend to converge to the maximum, and that the explained variance for the simpler images is higher than for the complex ones. We give more details of these results for the first class of each database.

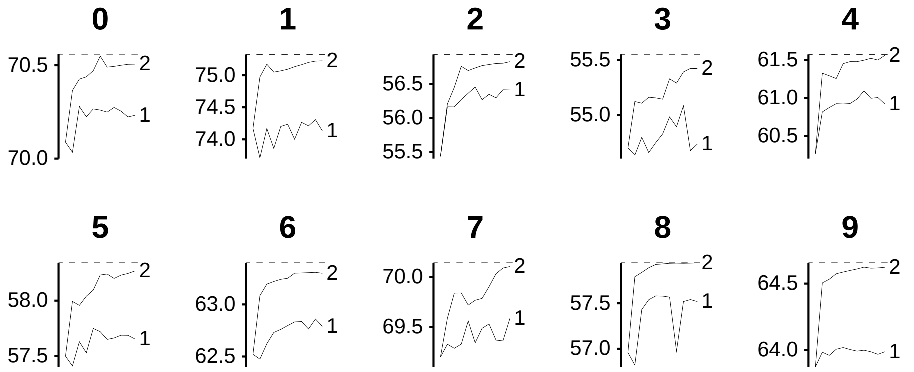

Figure 6: Convergence of explained variance of SPCA as a function of iterations, for different classes of the handwritten digits dataset.

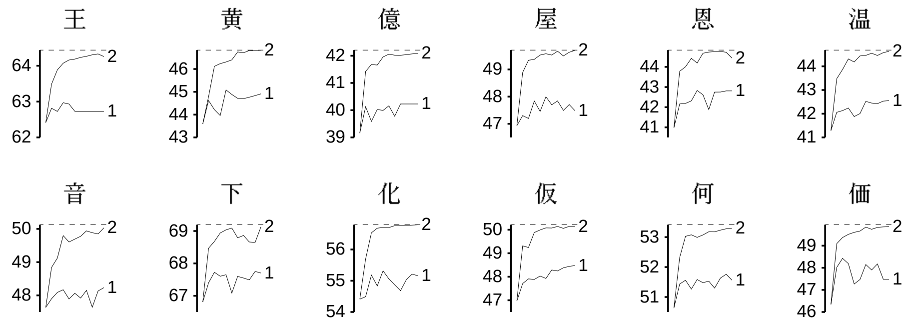

Figure 7: Convergence of explained variance of SPCA as a function of iterations, for different classes of the handwritten Japanese dataset.

Figure 8 shows the reconstructed images after compression using 10 components, we can see that this compression still retains most of the significant features of the image.

We now turn our attention to the speedup achieved with SPCA. Figures 9 and 10 show the explained variance of 10 components found by each method against the time taken to compute the solution. For the SPCA_1, SPCA_2, Oja, A-Oja and Power methods, the number of iterations performed was varied, giving different explained variances and different times.

Oja's single unit rule requires tuning the parameter β, and this varies for different datasets. After hundreds of experiments, the optimal value for the fixed β was found to be 0.00088 for the first class of the handwritten digits; 0.0028 for the first class of the Japanese characters. The optimal value for β_0 for A-Oja was found to be 0.00033 and 0.0021 for the first class of the handwritten digits and Japanese character datasets

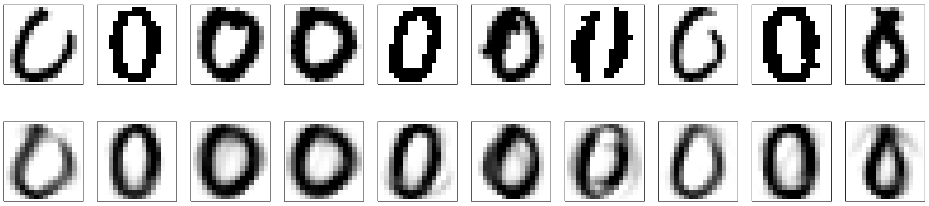

Figure 8: Original samples for handwritten 0s and the reconstructed samples after compression using 10 components.

respectively.

Data oriented methods may be sensitive to the order of presentation of the data, so in order to study the robustness, 50 trials of each method were run with different orderings of the data. (Oja's also had different initial random weight vectors over the 50 trials.) In the following plots of the results, the black line represents the median value, and the boundaries of the grey region represent the upper and lower quartiles.

The main advantage of SPCA is the speedup achieved over other methods. Times are exclusive of the time taken to read the files, or to remove the mean of the data—only the calculation of the covariance matrix and its diagonalisation are included for SVD and the power method; only the method as described is included in the timing for SPCA.

All implementations were compiled in C using the same compiler with the highest level of optimisation applied. All times were measured in seconds of user and system time on the same SUN 200 MHz UltraSPARC processor.

Note that only 10 of the 256 or 1024 possible principal components are calculated using SPCA, Oja's single unit rule and the Hotelling's power method, (whereas SVD and Householder-QR methods calculate all of them). However, as only 10 components are sought, this meets with the intention of the algorithm.

The "Maximum" line shows the maximum obtainable explained variance. The line marked "Covariance" shows the time required to just calculate the covariance matrix. Clearly this is a lower bound for the matrix methods. For comparison, the time required to generate the data in the new coordinates is shown as "Baseline". It is the amount of time required if we already knew the principal component directions are, and so represents an unobtainable lower bound on all methods.

5 Conclusion

We describe in this paper a new algorithm for performing PCA. The Simple PCA algorithm is a data oriented method, that does not require the computation of the variance-covariance matrix. It has similarities to the Hebbian learning methods for neural networks but uses a different set of functions for the correlation term, and it does not need any learning parameters.

Experimentally, we show that the method converges to solutions that preserve the same amount of information for the same compression rate as other data oriented (eg. Oja's rule for Hebbian learning) and matrix methods (e.g. power method and single value decomposition). It converges very fast so the explained variance obtained for a small number of components is high with little computation when compared to other methods. We have shown that the method is robust to different ordering of the data samples. It is also very simple to implement in software or hardware and does not require the tuning of learning parameters. Finally the simple geometric interpretation is useful for users without a strong mathematical or neural network background.

The issue of predicting what is the maximum explained variance for a given number of factors is still an open question. One approach might be to determine the eigenvalues (and not eigenvectors) of the covariance matrix, but, for SPCA, this is relatively very inefficient. This would be useful for a stopping criteria of the convergence of the algorithum. However, as more than 95% of the possible explained variance (for a given number of factors) is found after only a single iteration, the usefulness of even an efficient predictor is doubtful.

Therefore SPCA is a very useful set of techniques for dimensionality reduction in applications that require frequent updates (online) or with very high dimensionality.

The C++ code used in this paper is available from the authors upon request.

Acknowledgements

Matthew Partridge was supported by an Australian Postgraduate Award and a Norman I Price Scholarship. Rafael Calvo was supported by a scholarship from Universidad Nacional de Rosario and a Norman I Price Scholarship.

The authors would like to thank the reviewers as well as Andreas Weingessel for their very helpful comments.

References

[1] A. Basilevsky. *Statistical factor analysis and related methods: theory and applications*. Wiley, New York, 1994.

[2] A. Bravais. Analyse mathematique sur les probabilite's des erreurs de situation d'un point. *Sci. Math Phys.*, 9:255–332, 1846.

[3] C. Chatfield. *Introduction to multivariate analysis*. Chapman and Hall, London ; New York, 1980.

[4] K. Diamantaras and Kung. *Principal component neural networks : theory and applications*. Wiley, New York, 1996.

[5] Press W. [et al.]. *Numerical recipes in C : the art of scientific computing*. Cambridge University Press, Cambridge, 1988.

[6] Hotelling H. Analysis of a complex of statistical variables into principal components. *J. Educ. Psych.*, 24:417–441, 498–520, 1933.

[7] I. T. Jolliffe. *Principal component analysis*. Springer-Verlag, New York, 1986.

[8] K. Pearson. On lines and planes of closest fit to a system of points in space. *Philosophical Magazine*, 2:79–156, 1901.

[9] Frisch R. Correlation and scatter in statistical variables. *Nordic Stat. J*, 8:36–102, 1929.

[10] A. C. Rencher. *Methods of multivariate analysis*. Wiley, New York, 1995.

[11] P. Simard, Y Le Cun, and J. Denker. Efficient pattern recognition using a new transformation distance. In *NIPS*, volume 5, pages 50–58, 1993.

[12] J. H. Wilkinson. *Linear algebra*. Springer-Verlag, Berlin, New York, 1971.

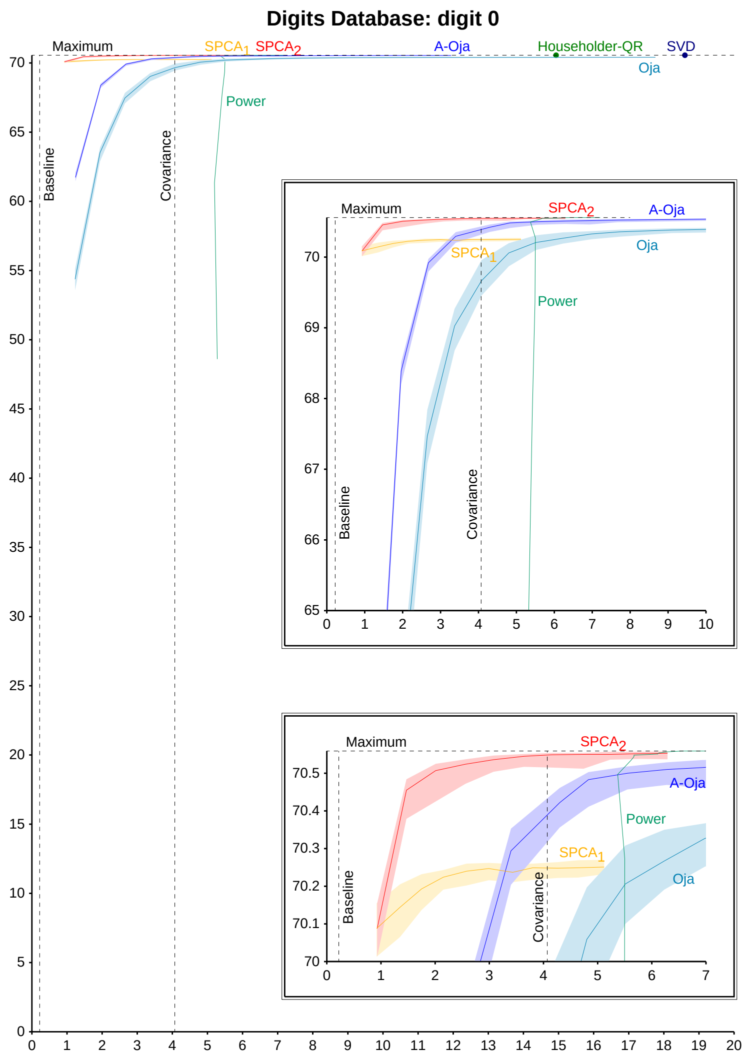

Figure 9: Percentage of the explained variance of the handwritten digits dataset, as function of CPU time spent in calculation by the different methods. The two insets are different magnifications of the graph. See text for a full description.

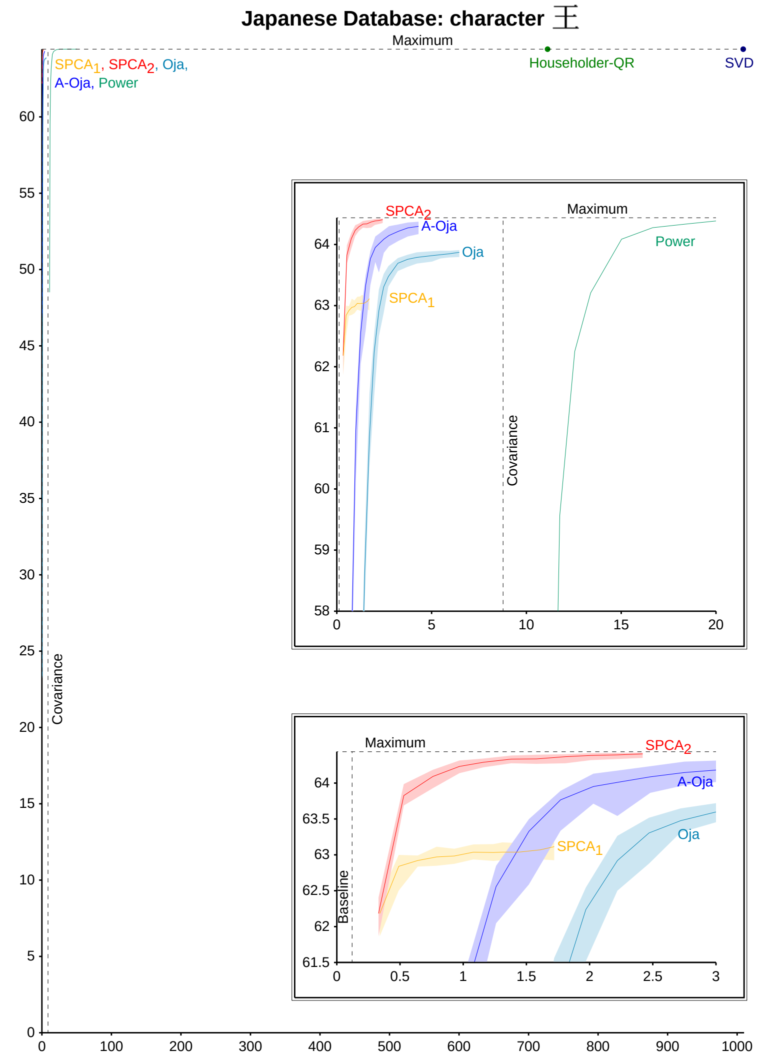

Figure 10: Percentage of the explained variance of the handwritten Japanese characters dataset, as function of CPU time spent in calculation by the different methods. The two insets are different magnifications of the graph. See text for a full description.