Trappl R. (ed.), *Cybernetics and Systems '96, volume 2.*
Austrian Society for Cybernetic Studies.
Vienna, Austria, 1996. pp. 1014–1019.

Adaptive Learning Algorithm for Principal Component Analysis With Partial Data

Andrzej Cichocki and **Włodzimierz Kasprzak** and **Władysław Skarbek**
Frontier Research Program RIKEN
Laboratory for Artificial Brain Systems
2–1 Hirosawa, Wako–shi
Saitama 351–01, JAPAN
e-mail: cia@kamo.riken.go.jp

Abstract

In this paper a fast and efficient adaptive learning algorithm for estimation of the principal components is developed. It seems to be especially useful in applications with changing environment, where the learning process has to be repeated in on–line manner. The approach can be called the cascade recursive least square (CRLS) method, as it combines a cascade (hierarchical) neural network scheme for input signal reduction with the RLS (recursive least square) filter for adaptation of learning rates. Successful application of the CRLS method for 2–D image compression–reconstruction and its performance in comparison to other known PCA adaptive algorithms are also documented.

1 Introduction

Principal Component Analysis (PCA) is a powerful data analysis tool in multivariate statistics [Jolliffe, 1986; Amari, 1977]. Up to now many neural network learning algorithms have been proposed for the PCA and its generalizations (for an overview see [Oja, 1992; Cichocki and Unbehauen, 1994; Karhunen and Joutsensalo, 1995]). The well known Oja's Learning Algorithm ([Oja, 1982]) has been further extended in [Sanger, 1989] (the GHA method), [Kung *et al.*, 1991] (the APEX method), [Abbas and Fahmy, 1993] (the SAMHL) and [Bannour and Azimi-Sadjadi, 1995] (the RLS method). In recent papers several new theoretic developments in neural network based PCA have been described (e.g. [Baldi and Hornik, 1989; Taylor and Coombes, 1993; Xu *et al.*, 1992]). PCA has also been widely studied and used in pattern recognition and signal processing [Niemann and Wu, 1993; Turk and Pentland, 1991].

The main purpose of this paper is to develop a reliable on–line adaptive learning algorithm for neural network. Its reliability means it converges for every signal of given class with well quality, and it precisely extracts an arbitrary number of principal components with high convergence speed. This fast learning is performed in one epoch of the input sample or even with partial data only, and it still can provide a better trade–off between learning speed and performance than the known adaptive algorithms. The application field in mind for the proposed algorithm is image processing – image restoration, compression and classification – if the observed scene is steadily changing (due to illumination change in case of a stationary camera or due to background or foreground change in case of an active camera). This assumption requires that the learning process is repeated in on–line manner.

The paper is organized in five sections. First a short introduction of the neural network based PCA is given and a discussion of previous and recent developments takes place. In section 3 we present a new method – the CRLS algorithm with adaptive learning rule. In section 4 test results for applying the above methods to single still images are summarized. Conclusions are drawn in the last section.

2 Neural Network Based PCA

The standard PCA, called also Karhunen–Loeve transformation, (KL) determines an optimal linear transformation of an input vector $\boldsymbol{x}$

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} \tag{1}$$

where $\boldsymbol{x} = [x_1(t), x_2(t), ..., x_n(t)]^T \in \mathcal{R}^n$ is a zero–mean input vector, $\boldsymbol{y} = [y_1(t), y_2(t), ..., y_m(t)]^T \in \mathcal{R}^m$ is the output vector, called *principal components*, and $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_n] \in \mathcal{R}^{m \times n}$ is a desired transformation matrix.

The orthogonal vectors $\boldsymbol{w}_i = [w_{i1}, w_{i2}, ..., w_{in}]^T$, ($\boldsymbol{w}_i^T \boldsymbol{w}_j = \delta_{ij}$ for $j \leq i$) are called *principal vectors.*

In the standard numerical approach for estimating the principal components first the autocorrelation matrix $\boldsymbol{R}_{xx} = \langle \boldsymbol{x}\boldsymbol{x}^T \rangle$ is computed and then its eigenvectors and associated eigenvalues are determined by one of the known numerical algorithms, e.g. the *QR algorithm* or the *SVD algorithm.* However, if the input data vectors have a large dimension, the correlation matrix $\boldsymbol{R}_{xx}$ is very large and this may lead to expensive computations. The neural network approach enables us to find the eigenvectors and the associated eigenvalues directly from the input vector $\boldsymbol{x}$ without the need to estimate the correlation matrix $\boldsymbol{R}_{xx}$. Such an approach is especially useful for non-stationary input data, i.e. in cases of tracking slow changes of correlation between the input data (signals) or in updating eigenvectors with new samples.

2.1 Robust Extraction of First PC – Extension of Oja's Rule

In order to find the optimal value of the vector $\boldsymbol{w}_1$ we can formulate (define) a suitable robust *loss function* as [Cichocki and Unbehauen, 1994]

$$E_1(\boldsymbol{w}_1, \boldsymbol{x}) = \rho(\boldsymbol{e}_1) = \rho(\boldsymbol{x} - y_1\boldsymbol{w}_1) = \sum_{i=1}^{n} \rho_i(e_{1i}), \quad (2)$$

where the $\rho_i(e_{1i})$ are the individual loss functions.

The minimization of the computational loss (cost) function according to the standard *gradient descent* approach leads to the generalized learning algorithm

$$\frac{d\boldsymbol{w}_1(t)}{dt} = -\mu_1(t)\frac{\partial E_1(\boldsymbol{w}_1)}{\partial \boldsymbol{w}_1} = \quad (3)$$

$$= \mu_1(t)\left[y_1(t)\Psi(\boldsymbol{e}_1) + \boldsymbol{x}(t)\boldsymbol{w}_1^T\Psi(\boldsymbol{e}_1)\right], \quad (4)$$

with $\boldsymbol{w}_1(0) \neq \boldsymbol{O}$, where $\mu_1(t) > 0$ is the learning rate and $\Psi_i(e_{1i}), (i = 1, 2, ..., n)$ are activation (influence) functions:

$$\Psi(\boldsymbol{e}_1) = [\Psi_1(e_{11}), \Psi_2(e_{12}), ..., \Psi_n(e_{1n})]^T,$$

$$\text{with } \Psi_i(e_{1i}) = \frac{\partial \rho(e_{1i})}{\partial e_{1i}}, (i = 1, 2, ..., n).$$

The standard loss function is a quadratic one (the 2–norm criterion) defined as

$$\rho(\boldsymbol{e}_1) = \frac{1}{2}||\boldsymbol{e}_1||^2 = \sum_{i=1}^{n} \rho_i(e_{1i}) = \frac{1}{2}\sum_{i=1}^{n} e_{1i}^2. \quad (5)$$

For this function the un-supervised learning rule (3) can be simplified to

$$\frac{d\boldsymbol{w}_1(t)}{dt} = \mu_1(t)\left[y_1(t)\boldsymbol{e}_1(t) + \boldsymbol{x}(t)\boldsymbol{w}_1^T(t)\boldsymbol{e}_1(t)\right] =$$

$$= \mu_1 y_1(\boldsymbol{x} - y_1\boldsymbol{w}_1) + \mu_1\boldsymbol{x}(1 - \boldsymbol{w}_1^T\boldsymbol{w}_1)y_1 =$$

$$= \mu_1 y_1(\boldsymbol{I} - \boldsymbol{w}_1\boldsymbol{w}_1^T)\boldsymbol{x} + \mu_1\boldsymbol{x}(1 - \boldsymbol{w}_1^T\boldsymbol{w}_1)y_1$$

Taking into account that $\boldsymbol{w}_1^T(t)\boldsymbol{w}_1(t) \to 1$ as $t \to \infty$ the second term tends quickly to zero (which has been confirmed by computer simulations) and it can be neglected without any influence on the final solution. Thus the learning rule can be simplified as follows:

$$\frac{d\boldsymbol{w}_1(t)}{dt} = \mu_1(t)y_1(t)\boldsymbol{e}_1(t) \quad (6)$$

$$= \mu_1(t)y_1(t)[\boldsymbol{x}(t) - y_1(t)\boldsymbol{w}_1(t)] \quad (7)$$

For discrete time data or a discrete time realization the learning algorithm in eq. (6) can be directly transformed to the well known *Oja's learning rule* [Oja, 1982]

$$\boldsymbol{w}_1(k+1) = \boldsymbol{w}_1(k) + \eta_1(k)y_1(k)[\boldsymbol{x}(k) - y_1(k)\boldsymbol{w}_1(k)], \quad (8)$$

where $\eta_1(k) > 0$ determines the learning rate.

2.2 Sanger's Generalized Hebbian Algorithm (GHA)

For many applications (e.g. image compression), it is important to extract or to estimate higher principal components in an exact and fast way. One of the first who proposed how to extend the Oja's rule for learning many principal components was Sanger [Sanger, 1989] (see Fig. 1). From the viewpoint of search for a general

FOR signal samples $\boldsymbol{x}(k)$: FROM $k = 1$ TO N
(1) set error signal $\boldsymbol{e}_0(k) = \boldsymbol{x}(k)$
FOR every neuron: FROM $j = 1$ TO m
(2) set $\boldsymbol{w}_j(0)$ randomly
(3) set η_j in accordance with input variance
FOR epoch index s: FROM $s = 1$ TO MAX_S
FOR input samples: FROM $k = 1$ TO N
(4) $y_j(k) = \boldsymbol{w}_j^T(k-1)\boldsymbol{x}(k)$
(5) $\boldsymbol{w}_j(k) = \boldsymbol{w}_j(k-1) +$ $+\eta_j y_j(k)[\boldsymbol{e}_{j-1}(k) - y_j(k)\boldsymbol{w}_j(k-1)]$
IF $\lvert\boldsymbol{w}_j(k-1) - \boldsymbol{w}_j(k)\rvert < \epsilon$
THEN (6) $\boldsymbol{w}_j = \boldsymbol{w}_j(k)$; GO TO STABLE
(7) decrease η_j exponentially
STABLE:
FOR input samples: FROM $k = 1$ TO N
(8) set $y_j(k) = \boldsymbol{w}_j^T\,\boldsymbol{x}(k)$
(9) set error $\boldsymbol{e}_j(k) = \boldsymbol{e}_{j-1}(k) - y_j(k)\,\boldsymbol{w}_j$

Figure 1: Sanger's GHA algorithm implementation.

method two main drawbacks of his algorithm exist: (1) the initialization of the learning rate η_j was done heuristically using a trial and error approach, and (2) the applied Gram–Schmidt orthogonalization usually tends to de-converge the calculations for higher principal components corresponding to small eigenvalues.

Several methods have been proposed to improve Sanger's GHA algorithm. Two of them are discussed below. Nearly all of them apply the GHA but with adaptive (adjustable) learning rate $\eta(k)$ in order to improve convergence speed and exactness.

2.3 Extensions

An extension of Sanger's algorithm, called **SAMHL**, was proposed in [Abbas and Fahmy, 1993]. Its main feature is a successive (sequential) application of the generalized Hebbian learning rule for each next higher principal component. The advantage of the **SAMHL** method over the Sanger's method is also the explicit calculation and adaptation of the learning rate. The parameter η_j is initialized according to the variance of the whole original input sequence σ_{all}^2 and to the set of eigenvalues λ_i, corresponding to the previous eigenvectors (step (3)):

$$\eta_j(0) = \frac{1}{\sigma_{all}^2 - \sum_{i=1}^{j-1}\lambda_i}\ ,\ \lambda_i = \frac{1}{N}\sum_{k=1}^{N} y_i(k)^2 \quad (9)$$

But the learning rate is adapted after some specified number of iteration steps q and not on–line. This is due to the fact, that the adaptation equation is relatively complex and requires the computation of current signal variances σ^2 and $\Delta\boldsymbol{w}(k)$ over the whole input set. The rule is based on a minimalization of a heuristic cost function and it has the following form (step (7)):

$$\eta_j(q_{k+1}) = \eta_j(q_k) + cN[1 - \eta_j(q_k)\sigma^2(q_k)] \times \quad (10)$$

$$\times \left[\sigma^2(q_k) + \frac{2}{N}\sum_{i=1}^{N} y_i(\boldsymbol{e}_j(q_k)\Delta\boldsymbol{w}_j(q_k))\right], \quad (11)$$

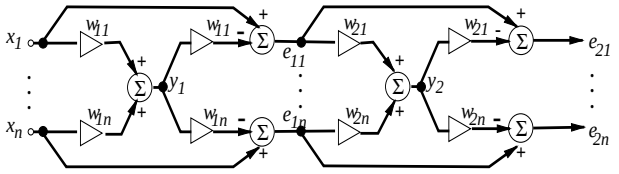

Figure 2: The architecture of a cascade neural network for PCA.

where c is a heuristic constant which is set to $c = 0.01/\lambda_{j-1}$, $(k = 1, 2, ...)$.

In [Bannour and Azimi-Sadjadi, 1995] the *recursive least square learning* (**RLS**) was proposed for Sanger's GHA algorithm. A dynamic Kalman gain K was introduced in the updating equation for the learning coefficient η_j. For this purpose an RLS algorithm was applied. The learning rate η_j is now modified for each new image sample according to current gain K (step 7):

$$K_j(k) = P_j(k-1)y_j(k)/[1 + P_j(k-1)y_j(k)^2] \quad (12)$$

$$P_j(k) = [1 - K_j(k)y_j(k)]P_j(k-1). \quad (13)$$

Step (5) takes now the following form:

$$\boldsymbol{w}_j(k) = \boldsymbol{w}_j(k-1) + K_j(\boldsymbol{e}_j(k) - \boldsymbol{w}_j(k-1)y_j(k))$$

Let us notice, that the adaptation of the learning rate (step 7) was now shifted to the iteration cycle and was placed between steps (4) and (5). But the authors applied the Gram–Schmidt orthogonalization (GHA Sanger's algorithm), which again prohibits a reliable extraction of all components (due to accumulation of errors).

3 The New CRLS Method

The proposed algorithm, called **CRLS** (cascade recursive least square) combines the advantages of both previous methods, **SAMHL** and **RLS**, and eliminates their drawbacks.

3.1 Cascade Neural Network

The task of principal component extraction can be easily accomplished by using a *cascade neural network* (see Fig. 2). Thus the learning algorithm for the estimation of the next largest principal component is performed in the same way as for the first component but we carry out the extraction process not directly from the input data (signals) $\boldsymbol{x}(t)$ but from the available errors (this procedure is called *deflation*). Depending on the number of required principal components m an appropriate number of m cascades can be stacked one after the other. Then, we have iteratively to supply the input of this network with the training data until all weight vectors $\boldsymbol{w}_j, (j = 1, ..., m)$ are stabilized.

3.2 The CRLS Algorithm

The algorithm of the **CRLS** (cascade recursive least square) method is given in Fig. 3. As a sequential application of the reduced input signal for the computation of the PC's y_j takes place, at the same time this algorithm can straightforward be mapped onto

FOR signal samples $\boldsymbol{x}(k)$: FROM $k = 1$ TO N
(1) set error signal $\boldsymbol{e}_0(k) = \boldsymbol{x}(k)$
FOR every neuron: FROM $j = 1$ TO m
(2) set $\boldsymbol{w}_j(0) = \mathbf{1}$
(3) set $\eta_j(0) = \sigma[\boldsymbol{e}_{j-1}]/N$, with $\sigma[\boldsymbol{e}] = \sum_t \sum_k e_{tk}^2$
FOR epoch index s: FROM $s = 1$ TO MAX_S
FOR input samples: FROM $k = 1$ TO N
(4) $y_j(k) = \boldsymbol{w}_j(k-1)\boldsymbol{e}_{j-1}(k)$
(7) $\eta_j(k) = y_j(k)^2 + \eta_j(k-1)$
(5) $\boldsymbol{w}_j(k) = \boldsymbol{w}_j(k-1) +$ $+[y_j(k)/\eta_j(k)][\boldsymbol{e}_{j-1}(k) - \boldsymbol{w}_j(k-1)y_j(k)]$
IF $\lvert\boldsymbol{w}_j(k-1) - \boldsymbol{w}_j(k)\rvert < \epsilon$
THEN (6) $\boldsymbol{w}_j = \boldsymbol{w}_j(k)$; GO TO STABLE
STABLE:
FOR input samples: FROM $k = 1$ TO N
(8) set $y_j(k) = \boldsymbol{w}_j\boldsymbol{e}_{j-1}(k)$
(9) set error $\boldsymbol{e}_j(k) = \boldsymbol{e}_{j-1}(k) - y_j(k)\boldsymbol{w}_j$

Figure 3: The new CRLS algorithm.

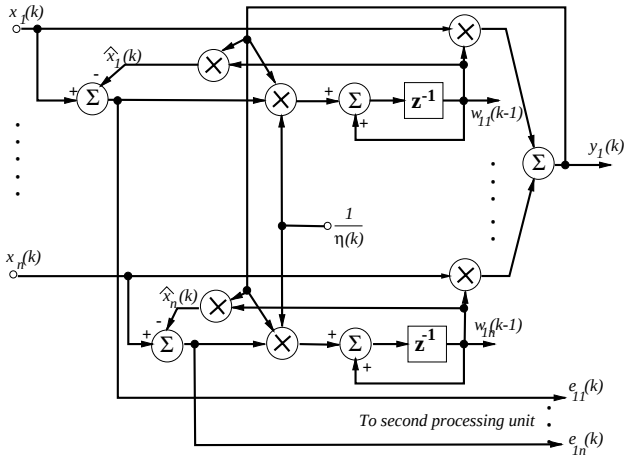

Figure 4: Functional block diagram illustrating the sequential extraction of PC's. The first processing unit is shown only.

a hardware realization that contains one processing block only, performing the calculations in an iterative manner (see Fig. 4).

3.3 Signal Reduction (Deflation)

For the calculation of the j-th eigenvalue, $\lambda_j = \mathrm{E}\{y_j^2\}$, the reduced input signal $\boldsymbol{E}_j = [\boldsymbol{e}_j(1), \boldsymbol{e}_j(2), ..., \boldsymbol{e}_j(N)]$ is applied (and not the original input signal) (step 4):

$$y_j(k) = \boldsymbol{w}_j(k-1)\boldsymbol{e}_j(k), \quad (14)$$

where $\boldsymbol{e}_j(k) = [e_{j1}(k), e_{j2}(k), ..., e_{jn}(k)]^T$, $(k = 1, 2, ..., N)$. This, at first view, slight modification has an important influence onto the orthogonality of the computed weights and consequently on their exactness. In contrast to other known algorithms the extraction of higher components is not tending to zero and the successive extractions are provided in a numerically stable way.

The parameter $\eta_1(0)$ is initialized according to the

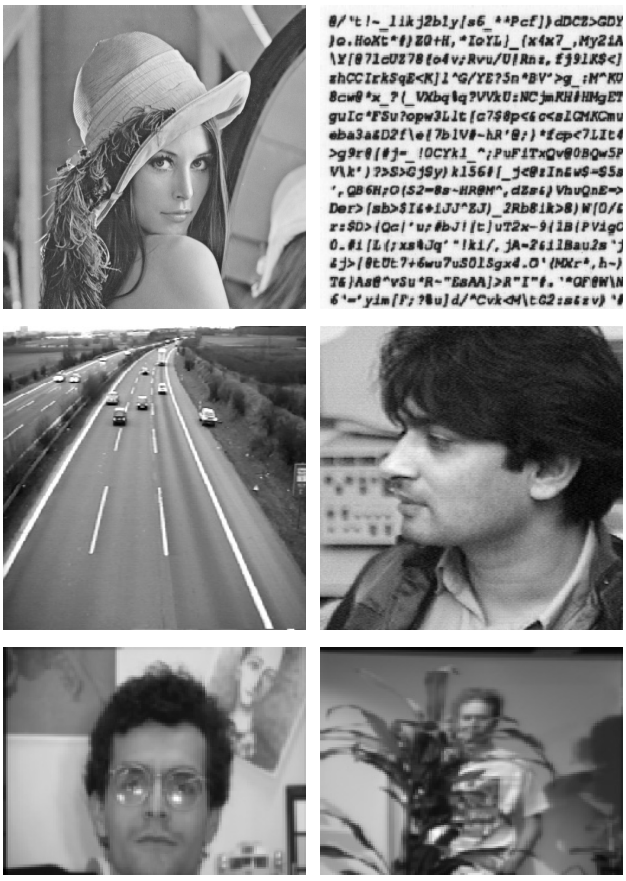

Figure 5: Images used for testing.

variance of the original input sequence $\sigma[\boldsymbol{x}]/N$ (step 3) and: $\eta_j(0) = \sigma[\boldsymbol{e}_{j-1}]/N = \eta_{j-1}(0) - \mathrm{E}[y_{j-1}^2]$.

3.4 RLS Adaptation of the Learning Rate

The learning rate is adaptively and optimal modified during a training process. We have transformed the Kalman form of the RLS model (12) to an equivalent and simple formula (step (7)). The equivalence between (12) of the RLS method and our step (7) of the CRLS method can be easily shown if one observes that $P_j(k) = (\eta_j(k))^{-1}$. This form of the learning rate ensures a considerable improvement in convergence speed. The developed algorithm needs one epoch for extraction of each component only.

4 Simulation Results

Several grey scale images (an 8 bit per pixel representation) have been used for tests of the PCA adaptive learning algorithms (Fig. 5). Their sizes ranges from 512 x 512 pixel for *Lenna* and the NIST subimage *Text* over 384 x 288 pixel for the traffic scene image *Road*, 350 x 400 for image *Ali* down to 128 x 100 pixel for the MIT Media Lab. images *Brad* and *Plant*. The evaluation of the PCA methods is performed according to the quality of image compression–reconstruction procedure, that is performed on the basis of extracted principal components.

PSNR = 24.46 dB PSNR = 28.12 dB

Figure 6: The reconstructed image *Lenna* after the 1st (left) and 3th (right) principal component.

4.1 Validity of the CRLS Method

In Fig. 6 some results of processing the well known image *Lenna* are provided. It can be observed visually, that already after the computation of three principal components the reconstructed image is of high quality. This quality is proved by the drawings in Fig. 8. The peak signal to noise ratio (PSNR) is shown for all 64 principal components for several sets of training data from the same image. No more than one epoch of the learning sample was required in order to fulfill the weight convergence condition $\Delta \boldsymbol{w}(k) < 10^{-5}$. It should be noted that the PSNR value is monotonically growing with the increasing number of extracted components, if a training set consists of more than 10 % of complete image.

This proves that one epoch is enough precisely to extract the components. Even the use of a part of training data (incomplete data case) still leads to a reliable extraction of the first 16 components.

4.2 Comparison of Methods

In order to compare our **CRLS** method with the **RLS** approach we have implemented the RLS–algorithm described in [Bannour and Azimi-Sadjadi, 1995]. The result of our simulation was rather surprising. CRLS and RLS converge both with similar speed and determine the first several principal components with similar quality, but RLS does it only up to a certain number of components. For example for the *Lenna* image the RLS algorithm after extraction of the first 10 components was not able to extract higher components (see Fig. 9). For the image *Plant* the situation was even worser – RLS has found four components only. In opposite to RLS our CRLS algorithm has monotonically increased the quality of reconstructed images as we add next components. The same conclusion can be drawn on the basis of the behavior of other performance factors, e.g. the signal to noise ratio (SNR) or normalized mean square error (NMSE).

4.3 Generalization Ability of PCA

One of the most important features of the PCA approach to data representation is its generalization power. We expect that the principal components (KL transform) obtained for one image I_1, when applied

PSNR = 33.41dB PSNR = 21.39dB PSNR = 29.73dB PSNR = 34.71dB

Figure 7: Image compression–reconstruction with components learned on *Lenna*: applying 8 principal components of *Lenna* for *Road* and *Text* compression–reconstruction, and 1 and 3 principal components of *Lenna* for *Ali* compression–reconstruction (from left to right, respectively).

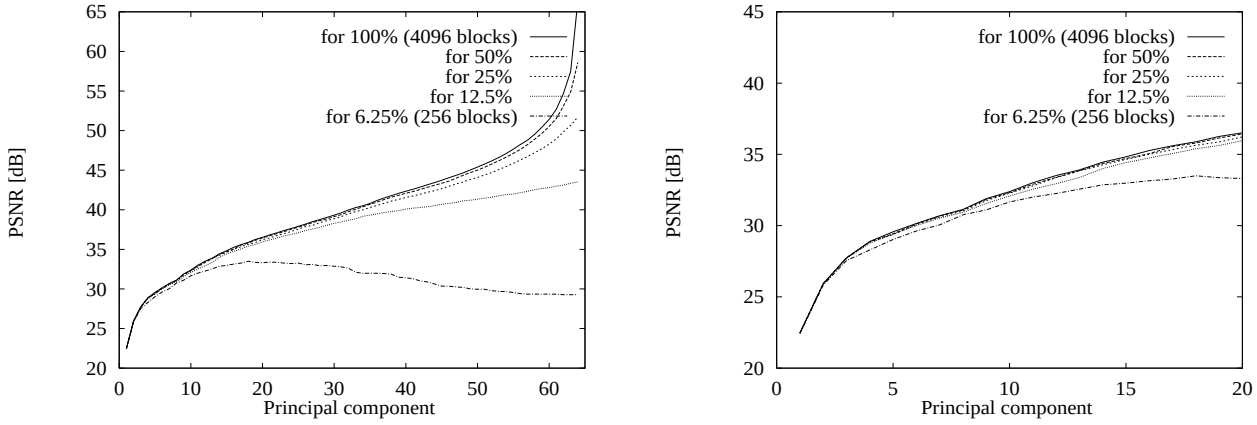

Figure 8: The PSNR of the reconstructed image depending on the part of the image used for training the **CRLS**–neural net. The algorithm achieves the convergence $\Delta \boldsymbol{w} < 10^{-5}$ already in one epoch (4096 blocks or 100% of the image). Even by using only 12.5% of the image pixels for training a very good quality of the reconstructed image was observed. The differences for the first 16 principal components (right drawing) are negligible.

to another image I_2 provide similar quality of reconstruction as the components (KL transform) obtained for the image I_2. In Fig. 10 we have illustrated the generalization power of PCA. The reconstructions of different images on the basis of PC's obtained from the image *Lenna* have been compared to reconstructions of these images on the basis of original image–own principal components. These experiments document that there is a certain difference between PSNR plots for both cases. However this disparity becomes significant for the components after the high quality limit, i.e. after the 16th component.

5 Conclusions

In this paper a new, fast and reliable, adaptive learning algorithm for the principal component analysis was proposed. The algorithm is useful for fast estimation of any number of eigenvectors and associated eigenvalues of a correlation matrix $\boldsymbol{R}_{xx} = E(\boldsymbol{x}\boldsymbol{x}^T)$ already from subsets of given zero–mean signals $\boldsymbol{x}(t)$ (so called *gappy* or incomplete data). The proposed algorithm can be considered as a kind of normalized self–adaptive Hebbian rule and as an modification and improvement of the RLS and SAMHL methods. The proposed neural networks attempt to find the minimum of a well–defined energy (loss, objective, cost) function. The architecture of the neural network is a cascade one, i.e. the first processing unit drives the second one which in turn drives the third one and so on. The neural network structure admits local self–adaptation of the synaptic weights.

References

[Abbas and Fahmy, 1993] H.M.Abbas and M.M.Fahmy. Neural model for Karhunen–Loéve transform with application to adaptive image compression. *IEE Proceedings–I*, 140(2):135–143, 1993.

[Amari, 1977] S. Amari. Neural theory of association and concept formation. *Biological Cybernetics*, 26:175–185, 1977.

[Baldi and Hornik, 1989] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58, 1989.

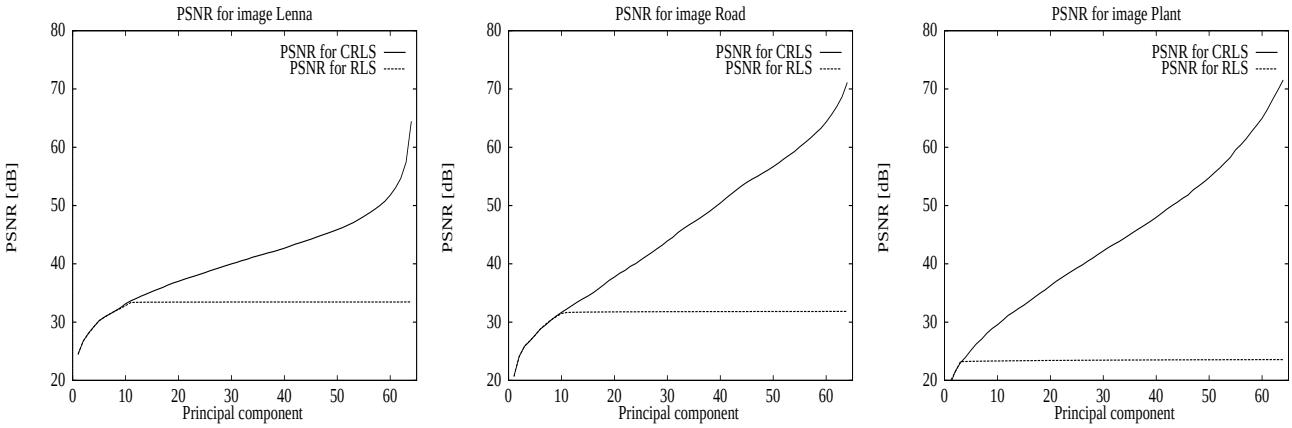

Figure 9: Comparison of image compression–reconstruction performances by using the CRLS and RLS algorithms. The plots show the achieved PSNR values of reconstructed images *Lenna*, *Road* and *Plant*.

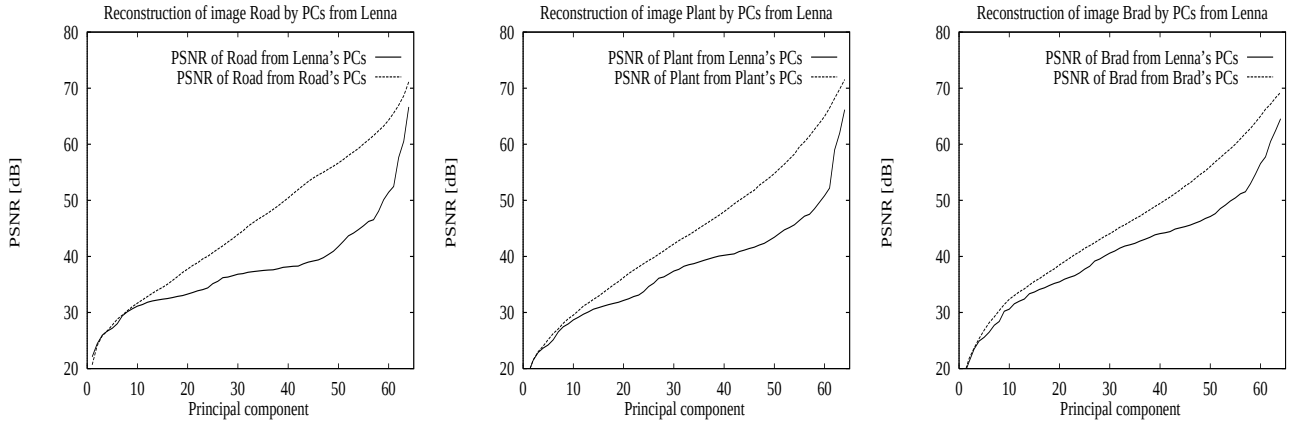

Figure 10: Performance factor PSNR for compression–reconstruction of test images by the use of principal components learned for the image *Lenna*. Comparison of these results with image compression–reconstruction performances by using the image–own principal components.

[Bannour and Azimi-Sadjadi, 1995] S. Bannour and R. Azimi-Sadjadi. Principal component extraction using recursive least squares learning. *IEEE Transactions on Neural Networks*, 6(2):457–469, 1995.

[Cichocki and Unbehauen, 1994] A. Cichocki and R. Unbehauen. *Neural Networks for Optimization and Signal Processing*. John Wiley & Sons, New York, 1994.

[Jolliffe, 1986] I.T. Jolliffe. *Principal Component Analysis*. Springer, New York, 1986.

[Karhunen and Joutsensalo, 1995] J. Karhunen and J. Joutsensalo. Generalization of principal component analysis, optimization problems, and neural networks. *Neural Networks*, 8(4):549–562, 1995.

[Kung *et al.*, 1991] S. Kung, K. Diamantaras, and J. Taur. Neural networks for extracting pure / constrained / oriented principal components. In: *SVD and Signal Processing*, pages 57–81. Elsevier Science, Amsterdam, 1991.

[Niemann and Wu, 1993] H. Niemann and J.-K. Wu. Neural network adaptive image coding. *IEEE Transactions on Neural Networks*, 4:615–627, 1993.

[Oja, 1982] E. Oja. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 16:267–273, 1982.

[Oja, 1992] E. Oja. Principal components, minor components and linear neural networks. *Neural Networks*, 5:927–935, 1992.

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

[Taylor and Coombes, 1993] J.G. Taylor and S. Coombes. Learning of higher order correlations. *Neural Networks*, 6:423–427, 1993.

[Turk and Pentland, 1991] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3:71–86, 1991.

[Xu *et al.*, 1992] L. Xu, E. Oja, and C.Y. Suen. Modified hebbian learning for curve and surface fitting. *Neural Networks*, 5:393–407, 1992.