

A Squared-Variance Principal Subspace Network

M. D. Plumbley†

† Department of Computer Science, King's College London
Strand, London WC2R 2LS, UK

1 Introduction

Over the last decade or so, a number of authors have investigated neural network algorithms which perform *principal component analysis* (PCA). Many of these are based on the Oja (1982) single-output principal component finder neuron, and find either the M largest principal components of their inputs, or the subspace spanned by those components. For background and references, see e.g. (Oja, 1989; Hornik & Kuan, 1992; Plumbley, 1994).

One feature that these algorithms share is that the variance of each selected principal component is preserved (or simply scaled) at the output of the network. In this paper, we consider an algorithm which will still select the principal subspace of the input, but whose output components will have variance which is the *square* of the variance of the corresponding input component.

2 Neural network PCA algorithms

Consider the network shown in Fig. 1. This is a linear network with the output \mathbf{y} given by

$$\mathbf{y}_t = \mathbf{W}_t \mathbf{x}_t. \quad (1)$$

The PCA algorithms mentioned in the introduction update the weight matrix \mathbf{W}_t at time t by an additional $\Delta \mathbf{W}_t$ at each presentation, according to the algorithm

$$\Delta \mathbf{W}_t = \eta (\mathbf{y}_t \mathbf{x}_t^T - \mathbf{K}_t \mathbf{W}_t) \quad (2)$$

where η is a small positive update factor, and \mathbf{K}_t is a function of \mathbf{x}_t and \mathbf{W}_t which varies from one algorithm to another. Table 1 lists the forms of \mathbf{K}_t used for some of these algorithms. For the algorithms in the table, \mathbf{K}_t or its expected value $\mathbf{K} = E(\mathbf{K}_t)$ satisfies

$$\mathbf{K}_t + \mathbf{K}_t^T = 2\mathbf{y}_t \mathbf{y}_t^T \quad \text{or} \quad \mathbf{K} + \mathbf{K}^T = 2\mathbf{W} \boldsymbol{\Sigma}_x \mathbf{W}^T \quad (3)$$

where $\boldsymbol{\Sigma}_x = E(\mathbf{x}\mathbf{x}^T)$ is the covariance matrix of \mathbf{x} (assuming \mathbf{x} has zero mean). Other PCA algorithms exist which do not satisfy this: these other algorithms will not concern us here.

In an earlier paper, the author showed that these algorithms converge to the principal subspace from almost everywhere, and that \mathbf{K} satisfying (3) produces outputs which preserve input component variance (Plumbley, 1994). Let us now consider a modification to \mathbf{K} which produces squared variance outputs.

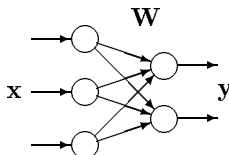


Figure 1: PCA network with input \mathbf{x} , weight matrix \mathbf{W} , and output $\mathbf{y} = \mathbf{W}\mathbf{x}$.

Algorithm	\mathbf{K}_t
Oja (1982) PCA neuron	y_t^2
Williams (1985) SEC	$\mathbf{y}_t \mathbf{y}_t^T$
Oja & Karhunen (1985) SGA	$\text{diag}(\mathbf{y}_t \mathbf{y}_t^T) + 2\text{LT}^+(\mathbf{y}_t \mathbf{y}_t^T)$

Table 1: Forms of \mathbf{K}_t used for some PCA algorithms. In this table, $\text{diag}(\cdot)$ sets off-diagonal entries to zero, while $\text{LT}^+(\cdot)$ sets entries on or above the diagonal to zero.

3 Squared-variance algorithm

Suppose that we now use algorithm (2), or strictly speaking its o.d.e. equivalent

$$d\mathbf{W}/dt = \mathbf{W}\Sigma_{\mathbf{x}} - \mathbf{K}\mathbf{W} \quad (4)$$

but with

$$\mathbf{K} = \mathbf{W}\mathbf{W}^T. \quad (5)$$

Then the following theorem concerning the behaviour of $\mathbf{W}\mathbf{W}^T$ holds.

Theorem 1 *Suppose that $\Sigma_{\mathbf{x}}$ is finite and nonsingular, and $\mathbf{W}\mathbf{W}^T$ is initially finite and nonsingular. Then $\mathbf{W}\mathbf{W}^T$ remains finite and nonsingular, and converges to satisfy $\mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T = \mathbf{I}$.*

Proof Consider the pair of cost functions

$$J_1 = \|\mathbf{I} - \mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T\|^2 = \text{Tr}((\mathbf{I} - \mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T)^2) \quad (6)$$

$$J_2 = \|\mathbf{I} - (\mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T)^{-1}\|^2 = \text{Tr}\left(\left(\mathbf{I} - (\mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T)^{-1}\right)^2\right). \quad (7)$$

Differentiating these, after substituting in (4) and a little manipulation we get

$$\begin{aligned} dJ_1/dt &= -4\text{Tr}((\mathbf{I} - \mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T)\mathbf{W}\mathbf{W}^T(\mathbf{I} - \mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T)) \\ &\leq 0 \end{aligned} \quad (8)$$

$$\begin{aligned} dJ_2/dt &= -\text{Tr}(\mathbf{W}^T(\mathbf{I} - (\mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T)^{-1})\mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T \\ &\quad (\mathbf{I} - (\mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T)^{-1})\mathbf{W}) \\ &\leq 0 \end{aligned} \quad (9)$$

so the norms of $\mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T$ and its inverse are bounded towards \mathbf{I} by their initial values, and consequently must remain finite and nonsingular. Since $\Sigma_{\mathbf{x}}$ is also finite and nonsingular, $\mathbf{W}\mathbf{W}^T$ must also remain finite and nonsingular.

Finally, since equality in (8) (or (9)) holds only when $\mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T - \mathbf{I} = \mathbf{0}$, J_1 (or J_2) is a Lyapunov function, and $\mathbf{W}\Sigma_{\mathbf{x}}^{-1}\mathbf{W}^T \rightarrow \mathbf{I}$ as $t \rightarrow \infty$. **QED.**

Furthermore, since we already know that $\mathbf{W}^T\mathbf{W}$ converges to span the principal subspace of $\Sigma_{\mathbf{x}}$ so that these two terms commute, we can show that $\mathbf{W}\Sigma_{\mathbf{x}}\mathbf{W}^T = \mathbf{W}\mathbf{W}^T\mathbf{W}\mathbf{W}^T$. Thus any non-zero component of $\mathbf{W}^T\mathbf{W}$ has eigenvalue identical to $\Sigma_{\mathbf{x}}$, thus producing a squared variance component at the output.

4 Implementation and simulation results

Implementation of this squared-variance algorithm is not quite as simple as some of the previous principal subspace algorithms such as the SEC algorithm (Williams, 1985). While the first term $\mathbf{y}\mathbf{x}^T$ can be calculated using purely local information to each weight, the second term $-\mathbf{K}\mathbf{W} = -\mathbf{W}\mathbf{W}^T\mathbf{W}$ is a little more awkward for a network relying on local calculations only.

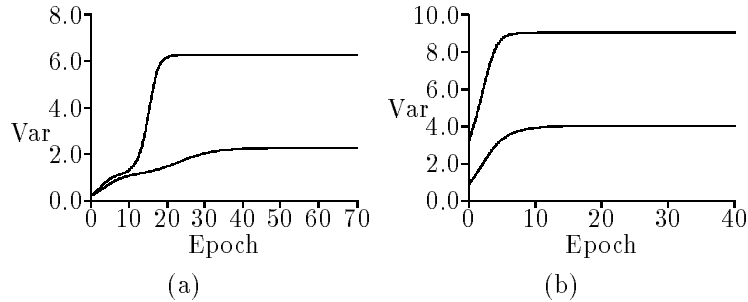


Figure 2: Simulation runs for (a) single-output network (b) two-output network.

The approach suggested here uses local calculations, but over a different phase (or a parallel information path) than that used to calculate the first term. We notice that we can write

$$\mathbf{W}\mathbf{W}^T\mathbf{W} = \mathbf{W}\mathbf{I}\mathbf{W}^T\mathbf{W} = E(\mathbf{y}_f\mathbf{x}_{fb}^T) \quad (10)$$

$$= \mathbf{W}\mathbf{W}^T\mathbf{I}\mathbf{W} = E(\mathbf{y}_{bf}\mathbf{x}_b^T) \quad (11)$$

where $\mathbf{y}_f = \mathbf{W}^T\mathbf{x}_f$ and $\mathbf{x}_{fb} = \mathbf{W}^T\mathbf{y}_f$ in (10), and $\mathbf{x}_b = \mathbf{W}^T\mathbf{y}_b$ and $\mathbf{y}_{bf} = \mathbf{W}\mathbf{x}_b$ in (11). In these expressions, \mathbf{x}_f or \mathbf{y}_b are simulated inputs (or outputs) which have $E(\mathbf{x}_f\mathbf{x}_f^T) = \mathbf{I}$, or $E(\mathbf{y}_b\mathbf{y}_b^T) = \mathbf{I}$. These could be simulated using uncorrelated equal-variance noise, or in a more practical system, by cycling through individual components using e.g. $\mathbf{y}_b(1) = [1, 0, 0, \dots, 0]$, $\mathbf{y}_b(2) = [0, 1, 0, \dots, 0]$, and so on. We denote (10) by the *forward-backward* form, with (11) as the *backward-forward* form.

In our simulations, we chose the backward-forward form (11). This is particularly simple in the single output case, since we can use a fixed $\mathbf{y}_b = 1$ for all t . It also involves a shorter number of componentwise cycles if the number of outputs is less than the number of inputs.

Fig. 2(a) shows two runs of a single-output network, with artificial data of (i) two inputs with component variances 1.5 and 1.0, and (ii) three inputs with component variances 2.5, 1.5 and 1.0. Both used $\eta = 0.1$. The respective output variances converge to (i) 2.25 and (ii) 6.25 as predicted. Fig. 2(b) shows a run of a two-output three-input network, with input component variances of 3.0, 2.0 and 1.0, using $\eta = 0.05$. The output component eigenvalues converge to 9.0 and 4.0 as predicted.

These simulations confirm that the output variances converge to the square of the input principal component variances. Other examination of the simulated networks confirm that the extracted subspace converges to the principal subspace.

5 Conclusions

We have presented a neural network algorithm which extracts the principal subspace of the input components, related to the class of algorithms derived from the Oja (1982) principal component finder. Theory and simulations show that this algorithm produces output components whose variances are the *square* of their corresponding input component variances. If a two-phase or double-path approach is taken, this algorithm can also be implemented using purely local calculations.

References

- Hornik, K. & Kuan, C.-M. (1992). Convergence analysis of local feature extraction algorithms. *Neural Networks*, **5**, 229–240.
- Oja, E. (1982). A simplified neuron model as a principal component analyser. *Journal of Mathematical Biology*, **15**, 267–273.

- Oja, E. (1989). Neural networks, principal components, and subspaces. *International Journal of Neural Systems*, **1**(1), 61–68.
- Oja, E. & Karhunen, J. (1985). On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of Mathematical Analysis and Applications*, **106**, 69–84.
- Plumbley, M. D. (1994). Lyapunov functions for convergence of principal component algorithms. *Neural Networks*. To appear.
- Williams, R. J. (1985). Feature discovery through error-correction learning. ICS Report 8501. University of California, San Diego.