USING RECURSIVE LEAST SQUARE LEARNING METHOD FOR PRINCIPAL AND MINOR COMPONENTS ANALYSIS

A. S. Y. Wong, K. W. Wong and C. S. Leung*
Dept. of EE, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong
*Dept. of CS, University of Wollongon, Northfields Avenue, Wollongon, NSW 2522, Australia

Abstract

In combining principal and minor components analysis, a parallel extraction method based on recursive least square algorithm is suggested to extract the principal components of the input vectors. After the extraction, the error covariance matrix obtained in the learning process is used to perform minor components analysis. The minor components found are then pruned so as to achieve a higher compression ratio. Simulation results show that both the convergent speed and the compression ratio are improved, which in turn indicate that our method effectively combines the extraction of the principal components and the pruning of the minor components.

1. Introduction

The development of principle components analysis (PCA) using neural networks has been grown from single-neuron to multi-neuron, from sequential extraction to parallel extraction, and from stationary process to non-stationary process. Oja first developed a learning rule [1] for a simple linear neuron. The rule is a linearized version of the normalized Hebbian learning rule. Sanger extended the rule to a multi-neuron network [2] for the extraction of the first n principal components of a stationary process. A major drawback of these methods is that the updating step size is fixed at a small value for accuracy of extraction, and so the speed of convergence is low. Moreover, the methods are suitable for stationary process only. In order to use an adaptive step size, Bannour and Azimi-Sadjadi introduced a recursive least square (RLS) method [3] that initially train the first principal component, then the second principal component, and finally the nth principal component. This RLS method estimates the best step size, i.e., the Kalman Gain for updating, thus the convergent speed is increased. However, the extraction of the principal components is still sequential. Leung *et al* generalised the RLS method for non-sequential extraction of principal components [4]. Their method is highly adaptive for non-stationary process. Although significant results have been obtained in the studies of PCA using RLS training method [3,5], their conjunction with minor components analysis (MCA) has not been discussed. In this paper, we combine the training of the network and the pruning of the minor components as a whole. In order to improve the convergence, a modified RLS algorithm is suggested for PCA. After the extraction of the principal components, the error covariance matrix, $P(k)$, obtained in the RLS learning procedures is used in the analysis of the minor components.

This paper is organised as follows. In Section 2, the modified RLS algorithm is introduced. The details of using RLS method for PCA and MCA are described in Section 3. Simulation results are reported in Section 4 to show the effectiveness of our algorithm; and finally, a conclusion is drawn in the last section.

2. The modified RLS-based Training Algorithm

Consider a fully connected two-layer neural network that operates in auto-association mode. If there are equal numbers of neurons in the input and output layers, and the number of hidden neurons is fewer than that in the input or output layer, then the bottleneck hidden layer will force the network to compress the input with fewer parameters. This means that the network will extract the principal components of the input. Owing to the hidden-layer representation, identical mapping from input to output is generally not achievable. The mean-squared-error in the reconstruction is what we wish to minimize.

Consider a general feedforward neural network with L layers (including the input and output layers) and there are N_l neurons in the lth layer. If the network is trained by the input vectors $a(t)$ with the desired output $d(t)$ set equal to $a(t)$, the operation of the network is characterised by,

$$a_{l+1,j}(t) = f\left(\sum_{i=1}^{N_l} w_{l,j,i}(t)a_{l,i}(t)\right), \quad (1)$$

$$e(t) = d(t) - a_L(t) \quad (2)$$

where $f(\cdot)$ is the sigmoid function $f(x)=\frac{1}{1+e^{-x}}$,

$a_{l,i}(t)$ is the output of the neuron $A_{l,i}$ (the ith neuron in the lth layer) in time t,
$w_{l,j,i}(t)$ is the weight connected from the ith neuron in the lth layer to the jth neuron in the (l+1)layer,
$e(t)$ is the error in the reconstruction, and
$a_L(t)$ is the output of the network at time t.

Iiguni and Sakai have developed a simplified RLS method [6] to estimate the weights of a multi-layer neural network. Their method only updates the weights that connect from all the neurons in the lth layer to the jth neuron in the (l+1)th layer in one iteration. Without deriving the equations again, the estimation of $w_{l,j}(t)$ is achieved by the following set of equations,

$$K_{l,j}(t)=P_{l,j}(t-1)H_{l,j}(t)\left[I+H_{l,j}(t)P_{l,j}(t-1)H_{l,j}(t)^T\right]^{-1} \quad (3)$$

$$w_{l,j}(t)=w_{l,j}(t-1)+K_{l,j}(t)\left[d(t)-a_L(t)\right] \quad (4)$$

$$P_{l,j}(t)=P_{l,j}(t-1)-K_{l,j}(t)H_{l,j}(t)P_{l,j}(t-1) \quad (5)$$

where $H_{l,j}(t)$ is defined as $\left.\frac{\partial a_L(t)}{\partial w_{l,j}(t)}\right|_{w=w(t-1)}$,

$P_{l,j}(t)$ is the error covariance matrix,
$K_{l,j}(t)$ is the Kalman gain, and
$w_{l,j}(t)$ is the column vector $\left[w_{l,j,1}(t)\ \ w_{l,j,2}(t)\ \ \cdots\ \ w_{l,j,N_l}(t)\right]^T$

In updating $w_{L\text{-}1}(t)$, ($\left[w_{L-1,1}(t)\ \ w_{L-1,2}(t)\ \ \cdots\ \ w_{L-1,N_l}(t)\right]^T$), if we take a closer look into the matrices $H_{L\text{-}1}$ and the $P_{L\text{-}1}$, they are in the form,

$$H_{L-1}=\left[H_{L-1,1}\ \ H_{L-1,2}\ \ \cdots\ \ H_{L-1,N_L}\right]$$

$$=\left[\begin{array}{cccc}\frac{\partial a_L(t)}{\partial w_{L-1,1}(t)} & 0 & \cdots & 0\\ 0 & \frac{\partial a_L(t)}{\partial w_{L-1,2}(t)} & & \vdots\\ \vdots & & \ddots & 0\\ 0 & \cdots & 0 & \frac{\partial a_L(t)}{\partial w_{L-1,N_L}(t)}\end{array}\right]_{w=w(t-1)} \quad (6)$$

$$P_{L-1}=\left[\begin{array}{cccc}P_{L-1,1} & 0 & \cdots & 0\\ 0 & P_{L-1,2} & & \vdots\\ \vdots & & \ddots & 0\\ 0 & \cdots & 0 & P_{L-1,N_L}\end{array}\right] \quad (7)$$

Since the off-diagonal elements in equations (6) and (7) are all zeroes, the estimation of $w_{L\text{-}1,j}(t+1)$ depends only on $a_{L,j}(t)$. The computational complexity of updating $P_{L\text{-}1}(t+1)$ and $w_{L\text{-}1}(t+1)$ is exactly the same as that of $\sum_{j=1}^{N_L} O\left(P_{L-1,j}(t+1)\right)$ and $\sum_{j=1}^{N_L} O\left(w_{L-1,j}(t+1)\right)$ respectively. Therefore, in order to achieve a faster convergence, we suggest that $w_{L\text{-}1}$ is updated simultaneously in one iteration rather than separately in N_L times. By using this modification, the weight vectors $w_{L\text{-}1}$ is updated locally in a layer level, and its estimation is more accurate than updated in a neuron level. Hence, without increasing the computational complexity, the convergent speed is improved. Moreover, this method extracts the principal components in a non-sequential way, i.e., the extraction of the ith component does not depend on the convergence of the (i-1)th component.

3. MCA with RLS Method

After the weight vectors have been converged to the optimal values or up to a desired degree of accuracy, the first n principal components are extracted in the hidden layer. Then the error covariance matrix obtained during the training process is used for minor component analysis. Based on the findings of Leung *et al* [7], the inverse of the error covariance matrix is approximately equal to the Hessian matrix of the network being trained. It can be used to measure the change in energy due to the changes in weights. After k training patterns are presented, the energy function that we are interested is,

$$E(w)=\frac{1}{2}\sum_{t=1}^{k}\mathrm{E}[e(t)]+\frac{1}{2}w^T P^{-1}(0)w \quad (8)$$

where $e(t) = d(t)\text{-}h(w(t), a(t))$, and $h(\cdot)$ is the function describing the network, and

$P(0)=\delta I_M$, with $\delta > 0$, and $M = \sum_{l=1}^{L-1} N_l N_{l+1}$

If $w = w_0$ is the optimum, then $\left.\frac{\partial E(w)}{\partial w}\right|_{w=w_0} = 0$

From [7], the measurement of the change in energy due to the changes in weights is,

$$\Delta E = \frac{1}{2}\Delta w^T P(k)^{-1}\Delta w \quad (9)$$

and the sensitivity of the energy with respect to the removal of each weight is,

$$\Delta E_i = \frac{1}{2} w_i(k)^2 q_{ii} \quad (10)$$

where q_{ii} is the ith diagonal element of the inverse of the matrix $P(k)$.

In MCA, the removal of some minor components is much more important than the pruning of some particular weights. If $a_{l'}(t)$ is the compressed representation of the input $a(t)$, then the energy change after the removal of $a_{l',j}(t)$ is,

$$\Delta E_{l',j} = \frac{1}{2}\times \left(\sum_{i=1}^{N_{l'-1}} w_{l'-1,j,i}(k)^2 q_{l'-1,j,ii}(k)+\sum_{y=1}^{N_{l'+1}} w_{l',y,j}(k)^2 q_{l',y,jj}(k)\right) \quad (11)$$

where $q_{l'-1,j,ii}$ and $q_{l',y,jj}$ are the ith and jth diagonal elements of $P_{l'-1,j}^{-1}$ and $P_{l',y}^{-1}$ respectively.

The first term in the r.h.s. of equation (11) is the energy change after the removal of $w_{l'-1,j}$ $(1 \le i \le N_{l'-1})$ while the second term is that for $w_{l',y,j}$ $(1 \le y \le N_{l'+1})$. The magnitude of $\Delta E_{l',j}$ reflects the importance of the extracted component, $a_{l',j}(t)$, that representing the input vector $a(t)$. If $\Delta E_{l',j}$ is small enough, then the effect of removing the neuron $\mathrm{A}_{l',j}$ can be ignored. This minor component is pruned away so as to achieve a higher compression ratio. For two-layer PCA networks, $l' = L-1$. So $w_{l'}$ is updated in layer level and its estimation is more accurate than that of $w_{l'-1}$. Thus, the second term in equation (11) shows the ranking of the components more precisely. We simplify the change of energy as $\Delta E'_{l',j} = \sum_{y=1}^{N_{l'+1}} w_{l',y,j}(k)^2 q_{l',y,jj}(k)$. Moreover, the computational complexity of $P_{l'}^{-1}$ is $O\left(N_{l'+1}N_{l'}^{\,2}\right)$ which is smaller than that of $P_{l'-1}^{-1}$, (which is $O\left(N_{l'}N_{l'-1}^{\,2}\right)$), as $N_{l'}$ is smaller than $N_{l'-1}$ and $N_{l'+1}$ for compression ratio greater than one. In short, our pruning algorithm for MCA is,

1) Sort $\Delta E'_{l',j}$ in ascending order as $\Delta E'_{l',s}$.

2) Prun away all $A_{l',s}$ for $1 \le s \le S$, where S is determined from $\sum_{s=1}^{S} \Delta E'_{l',s} < \lambda \sum_{j=1}^{N_{l'}} \Delta E'_{l',j}$, with $0 \le \lambda \le 1$.

In addition, the error covariance matrix is a block diagonal matrix with many zeroes. Therefore, the computation of the inverse of a huge matrix is avoided in the pruning procedures. This makes the combination of training and pruning more effectively. If bias terms are involved in the neural networks, the above derivation of the pruning algorithm is still valid because the bias vectors always carry heavy weights that can rarely be pruned.

4. Simulation Results

To show the performance of the proposed method, the test image Lenna with 512×512 pixels and 256 gray levels is used. The image is partitioned into a set of non-overlapping $b \times b$ blocks which are reshaped into vectors of size $b^2 \times 1$. The training vectors are then randomly fed into a neural network with b^2 inputs, N_2 hidden neurons and b^2 output neurons. The activation function used in the two layers are the nonlinear sigmoid function. Two bias vectors are also used for faster convergence. The pixel value is scaled to the range from 0 to 0.9 in order to avoid infinite growth of the weights. To start the training, the weights and the biases are randomly set between -0.5 and +0.5, and the error correlation matrix were initialised to $100*I_{M\mathrm{x}M}$. Where $M = \sum_{l=1}^{L-1}(N_l+1)N_{l+1}$ here.

The convergent speed of the proposed algorithm, (Layer-level update (LLU) method, which updates $w_{L\text{-}1}$ locally in layer level) is compared with that of the Neuron-level update (NLU) method (updates $w_{L\text{-}1}$ locally in neuron level). To train all the neurons in the network once, the LLU method requires $N_{L\text{-}1}$+1 iterations, while the NLU method needs $N_{L\text{-}1}$+N_L iterations; but the overall computation complexities are the same. Fig. 1 shows the signal-to-noise ratio (SNR=20log$_{10}$ Signal Power/Noise Power) of the image reconstruction after training through the network for a number of times. It shows that the reconstruction SNR of the LLU method is higher than that of the NLU method. Hence, it implies that the LLU method converges faster than the NLU method.

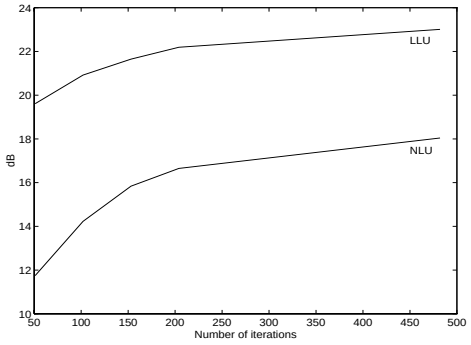
To show the effectiveness of our pruning algorithm, the network is trained with different input sizes and compression ratios. The number of iterations is fixed to $2\times$ sample size so that almost every sample has been trained twice. Table 1 summarises the reconstruction SNR with and without pruning. As observed from cases 1, 2 and 3, it is found that the reconstruction is better if the network is trained for more iterations. The reconstructed image after pruning in cases 2 and 3 are shown in Fig. 3 and 4 respectively. If the network is trained with higher initial compression ratios (cases 4 and 5), the reconstruction quality do not degrade too much. This shows that the training algorithm has fast convergent speed. As shown in the last two columns of Table 1, for whatever ways we compress the inputs, minor components are successfully pruned (loss in reconstruction is neglectable) to achieve a higher compression ratio.

In real-time learning, networks are usually only trained to an acceptable reconstruction accuracy with limited number of iterations and considerably smaller compression ratio. Therefore, weights are not converged properly and quite a number of the extracted components are not significant (as in case 3). In our approach, minor components found are successfully removed without much degrade in the reconstruction quality.

5. Conclusion

By using the fact that the derivative of the output of the ith output neuron with respect to the weights connected to the jth output neuron ($i \neq j$) is zero, a modified RLS method is proposed so that all the weights connected to the output layer are updated simultaneously in one iteration. With such modification, the convergent speed is improved without increase in computational complexity. After the network is trained, the error covariance matrix obtained in the RLS method is used for minor components analysis. Minor components found are then pruned to achieve a higher compression ratio. Simulation results show that we successfully combined the extraction of principal components and the pruning of minor components as a whole in an effective way.

Fig.1 Reconstruction SNR of the LLU and NLU methods



References

[1] Oja E., A simplified neuron model as a principal component analyzer. *Journal of Mathematics and Biology*. **15**, 267-273, 1982.

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

[3] Bannour S. and Azimi-Sadjadi M. R., Principal component extraction using recursive least squares learning. *IEEE Trans. Neural Networks*. **6**, 457-469, 1995.

[4] Leung C. S., Wong K.W. and Tsoi A. C., Recursive algorithms for principal component extraction. *Network: Computation in Neural Systems*, vol 8, **3**, 323-334, 1997.

[5] Kung S. Y. and Diamantaras K. I., A neural network learning algorithm for adaptive principal component extraction. *Proc. ICASSP*, 861-864, 1990.

[6] Iiguni Y. and Sakai H., A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter. *IEEE Trans. on Signal Processing*. **40**, 959-966, 1992.

[7] Leung C. S., Wong K. W., Sum J. and Chan L. W., On-line training and pruning for the recursive least square algorithms. *Electronics Letter*. **32**, 2152-2153, 1996.

Table 1. SNR of the reconstruction with and without pruning

input	N_2	iteration	SNR of the reconstruction with different numbers of minor components pruned /dB										loss in dB	increase in compression ratio
			0	1	2	3	4	5	6	7	8	9		
36	9	14450	24.69	24.66	--	--	--	--	--	--	--	--	0.03	12.5%
64	16	8192	23.77	23.77	23.71[1]	--	--	--	--	--	--	--	0.06	14.28%
100	25	5202	22.17	22.16	22.16	22.16	22.14	22.10	22.05	21.96	21.96	21.96[2]	0.21	56.25%
64	8	8192	22.21	22.21	21.06	--	--	--	--	--	--	--	0.15	33.33%
100	15	5202	22.26	22.22	22.12	--	--	--	--	--	--	--	0.14	15.38%

1,2 The results of the reconstruction are show in Fig. 3 and Fig. 4 respectively.
N_2 is the number of hidden neurons.

Fig. 2 The original Lenna image

Fig. 3 The reconstructed image with 23.71dB

Fig. 4 The reconstructed image with 21.96dB