TRAINING FEED-FORWARD NETWORKS WITH THE EXTENDED KALMAN ALGORITHM

Sharad Singhal and Lance Wu

Bell Communications Research, Inc.
Morristown, NJ 07960

ABSTRACT

A large fraction of recent work in artificial neural nets uses feed-forward networks trained with the back-propagation algorithm described by Rumelhart et. al. However, this algorithm converges slowly for large or complex problems such as speech recognition, where thousands of iterations may be needed for convergence even with small data sets.

In this paper, we show that training feed-forward networks is an identification problem for a nonlinear dynamic system which can be solved using the Extended Kalman Algorithm. Although computationally complex, the Kalman algorithm usually converges in a few iterations. We describe the algorithm and compare it with back-propagation using two-dimensional examples. In all cases examined, we find that the Kalman algorithm converges in fewer iterations than back-propagation and obtains solutions with fewer hidden nodes in the network.

INTRODUCTION

Artificial Neural Networks are an emerging field of research. Based on ideas from human cognition and biological systems, they are being applied to a multitude of problems including speech and image recognition. Feed-forward networks such as multilayer perceptrons are some of the most popular artificial neural net structures being used today. Continuous valued input vectors are presented to the networks and result in continuous valued output vectors. In multilayer perceptrons, the input and the output nodes are connected by one or more layers of interconnections using "hidden" nodes; the output of one layer being the input for the next layer. For feed-forward networks, interconnections are more general, with interconnections free to "skip" layers. Each node takes a weighted sum of its inputs and passes the result through a bounded nonlinearity to compute its output. In most applications, training data, consisting of matched input and output vectors, is applied to the network and the interconnection weights are adapted to minimize the mean squared error between the desired output and the output produced by the network. The so-called "back propagation" algorithm [1] is typically used to adapt the weights to the training data.

Although the back-propagation algorithm works well for small nets or simple problems, convergence is poor if the problem becomes complex or the number of nodes in the network become large [2]. This is because this algorithm is a LMS (Least Mean-Squared) gradient algorithm and ignores information from previous data. In problems such as speech recognition, tens of thousands of iterations may be required for convergence even with relatively small data-sets. Thus there is much interest [3,4] in other "training algorithms" which can compute the parameters faster than back-propagation and/or can handle much more complex problems.

In this paper, we show that training feed-forward networks can be viewed as an identification problem for a nonlinear dynamic system. For linear dynamic systems with white input and observation noise, the Kalman algorithm [5] is known to be an optimum algorithm. As opposed to gradient techniques, the Kalman algorithm computes the optimum value of the system parameters as each new data point is seen. Extended versions of the Kalman algorithm can be applied to nonlinear dynamic systems by linearizing the system around the current estimate of the parameters. Although it is computationally complex, this algorithm updates parameters consistent with all previously seen data and usually converges in a few iterations. In the following sections, we describe how this algorithm can be applied to feed-forward networks and compare its performance with back-propagation using some two-dimensional examples.

THE EXTENDED KALMAN FILTER

In this section we briefly outline the Extended Kalman filter. Mathematical derivations for the Extended Kalman filter are widely available in the literature [6-8] and beyond the scope of this paper.

Consider a nonlinear finite dimensional discrete time system of the form:

$$\begin{aligned} x(n+1) &= f_n(x(n)) + g_n(x(n))w(n), \\ d(n) &= h_n(x(n)) + v(n). \end{aligned} \tag{1}$$

Here the vector $x(n)$ is the state of the system at time n, $w(n)$ is the input, $d(n)$ is the observation, $v(n)$ is observation noise and $f_n(\cdot)$, $g_n(\cdot)$, and $h_n(\cdot)$ are nonlinear vector functions of the state with the subscript denoting possible dependence on time. We assume that the initial state, $x(0)$, and the sequences $\{v(n)\}$ and $\{w(n)\}$ are independent and gaussian with

$$\begin{aligned} &E[x(0)] = \bar{x}(0),\ E\{[x(0)-\bar{x}(0)][x(0)-\bar{x}(0)]^t\} = P(0), \\ &E[w(n)] = 0,\ E[w(n)w^t(l)] = Q(n)\delta_{nl}, \\ &E[v(n)] = 0,\ E[v(n)v^t(l)] = R(n)\delta_{nl}, \end{aligned} \tag{2}$$

where δ_{nl} is the Kronecker delta. Our problem is to find an estimate $\hat{x}(n+1)$ of $x(n+1)$ given $d(j)$, $0 \le j \le n$. We denote this estimate by $\hat{x}(n+1|n)$.

If the nonlinearities in (1) are sufficiently smooth, we can expand them using Taylor series about the state estimates $\hat{x}(n|n)$ and $\hat{x}(n|n-1)$ to obtain

$$\begin{aligned} f_n(x(n)) &= f_n(\hat{x}(n|n)) + F(n)[x(n)-\hat{x}(n|n)] + \cdots \\ g_n(x(n)) &= g_n(\hat{x}(n|n)) + \cdots = G(n) + \cdots \\ h_n(x(n)) &= h_n(\hat{x}(n|n-1)) + H^t(n)[x(n)-\hat{x}(n|n-1)] + \cdots \end{aligned}$$

where

$$G(n) = g_n(\hat{x}(n|n)),$$

$$F(n) = \left.\frac{\partial f_n(x)}{\partial x}\right|_{x=\hat{x}(n|n)},\quad H^t(n) = \left.\frac{\partial h_n(x)}{\partial x}\right|_{x=\hat{x}(n|n-1)}. \tag{3}$$

i.e. $G(n)$ is the value of the function $g_n(\cdot)$ at $\hat{x}(n|n)$ and the ijth components of $F(n)$ and $H^t(n)$ are the partial derivatives of the ith components of $f_n(\cdot)$ and $h_n(\cdot)$ respectively with respect to the jth component of $x(n)$ at the points indicated. Neglecting higher order terms and assuming knowledge of $\hat{x}(n|n)$ and $\hat{x}(n|n-1)$, the system in (3) can be approximated as

$$\begin{aligned} x(n+1) &= F(n)x(n) + G(n)w(n) + u(n) \quad n \ge 0 \\ z(n) &= H^t(n)x(n) + v(n) + y(n), \end{aligned} \tag{4}$$

where

$$\begin{aligned} u(n) &= f_n(\hat{x}(n|n)) - F(n)\hat{x}(n|n) \\ y(n) &= h_n(\hat{x}(n|n-1)) - H^t(n)\hat{x}(n|n-1). \end{aligned} \tag{5}$$

It can be shown [6] that the desired estimate $\hat{x}(n+1|n)$ can be obtained by the recursion

CH2673-2/89/0000-1187 $1.00 © 1989 IEEE

$$\hat{x}(n+1|n) = f_n(\hat{x}(n|n)) \quad (6)$$
$$\hat{x}(n|n) = \hat{x}(n|n-1) + K(n)[d(n) - h_n(\hat{x}(n|n-1))] \quad (7)$$
$$K(n) = P(n|n-1)H(n)[R(n)+H^t(n)P(n|n-1)H(n)]^{-1} \quad (8)$$
$$P(n+1|n) = F(n)P(n|n)F^t(n) + G(n)Q(n)G^t(n) \quad (9)$$
$$P(n|n) = P(n|n-1) - K(n)H^t(n)P(n|n-1) \quad (10)$$

with $P(1|0) = P(0)$. $K(n)$ is known as the Kalman gain. In case of a *linear* system, it can be shown that $P(n)$ is the conditional error covariance matrix associated with the state and the estimate $\hat{x}(n+1|n)$ is optimal in the sense that it approaches the conditional mean $E[x(n+1)|d(0)\cdots d(n)]$ for large n. However, for nonlinear systems, the filter is not optimal and the estimates can only loosely be termed conditional means.

TRAINING FEED-FORWARD NETWORKS

Training feed-forward networks can be considered a nonlinear estimation problem where the static weight values are unknowns that need to be estimated for the given set of input-output vectors. In this section we describe how feed-forward networks using smooth nonlinearities can be cast into a form suitable for recursive estimation using the extended Kalman algorithm. Without loss of generality, we will use multilayer perceptrons for illustration in this paper.

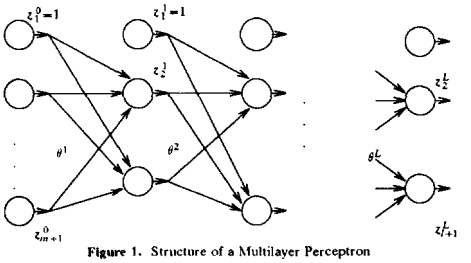

Figure 1. Structure of a Multilayer Perceptron

The network under consideration is shown in Figure 1. It is a L layer perceptron[1] with the ith input of the kth weight layer labeled as $z_i^{k-1}(n)$, the jth output being $z_j^k(n)$ and the weight connecting the ith input to the jth output being $\theta_{i,j}^k$. We assume that the net has m inputs and l outputs. Thresholds are implemented as weights connected from input nodes[2] with fixed unit strength inputs. Thus, if there are $N(k)$ nodes in the kth node layer, the total number of weights in the system is

$$M = \sum_{k=1}^{L} N(k-1)[N(k)-1]. \quad (11)$$

Although the inputs and outputs are dependent on time n, for notational brevity, we will not show this dependence unless explicitly needed.

In order to cast the problem in a form for recursive estimation, we let the weights in the network constitute the state x of the nonlinear system, i.e.

$$x = [\theta_{1,2}^1, \theta_{1,3}^1 \cdots \theta_{N(0),N(1)}^1]^t. \quad (12)$$

The vector x thus consists of all weights arranged in a linear array with dimension equal to the total number of weights M in the system. The system model thus is

$$x(n+1) = x(n) \quad n>0, \quad (13)$$
$$d(n) = z^L(n) + v(n) = h_n(x(n), z^0(n)) + v(n), \quad (14)$$

where $z^0(n)$ is the input vector corresponding to the desired output vector $d(n)$ in the training set and $z^L(n)$ is the corresponding output vector produced by the net. The components of $h_n(\cdot)$ define the nonlinear relationships between the inputs, weights and outputs of the net. If $\Gamma(\cdot)$ is the nonlinearity used, then $z^L(n) = h_n(x(n), z^0(n))$ is given by

$$z^L(n) = \Gamma\{(\theta^L)^t\Gamma\{(\theta^{L-1})^t\Gamma\cdots\Gamma\{(\theta^1)^t z^0(n)\}\cdots\}\}. \quad (15)$$

where Γ applies componentwise to vector arguments. Note that the input vectors appear only implicitly through the observation function $h_n(\cdot)$ in (14). The initial state (before training) $x(0)$ of the network is defined by populating the net with gaussian random variables with a $N(\bar{x}(0), P(0))$ distribution where $\bar{x}(0)$ and $P(0)$ reflect any apriori knowledge about the weights. In the absence of any such knowledge, a $N(0, 1/\epsilon\, I)$ distribution can be used, where ϵ is a small number and I is the identity matrix. For the system in (13) and (14), the extended Kalman filter recursion simplifies to

$$\hat{x}(n+1) = \hat{x}(n) + K(n)[d(n) - h_n(\hat{x}(n), z^0(n))] \quad (16)$$
$$K(n) = P(n)H(n)[R(n)+H^t(n)P(n)H(n)]^{-1} \quad (17)$$
$$P(n+1) = P(n) - K(n)H^t(n)P(n) \quad (18)$$

where $P(n)$ is the (approximate) conditional error covariance matrix.

Note that the inversion required in (17) has dimension equal to the number of outputs l, not the number of weights M, and thus does not grow as weights are added to the problem. The matrix updates require computation on the order of $O(M^2)$ whereas the delta rule used in back-propagation only requires $O(M)$ computation.

Usually, feed-forward nets use the sigmoid or the hyperbolic tangent functions as the nonlinearity $\Gamma(\cdot)$. The hyperbolic tangent function and its derivative are given by

$$z = \Gamma(x) = \tanh(x), \quad \partial z/\partial x = 1 - z^2. \quad (19)$$

Similarly the sigmoid and its derivative are

$$z = \Gamma(x) = \frac{1}{1+e^{-x}}, \quad \partial z/\partial x = z(1-z). \quad (20)$$

Note that the derivative is a function of the ordinate z alone in both cases and we will use the notation $\nabla_\Gamma(z)$ to represent it. For 3-layer perceptrons structured as shown in Fig. 1, elements of the gradient matrix H are given by

$$H_{j,i} = \partial z_i^L/\partial x_j \quad (21)$$

$$= \begin{cases} \nabla_\Gamma(z_i^L) z_k^{L-1} & \text{for } x_j = \theta_{k,i}^L \\ \nabla_\Gamma(z_i^L)\theta_{i,i}^N \nabla_\Gamma(z_i^{L-1}) z_k^{L-2} & \text{for } x_j = \theta_{k,\bar{i}}^{L-1} \\ \nabla_\Gamma(z_i^L)\nabla_\Gamma(z_i^{L-2}) z_k^{L-3} \sum_{p=2}^{N(L-1)} \theta_{k,p}^{L-2}\nabla_\Gamma(z_p^{L-1})\theta_{p,\bar{i}}^{L-1} & \text{for } x_j = \theta_{k,\bar{i}}^{L-2} \\ 0 & \text{otherwise} \end{cases}$$

where $L=3$ and the current estimate of θ is used for computation. For one- and 2-layer perceptrons, the first two expressions in (21) can be used to compute H. Similar expressions for for other feed-forward networks are easily derived. Equation (21) along with (16)-(18) forms the training algorithm.

EXAMPLES AND RESULTS

To evaluate the output and the convergence properties of the extended Kalman algorithm, we constructed mappings using two-dimensional inputs with two or four outputs as shown in Fig. 2. Limiting the input vector to 2 dimensions allows us to visualize the decision regions obtained by the net and to examine the outputs of any node in the net in a meaningful way. The x- and y-axes in Fig. 2 represent the two inputs, with the origin located at the center of the figures. The numbers in the figures represent the different output classes.

The training set for each example consisted of 1000 random vectors uniformly filling the region. The hyperbolic tangent nonlinearity was used as the nonlinear element in the networks.

1. We use the convention that the number of layers is equal to the number of weight layers. Thus we have L layers of *weights* labeled $1\cdots L$ and $L+1$ layers of *nodes* (including the input and output nodes) labeled $0\cdots L$. We will refer to the kth weight layer or the kth node layer unless the context is clear.

2. We adopt the convention that the 1st input node is the threshold, i.e. $\theta_{1,j}^k$ is the threshold for the jth output node from the kth weight layer.

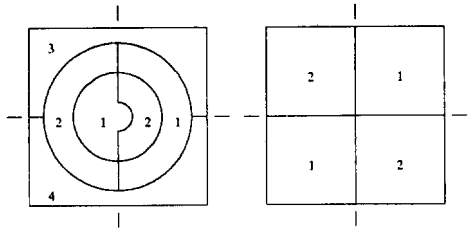

Figure 2. Output decision regions for two problems

The output corresponding to a class was set to 0.9 when the input vector belonged to that class, and to -0.9 otherwise. During training, the weights were adjusted after each data vector was presented. Up to 2000 sweeps through the input data were used with the stopping criteria described below to examine the convergence properties. The order in which data vectors were presented was randomized for each sweep through the data.

The Stopping Criteria: Training was considered complete if any one of the following conditions was satisfied:

a. 2000 sweeps through the input data were used,

b. the RMS (root mean squared) error at the output averaged over all training data during a sweep fell below a threshold t_1, or

c. the error reduction δ after the ith sweep through the data fell below a threshold t_2, where

$$\delta_i = \beta\delta_{i-1} + (1-\beta)\,|e_i - e_{i-1}|\,.$$

Here β is some positive constant less than unity, and e_i is the error defined in b.

In our simulations we set $\beta = 0.97$, $t_1 = 10^{-2}$ and $t_2 = 10^{-5}$.

Example 1 - Meshed, Disconnected Regions: Figure 2(a) shows the mapping with 2 disconnected, meshed regions surrounded by two regions that fill up the space. We used 3-layer perceptrons with 10 hidden nodes in each hidden layer to solve this problem.

We first set $R(n) = I$ in the Kalman recursions. This is equivalent to assuming that $v(n)$ is a zero mean unit variance gaussian process. The output error as a function of the number of sweeps is shown in Fig. 3(a) and the output decision regions obtained after training are shown in Fig. 4(a). From the error curve, the net appears to have converged. However, the output decision regions show that the solution is only partially correct. This behavior is caused by *data saturation*. Recall that the matrix P contains the algorithm's estimate of the error covariance matrix. As more and more terms are added to it, each new term is weighted less and less until the algorithm essentially ignores new data. To get around this problem, we need to give more weight to new data. This is done by weighting new data more heavily than old data using exponential windows. It can be shown [6] that this windowing can be included in the algorithm by replacing $R(n)$ in the recursion with $R(n)\cdot\alpha^{-n}$ for some $\alpha > 1$. Figure 3(b) shows the output error and Fig. 4(b) shows the decision regions formed when we set $R(k) = I\cdot e^{-k/50}$, where k is the number of sweeps through the training data. Within a sweep, R was held constant. Note that the net reaches a better solution in the second case. Similar results are obtained with other initial conditions (random starting weights) on the net.

Example 2 - 2 Input XOR: Figure 2(b) shows a generalized 2-input XOR with the first and third quadrants forming region 1 and the second and fourth quadrants forming region 2. We attempted the problem with two layer networks containing 2-4 nodes in the hidden layer. Figure 5 shows the results of training averaged over 10 different randomly chosen initial conditions. As the number of nodes in the hidden layer is increased, the net converges to smaller error values. When we examine the output decision regions, we find that none of the nets with 2 hidden nodes converges to the desired solution. With 3 hidden nodes, 6 of 10 initial conditions attempted led to the correct solution and with 4 hidden nodes, 9 of 10 initial conditions converged properly.

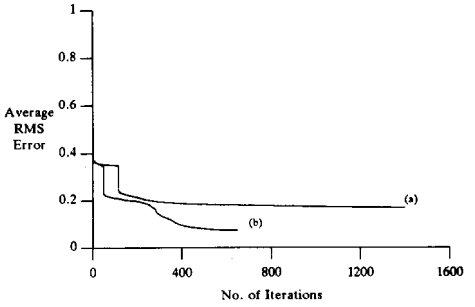

Figure 3. Error during training for (a) $R(k) = I$ and (b) $R(k) = I\cdot e^{-k/50}$

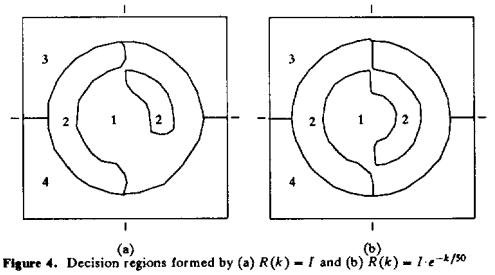

Figure 4. Decision regions formed by (a) $R(k) = I$ and (b) $R(k) = I\cdot e^{-k/50}$

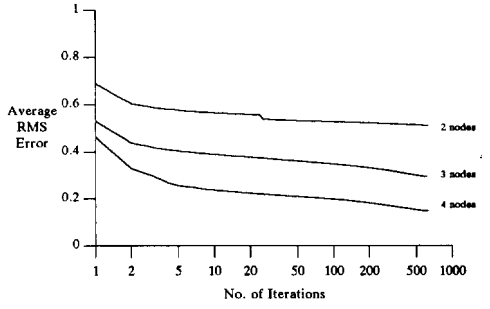

Figure 5. Average error during training for XOR problem with 2-4 hidden nodes

Comparison with Back-Propagation

To compare the results of the Kalman algorithm with back-propagation, we used back-propagation with a convergence constant of 0.1 and no "momentum" factor. In each case, back-propagation was used with the same initial conditions and network architecture as the Kalman algorithm. Figures 6 and 7 show the average RMS error during training for 10 different initial conditions. The number of sweeps through the data (x-axis) are plotted on a logarithmic scale to highlight the initial reduction for the Kalman algorithm.

Both the Kalman algorithm and back-propagation found similar solutions for the regions problem. However the Kalman algorithm was able to reduce the RMS error more than back-propagation and converged in fewer iterations. For the XOR problem, none of the nets converged to the desired solution when back-propagation was used. In almost all cases, back-propagation led to one of the solutions shown in Fig. 8. Recall that the Kalman algorithm found the correct solution 6 out of 10 times with 3 nodes and 9 out of 10 times with 4 hidden nodes. Also, convergence (or lack thereof) was usually apparent within

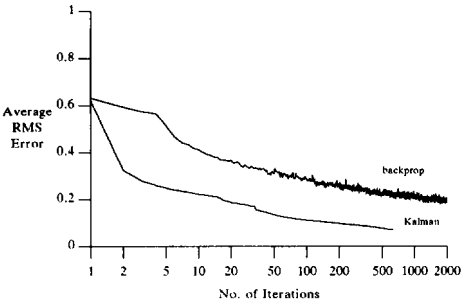

Figure 6. Average error during training for REGIONS using the Kalman algorithm and backprop

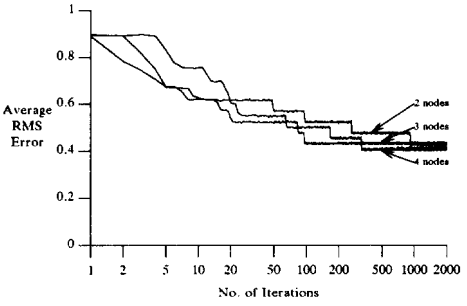

Figure 7. Average error during training for XOR problem with 2-4 hidden nodes using backprop

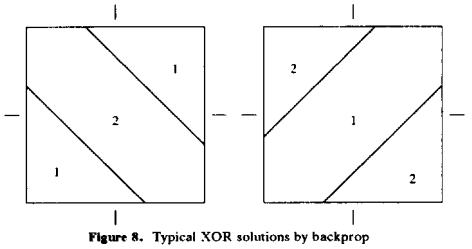

Figure 8. Typical XOR solutions by backprop

10-20 iterations through the data with the Kalman algorithm.

In all cases, the Kalman algorithm converged in fewer iterations than back-propagation.Also, in all but one case, the average RMS error after convergence was larger for back-propagation as compared to the Kalman algorithm. Back-propagation did achieve a lower error for the XOR with 2 hidden nodes; however, this case cannot be given much significance since neither algorithm achieved the desired solution. Table 1 summarizes the results of the comparison.

Table 1. Comparison of the Kalman Algorithm and Back-propagation for Regions and XOR Problems.

Problem	Kalman Algorithm		Back-Propagation	
	No. of iterations at termination	Avg. RMS Error	No. of iterations at termination	Avg. RMS Error
Regions	634	0.07	2000	0.21
XOR (2)	640	0.54	2000	0.43
XOR (3)	645	0.33	2000	0.43
XOR (4)	640	0.17	2000	0.41

CONCLUSIONS

In this paper, we showed that training feed-forward nets can be viewed as a system identification problem for a nonlinear dynamic system. For linear dynamic systems, the Kalman filter is known to produce an optimal estimator. Extended versions of the Kalman algorithm can be used to train feed-forward networks. We examined the performance of the Kalman algorithm using artificially constructed examples with two inputs and found that the algorithm typically converges in a few iterations. We also used back-propagation on the same examples and found that invariably, the Kalman algorithm converged in fewer iterations. For the XOR problem, back-propagation failed to converge on any of the cases considered while the Kalman algorithm was able to find solutions with the same network configurations.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagation," in D. E. Rumelhart and J. L. McCelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol 1: Foundations*. MIT Press, 1986.

[2] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K. Lang "Phoneme Recognition Using Time-Delay Neural Networks," *ATR internal Report* TR-I-0006, October 30, 1987.

[3] R. W. Prager and F. Fallside, "The Modified Kanerva Model for Automatic Speech Recognition," in *1988 IEEE Workshop on Speech Recognition*, Arden House, Harriman NY, May 31-June 3, 1988.

[4] B. Irie, and S. Miyake, "Capabilities of Three-layered Perceptrons," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, June 1988, Vol. I, pp. 641-648.

[5] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *J. Basic Eng., Trans. ASME*, Series D, Vol 82, No.1, 1960, pp. 35-45.

[6] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*, Prentice Hall, 1979.

[7] C. K. Chui and G. Chen, *Kalman Filtering*, Springer-Verlag, 1987.

[8] A. Gelb, Ed., *Applied Optimal Estimation*, MIT Press, 1974.