H_∞-Learning of Layered Neural Networks

Kiyoshi Nishiyama and Kiyohiko Suzuki

***Abstract*—Although the backpropagation (BP) scheme is widely used as a learning algorithm for multilayered neural networks, the learning speed of the BP algorithm to obtain acceptable errors is unsatisfactory in spite of some improvements such as introduction of a momentum factor and an adaptive learning rate in the weight adjustment. To solve this problem, a fast learning algorithm based on the extended Kalman filter (EKF) is presented and fortunately its computational complexity has been reduced by some simplifications. In general, however, the Kalman filtering algorithm is well known to be sensitive to the nature of noises which is generally assumed to be Gaussian. In addition, the H_∞ theory suggests that the maximum energy gain of the Kalman algorithm from disturbances (initial state, system, and observation noises) to the estimation error has no upper bound. That is, the Kalman filtering algorithm has a poor robustness to such disturbances. Therefore, the EKF-based learning algorithms should be further improved to enhance the robustness to variations in the initial values of link weights and thresholds as well as to the nature of noises. The aim of this paper is to propose H_∞-learning as a novel learning rule and to derive new globally and locally optimized learning algorithms based on H_∞-learning. Their learning behavior is analyzed from various points of view using computer simulations. The derived algorithms are also compared, in performance and computational cost, with the conventional BP and EKF learning algorithms.**

***Index Terms*—Backpropagation, H_∞ filter, H_∞-learning, Kalman filter, learning algorithm, neural network, robust estimation.**

I. Introduction

THE backpropagation (BP) scheme has been extensively used as a basic learning algorithm for training multilayered feedforward neural networks in many fields, including function approximation, pattern recognition and learning control for robotic manipulators [1]–[5]. The conventional BP algorithm iteratively adjusts the link weights and thresholds in the network using the steepest descent technique so as to minimize the difference between the actual network output and the desired output. However, because the learning rate is fixed, the convergence speed is inherently slow when a sufficiently trained network is desired. Although the convergence of the conventional BP algorithm has been somewhat improved by introducing a momentum factor into the weight adjustment, there still remains a need to carefully tune it in a heuristic manner for rapid learning and suppression of learning oscillation. Some methods with a variable learning rate have been examined for accelerating the convergence speed, which reveal the difficulties in scheduling the learning rate [6]–[8].

Manuscript received July 8, 1999; revised June 1, 2000 and June 5, 2001.

The authors are with the Department of Computer and Information Science, Faculty of Engineering, Iwate University, Morioka 020–8551, Japan (e-mail: nisiyama@cis.iwate-u.ac.jp).

Publisher Item Identifier S 1045-9227(01)09520-0.

As more powerful learning algorithms than the BP algorithms, some fast learning algorithms based on the extended Kalman filter (EKF) [9], [10] have been proposed [11]–[16] and statistically analyzed [17]. In these, some useful heuristics, including the decoupling, forgetting factor and teacher forcing, are used to improve their performance and computational cost. The motivation for the formulation of EKF learning stems from the fact that since multilayered neural networks are multiinput and multioutput nonlinear systems with a known output function, the learning in networks can be regarded as a parameter estimation for such nonlinear systems. In particular, a globally optimized EKF (g-EKF) learning algorithm, which is a straightforward implementation of the EKF, exhibits excellent performance at the expense of a large increase in computational time and storage. Indeed, although the g-EKF algorithm exhibits extremely fast learning as a function of the number of presentations of training data set, the computation time per instance scales as the square of the number of weights and thresholds in the network, thus rendering the algorithm impractical for many real-world problems.

A number of researchers have investigated simplifications of the g-EKF algorithm for reducing its computational cost [12]–[14]. The most popular simplification was achieved by ignoring the interdependencies of mutually exclusive groups of weights, which leads to lower computational complexity per training instance. This approach, called the decoupled EKF algorithm in [12], exhibited faster training both in terms of the number of presentations of training data and in total training time on a serial processor than a standard implementation of the BP for problems in pattern classification and function approximation. As another simplification, the covariance matrix of observation noise is treated as a diagonal matrix to avoid matrix inversion in the Kalman gain computation [13], [14]. Such simplifications allow the EKF-based learning algorithms to be applied more effectively to real-world problems [18]–[21].

In general, however, the Kalman filtering algorithm is well known to be strictly optimal only for Gaussian distribution of noises, while the H_∞ filtering algorithm makes no assumptions about the noise distribution. In addition, the H_∞ theory suggests that the maximum energy (worst case) gain of the Kalman algorithm from disturbances (initial state, system, and observation noises) to the estimation error has no upper bound because the H_∞ algorithm when $\gamma_f \rightarrow \infty$ is formally identical to the Kalman algorithm [23]–[25]. This leads to the conclusion that the Kalman algorithm suffers from poor robustness to such disturbances. Therefore, the EKF-based learning algorithms need to be further improved to enhance the robustness to variations in the initial values of link weights and thresholds as well as to the nature (i.e., distribution) of noises.

In this paper, to overcome the difficulties in the BP and EKF algorithms, a new fast and robust learning algorithm for multilayered neural networks is first derived by means of a worst-case optimization. This algorithm is called a globally optimized EHF (g-EHF) learning algorithm, which is derived by applying the H_∞ filter [23]–[25] to a linearized model of entire neural network with state-space representation. A simplification of the g-EHF algorithm is also given for single-output networks. Furthermore, a locally optimized EHF (l-EHF) learning algorithm is developed by creating disjoint subsets of weights to be decoupled, which is suitable for real-time learning in large-scale networks, especially multioutput networks. Several computer simulations demonstrate that the proposed EHF algorithms are more robust (i.e., less sensitive) than the EKF algorithms to variations in the initial values of link weights and thresholds under some conditions. Robustness is further explored for deterministic disturbances in observations.

This paper is organized as follows. In Section II, a globally optimized EHF (g-EHF) learning algorithm is derived as a fast and robust learning algorithm and its relationship with the corresponding g-EKF algorithm is analyzed. Section III provides a locally optimized EHF (l-EHF) learning algorithm through a simplified implementation of the g-EHF algorithm based on the so-called decoupling technique and some other ideas. In Section IV, we demonstrate the performance of the EHF algorithms and show that they are superior to the BP and the EKF algorithms using several computer simulations. Section V follows with some conclusions.

II. H_∞-Learning Algorithms

A fast and robust learning algorithm for training multilayered neural networks is derived by applying the H_∞ filter, which is equivalent to the Kalman filter in Krein space [24], [25], to the linearized neural-network model with state-space representation. This is called an extended H_∞ filter (EHF)-based learning algorithm in this paper. Before derivation of the EHF learning algorithms, some fundamentals are given in brief as necessary background.

A. Background

In this study, the artificial neural network to be trained is a multilayered feedforward network as shown in Fig. 1. The network consists of three layers of which the first, the second, and the third denote the input, the hidden, and the output layers, respectively, i.e., a three-layered network with one hidden layer. It is assumed that the input, the hidden, and the output layers have N_1, N_2, and N_3 neurons, respectively; this is termed an N_1-N_2-N_3 network.

Here, $z_i^n[p]$ represents the input to the ith neuron in the nth layer when the pth pair of input and output of a mapping is presented to the layered network; for instance, $z_1^3[p]$ is the output of the first neuron in the output layer as illustrated in Fig. 1. Also, $w_{i,j}^n$ denotes the connection strength (link weight) from the ith neuron in the nth layer to the jth neuron in the $n+1$th layer and θ_i^n is a threshold of the ith neuron in the $n+1$th layer.

The problem of adjusting the weights and thresholds in the network to produce the desired mapping using input–output pairs, which is called a learning problem of neural networks, can be regarded as an estimation problem of unknown constant parameters (link weights and thresholds) in a multiinput–multioutput (MIMO) nonlinear system with a known output function. Hence, the H_∞ filter or the Kalman filter, which is well known as a powerful state estimator in dynamical systems, can be utilized as a learning algorithm for training multilayered neural networks.

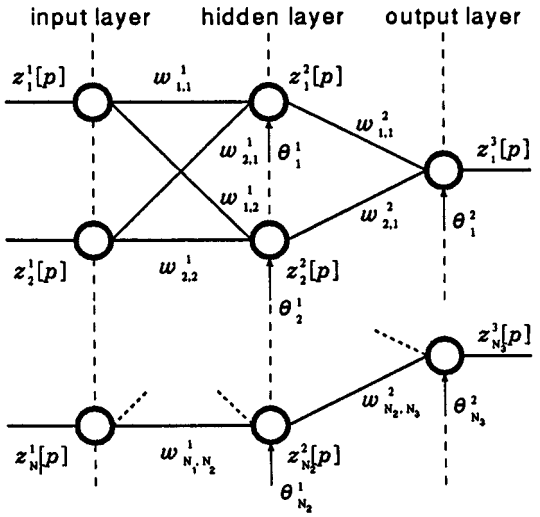

Fig. 1. Three-layered network with one hidden layer (an N_1-N_2-N_3 network).

As seen in Fig. 1, the operation of the ith neuron in the output layer is characterized by

$$z_i^3[p] = f\left(\sum_{j=1}^{N_2} w_{j,i}^2 f\left(\sum_{n=1}^{N_1} w_{n,j}^1 z_n^1[p] + \theta_j^1\right) + \theta_i^2\right) \quad i = 1, \ldots, N_3 \tag{1}$$

for $p = 1, \ldots, N_p$, where p denotes the index corresponding to an example ($\{z_n^1[p]\}$, $\{z_i^3[p]\}$), N_p the number of input–output pairs and an output function $f(x)$ is given by

$$f(x) = \frac{1}{1+\exp(-\alpha x)} \text{ or } f(x) = \frac{1-\exp(-\alpha x)}{1+\exp(-\alpha x)} \tag{2}$$

which is called a sigmoid function with gradient α.

As the first step to obtain the state-space representation of three-layered neural networks, defining a weight vector as

$$\begin{aligned} \boldsymbol{w} = &[\theta_1^1, w_{1,1}^1, w_{2,1}^1, \ldots, \theta_2^1, w_{1,2}^1 \\ &w_{2,2}^1, \ldots, \theta_1^2, w_{1,1}^2, w_{2,1}^2, \ldots]^T \in \mathcal{R}^{N_w} \\ N_w = &N_1 \times N_2 + N_2 \times N_3 + N_2 + N_3 \end{aligned} \tag{3}$$

and treating the right-hand side in (1) as a vector-valued nonlinear time-variant function $\boldsymbol{h}_p(\boldsymbol{w})$ of $\boldsymbol{w}$, we can rewrite (1) as

$$\boldsymbol{z}^3[p] = \boldsymbol{h}_p(\boldsymbol{w}) \in \mathcal{R}^{N_3} \tag{4}$$

where

$$\begin{aligned} \boldsymbol{z}^3[p] &= \left[z_1^3[p], \ldots, z_{N_3}^3[p]\right]^T \\ \boldsymbol{h}_p(\boldsymbol{w}) &= \left[h_{p,1}(\boldsymbol{w}), \ldots, h_{p,N_3}(\boldsymbol{w})\right]^T. \end{aligned} \tag{5}$$

Note that a similar model is obtained for networks with more than three layers.

Next, employing the trainable weight vector as a stationary state vector ($\boldsymbol{w}_k = \boldsymbol{w}$) at time step k in an unforced nonlinear dynamical system and assuming an observation noise $\boldsymbol{v}_k$ with zero mean, we achieve a nonlinear state-space model of the layered neural network

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k \quad \text{(state equation)} \tag{6}$$

$$\boldsymbol{y}_k = \boldsymbol{h}_k(\boldsymbol{w}_k) + \boldsymbol{v}_k \quad \text{(observation equation)} \tag{7}$$

where a pair of the input $\boldsymbol{z}^1[k]$ and the output $\boldsymbol{z}^3[k]$ is cyclically presented to the layered network in the order such as

$$\left(\boldsymbol{z}^1[p], \boldsymbol{z}^3[p]\right) = \left(\boldsymbol{z}^1\left[p + lN_p\right], \boldsymbol{z}^3\left[p + lN_p\right]\right)$$
$$p = 1, 2, \ldots, N_p$$

for learning iteration $l \geq 0$, where time step $k = p + lN_p$. Note that $\boldsymbol{y}_k = \boldsymbol{z}^3[k]$ when $\boldsymbol{v}_k$ is negligible. However, the ordinary H_∞ filter as well as the Kalman filter is still not directly applicable to the state-space model of neural networks because $\boldsymbol{h}_k(\boldsymbol{w}_k)$ is nonlinear with respect to $\boldsymbol{w}_k$. To overcome this, expanding the vector-valued function $\boldsymbol{h}_k(\boldsymbol{w}_k)$ into a Taylor series around the previous estimate $\hat{\boldsymbol{w}}_{k-1}$ of $\boldsymbol{w}_k$ and neglecting terms higher than the first order for linearization, we can obtain the following linear state-space model of the layered neural network:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k, \quad \boldsymbol{m}_k = \boldsymbol{H}_k \boldsymbol{w}_k + \boldsymbol{v}_k \tag{8}$$

where the N_3-dimensional observation vector $\boldsymbol{m}_k$ is redefined as

$$\boldsymbol{m}_k = \boldsymbol{y}_k - \boldsymbol{h}_k(\hat{\boldsymbol{w}}_{k-1}) + \boldsymbol{H}_k \hat{\boldsymbol{w}}_{k-1}$$
$$\boldsymbol{H}_k = \left.\frac{\partial \boldsymbol{h}_k(\boldsymbol{w})}{\partial \boldsymbol{w}}\right|_{w=\hat{w}_{k-1}} \in \mathcal{R}^{N_3 \times N_w}. \tag{9}$$

Note that the nonlinearity in the observation can be treated as if it were linearity perturbed by additive noise, so that $\boldsymbol{v}_k$ in (8) includes the residual in the Taylor expansion of $\boldsymbol{h}_k$. Also, an artificial system noise $\boldsymbol{u}_k$ with the covariance $\sigma_u^2 \boldsymbol{I}$, which is reported to work effectively in the EKF algorithm [12], [24], is often added into the state-space model such as $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \boldsymbol{u}_k$, although it will not appear in the exact modeling. The unknown quantities $\boldsymbol{w}_0$, $\boldsymbol{u}_k$, $\boldsymbol{v}_k$, which are assumed to be random vectors in the Kalman filter, are regarded as deterministic disturbances in the H_∞ filter and no assumptions about their distributional nature are made.

B. A Globally Optimized EHF Learning Algorithm

In the linearized state-space model, we would like to optimally estimate a linear combination of the weight vector $\boldsymbol{w}_k$, say $\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{w}_k$, using the observations $\{\boldsymbol{y}_i\}_{i=0}^k$.

Let $\check{\boldsymbol{z}}_{k|k} = \mathcal{F}_f(\boldsymbol{y}_0, \ldots, \boldsymbol{y}_k)$ denote the estimate of $\boldsymbol{z}_k$ given observations $\{\boldsymbol{y}_i\}_{i=0}^k$ from time t_0 to t_k. Then focusing on the following estimation error:

$$\boldsymbol{e}_{f,k} = \check{\boldsymbol{z}}_{k|k} - \boldsymbol{z}_k = \check{\boldsymbol{z}}_{k|k} - \boldsymbol{H}_k \boldsymbol{w}_k \tag{10}$$

we can define the transfer operator $\boldsymbol{T}_k(\mathcal{F}_f)$ that maps the unknown disturbances ($\boldsymbol{\Sigma}_0^{-1/2}(\boldsymbol{w}_0 - \check{\boldsymbol{w}}_0)$ and $\{\boldsymbol{v}_i\}_{i=0}^k$) to the filtered error ($\{\boldsymbol{e}_{f,i}\}_{i=0}^k$), where $\boldsymbol{\Sigma}_0$ is a positive-definite matrix that reflects *a priori* knowledge of how close $\boldsymbol{w}_0$ is to the initial guess $\check{\boldsymbol{w}}_0$.

Our objective is to choose a functional $\mathcal{F}_f$ so as to minimize the H_∞ norm of the transfer operator $\boldsymbol{T}_k(\mathcal{F}_f)$. This problem is called an optimal H_∞-learning problem in this paper.

Unfortunately, since the optimal H_∞-learning problem can not be solved in general, we settle for a solution of the following finite-time suboptimal problem.

[**Suboptimal H_∞-Learning Problem**]: For a given scalar $\gamma_f > 0$, find a (finite-time) suboptimal H_∞-learning strategy $\check{\boldsymbol{z}}_{k|k} = \mathcal{F}_f(\boldsymbol{y}_0, \ldots, \boldsymbol{y}_k)$ that achieves $\|\boldsymbol{T}_k(\mathcal{F}_f)\|_\infty < \gamma_f$. In other words, find a strategy that achieves

$$\|\boldsymbol{T}_k(\mathcal{F}_f)\|_\infty^2 = \sup_{w_0, \{v_i\}} \frac{\sum_{i=0}^k \|\boldsymbol{e}_{f,i}\|^2}{\|\boldsymbol{w}_0 - \check{\boldsymbol{w}}_0\|_{\boldsymbol{\Sigma}_0^{-1}}^2 + \sum_{i=0}^k \|\boldsymbol{v}_i\|^2} < {\gamma_f}^2. \tag{11}$$

This clearly requires investigating whether $\gamma_f > \gamma_{f,\mathrm{op}}$, where $\gamma_{f,\mathrm{op}} = \inf_{\mathcal{F}_f} \|\boldsymbol{T}_k(\mathcal{F}_f)\|_\infty$.

The quantity $\|\boldsymbol{T}_k(\mathcal{F}_f)\|_\infty^2$ can be interpreted as being the maximum energy gain from disturbances ($\boldsymbol{\Sigma}_0^{-1/2}(\boldsymbol{w}_0 - \check{\boldsymbol{w}}_0)$ and $\{\boldsymbol{v}_i\}_{i=0}^k$) to estimation error ($\{\boldsymbol{e}_{f,i}\}_{i=0}^k$). Hence, the maximum energy gain of the H_∞-learning is bounded over all possible disturbances, providing a worst-case learning. This fact brings out the robustness of the H_∞-learning.

The solution (the optimal H_∞-learning) to the optimal H_∞-learning problem could be obtained to the desired accuracy by means of the so-called γ-iteration, which is a well-known technique in H_∞ theory. Therefore, a solution to the suboptimal H_∞-learning problem is merely called an extended H_∞ filter (EHF)-based learning algorithm hereafter.

Based on the criterion (11) discussed above, we can derive a fast and robust learning algorithm to achieve $\|\boldsymbol{T}_k(\mathcal{F}_f)\|_\infty < \gamma_f$ for a prescribed γ_f, which leads to minimizing

$$J_{\mathrm{HF}} = \|\boldsymbol{T}_k(\mathcal{F}_f)\|_\infty^2 \tag{12}$$

with a possible decrease in γ_f from a relatively large positive value, checking the existence condition explained later.

Consequently, by applying the H_∞ filter [25] to the linearized state-space model of (8), we can derive a globally optimized EHF (g-EHF) learning algorithm for training multilayered feedforward neural networks as

$$\hat{\boldsymbol{w}}_{k+1} = \hat{\boldsymbol{w}}_k + \boldsymbol{K}_{s,k+1}\left(\boldsymbol{y}_{k+1} - \boldsymbol{h}_{k+1}(\hat{\boldsymbol{w}}_k)\right)$$
$$\boldsymbol{K}_{s,k+1} = \hat{\boldsymbol{P}}_{k+1|k}\boldsymbol{H}_{k+1}^T\left(\boldsymbol{H}_{k+1}\hat{\boldsymbol{P}}_{k+1|k}\boldsymbol{H}_{k+1}^T + \boldsymbol{I}\right)^{-1}$$
$$\hat{\boldsymbol{P}}_{k+1|k} = \hat{\boldsymbol{P}}_{k|k-1} - \hat{\boldsymbol{P}}_{k|k-1}\left[\boldsymbol{H}_k^T \quad \boldsymbol{H}_k^T\right] \cdot \boldsymbol{R}_{e,k}^{-1}\begin{bmatrix}\boldsymbol{H}_k \\ \boldsymbol{H}_k\end{bmatrix}\hat{\boldsymbol{P}}_{k|k-1}. \tag{13}$$

Here, the $2N_3 \times 2N_3$ matrix $\boldsymbol{R}_{e,k}$ is expressed by

$$\boldsymbol{R}_{e,k} = \boldsymbol{R}_k + \begin{bmatrix}\boldsymbol{H}_k \\ \boldsymbol{H}_k\end{bmatrix}\hat{\boldsymbol{P}}_{k|k-1}\left[\boldsymbol{H}_k^T \quad \boldsymbol{H}_k^T\right]$$
$$\boldsymbol{R}_k = \begin{bmatrix}\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\gamma_f^2 \boldsymbol{I}\end{bmatrix} \tag{14}$$

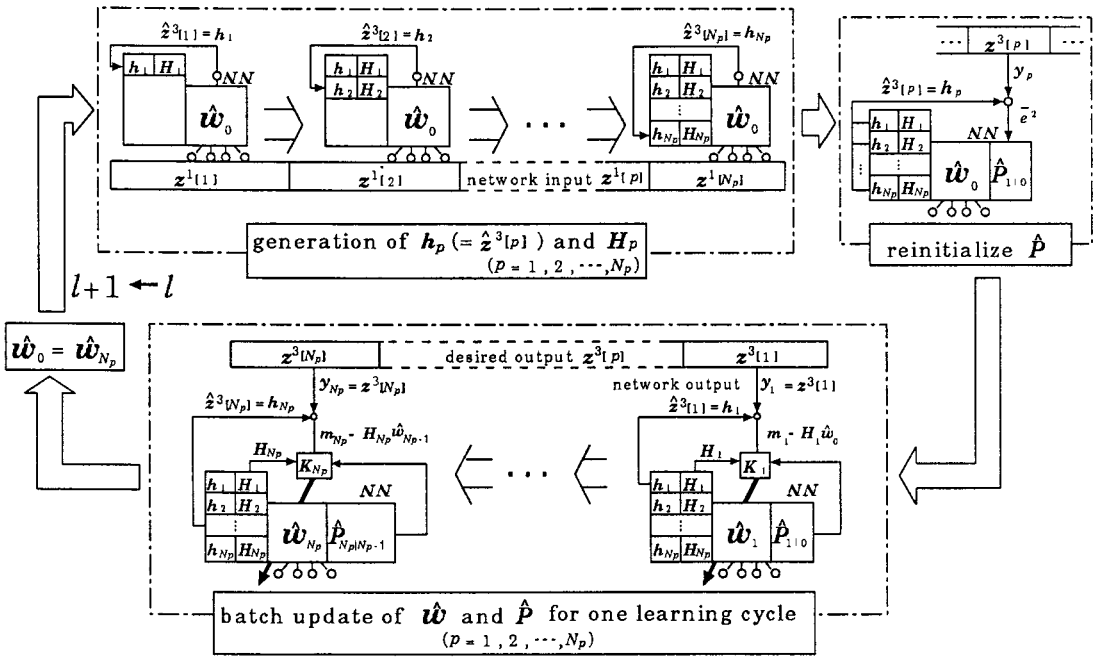

Fig. 2. Implementation of the g-EHF learning algorithm; the weight vector $\hat{w}_k$ is randomly initialized over a relatively small range and then is updated as $\hat{w}_k$, $k = p + lN_p, p = 1, 2, \ldots, N_p$ in the lth learning epoch.

and the parameter γ_f must be tuned so as to satisfy

$$\hat{\boldsymbol{P}}_{i|i}^{-1} = \hat{\boldsymbol{P}}_{i|i-1}^{-1} + \boldsymbol{H}_i^T \boldsymbol{H}_i - \gamma_f^{-2} \boldsymbol{H}_i^T \boldsymbol{H}_i > 0, \quad i = 0, \ldots, k \tag{15}$$

for the g-EHF algorithm to exist, i.e., to achieve (11). The requirement for existence is equivalent to the condition that the matrices $\boldsymbol{R}_k$ and $\boldsymbol{R}_{e,k}$ have the same inertia [24], [25], by which we mean the number of its positive, negative, and zero eigenvalues, namely

$$\begin{aligned} \boldsymbol{I} + \boldsymbol{H}_k \hat{\boldsymbol{P}}_{k|k-1} \boldsymbol{H}_k^T &> 0 \\ -\gamma_f^2 \boldsymbol{I} + \boldsymbol{H}_k \left(\hat{\boldsymbol{P}}_{k|k-1}^{-1} + \boldsymbol{H}_k^T \boldsymbol{H}_k \right)^{-1} \boldsymbol{H}_k^T &< 0. \end{aligned} \tag{16}$$

Here, it should be noted that the existence of the g-EHF algorithm depends on the value of γ_f; for instance, the g-EHF always exists when $\gamma_f = \infty$ [see (15)]. If the existence condition is unsatisfied, the optimality of the g-EHF algorithm is no longer guaranteed, leading to a drastic degradation in performance. The parameter γ_f should also be chosen to be as small as possible to maximize the robustness (i.e., minimize the sensitivity) of the g-EHF algorithm to variations in the initial weights and the nature of noises. Consequently, the determination of γ_f requires a tradeoff between robustness and existence.

The network's trainable weights and thresholds ($\hat{\boldsymbol{w}}_k$) are randomly initialized over a relatively small range for successful convergence and then are updated as $\hat{\boldsymbol{w}}_k$, $k = p + lN_p$, $p = 1, 2, \ldots, N_p$ in the lth learning epoch. The $N_w \times N_w$ matrix $\hat{\boldsymbol{P}}_{k|k-1}$ is also updated in the Riccati recursion, which corresponds to the error covariance matrix $\hat{\boldsymbol{\Sigma}}_{k|k-1}$ of $\boldsymbol{w}_k$ in Krein space divided by the variance σ_v^2 of the observation noise element, i.e., $\hat{\boldsymbol{P}}_{k|k-1} = \hat{\boldsymbol{\Sigma}}_{k|k-1}/\sigma_v^2$. In addition, to accelerate the convergence even when the output squared error is small, $\hat{\boldsymbol{P}}_{k|k-1}$ is reset at the beginning of the $(l+1)$th learning epoch as

$$\hat{\boldsymbol{P}}_{k|k-1} = \begin{cases} \frac{1}{\bar{e}_l^2} \boldsymbol{I}, & \bar{e}_l^2 < \frac{1}{\epsilon} \\ \epsilon \boldsymbol{I}, & \bar{e}_l^2 \geq \frac{1}{\epsilon} \end{cases}, \quad k = 1 + lN_p \tag{17}$$

using the average $\bar{e}_l^2$ of the squared errors in the output layer in the previous epoch. Here, the averaged squared error $\bar{e}_l^2$ is defined as

$$\bar{e}_l^2 = \frac{1}{N_p N_3} \sum_{p=1}^{N_p} \sum_{i=1}^{N_3} \left(z_i^3 [p + lN_p] - \hat{z}_i^3 [p + lN_p] \right)^2 \tag{18}$$

the parameter $\epsilon > 0$ in (17) is a certain positive constant ($\epsilon = 10$ is used throughout simulation), $\boldsymbol{I} = \text{diag}\{1, \ldots, 1\}$ is the $N_w \times N_w$ identity matrix and diag$\{\cdot\}$ denotes the diagonal matrix. The reinitialization is based on the assumption that each element of vector-valued observation noise $\boldsymbol{v}_k$ is stationary and mutually uncorrelated, say $\boldsymbol{\Sigma}_v = \text{diag}\{\sigma_v^2, \ldots, \sigma_v^2\}$ and the variance σ_v^2 corresponds to the average squared error $\bar{e}_l^2$. Then, if the numerator and denominator in the filter gain as well as both sides of the Riccati equation are divided by σ_v^2, the measurement error covariance can be expressed by the identity matrix, which represents the inverse of the learning rate. Under the assumption, the matrix $\hat{\boldsymbol{P}}_{k|k-1}$ is reset to $1/\bar{e}_l^2 \boldsymbol{I}$ at the beginning of the $l+1$th epoch, where $\hat{\boldsymbol{\Sigma}}_{k|k-1} = \boldsymbol{I}$ is assumed for $k = 1 + lN_P$, $l = 0, 1, \ldots, L$. However, as seen in (17), the reinitialized value $1/\bar{e}_l^2$ for the diagonal elements of $\hat{\boldsymbol{P}}_{k|k-1}$ is replaced by a relatively large positive number ϵ when $\bar{e}_l^2 \geq 1/\epsilon$, since the value of $1/\bar{e}_l^2$ in the epochs with large output error is relatively small so that the convergence becomes very slow.

Fig. 2 illustrates a straightforward implementation of the g-EHF learning algorithm. The g-EHF algorithm requires, in addition to the estimate of the weight vector, the storing and the

updating of $\hat{\boldsymbol{P}}_{k|k-1}$, which is used to model the correlations or interactions between each pair of weights in the network.

For each learning iteration, the average squared error in the output layer is calculated and dynamic derivatives of each component of $\boldsymbol{h}_p$ are formed with respect to the weights and thresholds in the network, which are evaluated at the current weight estimate $\hat{\boldsymbol{w}}_{lN_p}$. These derivatives are arranged into the matrix $\boldsymbol{H}_p$.

C. Relationship Between g-EHF and g-EKF Learning Algorithms

As seen in (13), the formula of the g-EHF learning algorithm is very similar to that of the g-EKF learning algorithm. However, for the g-EHF algorithm to be practical, the additional existence condition (15) or (16) must be satisfied. The indefinite matrix $\boldsymbol{R}_k = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\gamma_f^2 \boldsymbol{I} \end{bmatrix}$ also appears in the Riccati recursion, whereas it does not appear in the g-EKF algorithm.

The relationship between the g-EHF and g-EKF learning algorithms can be clarified by increasing γ_f to ∞. Indeed, in the limit, the inverse of $\boldsymbol{R}_{e,k}$ reduces to

$$\boldsymbol{R}_{e,k}^{-1} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} - \begin{bmatrix} \boldsymbol{H}_k \\ \boldsymbol{0} \end{bmatrix} \left(\boldsymbol{H}_k^T \boldsymbol{H}_k + \hat{\boldsymbol{P}}_{k|k-1}{}^{-1} \right)^{-1} \cdot [\boldsymbol{H}_k^T \quad \boldsymbol{0}] \tag{19}$$

because, using the matrix inversion lemma, it can be rewritten as

$$\begin{aligned} \boldsymbol{R}_{e,k}^{-1} = & \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\gamma_f^{-2} \boldsymbol{I} \end{bmatrix} - \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\gamma_f^{-2} \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{H}_k \\ \boldsymbol{H}_k \end{bmatrix} \\ & \cdot \left(\boldsymbol{H}_k^T \boldsymbol{H}_k - \gamma_f{}^{-2} \boldsymbol{H}_k^T \boldsymbol{H}_k + \hat{\boldsymbol{P}}_{k|k-1}^{-1} \right)^{-1} \\ & \cdot [\boldsymbol{H}_k^T \quad \boldsymbol{H}_k^T] \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & -\gamma_f^{-2} \boldsymbol{I} \end{bmatrix}. \end{aligned}$$

Substituting (19) into (13) provides the g-EHF algorithm when $\gamma_f \to \infty$ and it is presented by

$$\begin{aligned} \hat{\boldsymbol{w}}_{k+1} = & \hat{\boldsymbol{w}}_k + \boldsymbol{K}_{s,k+1} \left(\boldsymbol{y}_{k+1} - \boldsymbol{h}_{k+1}(\hat{\boldsymbol{w}}_k) \right) \\ \boldsymbol{K}_{s,k} = & \hat{\boldsymbol{P}}_{k|k-1} \boldsymbol{H}_k^T \left(\boldsymbol{H}_k \hat{\boldsymbol{P}}_{k|k-1} \boldsymbol{H}_k^T + \boldsymbol{I} \right)^{-1} \\ \hat{\boldsymbol{P}}_{k+1|k} = & \hat{\boldsymbol{P}}_{k|k-1} - \boldsymbol{K}_{s,k} \boldsymbol{H}_k \hat{\boldsymbol{P}}_{k|k-1}. \end{aligned} \tag{20}$$

This reduced version completely agrees with the g-EKF learning algorithm, which recursively provides the quasi-optimal estimate $\hat{\boldsymbol{w}}_{k|k} = E\{\boldsymbol{w}_k | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_k\}$ of $\boldsymbol{w}_k$ using the observations $\{\boldsymbol{y}_i\}_{i=0}^k$ [9], [10], where $\boldsymbol{w}_0$ and $\boldsymbol{v}_k$ are regarded as random vectors and $E\{\cdot\}$ denotes the expectation. Note that although the g-EHF algorithm is formally identical to the g-EKF algorithm when $\gamma_f \to \infty$, the criteria (or cost functions) used for derivation of the two algorithms are different.

The asymptotical agreement means that the maximum energy gain of the g-EKF algorithm is not upper bounded. In other words, the g-EKF algorithm may have quite a large H_∞ norm. Hence, it is clear that the g-EKF algorithm has a higher sensitivity to variations in the initial weights ($\boldsymbol{w}_0$) and observation noises ($\{\boldsymbol{v}_i\}_{i=0}^k$) and may, therefore, have extremely slow convergence. This is why we believe that the H_∞-learning leads to greater robustness to variations in weight initialization or to deterministic disturbance in observation.

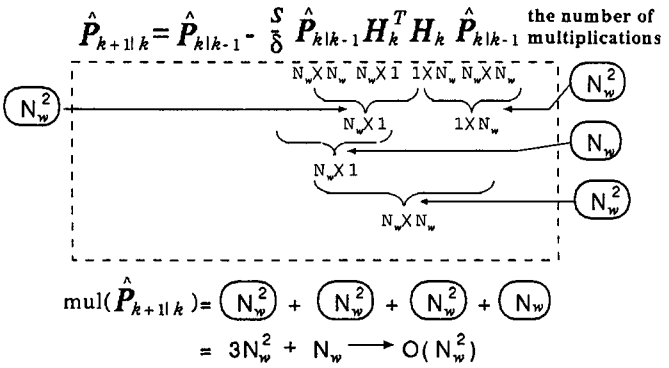

Fig. 3. Computational complexity per time step for the Riccati recursion in the reduced version.

This statement will be confirmed by several computer simulations in Section IV.

D. Reduction of Computational Burden

Roughly speaking, in a straightforward implementation, the computational complexity of the g-EHF algorithm is about two times as large as that of the g-EKF algorithm. To reduce the gap in computational burden, a simplification of the g-EHF algorithm for single-output networks, i.e., $N_3 = 1$, is described here.

First, focusing on the symmetry of the $N_w \times N_w$ matrix

$$[\boldsymbol{H}_k^T \quad \boldsymbol{H}_k^T] \boldsymbol{R}_{e,k}^{-1} \begin{bmatrix} \boldsymbol{H}_k \\ \boldsymbol{H}_k \end{bmatrix}$$

and algebraically calculating the inverse of the 2×2 matrix $\boldsymbol{R}_{e,k} = \{r_{i,j}\}$ before executing, we find that the Riccati recursion in the g-EHF algorithm for single-output networks reduces to

$$\hat{\boldsymbol{P}}_{k+1|k} = \hat{\boldsymbol{P}}_{k|k-1} - \frac{s}{\delta} \hat{\boldsymbol{P}}_{k|k-1} \boldsymbol{H}_k^T \boldsymbol{H}_k \hat{\boldsymbol{P}}_{k|k-1} \tag{21}$$

where

$$s = (r_{1,1} + r_{2,2}) - (r_{1,2} + r_{2,1}), \quad \delta = r_{1,1} r_{2,2} - r_{1,2} r_{2,1}. \tag{22}$$

Note that the parameters s and δ take a scalar value and the inverse of $\boldsymbol{R}_{e,k}$ is removed in the update of $\hat{\boldsymbol{P}}_{k|k-1}$. This technique decreases the number of required multiplications from $6N_w^2 + 4N_w$ in the original Riccati recursion to $3N_w^2 + N_w$, as shown in Fig. 3.

Taking into account the computational cost in the filter equation, the total number of multiplications required for the reduced version of the g-EHF algorithm becomes $5N_w^2 + 4N_w$. On the other hand, the standard g-EHF algorithm needs $8N_w^2 + 7N_w$ multiplications per time step.

Therefore, the computational burden is reduced by a factor of

$$\begin{aligned} \frac{\text{mul(reduced } g\text{-EHF)}}{\text{mul}(g\text{-EHF})} = & \frac{5N_w^2 + 4N_w}{8N_w^2 + 7N_w} \\ = & \frac{5}{8} \cdot \frac{N_w + 0.8}{N_w + 0.875} \approx 0.625 \end{aligned} \tag{23}$$

where mul$(\cdot)$ implies the number of multiplications per time step.

$$
\begin{aligned}
&for(k = 1, \cdots, N_p, 1 + N_p, \cdots, 2N_p, 1 + 2N_p, \cdots)\\
&\{\\
&\quad \hat{\boldsymbol{z}}^3[k] = \boldsymbol{h}_k(\hat{\boldsymbol{w}}_{k-1})\\
&\quad \hat{\sigma}^2_{v_k} = \hat{\sigma}^2_{v_{k-1}} + \mu[k]\left[\frac{(\boldsymbol{y}_k - \hat{\boldsymbol{z}}^3[k])^T(\boldsymbol{y}_k - \hat{\boldsymbol{z}}^3[k])}{N_3} - \hat{\sigma}^2_{v_{k-1}}\right]\\
&\quad \hat{\boldsymbol{h}}^2_{1,k} = \hat{\boldsymbol{z}}^3[k]\\
&\quad for(n = 2;\ n \geq 1;\ n \leftarrow n-1)\\
&\quad \{\\
&\quad\quad for(i = 1;\ i \leq N_{n+1};\ i \leftarrow i+1)\\
&\quad\quad \{\\
&\quad\quad\quad \boldsymbol{u}^n_{i,k} = \begin{cases} \alpha\hat{z}^3_i[k](1-\hat{z}^3_i[k])[0,\cdots,0,\overset{i}{1},0,\cdots,0]^T, & n = 2\\ \alpha\hat{z}^2_i[k](1-\hat{z}^2_i[k])\sum_{j=1}^{N_3}\hat{w}^2_{i,j}\boldsymbol{u}^{n+1}_{j,k}, & n = 1 \end{cases}\\
&\quad\quad\quad \boldsymbol{\psi}^n_{i,k} = \hat{\boldsymbol{\Sigma}}^n_{i,k|k-1}\hat{\boldsymbol{z}}^n[k]\\
&\quad\quad\quad \alpha^n_i[k] = \hat{\boldsymbol{z}}^n[k]^T\boldsymbol{\psi}^n_{i,k}, \qquad \beta^n_i[k] = {\boldsymbol{u}^n_{i,k}}^T\boldsymbol{u}^n_{i,k}\\
&\quad\quad\quad \hat{\boldsymbol{w}}^n_{i,k} = \hat{\boldsymbol{w}}^n_{i,k-1} + \frac{{\boldsymbol{u}^n_{i,k}}^T[\boldsymbol{y}_k - \hat{\boldsymbol{h}}^n_{i,k}]}{\hat{\sigma}^2_{v_k} + \alpha^n_i[k]\beta^n_i[k]}\boldsymbol{\psi}^n_{i,k}\\
&\quad\quad\quad \hat{\boldsymbol{\Sigma}}^n_{i,k+1|k} = \hat{\boldsymbol{\Sigma}}^n_{i,k|k-1} - \frac{(1-\gamma_f^{-2}\hat{\sigma}^2_{v_k})\beta^n_i[k]}{\hat{\sigma}^2_{v_k} + (1-\gamma_f^{-2}\hat{\sigma}^2_{v_k})\alpha^n_i[k]\beta^n_i[k]}\boldsymbol{\psi}^n_{i,k}{\boldsymbol{\psi}^n_{i,k}}^T\\
&\quad\quad\quad \hat{\boldsymbol{h}}^n_{i+1,k} = \hat{\boldsymbol{h}}^n_{i,k} + \boldsymbol{u}^n_{i,k}\hat{\boldsymbol{z}}^n[k]^T(\hat{\boldsymbol{w}}^n_{i,k} - \hat{\boldsymbol{w}}^n_{i,k-1})\\
&\quad\quad \}\\
&\quad\quad \hat{\boldsymbol{h}}^{n-1}_{1,k} = \hat{\boldsymbol{h}}^n_{N_{n+1}+1,k}\\
&\quad \}\\
&\quad \hat{\boldsymbol{w}}_k = \left[{\hat{\boldsymbol{w}}^1_{1,k}}^T, \cdots, {\hat{\boldsymbol{w}}^1_{N_2,k}}^T, {\hat{\boldsymbol{w}}^2_{1,k}}^T, \cdots, {\hat{\boldsymbol{w}}^2_{N_3,k}}^T\right]^T\\
&\}
\end{aligned}
$$

Fig. 4. The l-EHF learning algorithm; $\hat{mbiw}^n_{i,k} = \left[\hat{\theta}^n_i[k], \hat{w}^n_{1,i}[k], \ldots, \hat{w}^n_{N_n,i}[k]\right]^T$, $\hat{z}^n[k] = \left[1, \hat{z}^n_1[k], \ldots, \hat{z}^n_{N_n}[k]\right]^T$, $\hat{z}^1_i[k] = z^1_i[k]$ $\hat{\Sigma}^n_{i,1|0} = \epsilon\boldsymbol{I}$, $\mu[k] = \begin{cases} \frac{1}{k}, & k \leq T_{\max} \\ \frac{1}{T_{\max}}, & \text{otherwise} \end{cases}$, $T_{\max} = 10 \cdot N_p$.

Compared with the corresponding g-EKF algorithm, the complexity of the reduced g-EHF algorithm maintains a slight increase by a factor of

$$
\begin{aligned}
\frac{\text{mul(reduced } g\text{-EHF)}}{\text{mul}(g\text{-EKF})} &= \frac{5N_w^2 + 4N_w}{4N_w^2 + 3N_w}\\
&= \frac{5}{4}\cdot\frac{N_w + 0.8}{N_w + 0.75} \approx 1.25. \qquad (24)
\end{aligned}
$$

Here, recalling that the g-EHF algorithm requires tens of times larger computational burden than the BP algorithm for each learning epoch, we find that this method of computational reduction is quite effective for the g-EHF algorithm in the case of single-output networks.

III. Simplified Implementation

One of the main disadvantages of using the g-EHF algorithm is its rather severe computational cost in large networks in spite of the computational reduction outlined in the previous section. Indeed, its computational requirements are on the order of $\mathcal{O}(N_w^2)$, which becomes intractable as the size of the network to be trained increases, where N_w is the total number of trainable weights and thresholds in the network under consideration. A matrix inversion in the filter gain computation also requires considerable computational effort for multioutput networks. In addition, the storage requirements are dominated by the need to store the matrix $\hat{\boldsymbol{P}}_{k|k-1}$, which contains N_w^2 elements, each of which must be updated for every time step. Thus, there is a need for a suitable simplification that preserves the most useful properties of the g-EHF algorithm while requiring less computation and storage per time step.

Figs. 4 and 5 present a computationally attractive EHF learning algorithm, which is called a locally optimized EHF (l-EHF) learning algorithm here. The l-EHF algorithm reduces the order of computational complexity to $\mathcal{O}(\sum_{i=1}^{2}(N_i+1)^2 \times N_{i+1})$ per time step, which is suitable for real-time learning (or on-line learning) of large-scale networks. For instance, in the case of a 4–16–1 network, the number of multiplications to be required per time step is almost reduced by a factor of $1/14$. Derivation of the l-EHF algorithm is based on creating disjoint subsets of weights to be decoupled, estimating the variance $\sigma^2_{v_k}$ of observation noise element

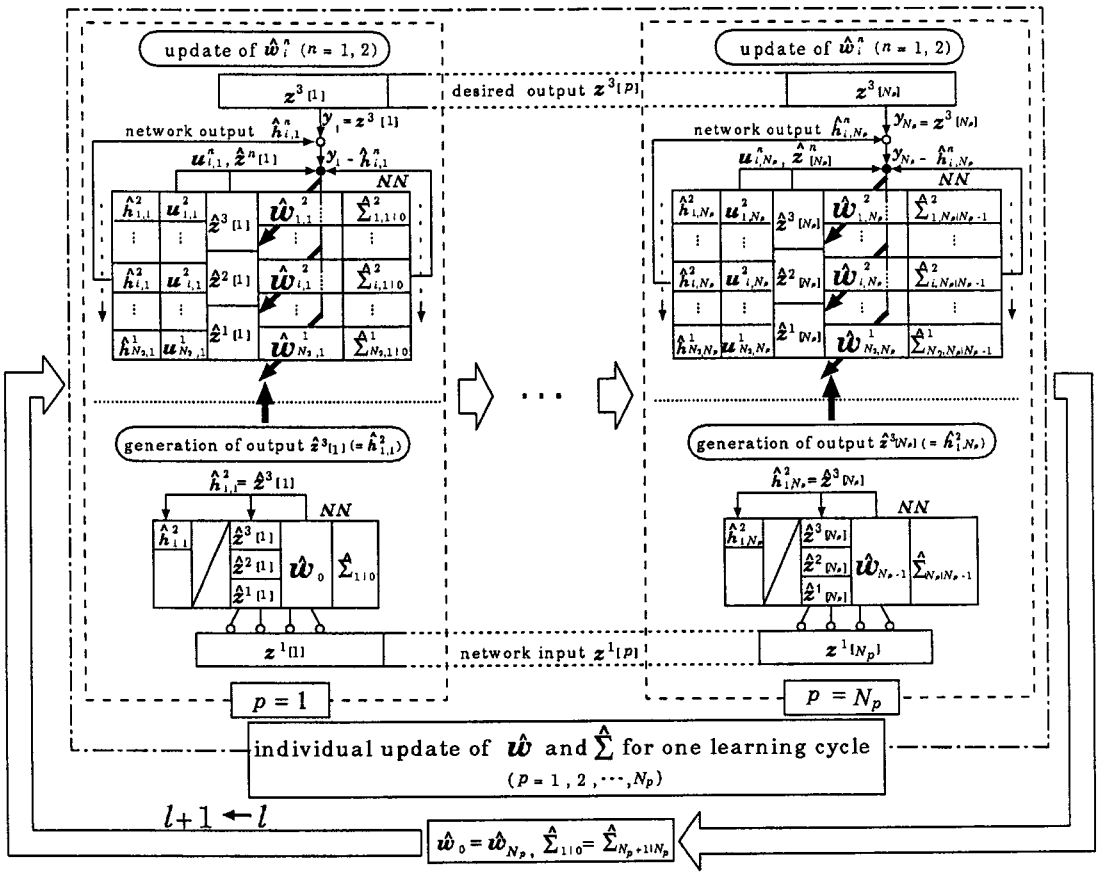

Fig. 5. Implementation of the l-EHF learning algorithm; the weight vector $\hat{\boldsymbol{w}}_{i,k}^n$ for neuron i in the $n+1$th layer is updated in an ascending order of i and then in a descending order of n during each time step.

on-line and locally updating the output $\boldsymbol{h}_k$ in the output layer as $\hat{\boldsymbol{h}}_{i,k}^n$ without feedforward propagation for further improvement of estimation accuracy. A significant part of the derivation of the l-EHF algorithm is described in the Appendix. It should be noted that the reinitialization of $\hat{\boldsymbol{P}}_{k|k-1}$ used in the g-EHF and g-EKF algorithms is not employed here, resulting in the so-called individual update scheme. The l-EHF algorithm needs storage and updating of a matrix for each neuron, but it does not require any matrix inversion.

Furthermore, the l-EHF algorithm corresponds to an H_∞-version of the locally optimized EKF algorithm presented in [13], which is called a locally optimized EKF (l-EKF) algorithm in this paper. Fortunately, the two locally optimized algorithms are in complete agreement when $\gamma_f \to \infty$ and the computational complexity of the l-EHF algorithm is nearly equal to that of the l-EKF algorithm. Such a successful simplification may allow one to optimize the tradeoff between learning ability and computational effort.

In this paper, the l-EHF algorithm also has a very important role in investigating the dependency of the H_∞-learning on implementation, as seen later.

IV. Simulation Study

We carry out several computer simulations to compare the present EHF algorithms with the conventional BP and EKF algorithms in terms of learning performance. In the comparison, it will be verified that the EHF algorithms are more robust than the conventional algorithms to variations in the initial weights and thresholds as well as to deterministic disturbances in observation. Robustness is also measured using the variance σ_L^2 (or normalized variance $\sigma_L^2/\bar{L}^2$) of the number L of learning iterations for 10 000 independent trials. The following BP algorithm is used throughout the simulation:

$$\begin{aligned} w_{j,i}^n[l] &= w_{j,i}^n[l-1] + \tilde{\Delta} w_{j,i}^n[l] \\ \tilde{\Delta} w_{j,i}^n[l] &= \beta \tilde{\Delta} w_{j,i}^n[l-1] + \Delta w_{j,i}^n[l], \quad l = 1, 2, \ldots \end{aligned} \tag{25}$$

where $\Delta w_{j,i}^n = -\eta(\partial J_{\mathrm{BP}})/(\partial w_{j,i}^n)$, $J_{\mathrm{BP}} = (1/2)\sum_{p=1}^{N_p} \sum_{i=1}^{N_3} (z_i^3[p] - \hat{z}_i^3[p])^2$, $\tilde{\Delta} w_{j,i}^n[0] = 0$, and η and β are called the learning rate and the momentum factor, respectively. Note that the initial conditions for $\{w_{j,i}^n\}$ are different in each of the following problems but are kept the same for each of algorithms applied to the same problem.

A. Robustness to Variations in Initial Weights and Thresholds

1) XOR Problem: As the first experiment, we consider the XOR problem in which a three-layered neural network is trained to assign the inputs $(z_1^1[p], z_2^1[p])$ to the desired output (target) $z_1^3[p]$ as

$$\{(0,0),0\},\ \{(0,1),1\},\ \{(1,0),1\},\ \{(1,1),0\}.$$

In the training procedure, all examples in the training set $\{(z_1^1[p], z_2^1[p]), z_1^3[p]\}$ are periodically presented to the net-

TABLE I
ROBUSTNESS OF EACH LEARNING ALGORITHM TO VARIATIONS IN INITIAL WEIGHTS, $(-0.1, 0.1)_{iw}$, FOR THE XOR PROBLEM

	$N_2 = 4$			$N_2 = 8$		
algorithm	BP (η=0.8, β=0.8)	g-EKF	g-EHF (γ_f=1.7)	BP (η=0.9, β=0.8)	g-EKF	g-EHF (γ_f=1.4)
$\bar{L}$	121.0	47.1	23.9	67.0	33.4	22.7
$\sigma_L^2/\bar{L}^2$	0.734	0.244	0.018	0.036	0.082	0.012
N_{term}	0	0	0	0	0	0
$t_{\bar{L}}$ [ms]	24.2	89.5	47.8	26.8	237.0	161.0

work in the order such as $\{(0,0),0\}$, $\{(0,1),1\}$, $\{(1,0),1\}$, $\{(1,1),0\}$, $\{(0,0),0\},\ldots$ and so on. The weights and thresholds in the network are initialized to random values uniformly distributed between -0.1 and 0.1 and are then updated iteratively. For simplicity, the initial setting is denoted by $(-0.1, 0.1)_{iw}$. The updating method is to cycle through all examples before updating the weights and thresholds, resulting in the so-called batch update scheme (see Fig. 2). Note that the output becomes $y_k = z_1^3[k]$ since the binary classification has no noise. Training is terminated when the total squared error $J = \sum_{p=1}^{4}(\hat{z}_1^3[p] - z_1^3[p])^2$ falls below 0.01 or when convergence is not obtained within a prespecified number, $L_{\rm term}$, of learning iterations. Here, J is evaluated typically at each iteration on the whole pattern set.

To statistically evaluate the convergence of each learning algorithm, 10 000 independent sets of initial weights and thresholds were randomly chosen and then the layered network was trained so as to satisfy the XOR mapping ($\{0,1\}^2 \rightarrow \{0,1\}$) for each initial weight set. Here, small randomly generated initial values are employed to emphasize the influence of learning algorithms, rather than initial weight settings, on convergence. Indeed, such an initial setting tends to prevent their adjustment from falling into and remaining within the flat regions in the error surfaces.

For quantitative comparison of convergence speed and robustness, Table I lists the average $\bar{L}$ of the number L of learning iterations to attain the desired total squared error and its normalized variance $\sigma_L^2/\bar{L}^2$ over 10 000 independent trials, leaving out the trials which did not converge within $L_{\rm term}$. The tuning parameters of each algorithm are carefully chosen and the sigmoid function has a considerably large gradient ($\alpha = 2.5$) to hasten the convergence. Here, $N_{\rm term}$ is the number of trials which do not converge within $L_{\rm term} = 10^5$ and $t_{\bar{L}}$ stands for the average CPU time ($\bar{L} \times \bar{t}_{\rm step}$) required for running $\bar{L}$ iterations, where $\bar{t}_{\rm step}$ is the CPU time per unit step averaged over 10^4 steps and the CPU time is observed with the *time* command in Solaris 2.5 on SUN SS5 (SPARC 170MHz, 32MB).

Fig. 6 shows the convergence processes (learning trajectories) during the first 100 trials within a total of 10 000 trials for 1) the BP algorithm with $\eta = 0.8$ and $\beta = 0.8$; 2) the g-EKF algorithm; and 3) the g-EHF algorithm with $\gamma_f = 1.7$. Here, a 2–4–1 network is used and each curve of the total squared error is plotted as a function of learning iterations. These results show that tuning γ_f allows the g-EHF algorithm to accomplish a dramatic improvement in the robustness to variations in the

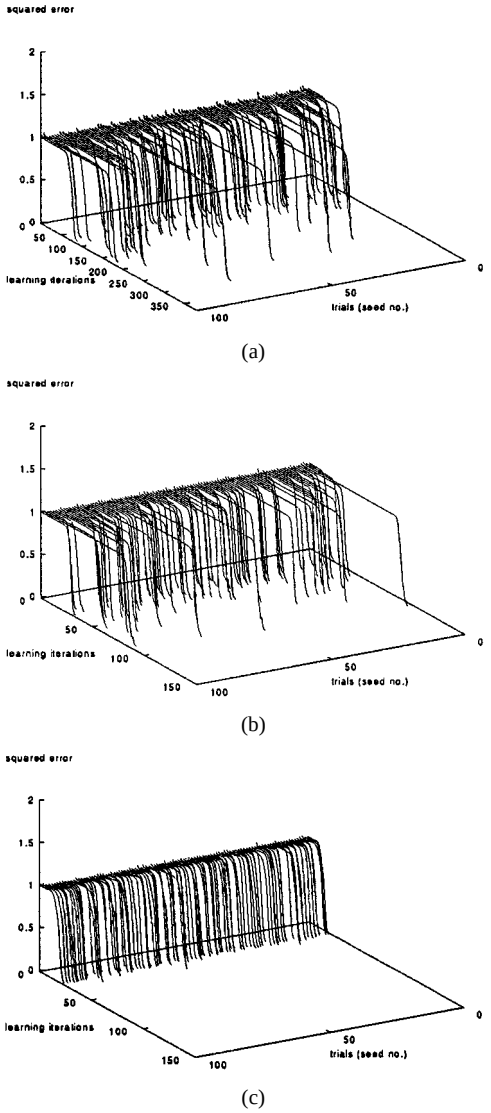

Fig. 6. Learning trajectories in the first 100 trials for the XOR problem using a 2–4–1 network; $\alpha = 2.5$, $(-0.1, 0.1)_{iw}$. (a) The BP algorithm; $\eta = 0.8$, $\beta = 0.8$. (b) The g-EKF algorithm. (c) The g-EHF algorithm; $\gamma_f = 1.7$.

initial weights and thresholds, obviously reducing the dependence on the choice of initial conditions for $\hat{\boldsymbol{w}}_k$. The phenomenon observed in the learning trajectories could be characterized by $\sigma_L^2/\bar{L}^2$. In addition, Table I suggests that the g-EHF algorithm successfully converges in a smaller number of iterations and seems to be more robust to changes in the number of neurons in the hidden layer. Here it should be noted that the g-EKF algorithm reinitializes $\hat{\boldsymbol{P}}_{k|k-1}$ in the same manner as the g-EHF algorithm does. The convergence of the BP algorithm is slower on average even when η and β are carefully chosen, but needs no more computation time than the g-EKF and g-EHF algorithms. In the worst case of convergence, the maximum number $L_{\max}$ of iterations for the BP, the g-EKF and the g-EHF algorithms were 9044, 389, and 40, respectively, so that the CPU time required for the worst convergence is estimated to be 1.81×10^3, 7.40×10^2 and 80.0 ms.

Throughout the simulations, the value of γ_f is experimentally determined. If γ_f is set to a large positive number (for instance $\gamma_f = 10^4$) then the results of the g-EKF and the g-EHF algo-

TABLE II
ROBUSTNESS OF EACH LEARNING ALGORITHM TO VARIATIONS IN INITIAL WEIGHTS, $(-0.2, 0.2)_{iw}$, FOR THE FOUR-INPUT PARITY PROBLEM

	$N_2 = 8$			$N_2 = 12$		
algorithm	BP (η=0.2, β=0.5)	g-EKF	g-EHF (γ_f=1.1)	BP (η=0.2, β=0.5)	g-EKF	g-EHF (γ_f=1.1)
$\bar{L}$	2484.8	94.3	60.8	1817.3	54.6	47.7
$\sigma_L^2/\bar{L}^2$	5.92	14.6	17.0	3.68	4.19	79.3
N_{term}	63	2	1	25	6	1
t_L (s)	5.22	5.79	3.73	5.45	7.35	6.77

rithms become close enough to be indistinguishable. This means that when the robustness is not required, the g-EHF algorithm can be easily switched over to the g-EKF algorithm by setting γ_f to a large positive number. This is one of the attractive features of the proposed g-EHF algorithm. In general, the determination of γ_f is very important to exhibit the excellent performance of the g-EHF algorithm. In this work, the best value of γ_f was found by gradually decreasing γ_f from a relatively large positive number to 1.0 for checking the performance degradation or the positiveness of the smallest eigenvalue of $\hat{\boldsymbol{P}}_{k|k}$. It was found that the range of γ_f from 1.0 to 3.0 is the most important in many cases. It is also known that the parameter γ_f takes no values below 1.0. A better insight into the relevance of γ_f requires further detailed study.

According to the literature [12], [22], the Riccati recursion for updating the matrix $\hat{\boldsymbol{P}}_{k|k-1}$ can be augmented by an artificial system noise term that is known to accelerate training and lead to better solutions. As a test, the system noise $\boldsymbol{u}_k$ with a heuristically determined covariance $\sigma_u^2\boldsymbol{I}$, $\sigma_u^2 = 0.1$ was added into the g-EKF algorithm for training a 2–4–1 network. The normalized variance $\sigma_L^2/\bar{L}^2$ of iteration number was improved to 0.135, but the average $\bar{L}$ increases slightly to 49.0. Adding a system noise yielded no significant improvement to the g-EHF algorithm in this case.

2) Parity Problem: The second experiment is a four-input parity problem where the desired output is one if the input pattern (code) contains an odd number of ones, otherwise it is zero. Sixteen different pairs of input and desired output are used for training a three-layered network. To statistically evaluate the convergence of each learning algorithm, the layered network is trained for 10 000 independent trials, in which the weights and thresholds are initialized to random values uniformly distributed between -0.2 and 0.2, i.e., $(-0.2, 0.2)_{iw}$. Training was terminated when the squared error totaled over all 16 examples fell below 0.01 or when the convergence was not obtained within a maximum number $L_{\rm term} = 10^5$ of iterations.

Table II lists the average number $\bar{L}$ of learning iterations and the normalized variance $\sigma_L^2/\bar{L}^2$ over 10 000 independent trials, leaving out the trials which did not converge within $L_{\rm term}$. Here η and β in the BP and γ_f in the g-EHF are carefully selected and are different from the values used in the XOR problem. In this case, extremely slow convergence occurs for every algorithm, which terminates the learning after $L_{\rm term} = 10^5$ iterations. It should be noted here that the BP algorithm requires too specific a parameter adjustment for reaching the acceptable results. Also, $\bar{L}$ is not critically affected by γ_f, but it is drastically affected by η and β. This makes applications of the g-EHF algorithm easier as compared with the BP algorithm. As seen in Table II, we obtain the results similar to those for the XOR problem. That is, the convergence of the BP algorithm was much slower on average than that of the g-EHF and g-EKF algorithms, whereas the g-EHF algorithm has more robustness to variations in initial weights and thresholds. Here, it seems that the g-EHF algorithm is inferior to the g-EKF algorithm because of the much bigger variance of L when $N_2 = 12$. However, such a conclusion is incorrect because six terminated trials in the g-EKF algorithm are not taken into account for calculating the variance. On the other hand, in the g-EHF algorithm, only one trial is terminated although considerably slower convergence within $L_{\rm term}$ sometimes occurs.

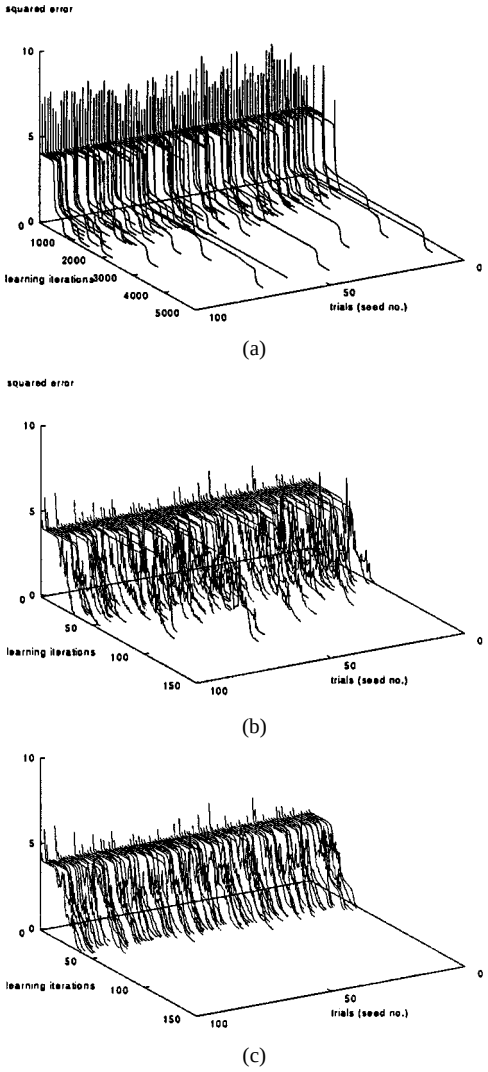

Fig. 7. Learning trajectories in the first 100 trials for the four-input parity problem using a 4–12–1 network; $\alpha = 2.5$, $(-0.2, 0.2)_{iw}$. (a) The BF algorithm; $\eta = 0.2$. $\beta = 0.5$. (b) The g-EKF algorithm. (c) g-EHF algorithm; $\gamma_f = 1.1$.

Fig. 7 shows the learning trajectories in the first 100 trials for the BP, the g-EKF and the g-EHF algorithms when a 4–12–1 network is trained. The convergence of the BP and the g-EKF algorithms is considerably affected by the initial weights and thresholds. In contrast, the g-EHF algorithm successfully ex-

TABLE III
ROBUSTNESS OF EACH LEARNING ALGORITHM TO VARIATIONS IN INITIAL WEIGHTS, $(-0.5, 0.5)_{iw}$, FOR THE FOUR-INPUT PARITY PROBLEM

	$N_2 = 8$			$N_2 = 12$		
algorithm	BP (η=0.2, β=0.7)	g-EKF	g-EHF (γ_f=1.5)	BP (η=0.2, β=0.6)	g-EKF	g-EHF (γ_f=2.1)
$\bar{L}$	780.5	41.1	40.6	637.7	27.4	26.7
$\sigma_L^2/\bar{L}^2$	14.05	5.63	4.57	6.93	1.14	0.86
N_{term}	18	0	0	10	0	0
t_L (s)	1.64	2.52	2.49	1.91	3.78	3.79

hibits robustness in convergence to the initial weight variations in spite of the large value of $\sigma_L^2/\bar{L}^2$.

Table III shows the statistical results of learning for the same parity problem when the weights and thresholds were initialized to random variables uniformaly distributed between -0.5 and 0.5. In this larger weight initialization, both g-EHF and g-EKF algorithms converge within $L_{\rm term} = 10^5$ for every trial and in fewer iterations on average than in the previous case shown in Table II. Furthermore, the number of iterations for the g-EHF algorithm has a smaller spread around the mean compared to that of the other algorithms. Nevertheless, an investigation of the learning trajectories shows that the g-EHF algorithm cannot completely suppress the worst case of convergence. The enhancement of the worst-case suppression is one of the most important open problems.

From this one can conclude that, for the parity problem, the g-EHF learning algorithm provides the most desirable performance among the three learning algorithms at the expense of an increase in computational burden.

B. Robustness to Disturbances in Observations

To explore the superiority of the g-EHF algorithm from a different point of view, we consider the problem of predictive learning, i.e., estimating an unknown dependency from known observations (or training samples) with disturbances. The problem of predictive learning has became increasingly important. Once a dependency has been learned, it can be used to predict future data. The problem of predictive learning is inherently difficult (ill-posed), due to the general lack of knowledge about the underlying dependency and the finiteness of available training data.

Fig. 8 shows an example of learning a sinusoid $y_k = 0.6\cos(\pi k/24)$ with a 4–8–1 network using the data $\{y_k\}_{k=0}^{35}$, a part of which is lacking as $y_6 = 0$ and $y_7 = 0$. Here the network is randomly initialized in the range of (−0.5,0.5) and trained sufficiently to retain the total squared error constant using a sigmoid function defined as $f(x) = (1-\exp(-\alpha x))/(1+\exp(-\alpha x))$. A part of the time-series to be predicted by the trained network is $\{y_k\}_{k=36}^{48}$, in which the output of the network is fed back to the last input neuron since the input data are not available in the period of prediction. In the learning process, the total squared error J of the BP and the g-EKF algorithms attained the value 0.49 and that of the g-EHF algorithm and the g-EKF with the system noise of $\sigma_u^2 = 0.012$ remained constant at 0.55. Note that $L_{\rm term}$ is set to 5×10^4 for the BP and to 10^3 for the g-EHF and g-EKF algorithms, respectively. It is also confirmed in some simulations that the g-EHF algorithm creates an equilibrium point at larger squared error than the BP and the g-EKF (with $\sigma_u^2 = 0$). As seen in Fig. 8, the BP and the g-EKF algorithms provide no acceptable recall results, even when system noise is added. That is, although introduction of system noise improves the tracking ability, the asymptotic performance deteriorates with added system noise. In contrast, the g-EHF algorithm enables the trained network to considerably suppress the influence of the deterministic disturbance in observations. In the field of learning with neural networks, this phenomenon is very interesting, because it suggests that the g-EHF algorithm tends to avoid over-learning, thus exhibiting a good recall ability. This extends the potential of the H_∞-learning.

C. Dependency on Implementation

How does the favorable performance of H_∞-learning depend on implementation? To answer this question, an experiment similar to that in Section IV-A.2 was carried out for a locally optimized EHF (l-EHF) algorithm, which is derived through a simplified implementation presented in Section III. Table IV summarizes the comparison results for a four-input parity problem, yielding a statistical insight into the convergence property. In the right-hand column of Table IV, a 4–16–2 network is trained so that the desired outputs are (1,0) if the input pattern contains an odd number of ones and otherwise (0,1). These results show that the l-EHF and l-EKF algorithms realize much faster convergnce speed as well as less computation time on average than the BP algorithm for training the 4–16–1 network, where $L_{\rm term} = 10^5$ and $\epsilon = 1$ are set. Note that the l-EKF algorithm in this paper is identical to a learning algorithm presented in [13]. Furthermore, the l-EHF algorithm is more robust than the l-EKF algorithm for variations in the initial weights and thresholds, leading to a fast and robust learning algorithm. Here, it is found that the determination of γ_f in the l-EHF algorithm is more difficult than that in the g-EHF algorithm. Fig. 9 demonstrates variations of the number L of learning iterations in the l-EKF and l-EHF algorithms for 10 000 trials when a 4–16–1 network is trained. The outliers in convergence are seen to be successfully suppressed in the l-EHF algorithm. On the other hand, in the training of a 4–16–2 network, the maximum number $L_{\rm max}$ of L over 10 000 trials for the BP, the l-EKF and the l-EHF algorithms were 60046, 2786, and 1305, respectively, so that the CPU time required for the worst convergence is estimated to be 2.82×10^2, 1.30×10^2, and 6.09×10^1 s. These results also verify the robustness of the l-EHF algorithm to variations in the initial weight vector.

Table IV is also helpful to assess the influence of decoupling (i.e., segmentation of the neural network) on performance. In comparison with the g-EHF and g-EKF algorithms, the l-EHF and l-EKF algorithms tend to raise the average iteration number $\bar{L}$ over 10 000 independent trials (see Table II). There is also a remarkable increase in $\bar{L}$ when the hidden layer has only a few neurons. This degradation is considered to be due to approximating $\hat{\Sigma}_{k|k-1}$ to a block diagonal matrix using

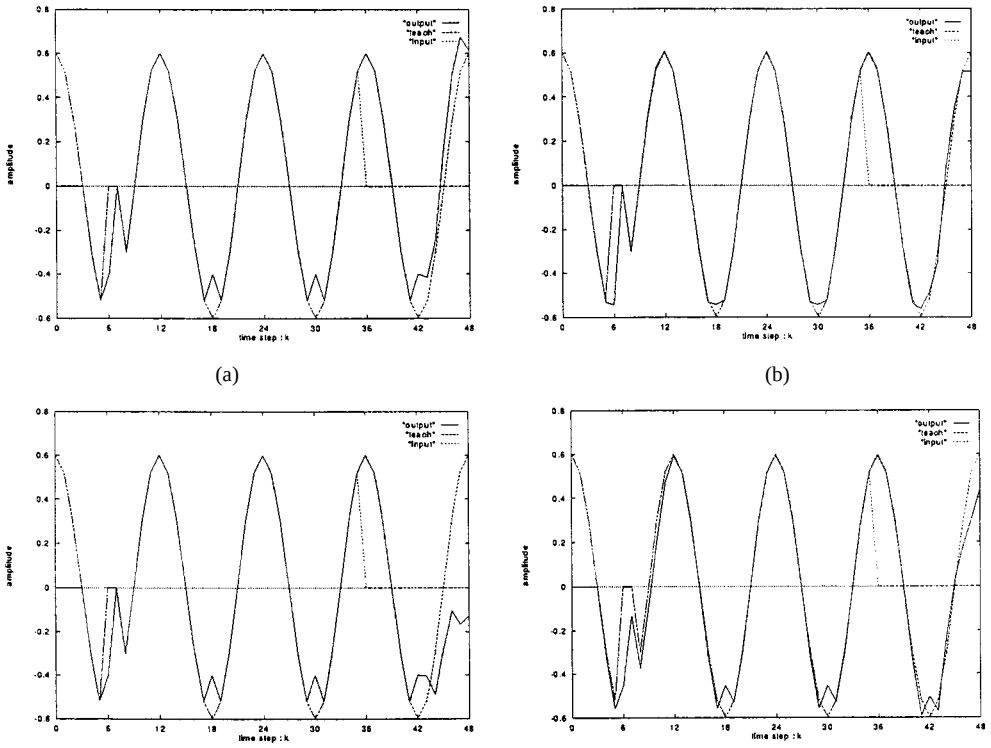

Fig. 8. Recalled and predicted results via each learning algorithm in the learning of a sinusoid with disturbances using a 4–8–1 network; $\alpha = 1.5$. (a) The BP algorthim; $\eta = 0.2$, $\beta = 0.6$. (b) The g-EHF algorithm; $\eta_f = 1.2$. (c) The g-EKF algorithm; $\sigma_u^2 = 0$. (d) The g-EKF algorithm; $\sigma_u^2 = 0.012$.

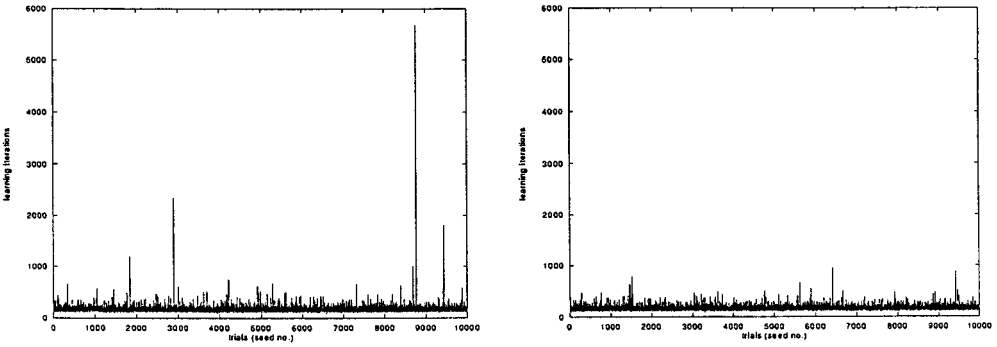

Fig. 9. Distribution of the number L of learning iterations for 10 000 trials in the four-input parity problem using a 4–16–1 network; $\alpha = 2.5$, $(-0.2, 0.2)_{iw}$. (a) the l-EKF algorithm and (b) the l-EHF algorithm; $\eta_f = 1.2$.

TABLE IV
ROBUSTNESS OF EACH LEARNING ALGORITHM TO VARIATIONS IN INITIAL WEIGHTS, $(-0.2, 0.2)_{iw}$, FOR THE FOUR-INPUT PARITY PROBLEM

	$N_2 = 16,\ N_3 = 1$			$N_2 = 16,\ N_3 = 2$		
algorithm	BP	l-EKF	l-EHF	BP	l-EKF	l-EHF
	(η=0.2, β=0.2)		(γ_f=1.2)	(η=0.2, β=0.4)		(γ_f=3.9)
$\bar{L}$	2977.3	174.0	165.3	1460.5	149.6	148.5
$\sigma_L^2/\bar{L}^2$	2.41	0.20	0.08	0.54	0.19	0.13
N_{term}	20	0	0	0	0	0
$t_{\bar{L}}$ (s)	11.6	5.8	5.5	6.9	7.0	6.9

decoupling techniques. However, the principal robustness of the H_∞-learning is preserved in spite of such an approximation and thus is essentially independent of implimentation.

V. CONCLUSION

In this paper, H_∞-learning have been proposed for training multilayered feedforward neural networks and some learning algorithms have been derived by applying an H_∞ filter to the linearized neural-network model with state-space representation. In addition, exploring the relationship between the H_∞-based (EHF) and the Kalman-based (EKF) algorithms has provided an

insight into the working of H_∞-learning. The EHF algorithms, especially a globally optimized EHF (g-EHF) algorithm, successfully converges in a smaller number of learning iterations than the BP algorithm and exhibit more robust behavior than the EKF algorithms to variations in the initial weight assignment. Furthermore, since only one added tuning parameter γ_f is required, the additional tuning effort is not large in comparison with the EKF algorithms.

In the g-EHF algorithm, however, the computation time required for convergence is not reduced proportionally to the number of iterations due to an increase in computation time per iteration. Nevertheless, there are some applications, such as neural controllers, in which the number of iterations is substantially more important than the computation time so that the g-EHF algorithm could become most useful for these. The robustness to deterministic disturbances in observations was also clarified using the time-series prediction problem, which leads to the understanding of over-learning.

To reduce the computational burden, a simplified implementation of the g-EHF algorithm was also developed by means of local optimization for each neuron. The success of the locally optimized EHF (l-EHF) algorithm supports our conclusion that the robustness of H_∞-learning does not essentially depend on implementation. Our ultimate objective is the development of algorithms that permits an optimal tradeoff between learning ability and computational effort.

In the future, effective determination of γ_f in H_∞-learning will be studied through implementations, including the on-line estimation of the system noise [22], in more detail and its applications will be examined for various real-world problems.

Appendix
Derivation of the l-EHF Algorithm

The standard Riccati recursion in the l-EHF algorithm for each neuron i in the $n+1$th layer is represented as

$$\hat{\boldsymbol{\Sigma}}^n_{i,k+1|k} = \hat{\boldsymbol{\Sigma}}^n_{i,k|k-1} - \hat{\boldsymbol{\Sigma}}^n_{i,k|k-1}\left[\boldsymbol{H}^{n}_{i,k}{}^T \quad \boldsymbol{H}^{n}_{i,k}{}^T\right] \cdot \boldsymbol{R}^{n,i}_{e,k}{}^{-1}\begin{bmatrix}\boldsymbol{H}^n_{i,k}\\ \boldsymbol{H}^n_{i,k}\end{bmatrix}\hat{\boldsymbol{\Sigma}}^n_{i,k|k-1} \quad (26)$$

where

$$\boldsymbol{R}^{n,i}_{e,k} = \begin{bmatrix}\hat{\sigma}^2_{vk}\boldsymbol{I} & 0\\ 0 & -\gamma_f^2\boldsymbol{I}\end{bmatrix} + \begin{bmatrix}\boldsymbol{H}^n_{i,k}\\ \boldsymbol{H}^n_{i,k}\end{bmatrix}\hat{\boldsymbol{\Sigma}}^n_{i,k|k-1} \cdot \left[\boldsymbol{H}^{n}_{i,k}{}^T \quad \boldsymbol{H}^{n}_{i,k}{}^T\right]. \quad (27)$$

Let $\boldsymbol{R}^{n,i}_{e,k}{}^{-1}$ be a 2×2 block matrix such that

$$\boldsymbol{R}^{n,i}_{e,k}{}^{-1} = \begin{bmatrix}\boldsymbol{L}^n_{i,k} & \boldsymbol{M}^n_{i,k}\\ \boldsymbol{N}^n_{i,k} & \boldsymbol{O}^n_{i,k}\end{bmatrix} \quad (28)$$

for convenience. Then, defining

$$\boldsymbol{S}^n_{i,k} = \boldsymbol{L}^n_{i,k} + \boldsymbol{M}^n_{i,k} + \boldsymbol{N}^n_{i,k} + \boldsymbol{O}^n_{i,k} \quad (29)$$

we can rewrite (26) as

$$\hat{\boldsymbol{\Sigma}}^n_{i,k+1|k} = \hat{\boldsymbol{\Sigma}}^n_{i,k|k-1} - \hat{\boldsymbol{\Sigma}}^n_{i,k|k-1}\boldsymbol{H}^{n}_{i,k}{}^T\boldsymbol{S}^n_{i,k}\boldsymbol{H}^n_{i,k}\hat{\boldsymbol{\Sigma}}^n_{i,k|k-1} \quad (30)$$

which is computationally simpler. Besides, the inverse of $\boldsymbol{R}^{n,i}_{e,k}$ can be factorized as

$$\boldsymbol{R}^{n,i}_{e,k}{}^{-1} = \begin{bmatrix}\boldsymbol{I} & \boldsymbol{A}^n_{i,k}\\ 0 & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{B}^n_{i,k} & 0\\ 0 & \boldsymbol{C}^n_{i,k}\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & 0\\ \boldsymbol{A}^n_{i,k} & \boldsymbol{I}\end{bmatrix} = \begin{bmatrix}\boldsymbol{B}^n_{i,k} + \boldsymbol{A}^n_{i,k}\boldsymbol{C}^n_{i,k}\boldsymbol{A}^n_{i,k} & \boldsymbol{A}^n_{i,k}\boldsymbol{C}^n_{i,k}\\ \boldsymbol{C}^n_{i,k}\boldsymbol{A}^n_{i,k} & \boldsymbol{C}^n_{i,k}\end{bmatrix}. \quad (31)$$

in which, using the matrix inversion lemma and $\boldsymbol{H}^n_{i,k} = \boldsymbol{u}^n_{i,k}\hat{\boldsymbol{z}}^n[k]^T$, the matrices $\boldsymbol{A}^n_{i,k}$, $\boldsymbol{B}^n_{i,k}$, and $\boldsymbol{C}^n_{i,k}$ are reduced to

$$\boldsymbol{A}^n_{i,k} = -\frac{\alpha^n_i[k]}{\hat{\sigma}^2_{vk} + \alpha^n_i[k]\beta^n_i[k]}\boldsymbol{u}^n_{i,k}\boldsymbol{u}^n_{i,k}{}^T \quad (32)$$

$$\boldsymbol{B}^n_{i,k} = \frac{1}{\hat{\sigma}^2_{vk}}\left[\boldsymbol{I} - \frac{\alpha^n_i[k]}{\hat{\sigma}^2_{vk} + \alpha^n_i[k]\beta^n_i[k]}\boldsymbol{u}^n_{i,k}\boldsymbol{u}^n_{i,k}{}^T\right] = \frac{1}{\hat{\sigma}^2_{vk}}\left[\boldsymbol{I} + \boldsymbol{A}^n_{i,k}\right] \quad (33)$$

$$\boldsymbol{C}^n_{i,k} = \frac{1}{\gamma_f^2}\left[\frac{\hat{\sigma}^2_{vk}\alpha^n_i[k]}{\hat{\sigma}^2_{vk}\alpha^n_i[k]\beta^n_i[k] - \gamma_f^2\left(\hat{\sigma}^2_{vk} + \alpha^n_i[k]\beta^n_i[k]\right)} \cdot \boldsymbol{u}^n_{i,k}\boldsymbol{u}^n_{i,k}{}^T - \boldsymbol{I}\right] \quad (34)$$

where

$$\alpha^n_i[k] = \hat{\boldsymbol{z}}^n[k]^T\hat{\boldsymbol{\Sigma}}^n_{i,k|k-1}\hat{\boldsymbol{z}}^n[k], \quad \beta^n_i[k] = \boldsymbol{u}^n_{i,k}{}^T\boldsymbol{u}^n_{i,k} \quad (35)$$

and $\boldsymbol{u}^n_{i,k}$ is determined as shown in Fig. 4. Comparing (28) with (31), we have

$$\begin{aligned}\boldsymbol{S}^n_{i,k} &= \left(\boldsymbol{B}^n_{i,k} + \boldsymbol{A}^n_{i,k}\boldsymbol{C}^n_{i,k}\boldsymbol{A}^n_{i,k}\right) + \boldsymbol{A}^n_{i,k}\boldsymbol{C}^n_{i,k} + \boldsymbol{C}^n_{i,k}\boldsymbol{A}^n_{i,k} + \boldsymbol{C}^n_{i,k}\\ &= \left(\boldsymbol{A}^n_{i,k} + \boldsymbol{I}\right)\left\{\frac{1}{\hat{\sigma}^2_{vk}}\boldsymbol{I} + \boldsymbol{C}^n_{i,k}\left(\boldsymbol{A}^n_{i,k} + \boldsymbol{I}\right)\right\}\\ &= \frac{\left(1-\gamma_f^{-2}\hat{\sigma}^2_{vk}\right)}{\hat{\sigma}^2_{v,k}}\left[\boldsymbol{I} - \frac{\left(1-\gamma_f^{-2}\hat{\sigma}^2_{v,k}\right)\alpha^n_i[k]}{\hat{\sigma}^2_{vk} + \left(1-\gamma_f^{-2}\hat{\sigma}^2_{vk}\right)\alpha^n_i[k]\beta^n_i[k]} \cdot \boldsymbol{u}^n_{i,k}\boldsymbol{u}^n_{i,k}{}^T\right]\end{aligned} \quad (36)$$

which leads to

$$\boldsymbol{u}^n_{i,k}{}^T\boldsymbol{S}^n_{i,k}\boldsymbol{u}^n_{i,k} = \frac{\left(1-\gamma_f^{-2}\hat{\sigma}^2_{vk}\right)\beta^n_i[k]}{\hat{\sigma}^2_{vk} + \left(1-\gamma_f^{-2}\hat{\sigma}^2_{vk}\right)\alpha^n_i[k]\beta^n_i[k]}. \quad (37)$$

Substituting (37) into the second term on the right-hand side of (30) arranged as

$$\hat{\boldsymbol{\Sigma}}^n_{i,k|k-1}\boldsymbol{H}^{n}_{i,k}{}^T\boldsymbol{S}^n_{i,k}\boldsymbol{H}^n_{i,k}\hat{\boldsymbol{\Sigma}}^n_{i,k|k-1} = \hat{\boldsymbol{\Sigma}}^n_{i,k|k-1}\hat{\boldsymbol{z}}^n[k]\left(\boldsymbol{u}^n_{i,k}{}^T\boldsymbol{S}^n_{i,k}\boldsymbol{u}^n_{i,k}\right)\hat{\boldsymbol{z}}^n[k]^T\hat{\boldsymbol{\Sigma}}^n_{i,k|k-1}$$

we then obtain

$$\hat{\boldsymbol{\Sigma}}^n_{i,k+1|k} = \hat{\boldsymbol{\Sigma}}^n_{i,k|k-1} - \frac{\left(1-\gamma_f^{-2}\hat{\sigma}^2_{vk}\right)\beta^n_i[k]}{\hat{\sigma}^2_{vk} + \left(1-\gamma_f^{-2}\hat{\sigma}^2_{vk}\right)\alpha^n_i[k]\beta^n_i[k]}\boldsymbol{\psi}^n_{i,k}\boldsymbol{\psi}^n_{i,k}{}^T \quad (38)$$

where $\boldsymbol{\psi}^n_{i,k} = \hat{\boldsymbol{\Sigma}}^n_{i,k|k-1}\hat{\boldsymbol{z}}^n[k]$.

ACKNOWLEDGMENT

The authors would like to acknowledge the valuable suggestions and comments from the anonymous reviewers and Prof. S. Omatu, Osaka Prefecture University.

REFERENCES

[1] D. E. Rumelhart and J. L. McCelland, *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, vol. 1.
[2] R. Hecht-Nielsen, *Neurocomputing*. Reading, MA: Addison-Wesley, 1990.
[3] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Clarendon.
[4] W. T. Miller, R. S. Sutton, and P. J. Werbos, *Neural Networks for Control*. Cambridge, MA: MIT Press, 1990.
[5] P. J. Antsaklis *et al.*, "Neural networks in control systems," *IEEE Contr. Syst. Mag.*, pp. 3–86, 1990.
[6] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295–307, 1988.
[7] A. G. Parlos *et al.*, "An accelerating learning algorithm for multilayer perceptron networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 493–497, May 1994.
[8] X. H. Yu, G. A. Chen, and S. X. Cheng, "Dynamic learning rate optimization of the backpropagation algorithm," *IEEE Trans. Neural Networks*, vol. 6, pp. 669–677, May 1995.
[9] R. E. Kalman, "A new approach to linear filtering and prediction problem," *J. Basic Eng.*, vol. 82, pp. 35–45, 1960.
[10] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall, 1979.
[11] S. Singhal and L. Wu, "Training feedforward networks with the extended Kalman filter," in *Proc. IEEE Int. Conf. ASSP*, 1989, pp. 1187–1190.
[12] G. V. Puskorious and L. A. Feldkamp, "Decoupled extended Kalman filter training of feedforward layered networks," *IEEE Trans. Neural Networks*, vol. 2, pp. 771–777, 1991.
[13] Y. Iiguni, H. Sakai, and H. Tokumaru, "A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter," *IEEE Trans. Signal Processing*, vol. 40, pp. 959–966, 1992.
[14] S. Shah *et al.*, "Optimal filtering algorithms for fast learning in feedforward neural networks," *Neural Networks*, pp. 779–787, 1992.
[15] R. J. Williams, "Training recurrent networks using the extended Kalman filter," in *Proc. Int. Joint Conf. Neural Networks*, 1992, pp. 241–246.
[16] G. V. Puskorious and L. A. Feldkamp, "Extensions and enhancements of decoupled extended Kalman filter training," in *Proc. IEEE Int. Conf. Neural Networks*, 1997, pp. 1879–1883.
[17] B. Schottky and D. Saad, "Statistical mechanics of EKF learning in neural networks," *J. Phys. A*, pp. 1605–1621, 1999.
[18] K. S. Narendra and K. Parthasaarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, 1990.
[19] M. B. Matthews, "Neural-network nonlinear adaptive filtering using the extended Kalman filter," in *Proc. Int. Neural Network Conf.*, vol. I, 1990, pp. 115–119.
[20] G. V. Puskorious and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 279–297, Mar. 1991.
[21] E. Wan, R. Merwe, and A. Nelson, "Dual estimation and the unscented transformation," *Advances Neural Inform. Processing Syst.*, vol. 12, pp. 666–672, 2000.
[22] W. D. Penny and S. J. Roberts, "Dynamic models for nonstationary signal segmentation," *Comput. Biomed. Res.*, vol. 32, no. 6, pp. 483–502, 1999.
[23] U. Shaked and Y. Theodor, "H^∞-optimal estimation: A tutorial," in *Proc. IEEE Conf. Decision Contr.*, 1992, pp. 2278–2286.
[24] B. Hassibi, A. H. Sayed, and T. Kailath, "Linear estimation in Krein spaces—part I: Theory," *IEEE Trans. Automat. Contr.*, vol. 41, pp. 18–33, 1996.
[25] ——, "Linear estimation in Krein spaces—part II: Applications," *IEEE Trans. Automat. Contr.*, vol. 41, pp. 34–49, 1996.

Kiyoshi Nishiyama was born in Tokyo, Japan, in 1957. He received the M.E. degree in electrical engineering from Chiba University, Japan, in 1985 and the Dr. Eng. degree from Tokyo Institute of Technology, Tokyo, Japan, in 1991.

He joined the Department of Information and Computer Science, Iwate University, Morioka, Japan, in 1998. He is presently an Associate Professor. His research interests include digital signal processing and neural networks, especially in learning theory.

Dr. Nishiyama is a member of the Institute of Electronics, Information, and Communication Engineers (IEICE) of Japan and the Society of Instrument and Control Engineers of Japan.

Kiyohiko Suzuki was born in Chiba, Japan, in 1977. He received the B.E. degree in information and computer science from Iwate University, Morioka, Japan, in 2000. He is presently working toward the M.E. degree at Iwate University. His research interest is in learning of neural networks.

Mr. Suzuki is a student member of the Institute of Electronics, Information and Communication Engineers (IEICE) of Japan.