Fast Training of Recurrent Networks Based on the EM Algorithm

Sheng Ma and Chuanyi Ji, *Member, IEEE*

***Abstract*— In this work, a probabilistic model is established for recurrent networks. The EM (expectation-maximization) algorithm is then applied to derive a new fast training algorithm for recurrent networks through mean-field approximation. This new algorithm converts training a complicated recurrent network into training an array of individual feedforward neurons. These neurons are then trained via a linear weighted regression algorithm. The training time has been improved by five to 15 times on benchmark problems.**

***Index Terms*— EM algorithm, fast, mean-field approximation, moving targets, probability model, recurrent networks.**

I. Nomenclature

N:	Total number of training samples.
L:	Number of hidden nodes.
d:	Input dimension.
$\vec{x}(n)$:	nth input vector.
$y(n+1)$:	$n+1$th desired output.
$\vec{z}(n+1)$:	$n+1$th desired hidden state.
$\vec{h}(n+1)$:	$n+1$th actual hidden state when previous desired hidden states $\vec{z}(n+1)$ and previous input $\vec{x}(n+1)$ are fed through.
$\vec{w}_j^{(1)}$:	Weight vector connecting the jth hidden unit to the inputs.
$\vec{w}^{(2)}$:	Weight vector at the second layer.
$w_j^{(2)}$:	Weight connecting the jth hidden unit to the output.
$\vec{w}_j^{(3)}$:	Weight vector connecting the jth hidden unit to other hidden units.
$g(\cdot)$:	Sigmoidal transfer function.
Θ:	Parameter set.
$\boldsymbol{E}(\cdot)$:	Expectation operation.
$P(\cdot)$:	Probability density function.
E_1:	Error between the desired and the actual ouputs of hidden units.
λ_1:	Weighting factor for E_1.
E_2:	Error between the desired and the actual ouputs of the network when the desired hidden values are fed.
λ_2:	Weighting factor for E_2.
E_3:	Error between the desired and the actual outputs of the network.
λ_3:	Weighting factor for E_3.
$\hat{\vec{z}}(n)$:	Expectation of $\vec{z}(n)$.
$\tilde{v}$:	Estimated v, where v is a arbitrary variable.
$\det(\cdot)$:	Determinant operation.
$\|\|\cdot\|\|$:	Euclidean norm.
I:	Identity matrix.
$\{\cdot\}_{n=1}^{N}$:	A collection of variable whose index n runs from one to N.
p:	Iteration index.

Manuscript received June 9, 1995; revised November 11, 1996 and September 30, 1997. This work was supported by the National Science Foundation (ECS-9 312 504).

The authors are with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180 USA.

Publisher Item Identifier S 1045-9227(98)00419-6.

I. Introduction

IN MANY important application areas like control and signal processing, a common class of important problems is how to model or identify underlying temporal processes. Most of these problems demand nonlinear adaptive systems which can learn from observed data. Recurrent neural networks with arbitrarily connected neurons have great potential to meet this demand, since they have been shown to be capable of approximating any dynamical systems [30]. However, how to train recurrent networks effectively remains an open problem, which hinders wide applications of recurrent networks in the aforementioned areas. In addition, since recurrent networks are very complicated, training methods are often developed in an *ad hoc* fashion.

A lot of algorithms have been developed to facilitate training recurrent networks. One type of such algorithms is based on finding good internal representations for recurrent neurons [9], [24], which is essentially motivated by methods using internal representations for feedforward networks [12], [16], [25], [26], [32]. The advantage of introducing internal representations is that it makes the training easier by constraining the search in a smaller solution space defined by internal representations, and allows prior knowledge to be incorporated when available [2], [22], [23], [28], [31]. Although the algorithms do facilitate training recurrent networks, the methods used to find internal representations are usually heuristic. On the other hand, a lot of methods exist which formulate training a feedforward network as a maximum likelihood problem through a probabilistic framework [18], [20]. Therefore, a natural question to ask is whether such a framework can be used to unify the idea of using internal representations. In our previous work [19], we have developed a probabilistic framework to formalize the ideas of internal representations for feedforward networks, and demonstrated that expectation-maximization (EM) algorithm [10], [15] can be used to derive a fast training algorithm. In this work, we will extend our work for feedforward networks to recurrent networks. Our goal is twofold. First, we would

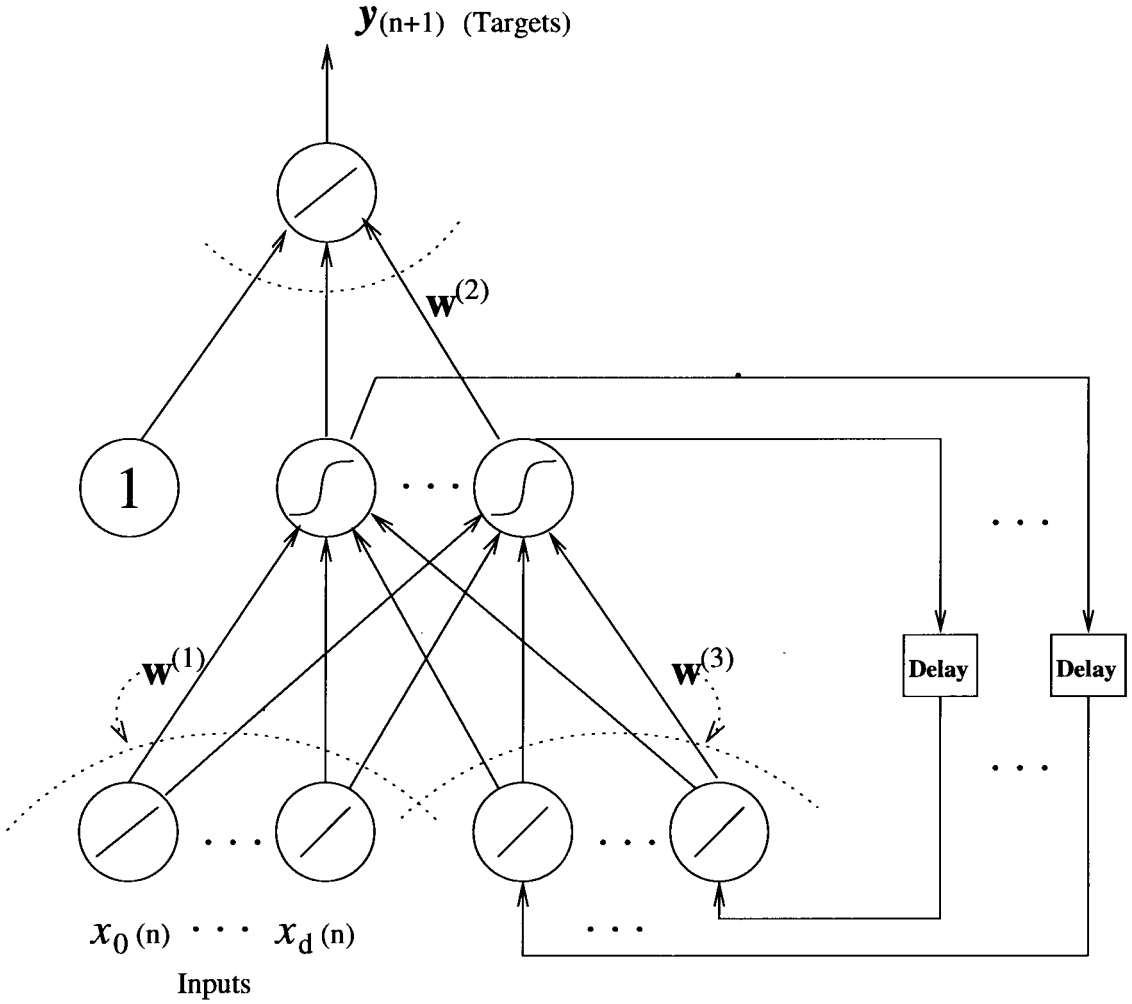


Fig. 1. The fully-connected-hidden-layer recurrent network.

like to develop a theoretical framework for training recurrent networks based on internal representations. Second, we would like to apply the EM algorithm to develop a fast training algorithm for recurrent networks. Our focus will be on fully connected recurrent networks. This differs from the previous work on fast training algorithms, such as recurrent cascade correlation [11], where recurrent connections were only feedbacks from individual neurons but no feedbacks among them.

Specifically, we will combine the ideas of internal representations used in Rowher's algorithm [24], and the EM framework [10], [15] to derive an EM-based algorithm for recurrent networks. We will first establish probabilistic models for (hidden) targets of recurrent hidden units. Then the EM algorithm as well as the mean-field approximation will be used to obtain an algorithm which decomposes training a complicated recurrent network into training a set of feedforward neurons. This makes it possible to apply fast training algorithms such as linear weighted regressions to train individual feedforward neurons, and thus reduce the training time drastically.

This paper is organized as follows. In Section II, we will describe our notations and the architecture of the network used. In Section III, we will briefly describe the EM algorithm. In Section IV, we will develop a probabilistic model for recurrent networks. In Section V, we will apply the EM algorithm to recurrent neural networks using this probabilistic model. Explanations of the new algorithm and discussions on other related work can be found in Section VI. We will give simulation results in Section VII, and then conclude the paper with Section VIII. Most of the mathematical derivations are left to the Appendixes.

II. Neural-Network Structure and Notation

In this paper, we consider discrete-time recurrent networks with a fully connected hidden layer as shown in Fig. 1. The network has one linear output,[1] d inputs, and L fully connected recurrent units. The recurrent (hidden) units are sigmoidal units with a transfer function

$$g(v) = \frac{1}{1+e^{-v}}. \tag{1}$$

We will use $\vec{w}_j^{(1)}$ and $\vec{w}_j^{(3)}$ to denote the weight vectors connecting the external inputs and the other recurrent units to the jth hidden unit, respectively, for $1 \le j \le L$. $W^{(1)}$ and $W^{(3)}$ are the weight matrices containing all $\vec{w}_j^{(1)}$ and $\vec{w}_j^{(3)}$,

[1] Extension to multiple outputs is straightforward.

respectively. $\vec{w}^{(2)}$ represents the weight vector connecting the recurrent hidden units to the output. $\{\vec{x}(n), y(n+1)\}_{n=1}^{N}$ is a set of N training samples, where $\vec{x}(n)$ is a $(d+1)\times 1$ input vector at the nth time unit,[2] and $y(n+1)$ is the corresponding desired output.[3]

III. The EM Algorithm

The method we will use to develop our algorithm is based on the EM algorithm for maximum likelihood estimation [10].

A. The Original EM Algorithm

Let D_I be a set of data directly observed for a set of random variables and Θ be a set of parameters characterizing the corresponding distributions of the random variables. Then $P(D_I|\Theta)$ is the probability density function of the observed data D_I given the set of paramenters Θ. The original maximum likelihood problem is to find an optimal set of parameters Θ_{opt} through maximizing the log likelihood of data, i.e., $\Theta_{\text{opt}} = \arg\max_{\Theta} \ln P(D_I|\Theta)$. Since directly maximizing the original log likelihood function can be extremely difficult, the EM algorithm has been introduced to simplify this process. In particular, a set of hidden (random) variables are introduced. The corresponding data for the hidden variables are called missing data denoted as D_{mis}, which can not be observed directly but can be estimated from the original data set D_I. D_I, which is called the incomplete data, together with the missing data form a complete data set D_c. As a result, the EM algorithm converts the original likelihood maximizing process into two steps: the E-step and the M-step.

In the E-step, the following conditional expectation (with respect to the hidden variables) is evaluated through

$$\begin{aligned} Q(\Theta|\Theta^p) &= \boldsymbol{E}\{\ln P(D_c|\Theta)|D_I, \Theta^p\} \\ &= \int P(D_{\text{mis}}|D_I, \Theta^p) \ln P(D_c|\Theta)\, dD_{\text{mis}} \end{aligned} \quad (2)$$

where Θ is the set of parameters to be obtained in the M-step. $P(\cdot)$ denotes probability density functions, and $\boldsymbol{E}\{\cdot\}$ stands for the expectation operation. The notation $\boldsymbol{E}\{\ln P(D_c|\Theta)|D_I, \Theta^p\}$ indicates that the conditional expectation $Q(\Theta|\Theta^p)$ is an expectation of the log likelihood for the complete data given incomplete data D_I and the set of previous parameters Θ^p, hence it is a function of Θ.

In the M-step, the new set of parameters Θ^{p+1} are found by maximizing the function $Q(\Theta|\Theta^p)$, i.e.,

$$\Theta^{p+1} = \arg\max_{\Theta} Q(\Theta|\Theta^p). \quad (3)$$

The algorithm alternates between the E-step and the M-step until a certain convergence criterion is satisfied. Local convergence of the EM Algorithm is guaranteed [10]. That is, after every (E-M) iteration, the original log likelihood will increase

$$\ln P(D_I|\Theta^p) \le \ln P(D_I|\Theta^{p+1}). \quad (4)$$

[2] The bias corresponds to an input $x_0(n) = 1$.

[3] One unit of time delay occurs between an input vector and its corresponding target due to the delay in a recurrent hidden unit.

Eventually an optimal set of parameters can be obtained as a local maximum of the original log likelihood function $\ln P(D_I|\Theta)$.

B. Choice of Hidden Random Variables

In our work, we will apply the EM algorithm in a supervised learning environment as [15]. In particular, we choose the hidden random variables to be the desired hidden states (or so called hidden targets) of a recurrent network denoted as $\vec{z}$. The complete data set therefore contains the labeled training samples and the desired hidden states, i.e, $D_c = \{\vec{x}(n), y(n+1), \vec{z}(n+1)\}_{n=1}^{N}$, where $\{\vec{z}(n+1)\}_{n=1}^{N}$ is the set of missing data. The incomplete data set contains $\{\vec{x}(n), y(n+1)\}_{n=1}^{N}$. For simplicity, we use the notation $\{\vec{x}\}, \{y\}$, and $\{\vec{z}\}$ to represent $\{\vec{x}(n)\}_{n=1}^{N}, \{y(n+1)\}_{n=1}^{N}$, and $\{\vec{z}(n+1)\}_{n=1}^{N}$, respectively, and Θ to represent all the weights in the network, i.e., $\Theta = \{\vec{w}_j^{(1)}, \vec{w}^{(2)}, \vec{w}_j^{(3)}\}$ for all $1 \le j \le L$. Using the above notations, the conditional expectation of the log likelihood of the complete data, $Q(\Theta|\Theta^p)$ as defined in (2), can be expressed as

$$\begin{aligned} Q(\Theta|\Theta^p) = \ln P(\{\vec{x}\}|\Theta) + \int P(\{\vec{z}\}|\{y\}, \{\vec{x}\}, \Theta^p) \\ \cdot \ln P(\{y\}, \{\vec{z}\}|\{\vec{x}\}, \Theta)\, d\{\vec{z}\}. \end{aligned} \quad (5)$$

IV. Probabilistic Models

In order to apply the EM framework to recurrent networks, we need to establish a probabilistic model for recurrent networks. Specifically, we need to obtain $P(\{\vec{z}\}|\{y\}, \{\vec{x}\}, \Theta)$ and $P(\{y\}, \{\vec{z}\}|\{\vec{x}\}, \Theta)$, which are the probability density function of desired hidden states $\{\vec{z}\}$ given $(\{y\}, \{\vec{x}\}, \Theta)$ and the joint probability density function of desired outputs $\{y\}$ and desired hidden states $\{\vec{z}\}$ given $(\{\vec{x}\}, \Theta)$, respectively.

Since the output of the recurrent network and hidden states at the current time step depend only on the hidden states at the previous time step, it is reasonable to use the Markov property to model recurrent networks. That is, the probability density functions of the desired hidden states and the desired output of the network at the $n+1$th time step only depend on the previous desired hidden states $\vec{z}(n)$ and the previous inputs $\vec{x}(n)$

$$\begin{aligned} &P(\vec{z}(n+1)|\{\vec{z}\}_{k=1}^{n}, \{t\}, \{\vec{x}\}, \Theta) \\ &\quad = P(\vec{z}(n+1)|y(n+1), \vec{z}(n), \vec{x}(n), \Theta) \end{aligned} \quad (6)$$

and

$$\begin{aligned} &P(y(n+1), \vec{z}(n+1)|\{\vec{z}(k)\}_{k=1}^{n}, \{y(k)\}_{k\neq n+1}, \{\vec{x}\}, \Theta) \\ &\quad = P(y(n+1), \vec{z}(n+1)|\vec{z}(n), \vec{x}(n), \Theta). \end{aligned} \quad (7)$$

These two equations will lead to the following joint probability density functions:[4]

$$\begin{aligned} &P(\{\vec{z}\}|\{t\}, \{\vec{x}\}, \Theta) \\ &\quad = \prod_{n=1}^{N} P(\vec{z}(n+1)|\vec{z}(n), y(n+1), \vec{x}(n), \Theta) \end{aligned} \quad (8)$$

[4] The initial states $\vec{z}(1)$ are assumed to be independent of the parameters Θ and the inputs $\{\vec{x}(n)\}$. Furthermore, we assume $\vec{z}(1) = \vec{h}(1)$ with probability 1, where $\vec{h}(1)$ are the given initial hidden states.

and

$$P(\{t\},\{\vec{z}\}|\{\vec{x}\},\Theta) = \prod_{n=1}^{N} P(y(n+1),\vec{z}(n)f|\vec{z}(n),\vec{x}(n),\Theta). \quad (9)$$

Therefore, obtaining $P(\{\vec{z}\}|\{t\},\{\vec{x}\},\Theta)$ and $P(\{t\},\{\vec{z}\}|\{\vec{x}\},\Theta)$ is reduced to finding $P(\vec{z}(n+1)|y(n+1),\vec{z}(n),\vec{x}(n),\Theta)$ and $P(y(n+1),\vec{z}(n+1)|\vec{z}(n),\vec{x}(n),\Theta)$.

In this work, we focus on investigating the simple probabilistic model, Gaussian distributions.[5] In particular, we choose the conditional distribution of hidden variables $\vec{z}(n+1)$ given the previous desired hidden states $\vec{z}(n)$ and the previous inputs $\vec{x}(n)$, and the conditional distribution of the output $y(n+1)$ given the current desired hidden states $\vec{z}(n+1)$ as[6]

$$P(\vec{z}(n+1)|\vec{z}(n),\vec{x}(n),\Theta) = B_1 \exp(-\lambda_1 E_1(n+1)) \quad (10)$$

$$P(y(n+1)|\vec{z}(n+1),\Theta) = B_2 \exp(-\lambda_2 E_2(n+1)) \quad (11)$$

where B_1 and B_2 are normalization factors, which can be expressed as $B_1 = \sqrt{(\pi/\lambda_1)^L}$ and $B_2 = \sqrt{(\pi/\lambda_2)}$. λ_1 and λ_2 are weighting factors for $E_1(n+1)$ and $E_2(n+1)$, respectively. $E_1(n+1)$ can be expressed as

$$E_1(n+1) = ||\vec{z}(n+1) - \vec{h}(n+1)|^2 \quad (12)$$

where the jth element of $\vec{h}(n+1)$ for $1 \le j \le L$ is

$$h_j(n+1) = g(\vec{x}(n)^T \cdot \vec{w}_j^{(1)} + \vec{z}(n)^T \cdot \vec{w}_j^{(3)}). \quad (13)$$

$\vec{h}(n+1)$ contains the actual outputs of hidden units when the previous desired hidden states $\vec{z}(n)$ as well as inputs $\vec{x}(n)$ are the (equivalent) inputs to a recurrent hidden unit as shown in Fig. 2(a). Therefore, $E_1(n+1)$ measures the error between the desired hidden states $\vec{z}(n+1)$ and $\vec{h}(n+1)$ [see Fig. 2(a)]. Likewise, $E_2(n+1)$ characterizes the error between the desired and the actual output of the network when the desired hidden states are used as inputs to the output layer [see Fig. 2(b)], which is determined by

$$E_2(n+1) = (y(n+1) - \vec{z}(n+1)^T \cdot \vec{w}^{(2)})^2. \quad (14)$$

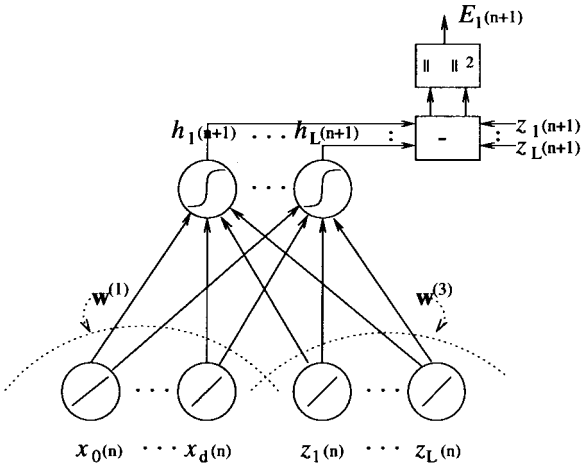
To obtain the joint conditional distribution of the output and the desired hidden states given the previous inputs and the previous desired hidden states, we further assume that the probability of the current output given both the desired hidden states and the inputs depends only on the current desired hidden states, i.e.,

$$P(y(n+1)|\vec{z}(n+1),\vec{x}(n),\Theta) = P(y(n+1)|\vec{z}(n+1),\Theta). \quad (15)$$

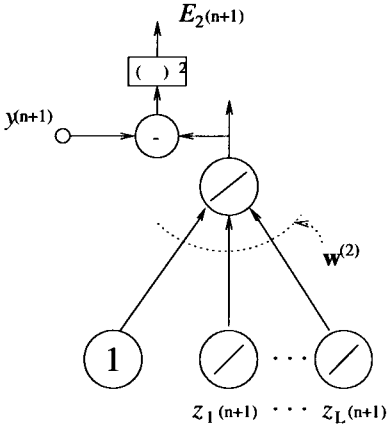
This is a valid assumption for the recurrent network described in Section II, because the output of recurrent networks

[5]The motivation for using Gaussian distributions stems from the fact that they correspond to commonly used quadratic error functions. They were successfully used by Levin *et al.* [18] and Mackey [20] in feedforward networks to model the outputs of a feedforward network given the inputs. As will be seen later, they facilitate the analysis in this work.

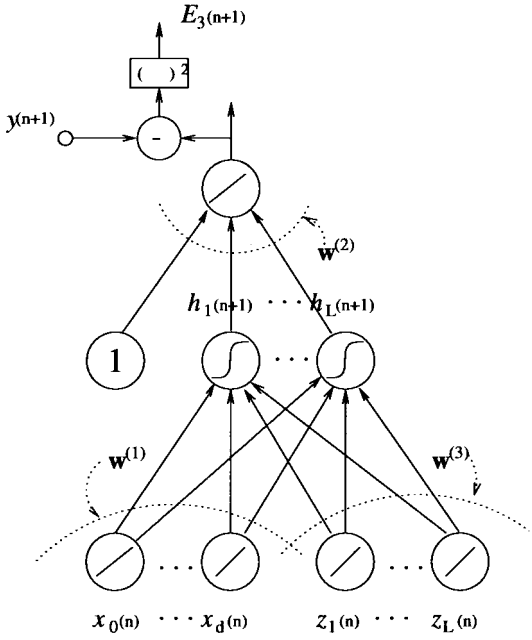
[6]It will be clear later that these two conditional distributions are building blocks for compute $P(\vec{z}(n+1)|y(n+1),\vec{z}(n),\vec{x}(n),\Theta)$.



(a)



(b)



(c)

Fig. 2. (a) Computation of $E_1(n+1)$. (b) Computation of $E_2(n+1)$. (c) Computation of $E_3(n+1)$.

would depend on current desired hidden states alone if they were known. Thus the joint distribution can be obtained as

$$\begin{aligned} &P(y(n+1),\vec{z}(n+1)|\vec{z}(n),\vec{x}(n),\Theta) \\ &\quad = P(y(n+1)|\vec{z}(n+1),\vec{z}(n),\vec{x}(n),\Theta)P(\vec{z}(n+1)|\vec{z}(n) \\ &\quad\quad \vec{x}(n),\Theta) \\ &\quad = P(y(n+1)|\vec{z}(n+1),\Theta)P(\vec{z}(n+1)|\vec{z}(n),\vec{x}(n),\Theta) \\ &\quad = A_{yz}\exp\left(-\lambda_1 E_1(n+1)-\lambda_2 E_2(n+1)\right) \end{aligned} \tag{16}$$

where the normalization coefficient A_{yz} satisfies

$$A_{yz}=\sqrt{\frac{\lambda_2}{\pi}\left(\frac{\lambda_1}{\pi}\right)^L}. \tag{17}$$

Furthermore, $P(y(n+1)|\vec{z}(n),\vec{x}(n),\Theta)$ can also be chosen as a Gaussian distribution with a certain constrain shown later, i.e.,

$$P(y(n+1)|\vec{z}(n),\vec{x}(n),\Theta)=A_y\exp\left(-\lambda_3 E_3(n+1)\right) \tag{18}$$

where $E_3(n+1)$ is a quadratic energy function measuring the error at the output of the network when the previous desired hidden states are used as inputs [see Fig. 2(c)], i.e.,

$$E_3(n+1)=(y(n+1)-\vec{h}(n+1)^T\cdot\vec{w}^{(2)})^2. \tag{19}$$

λ_3 is a weighting factor for $E_3(n+1)$, and is related to the variance of noise in the training set. A_y is a normalization coefficient satisfying

$$A_y=\sqrt{\frac{\lambda_3}{\pi}}. \tag{20}$$

In addition, the probability density functions given in (16) and (18) should satisfy the relationship

$$\begin{aligned} &P(y(n+1)|\vec{z}(n),\vec{x}(n),\Theta) \\ &\quad = \int_{-\infty}^{+\infty} P(y(n+1),\vec{z}(n+1)|\vec{z}(n),\vec{x}(n),\Theta) \\ &\quad\quad \cdot d\vec{z}(n+1) \end{aligned} \tag{21}$$

which leads to the following constraint:[7]

$$\lambda_3=\frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2||\vec{w}^{(2)}||^2}. \tag{22}$$

Intuitively, the constraint in (22) provides a relationship among the variance of noise in a training set (λ_3), the variance of noise in the desired hidden states and the relative importance of $E_1(n+1)$ and $E_2(n+1)$ (λ_1 and λ_2). Notice the following equation for conditional probabilities:

$$\begin{aligned} &P(\vec{z}(n+1)|y(n+1),\vec{z}(n),\vec{x}(n),\Theta) \\ &\quad = \frac{P(y(n+1),\vec{z}(n+1)|\vec{z}(n),\vec{x}(n),\Theta)}{P(y(n+1)|\vec{z}(n),\vec{x}(n),\Theta)}. \end{aligned} \tag{23}$$

Therefore by putting (16) and (18) into the above equation, we can obtain[8]

$$\begin{aligned} &P(\vec{z}(n+1)|y(n+1),\vec{z}(n),\vec{x}(n),\Theta) \\ &\quad = A_z\exp\left(-\tfrac{1}{2}(\vec{z}(n+1)-\hat{\vec{z}}(n+1))^T\right. \\ &\quad\quad \left.\cdot\,\Sigma^{-1}(\vec{z}(n+1)-\hat{\vec{z}}(n+1))\right) \end{aligned} \tag{24}$$

[7]Detailed derivations are given in Appendix 1.

[8]Detailed derivations are given in Appendix 1.

where the normalizaiton factor A_z satisfies $A_z = 1/\sqrt{(2\pi)^L\det(\Sigma)}$. Σ is a $L\times L$ (covariance) matrix whose inverse satisfies $\Sigma^{-1}=2(\lambda_1 I+\lambda_2\vec{w}^{(2)}\cdot\vec{w}^{(2)^T})$. I is an $L\times L$ identity matrix. $\hat{\vec{z}}(n+1)$ is the conditional expectation of $\vec{z}(n+1)$ conditioned upon $y(n+1),\vec{z}(n)$, and $\vec{x}(n)$. The jth element of $\hat{\vec{z}}(n+1)$ can be expressed as

$$\hat{z}_j(n+1)=h_j(n+1)+\frac{\lambda_2 w_j^{(2)}}{||\lambda_1+\lambda_2||\vec{w}^{(2)}||^2}e(n+1) \tag{25}$$

with

$$e(n+1)=y(n+1)-\vec{h}(n+1)^T\cdot\vec{w}^{(2)} \tag{26}$$

where $\vec{h}$ is defined by (13). Finally, by substituting (24) and (16) into (8) and (9), we obtain the two distributions we need

$$\begin{aligned} &P(\{\vec{z}\}|\{t\},\{\vec{x}\},\Theta) \\ &\quad = \prod_{n=1}^{N} A_z\exp\left(-\tfrac{1}{2}(\vec{z}(n+1)-\hat{\vec{z}}(n+1))^T\right. \\ &\quad\quad \left.\cdot\,\Sigma^{-1}(\vec{z}(n+1)-\hat{\vec{z}}(n+1))\right) \end{aligned} \tag{27}$$

and

$$\begin{aligned} &P(\{t\},\{\vec{z}\}|\{\vec{x}\},\Theta) \\ &\quad = \prod_{n=1}^{N} A_{yz}\exp\left(-\lambda_1 E_1(n+1)-\lambda_2 E_2(n+1)\right) \end{aligned} \tag{28}$$

where $E_1(n+1)$ and $E_2(n+1)$ are defined by (12) and (14), respectively.

V. Applying the EM Algorithm to Recurrent Networks

A. E-Step and Mean-Field Approximation

Applying the established probabilistic models to the EM algorithm, we obtain the expectation of the log likelihood for the complete data after inserting (24) and (16) into (5)

$$\begin{aligned} Q(\Theta|\Theta^p) = \ln P(\{\vec{x}\}|\Theta)+\sum_{k=1}^{N}\int\prod_{n=1}^{N} \\ \cdot\,P(\vec{z}(n+1)|y(n+1),\vec{z}(n),\vec{x}(n),\Theta^p) \\ \cdot\ln P(y(k+1),\vec{z}(k+1)|\vec{z}(k),\vec{x}(k),\Theta) \\ \cdot\,d\{\vec{z}\}. \end{aligned} \tag{29}$$

The task of the E-step is to evaluate $Q(\Theta|\Theta^p)$, where the superscript p indicates that the evaluation is at the pth iteration. It is very difficult, however, to evaluate this integral directly due to the nonlinearity of sigmoidal functions. Therefore, we use the mean-field approximation to obtain an approximation for the original integral. The mean-field approximation has been used widely in simplifying the training process of Boltzmann machines [13], sigmoid belief networks [27], Markov random fields, and hidden Markov models (HMM's) [35].

The mean-field approximation we will use is called the self-consistent mean-field approximation described in [35].

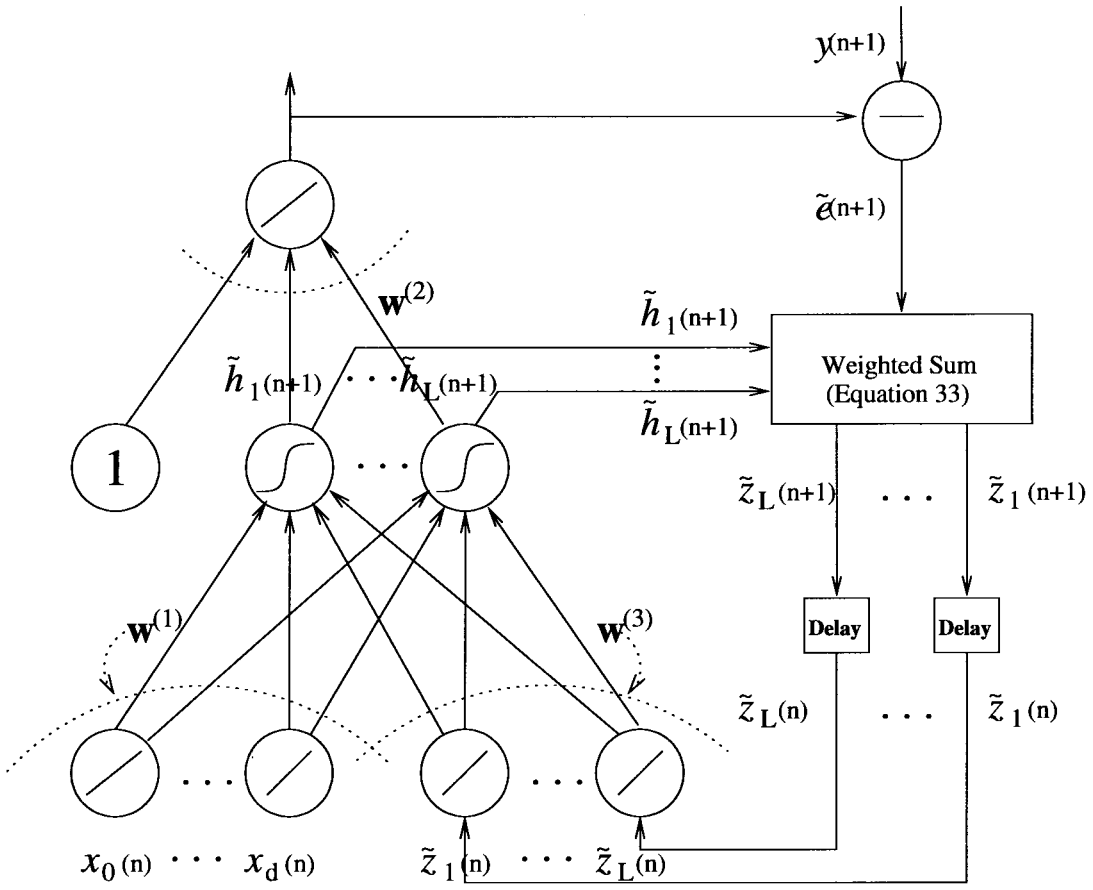


Fig. 3. Iterative computation of $\tilde{z}(n)$.

The main idea of the self-consistent mean-field approximation can be described as follows. Consider a random variable u_i which depends on other random variables $u_j, j \neq i$. When evaluating the mean of u_i, the dependence of u_j's on u_i can be approximated by dependence of u_i on mean of u_j's, this can be described as

$$\begin{aligned} \boldsymbol{E}u_i &= \iint u_i P(u_i|u_j)P(u_j)\,du_i\,du_j \\ &\approx \int u_i P(u_i|\boldsymbol{E}u_j)\,du_i \end{aligned} \tag{30}$$

which is mathematically equivalent to letting

$$P(u_i|u_j) \approx P(u_i|\boldsymbol{E}u_j) \tag{31}$$

as mentioned previously. In other words, (31) indicates that the condition on the original random variable can be approximated by the condition on the expected value of the random variable.

It should be noted that the self-consistent mean-field approximation is only one of many ways to approximate the dependencies among random variables. A better approximation can be obtained by approximating the conditional probability given in the above equation using another probability which maximizes the Kullback–Liebler divergence [8]. Interested readers may find the details on this subject in [27].

By applying the idea of self-consistent mean-field approximation to our case, we have an approximation of the following conditional probability:

$$\begin{aligned} &P(\vec{z}(n+1)|y(n+1),\vec{z}(n),\vec{x}(n),\Theta^p) \\ &\approx P(\vec{z}(n+1)|y(n+1),\tilde{\vec{z}}(n),\vec{x}(n),\Theta^p) \end{aligned} \tag{32}$$

where $\tilde{z}_j(n)$ is the jth element of $\tilde{\vec{z}}(n)$, which is an approximation to the expectation of $\vec{z}(n)$ by the mean-field approximation, and can be computed recursively through the following equations:

$$\tilde{h}_j(n+1) = g(\vec{x}(n)^T \cdot \vec{w}_j^{(1)} + \tilde{\vec{z}}(n)^T \cdot \vec{w}_j^{(3)}) \tag{33}$$

$$\tilde{e}(n+1) = y(n+1) - \tilde{\vec{h}}(n+1)^T \cdot \vec{w}^{(2)^p} \tag{34}$$

$$\tilde{z}_j(n+1) \approx \tilde{h}_j(n+1) + \frac{\lambda_2 w_j^{(2)^p}}{\lambda_1 + \lambda_2 ||\vec{w}^{(2)^p}||}\tilde{e}(n+1) \tag{35}$$

for every $1 \leq j \leq L$ and $1 \leq n \leq N$. $\tilde{\vec{h}}(n+1)$ is a $L \times 1$ vector whose jth element is defined by (33). The initial condition $\tilde{\vec{z}}(1)$ is assumed to be the actual initial states $\vec{h}(1)$. These recursive relations in (35) say that the estimated mean value of the current hidden target is equal to a correct term plus the state ouputs of feeding through the estimated mean value of the previous hidden targets and previous inputs. This can be further illustrated by an equivalent feedforward network in Fig. 3. Therefore, using the idea from the mean-field approximations, the conditional probability given in (32) is approximated by replacing the condition on $\vec{z}(n)$ by its

estimated expected value $\tilde{\vec{z}}(n)$, where $\tilde{\vec{z}}(n)$ can be obtained through the recursive relations.

Equation (29) is then reduced to[9]

$$Q(\Theta|\Theta^p) = \boldsymbol{E}_z\{\ln P(\{y\},\{\vec{z}\},\{\vec{x}\}|\Theta)||\{y\},\{\vec{x}\},\Theta^p\} \approx E_c - E_p - E_h - E_o \quad (36)$$

where E_c, E_p, E_h, E_o can be expressed explicitly as[10]

$$E_c = 0.5N\ln\left(\frac{\lambda_2}{\pi}\left(\frac{\lambda_1}{\pi}\right)^L\right) - \lambda_1\sum_n\sum_j \sigma^2_{z_j(n)} + \ln P(\{\vec{x}\}|\Theta) \quad (37)$$

$$E_p = \lambda_2 N \vec{w}^{(2)^T}\cdot\Sigma\cdot\vec{w}^{(2)} \quad (38)$$

$$E_h = \lambda_1\sum_n\sum_j(\tilde{z}_j(n) - g(\vec{x}(n)^T\cdot\vec{w}_j^{(1)} + \tilde{\vec{z}}(n)^T\cdot\vec{w}_j^{(3)}))^2 \quad (39)$$

$$E_o = \lambda_2\sum_n(\vec{w}^{(2)^T}\cdot\tilde{\vec{z}}(n) - y(n))^2. \quad (40)$$

$\sigma^2_{z_j(n)}$ and Σ are the variance and the covariance matrix of the jth element of desired hidden state $z_j(n)$ and desired hidden states $\vec{z}(n)$, respectively, which only depend on the previous parameters Θ^p.

As a result, the evaluation of the expectation of the complete log likelihood in the E-step is reduced to finding the expectation of the desired hidden states through (33)–(35). The expectation of the current desired hidden states can be computed easily through evaluating activations of hidden units and the weighted error at the output of the network when the previous expectation of the desired hidden states and external inputs are used together as effective inputs to a (recurrent hidden) unit (see Fig. 3). Therefore, computations needed in the E-step only involve calculating the "output" of the network using the previous $\tilde{\vec{z}}$ and $\vec{x}$ as inputs through the equivalent (feedforward) network given in Fig. 3.

B. The M-Step

In the maximization step of the EM algorithm, new weights need to be obtained through maximizing the expectation of the log likelihood for the complete data, $Q(\Theta|\Theta^p)$, given in (36).

To see how the mean-field approximation simplifies the M-step, let us examine (37) through (40). If we assume $P(\vec{x}|\Theta)$ is independent of Θ, then E_c only depends on λ_1 and λ_2. To obtain a simple result, we assume λ_1 and λ_2 are constants,[11] hence E_c is also a constant. E_h measures the error between the actual outputs of hidden units and the expectations of the desired hidden states. It is a function of $\vec{w}_j^{(1)}$ and $\vec{w}_j^{(3)}$ $(1 \le j \le L)$ which are the weight vectors connecting the jth unit to its aforementioned effective inputs [see Fig. 4(a)]. E_p and E_o only depend on $\vec{w}^{(2)}$ which is a weight vector connecting the hidden units to the output unit. E_o characterizes the error between the actual and desired outputs of the network when the expectation of the desired hidden states are fed as inputs to the output unit [See Fig. 4(b)]. Therefore, maximizing $Q(\Theta|\Theta^p)$ in M-step reduces to solving the following two separated minimization problems:

$$(\vec{w}_j^{(1)^{p+1}}, \vec{w}_j^{(3)^{p+1}}) = \arg\min_{w_j^{(1)}, w_j^{(3)}}\sum_n(\tilde{z}_j(n) - g(\vec{x}(n)^T\cdot\vec{w}_j^{(1)} + \tilde{\vec{z}}(n)^T\cdot\vec{w}_j^{(3)}))^2 \quad (41)$$

where $\vec{w}_j^{(1)^{p+1}}$ and $\vec{w}_j^{(3)^{p+1}}$ stand for the new weight vectors connecting the jth unit to the inputs and the other recurrent hidden units, for $1 \le j \le L$. This step is equivalent to training individual sigmoidal units in parallel. In addition, we have

$$\vec{w}^{(2)^{p+1}} = \arg\min_{\vec{w}^{(2)}}(E_o + E_p) = \arg\min_{\vec{w}^{(2)}}\lambda_2\sum_n(\vec{w}^{(2)^T}\tilde{\vec{z}}(n) - y(n))^2 + E_p \quad (42)$$

where $\vec{w}^{(2)^{p+1}}$ contains the new weights connecting the hidden units to the output of the network. Furthermore, by expanding (38), we obtain

$$E_p = \frac{\lambda_2 N||\vec{w}^{(2)}||^2}{2(\lambda_1 + \lambda_2||\vec{w}^{(2)^p}||^2)} + \frac{\lambda_2 N(||\vec{w}^{(2)}||^2||\vec{w}^{(2)^p}||^2 - (\vec{w}^{(2)})^T(\vec{w}^{(2)^p})(\vec{w}^{(2)^p})^T\vec{w}^{(2)})}{2\lambda_1(\lambda_1 + \lambda_2||\vec{w}^{(2)^p}||^2} \quad (43)$$

where $\vec{w}^{(2)^p}$ denotes the corresponding weights at the previous step. Intuitively, the first term on the right side of (43) minimizes the magnitude of the weights $\vec{w}^{(2)}$, whereas the second term will push $\vec{w}^{(2)}$ toward $\vec{w}^{(2)^p}$. Equation (42) minimizes a quadratic function of $\vec{w}^{(2)}$ which has a closed form solution. Therefore, the major computational cost of the M-step consists of training individual sigmoidal neurons given by (41).

C. A Fast Training Algorithm for a Single Neuron

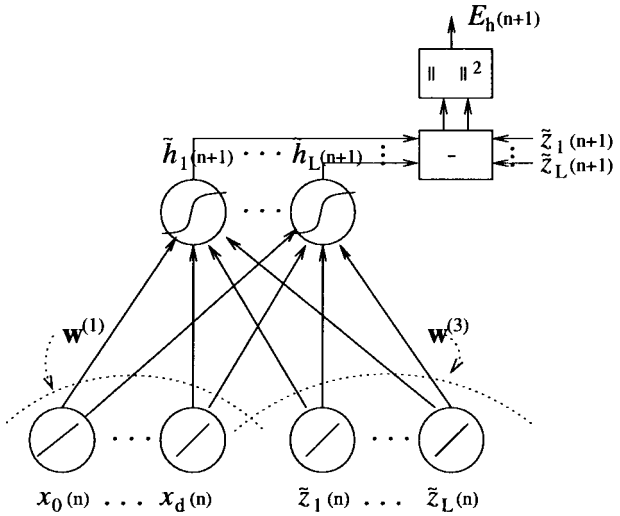
Several fast training algorithms exist to train a single sigmoidal neuron, which include the ones by Breiman *et al*. [6] and by Shadafan *et al*. [29]. However, the former is better suited for a neuron with a ramp activation function and may not be appropriate for approximating subtle features in a smooth target function, while the latter one uses complicated heuristic criteria to assist learning. In what follows, we will develop an algorithm suitable for our case, which is essentially a linear weighted regression algorithm.

Recall that the training data used to train a single (the jth) hidden neuron is $\{\vec{x}(n), \tilde{\vec{z}}(n), \tilde{z}_j(n+1)\}_{n=1}^N$, where $\vec{x}(n), \tilde{\vec{z}}(n)$ are the inputs and $\tilde{z}_j(n+1)$ is their corresponding target. The task is to find the optimum weight vectors given by (41). For notational convenience, in the rest of the derivations in this section, we will use w, u, and v to represent weights, inputs, and targets, i.e., we let $w^T = (\vec{w}_j^{(1)^T}, \vec{w}_j^{(3)^T}), u^T(n) =$

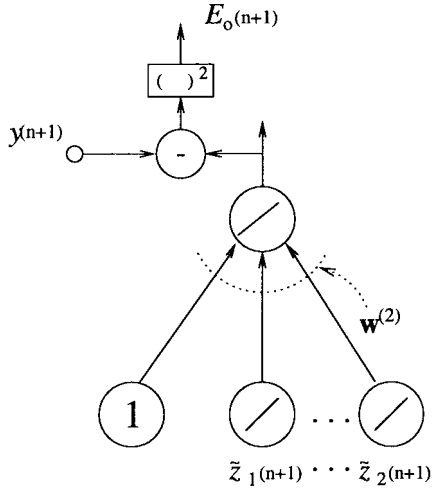
[9] Detailed derivations are given in Appendix 2.

[10] It will soon become clear that the subscripts "*c*," "*p*," "*h*," and "*o*" indicate that the corresponding error terms are a constant, a penalty term, error terms for the expectation of the desired hidden states and for outputs, respectively.

[11] In general, when GEM (generalized EM algorithm [10]) is used, λ_1 and λ_2 can also be treated as parameters and be updated in the M-step.



(a)



(b)

Fig. 4. (a) Computation of $E_h(n+1)$. (b) Computation of $E_o(n+1)$.

$(\vec{x}(n)^T, \tilde{\vec{z}}(n)^T)$ and $v(n) = \tilde{z}_j(n+1)$. Then (41) is equivalent to

$$w_{\text{opt}} = \arg\min_w \sum_{n=1}^{N} (v(n) - g(w^T \cdot u(n)))^2. \quad (44)$$

Due to the nonlinearity of the sigmoidal transfer function, when the weights in (44) are trained directly using gradient descent, no significant improvement on training time has been observed. However, if targets were available before the sigmoidal transfer function, the problem would become a linear optimization problem which could be solved easily. Based on this observation, the method we use to speed up training is to convert the original targets ($v(n)$'s) to the equivalent targets before the transfer function with proper weightings resulting from a Taylor series expansion,[12] i.e., to expand $g(w \cdot u(n))$ about $g^{-1}(v(n))$ using Taylor series

$$g(w^T \cdot u(n)) = v(n) + c(n)(w^T \cdot u(n) - g^{-1}(v(n))) + o(|w^T \cdot u(n) - g^{-1}(v(n))|) \quad (45)$$

where $o(\cdot)$ represents higher order terms and $c(n) = g'(g^{-1}(v(n)))$. When the difference between $w^T \cdot u(n)$ and $g^{-1}(v(n))$ is small, the higher order terms can be neglected.[13] Putting (45) into (44), we have

$$w_{\text{opt}} \approx \arg\min_w \sum_{n=1}^{N} c(n)^2 (w^T \cdot u(n) - g^{-1}(v(n)))^2 \quad (46)$$

where the function to be optimized is quadratic in terms of w. Therefore the original nonlinear optimization problem has been reduced to a linear weighted regression problem which has a closed form solution:[14]

$$w_{\text{opt}} = (U^T G^T G U)^{-1} U^T G^T V \quad (47)$$

where U is an $N \times (d+L+1)$ matrix with $(U)_{n,k} = u_k(n)$ (the kth element of $u(n)$ is defined previously), G is a $N \times N$ matrix with $(G)_{n,m} = 0$ if $n \neq m$ and $(G)_{n,n} = c(n)$, and V is an $N \times 1$ vector with $(V)_n = v(n)$.

The solution to linear weighted regressions can also be obtained iteratively using least-mean-square or recursive-least-square methods when on-line training is needed [15], [29].

D. Summary of the Algorithm

Combining the E-step and the M-step, we obtain our algorithm for recurrent networks called EMR (EM algorithm for recurrent networks). Specifically, the training procedures can be summarized as follows:

Randomly initialize $\vec{w}_j^{(1)}, \vec{w}_j^{(3)}$ and $w_j^{(2)}$ for $1 \leq j \leq L$.
E-step:
Compute the expectation of the desired hidden states $\tilde{\vec{z}}(n)$ recursively according to (33)–(35) (illustrated by Fig. 3).
M-step:
a) Compute $\vec{w}_j^{(1)}$ and $\vec{w}_j^{(3)}$ given by (41) through linear weighted regressions ((47)).
b) Compute $\vec{w}^{(2)}$ by solving (42).
Go back to the E-step until certain convergence criteria are satisfied.

VI. Discussions

A. More Explanations on the Algorithm

Our new (EMR) algorithm contains two steps. In the E-step, the approximated expectation of the desired hidden states

[12] This is similar to using the first-order derivative of the sigmoidal function as a distance measure to replace the first heuristic assumption in [29].

[13] To see how the assumption used above can be satisfied for our case, we recall that $\tilde{z}_j(n)$ can be expressed through (35). Therefore, if the second term in this expression is small at each M-step, the assumption is valid. This can be made true through controlling λ_1/λ_2. In testing our algorithm on the bench mark problems, we have observed that the higher order terms remain to be small in all the experiments.

[14] We assume that $U^T G^T G U$ is invertible.

$\tilde{\tilde{z}}$ is evaluated through a closed-form expression obtained by the mean-field approximation. Specifically, $\tilde{\tilde{z}}$ consists of two major components: the output of a recurrent hidden unit based on the expectation of $\tilde{\tilde{z}}$ in the previous time step, and the training error when the previous expectation of the desired hidden states and inputs are used as inputs to the recurrent hidden units (see Fig. 3). In the M-step, the expectation of the desired hidden states serve as the targets for the current states of the recurrent neurons as well as the inputs to these neurons at the next time step as illustrated in Fig. 4(a). Meanwhile, the expectation of the desired hidden states serve as the inputs to the output unit as illustrated in Fig. 4(b). In other words, at each E-M step, the original recurrent network is unfolded into equivalent feedforward networks shown in Fig. 4(a) and (b). Therefore, the hidden targets at the hidden layer decompose finding the weights for the recurrent network into finding the weights for a set of feedforward networks which can be approximated through weighted linear regressions. This speeds up learning.

The simplification of the E-step and the M-step results from the mean-field approximation. Intuitively, this approximation is acceptable if it directs the algorithm toward finding good hidden states as training proceeds. This means that the over- or under-estimated desired hidden states at one particular EM step due to the mean-field approximation should get corrected in subsequent iterations. Our results, which will be shown later, shows empirically the effectiveness of the mean-field approximation in terms of finding a good solution. Systematic experiments have been done in [27] to show the power of the mean-field approximation for sigmoidal belief networks. Theoretically, it has also been shown in [27] that using the mean-field approximation, the algorithm is essentially maximizing a lower bound of the original log-likelihood function.

B. Related Works

To relate our work to existing methods, we would like to point out that our work is most closely related to the ideas of "moving targets" for recurrent networks by Rohwer [24], in which an error function was first minimized to find targets for the values before the transfer function of recurrent hidden nodes, and then the targets were used to find weights. There are mainly two differences between the "moving target" algorithm and our algorithm. First, in terms of selecting hidden targets, Rohwer chose the value before the transfer function of recurrent hidden units, while we choose the one after the transfer function. Although finding targets before the transfer function can simplify the process of finding the weights of the recurrent networks, it complicates the process of finding the desired targets, which has to be computed iteratively through gradient descent. When the desired hidden states are used as hidden variables (the values after transfer function) in our algorithm, the expectation of the desired hidden states can be recursively estimated easily in the E-step, and then linear weighted regression can be applied to simplify the computation in the M-step. Therefore, our algorithm is much faster. From a theoretical stand point, our algorithm is derived through a probabilistic framework, while Rohwer's algorithm was based on an optimization framework.

The EM algorithm has also been used in Baum–Welch's algorithm for training HMM's [3], and in the input–output HMM, a more general version of HMM, by Bengio *et al*. [4]. In those two algorithms, both forward and backward iterations are needed, whereas our EMR algorithm only requires forward iterations. This is due to the mean-field approximation used, which can also reduce the computational complexity for HMM as shown in [35]. In terms of architecture, the hidden states of HMM and the input–output HMM linearly depend on previous hidden states. The hidden states of the recurrent networks depend on previous states nonlinearly, which are more general than HMM's. As for the formulations of the algorithms are concerned, Bengio *et al*. [4], adopted the idea of mixture models as described by Jordan and Jacobs [15], and chose the "paths" as hidden variables and used "softmax" models. We choose the desired hidden states as hidden variables and use Gaussian models.

One of the most important examples investigated in Bengio *et al*. [4] is how to track the long-term dependence in the data. Extensive study on this problem is beyond the scope of this work. We expect, however, that the performance of our algorithm would be similar to that of BPTT's, if no prior knowledge is incorporated. However, the formulation of EMR does provide an opportunity to incorporate prior knowledge to learn the long-term dependence in the data. The prior knowledge, if available, may be incorporated into the formulation for hidden targets as well as in the initial conditions to further facilitate training.

The EM algorithm has also been applied to training Boltzmann machines [1], [7], where the generalized EM algorithm is usually used which makes small adjustments on weights.

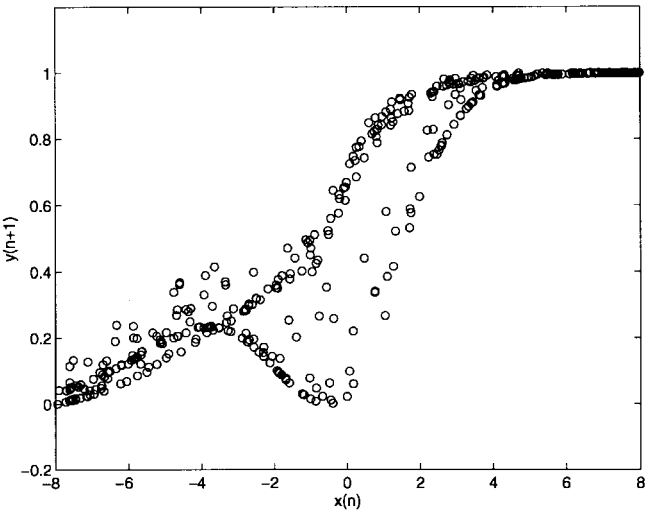
VII. Simulation Results

A. Problems

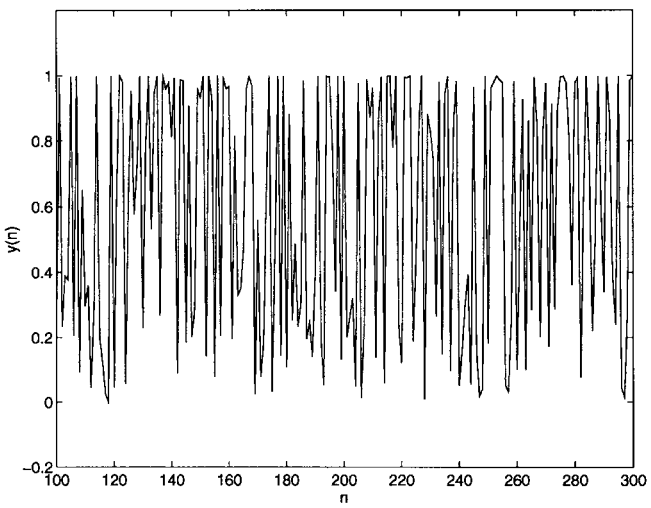
In order to test our algorithm and compare its performance with those of other methods, we choose two test problems: 1) learning underlying target recurrent networks and 2) a nonlinear system identification problem.

In the problem of learning underlying target recurrent networks, the weights of target networks are randomly chosen. Then input sequences are randomly generated from a uniform distribution over the interval $[-8.0, 8.0]$.[15] The corresponding outputs from a target network are used as the desired outputs in a training set. The advantage of using such underlying recurrent networks is that we can control the complexity of the problem easily by changing the number of recurrent hidden units of an underlying target network. Specifically, target recurrent networks with four and eight fully connected hidden units are used in our experiments. To give a rough idea about the problems, the targets versus their corresponding inputs and their corresponding sequence numbers are given in

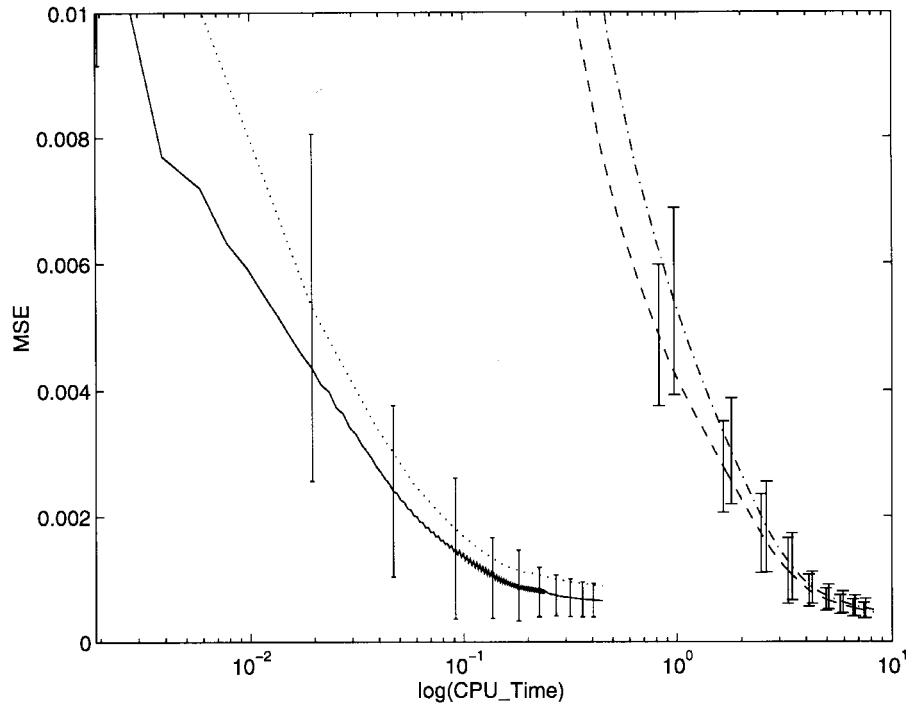
[15] The interval is so chosen that the nonlinearity of hidden units can be fully utilized.



(a)



(b)



(c)

Fig. 5. (a) Targets $y(n+1)$ versus inputs $x(n)$. (b) Targets $y(n)$ versus the sequence number n. (c) The results of learning an underlying target network with four hidden nodes. MSE: averaged mean square error over 20 random runs. log(CPU_Time): the log CPU-Time in minutes on SPARC 10 sun workstation. "-.," "$^-$," "$\cdots$," and "-" represent the training and testing error of BPTT and EMR, respectively. The error bar "I" represents the standard derivation.

Figs. 5(a), 6(a), 5(b), and 6(b), respectively. From Figs. 5(a) and 6(a), it is easy to tell that this problem cannot be modeled by feedforward networks, since a single input can correspond to many different outputs.[16]

The second test problem is a nonlinear system identification problem, which can be described as follows. Let $s_1(k)$ and $s_2(k)$ be two states of the system at time k. Let $u(k)$ and $y(k)$ be the input and the corresponding output of the system at time k $(k \geq 0)$. The underlying mapping between input $u(k)$ and output $y(k)$ is then given by

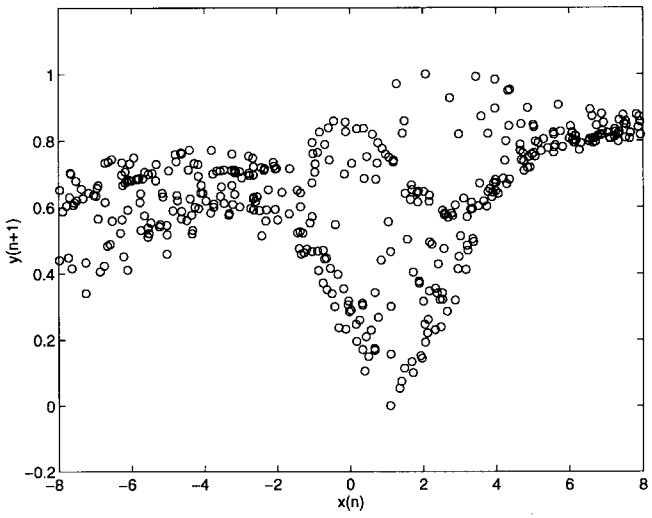
$$s_1(k+1) = \frac{s_1(k) + 2s_2(k)}{1 + s_2^2(k)} + u(k) \tag{48}$$

$$s_2(k+1) = \frac{s_1(k)s_2(k)}{1 + s_2^2(k)} + u(k) \tag{49}$$

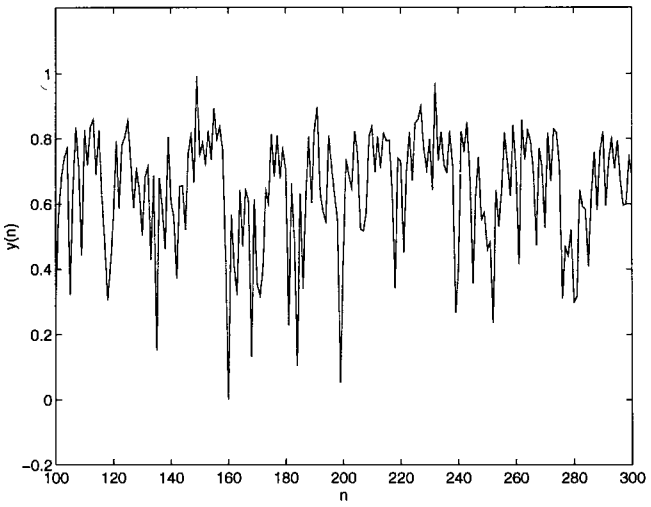
$$y(k+1) = s_1(k+1) + s_2(k+1). \tag{50}$$

This problem was first proposed by Narendra [21], and later used by Horne and Giles [14] to compare different architectures and algorithms of recurrent networks.

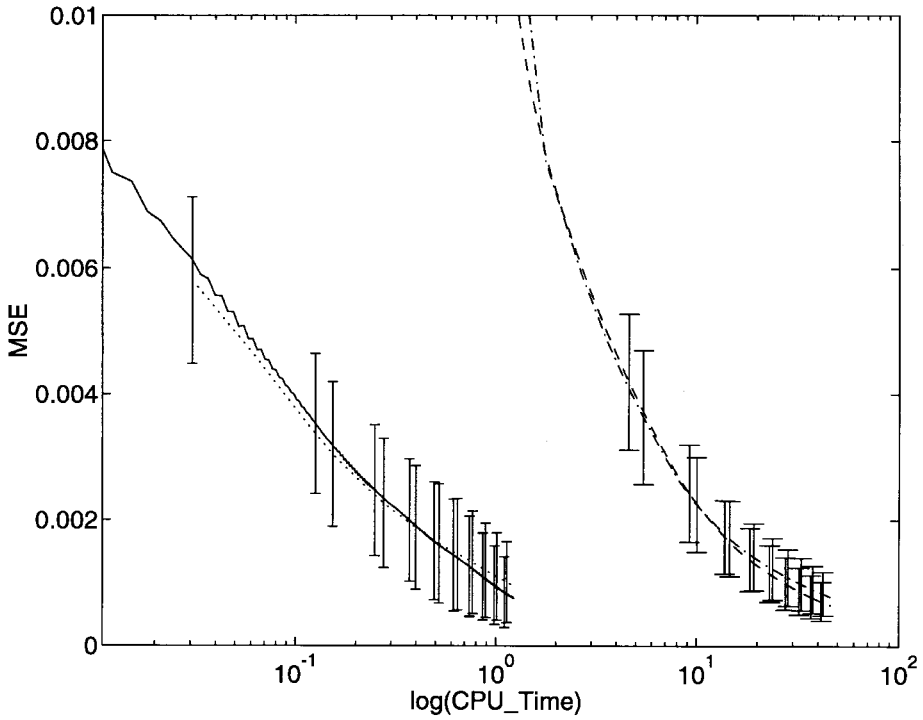
[16]This will be demonstrated again through the results obtained by time-delay-neural-networks (TDNN) given in the next section.



(a)



(b)



(c)

Fig. 6. (a) Targets $y(n+1)$ versus inputs $x(n)$. (b) Targets $y(n)$ versus the sequence number n. (c) The results of learning an underlying target nework with eight hidden nodes. MSE: averaged mean-square error over 20 random runs. log (CPU_Time): the log CPU-Time in minutes on SPARC 10 sun workstation. "-.," "—," "··," and "-" represent the training and testing error of BPTT and EMR, respectively. The error bar "I" represents the standard deviation.

B. Results

Two versions of our EMR algorithm have been tested. One is for off-line training, which uses all the training sequences to update the weights. Another is for on-line learning, which uses randomly drawn sequences, one at a time, to update the weights.[17] Meanwhile, backpropagation-through-time (BPTT) (both off-line and on-line), and time-delay-neural-networks (TDNN's)[18] are used for comparison.[19] We use the maximum number of training epochs to terminate training. The way we choose the maximum number of epochs is that further training can not achieve better generalization performance. In

[17] The weights get updated at the end of each sequence.

[18] TDNN is a feedforward network with current and previous inputs as input to the network. In our experiments for TDNN, we focus on one hidden layer feedforward networks trained by the standard backpropagation.

[19] The real-time-recurrent-learning algorithm have also been tested on the same problems. Since the results were worse than those given by BPTT, they are not being reported in the paper.

particular, we choose 10 000 epochs for off-line BPTT, 20 for on-line BPTT, 5000 for TDNN, 400 for off-line EMR, and five for on-line EMR. Twenty runs are conducted for each case to obtain average training and test errors as well as standard deviations.

For the problem of learning target recurrent networks, a sequence of length 400 is used for training, and a sequence of length 800 is used for testing. To be more specific, a random seed is chosen as the initial state of a sequence. Then the inputs are generated randomly and sequentially up to either 400 or 800 steps. The corresponding outputs of the target network are recorded as desired outputs. The same size networks are trained using EMR and BPTT. For TDNN, we choose the number of hidden units the same as that in the recurrent target networks. We also choose the appropriate number of delays so that the time-delay networks have the same number of weights as those of the target networks. The average training and test errors as well as the standard deviations (over 20 runs with different initial conditions) during the entire training process have been given in Fig. 5(c) (for four-hidden-node target network) and 6(c) (for eight-hidden-node target network) to compare EMR with BPTT for off-line learning. The average test error and the standard deviation when training terminates are given in Table I.

The results show that the EMR is over ten times faster than BPTT with a similar performance in terms of test error in the batch mode, and about four times faster than TDNN. The performance (the test error) of TDNN, however, is far worse than those of EMR's and BPTT's, since a TDNN is a feedforward network, and is not able to model the complicated dynamic behavior of the recurrent network unless more delays and hidden units are used. However, since more delays and more hidden units result in more weights in a TDNN, a good generalization performance can only be obtained when the number of hidden units as well as the delays are properly chosen. In our experiments for learning the four hidden node target network, we find that the best generalization performance (in terms of test error) achieved for TDNN is 1.1 $\times 10^{-3}$ with 20 hidden units and five time delays as inputs. For this problem, since the training data is generated as one sequence, the off-line training and the on-line training are essentially the same. It will be demonstrated in the second test problem that on-line training is much faster than the off-line training when the number of training sequences is large.

In the second test problem on learning the dynamic mapping given by (50), 100 randomly drawn training sequences of length 50 are generated as training samples. In particular, the input sequences $u(k)$'s are selected randomly from $[-2, 2]$, and the corresponding $y(k)$'s are used as the desired targets. The training sequences are illustrated in Fig. 7(a) and (b). Fully connected recurrent networks with six hidden neurons are used. The trained network is tested on a sine wave of frequency 0.04 rad/s which is totally different from the training sequences. The same experimental setting is used as that used in [14] for the purpose of making comparisons.

Both on-line and off-line training is used for BPTT and EMR, where for on-line training, one sequence is chosen randomly at a time, and weights get updated at the end of each sequence. The results are given in Table I. The learning curves for the off-line versions of our algorithm and BPTT are plotted in Fig. 7(c) for comparisons. Similar to what was obtained in the case of learning target networks, the performance of EMR is comparable to that of BPTT. EMR is ten to 15 times faster than off-line BPTT, about five to six times faster than on-line BPTT, and ten times faster than on-line backpropagation for TDNN. Performance of TDNN is again worse than that of EMR and BPTT. The results on the performance of TDNN are from [14].[20] We have run additional simulations on training time for TDNN with the same experimental setup as in [14] for comparisons.

As can be seen from the results, the performance of our algorithm is comparable to that of BPTT. The training time, however, has been improved when EMR is used. The cost for the improved training time seems to be a large variance shown by the learning curves given in Figs. 5(c), 6(c), and 7(c). The reason for a larger variance may be interpretated as follows. In the E-step, our EMR algorithm finds hidden targets based on current weights. In the M-step, our algorithm obtains optimal weights based on the hidden targets. In other words, our algorithm makes a big step in terms of updating the weights at each EM-step. If the estimated hidden targets though the mean-field approximation is inaccurate at an E-step, the optimized weights based on the inaccurate hidden targets will result in a large error. Although such an error can be reduced at the subsequent EM steps when the hidden targets get reestimated, this will cause a large variance in the generalization error. BPTT, on the other hand, only makes small incremental changes to the weights at each step specified by the learning rate, and thus results in a smaller variance in error. Training speed of BPTT, however, will be a lot slower than that of EMR.

VIII. Conclusions and Future Work

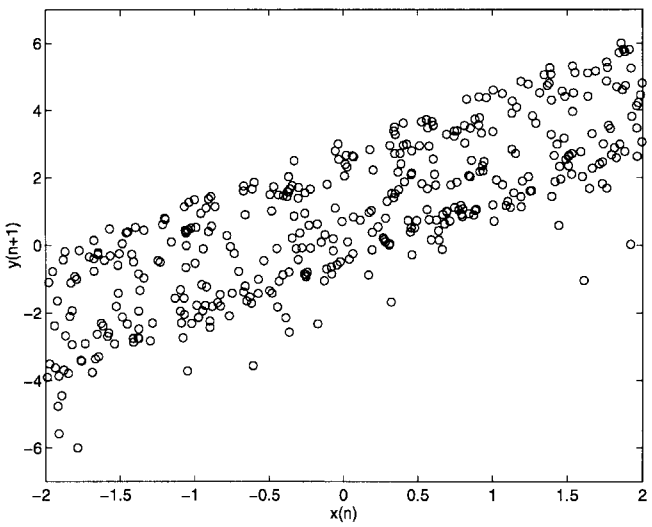
In the work presented, we have established the probabilistic model of desired hidden states for recurrent networks. Based on this model, we have applied the EM algorithm to recurrent networks. Using the mean-field approximation in the E-step, we obtain a novel training algorithm which converts training a complicated recurrent network into training a set of individual feedforward neurons. Since linear weighted regression algorithms can then be used to train these neurons, significant improvements have been made on training speed by our algorithm.

In our future work, we will investigate how widely our simple model for desired hidden states is applicable for real problems. In the mean time, we will consider alternative models for desired hidden states.

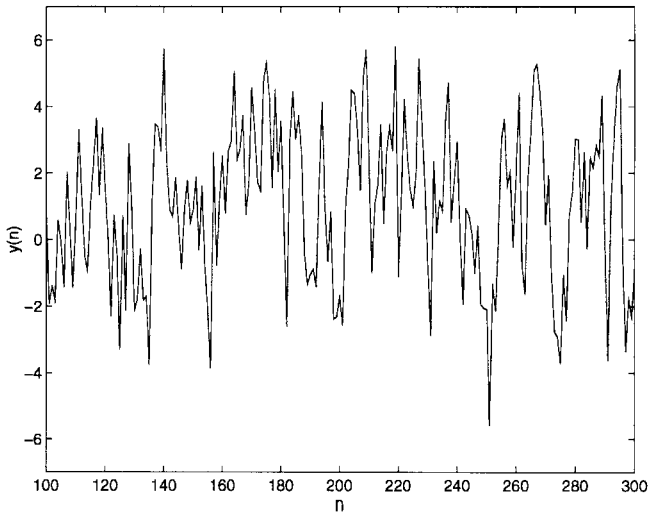
Appendix 1

To show that the constraint given by (22) and the expression for $P(\vec{z}(n+1)|y(n+1), \vec{z}(n), \vec{x}(n), \Theta)$ given by (24) hold,

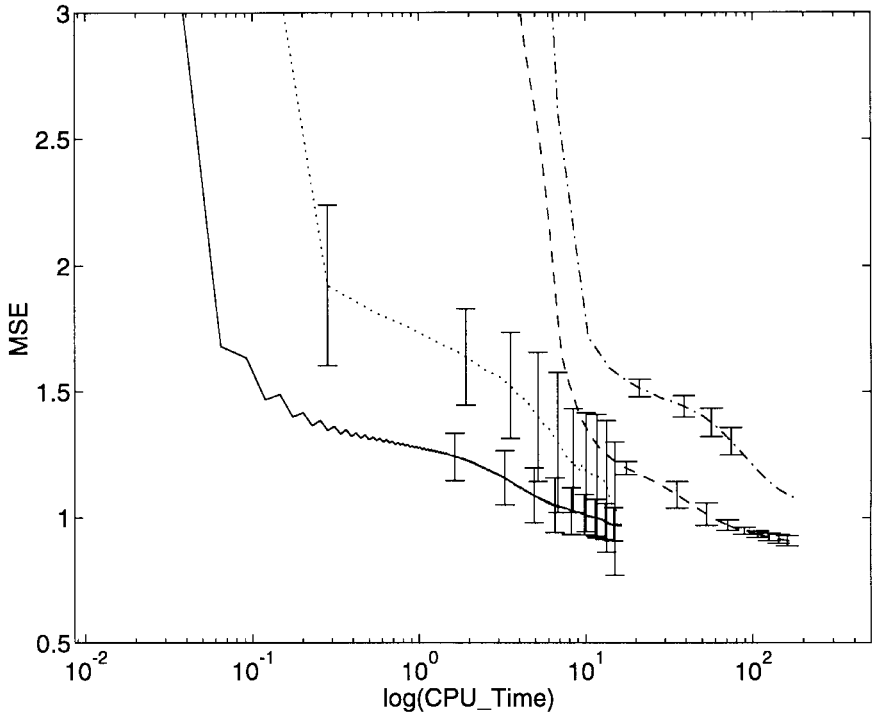
[20] The standard deviation is not given in [14].



(a)



(b)



(c)

Fig. 7. (a) Targets $y(n+1)$ versus inputs $x(n)$. (b) Targets $y(n)$ versus the sequence number n. (c) The results of learning a nonlinear system identification problem. MSE: averaged mean square error over 20 random runs. log (CPU_Time): the log CPU-Time in minutes on SPARC 10 workstation. "-.," "$-$," "$\cdot\cdot$," and "-" represent the training and testing error of BPTT and EMR, respectively. The error bar "I" represents the standard deviation.

we will first show that

$$\lambda_1 E_1(n) + \lambda_2 E_2(n) = \tfrac{1}{2}(\vec{z}(n) - \hat{\vec{z}}(n))^T \Sigma^{-1} (\vec{z}(n) - \hat{\vec{z}}(n)) + Q(n) \quad (51)$$

where

$$\Sigma^{-1} = 2(\lambda_1 I + \lambda_2 \vec{w}^{(2)} \vec{w}^{(2)^T}) \quad (52)$$

$$\hat{\vec{z}}(n) = 2\Sigma(\lambda_1 \vec{h}(n) + \lambda_2 y(n) \vec{w}^{(2)}) = \vec{h}(n) + \frac{\lambda_2 \vec{w}^{(2)}}{\lambda_1 + \lambda_2 ||\vec{w}^{(2)}||^2} (y(n) - \vec{w}^{(2)^T} \vec{h}(n)) \quad (53)$$

$$Q(n) = \lambda_1 \vec{h}(n)^T \vec{h}(n) + \lambda_2 y(n)^2 - \frac{1}{2} \hat{\vec{z}}(n)^T \Sigma^{-1} \hat{\vec{z}}(n) = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2 ||\vec{w}^{(2)}||^2} (y(n) - \vec{w}^{(2)^T} \vec{h}(n))^2 \quad (55)$$

where $\vec{h}(n)$ is determined by (13).

Proof: For $E_1(n)$ and $E_2(n)$ given in (12) and (14), using matrix notations, we can obtain

$$\begin{aligned} \lambda_1 E_1(n) + \lambda_2 E_2(n) &= \vec{z}(n)^T (\lambda_1 I + \lambda_2 \vec{w}^{(2)} \vec{w}^{(2)^T}) \vec{z}(n) \\ &\quad - 2\vec{z}(n)^T (\lambda_1 \vec{h}(n) + \lambda_2 y(n) \vec{w}^{(2)}) \\ &\quad + \lambda_1 \vec{h}(n)^T \vec{h}(n) + \lambda_2 y(n)^2. \end{aligned} \quad (56)$$

TABLE I
1) THE MEAN-SQUARE ERROR ON THE TEST SET. 2) THE STANDARD DEVIATION. 3) THE CPU-TIME ON SPARC 10. ALL RESULTS ARE AVERAGED OVER 20 RUNS. THE AVERAGE TEST ERRORS AND THE STANDARD DEVIATIONS ARE IN THE ORDER OF 10^{-3}

	BPTT		EMR		TDNN
	off-line	on-line	off-line	on-line	
4-hidden unit	0.502 (0.148)		0.884 (0.439)		5.33 (1.00)
target network	8.22min [3]		0.445min		2.42min
8-hidden unit	1.26 (0.497)		0.95(0.597)		4.53 (0.804)
target network	30.7min		1.82min		7.65min
System Identification	107.6 (4.80)	144.0 (1.30)	98.4 (25.2)	141.0 (31.0)	160.0
Problem	178min	2.40min	16.3min	0.427min	5.13 min

On the other hand

$$\begin{aligned}&\tfrac{1}{2}\,(\vec{z}(n)-\hat{\vec{z}}(n))^T\Sigma^{-1}(\vec{z}(n)-\hat{\vec{z}}(n))+Q(n)\\&\quad=\tfrac{1}{2}\,\vec{z}(n)^T\Sigma^{-1}\vec{z}(n)-\vec{z}(n)^T\Sigma^{-1}\hat{\vec{z}}(n)\\&\qquad+\tfrac{1}{2}\,\hat{\vec{z}}(n)^T\Sigma^{-1}\hat{\vec{z}}(n)+Q(n).\end{aligned}\tag{57}$$

Since we want $\lambda_1E_1(n)+\lambda_2E_2(n)\equiv\frac{1}{2}\,(\vec{z}(n)-\hat{\vec{z}}(n))^T\Sigma^{-1}(\vec{z}(n)-\hat{\vec{z}}(n))+Q(n)$, the result is obtained by comparing the right-hand sides of (56) and (57).

Now, we are ready to show (1) the constraint given by (22) holds, and (2) $P(\vec{z}(n+1)|y(n+1),\vec{z}(n),\vec{x}(n),\Theta)$ can be determined by (24).

Inserting (51) into (16), we have

$$\begin{aligned}&P(y(n+1),\vec{z}(n+1)|\vec{z}(n),\vec{x}(n),\Theta)\\&\quad=A_{yz}\exp\left(-\lambda_1E_1-\lambda_2E_2\right)\\&\quad=A_{yz}e^{(-Q(n+1))}\exp\left(-\tfrac{1}{2}\,(\vec{z}(n+1)-\hat{\vec{z}}(n+1))^T\right.\\&\qquad\left.\cdot\,\Sigma^{-1}(\vec{z}(n+1)-\hat{\vec{z}}(n+1))\right)\end{aligned}\tag{58}$$

where $A_{yz},Q(n)$, and Σ^{-1} are defined by (17), (55), and (52), respectively.

Then by the relationship $P(y(n+1)|\vec{z}(n),\vec{x}(n),\Theta)=\int_{-\infty}^{+\infty}P(y(n+1),\vec{z}(n+1)|\vec{z}(n),\vec{x}(n),\Theta)\,d\vec{z}(n+1)$, and using $P(y(n+1)|\vec{z}(n),\vec{x}(n),\Theta)$ given in (18), we obtain

$$A_y\exp\left(-\lambda_3E_3(n+1)\right)=A_{yz}\sqrt{(2\pi)^L\det(\Sigma)}e^{(-Q(n+1))}.\tag{59}$$

Since among L eigenvalues of $\Sigma^{-1},L-1$ of them are identical and equal to $2\lambda_1$, while the additional eigenvalue is $2(\lambda_1+\lambda_2||\vec{w}^{(2)}||^2)$, we have

$$\det(\Sigma)=\frac{1}{\det(\Sigma^{-1})}=\frac{1}{(2\lambda_1)^{L-1}2(\lambda_1+\lambda_2||\vec{w}^{(2)}||^2)}.\tag{60}$$

Therefore, plugging $\det(\Sigma),Q(n+1),A_t$ and A_{yz} into (59), we have

$$\begin{aligned}&\sqrt{\frac{\lambda_3}{\pi}}\exp\left(-\lambda_3E_3(n+1)\right)\\&\quad=\sqrt{\frac{\lambda_1\lambda_2}{\pi(\lambda_1+\lambda_2||\vec{w}^{(2)}||^2)}}\\&\qquad\cdot\exp\left(-\frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2||\vec{w}^{(2)}||^2}E_3(n+1)\right).\end{aligned}\tag{61}$$

Then we can obtain immediately

$$\lambda_3=\frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2||\vec{w}^{(2)}||^2}.\tag{62}$$

Furthermore, using this constraint, we can further show that

$$\begin{aligned}&P(\vec{z}(n+1)|y(n+1),\vec{z}(n),\vec{x}(n),\Theta)\\&\quad=\frac{P(y(n+1),\vec{z}(n+1)|\vec{z}(n),\vec{x}(n),\Theta)}{P(y(n+1)|\vec{z}(n),\vec{x}(n),\Theta)}\\&\quad=\frac{\sqrt{\frac{\lambda_2}{\pi}\left(\frac{\lambda_1}{\pi}\right)^L}\exp\left(-\lambda_1E_1(n+1)-\lambda_2E_2(n+1)\right)}{\sqrt{\frac{\lambda_3}{\pi}}\exp\left(-\lambda_3E_3(n+1)\right)}\\&\quad=\sqrt{\frac{1}{(2\pi)^L\det(\Sigma)}}\exp\left(-\lambda_1E_1(n+1)-\lambda_2E_2(n+1)\right.\\&\qquad\left.+Q(n+1)\right)\end{aligned}\tag{63}$$

$$\begin{aligned}&=\sqrt{\frac{1}{(2\pi)^L\det(\Sigma)}}\exp\left(-\frac{1}{2}(\vec{z}(n+1)-\hat{\vec{z}}(n+1))^T\right.\\&\qquad\left.\cdot\,\Sigma^{-1}(\vec{z}(n+1)-\hat{\vec{z}}(n+1))\right).\end{aligned}\tag{64}$$

Equation (63) is derived by inserting (61) and (62), and gives us $P(\vec{z}(n+1)|y(n+1),\vec{z}(n),\vec{x}(n),\Theta)$.

APPENDIX 2

We shall derive in this section (36), i.e.,

$$Q(\Theta|\Theta^p) \approx E_c - E_p - E_h - E_o \tag{65}$$

where E_c, E_p, E_h, E_o are expressed explicitly by (37)–(40).

From the definition of $Q(\Theta|\Theta^p)$ as in (2), we have

$$\begin{aligned} Q(\Theta|\Theta^p) &= \boldsymbol{E}_z\{\ln P(\{y\},\{\vec{z}\},\{\vec{x}\}|\Theta)||\{y\},\{\vec{x}\},\Theta^p\} \\ &= \int P(\{\vec{z}\}|\{\vec{x}\},\{y\},\Theta^p)\ln P(\{y\},\{\vec{z}\},\{\vec{x}\}|\Theta) \\ &\quad \cdot d(\{\vec{z}\}) \\ &= \int P(\{\vec{z}\}|\{\vec{x}\},\{y\},\Theta^p)\ln P(\{y\},\{\vec{z}\}|\{\vec{x}\},\Theta) \\ &\quad \cdot d(\{\vec{z}\}) + \ln\left(P(\{\vec{x}\}|\Theta)\right). \end{aligned} \tag{66}$$

Since the input training data $\{\vec{x}\}$ is assumed to be drawn from a fixed distribution and is independent of $\Theta, \ln(P(\{\vec{x}\}|\Theta)$ is a constant with respect to Θ.

Substituting $P(\{y\},\{\vec{z}\},|\{\vec{x}\},\Theta)$ given by (28) into $Q(\Theta|\Theta^p)$ given by (66), we can obtain

$$\begin{aligned} Q(\Theta|\Theta^p) &= \int P(\{\vec{z}\}|\{y\},\{\vec{x}\},\Theta^p)(N\ln A_{yz} - \lambda_1 E_1 \\ &\quad - \lambda_2 E_2)\, d(\{\vec{z}\}) + \ln\left(P(\{\vec{x}\}|\Theta)\right) \\ &= \boldsymbol{E}_z(\ln A_{yz}^N) - \boldsymbol{E}_z(\lambda_1 E_1) - \boldsymbol{E}_z(\lambda_2 E_2) \\ &\quad + \ln\left(P(\{\vec{x}\}|\Theta)\right) \end{aligned} \tag{67}$$

where $E_1 = \Sigma_{n=1}^N E_1(n+1)$ and $E_2 = \Sigma_{n=1}^N E_2(n+1)$. Furthermore, $\boldsymbol{E}_z(\ln A_{yz}^N), \boldsymbol{E}_z(\lambda_1 E_1)$ and $\boldsymbol{E}_z(\lambda_2 E_2)$ can be evaluated as follows:

$$\begin{aligned} \boldsymbol{E}_z(N\ln A_{yz}) &= \int N\ln A_{yz} P(\{\vec{z}\}|\{y\},\{\vec{x}\},\Theta^p)\, d(\{\vec{z}\}) \\ &= N\ln A_{yz} \\ &= 0.5N\ln\left(\frac{\lambda_2}{\pi}\left(\frac{\lambda_1}{\pi}\right)^L\right). \end{aligned} \tag{68}$$

The above result is obtained by using the assumption that λ_1 and λ_2 are constants independent of hidden variables $\{\vec{z}\}$ and by plugging in A_{yz} given by (17).

Substituting E_1 by (12) and $P(\{\vec{z}\}|\{\vec{x}\},\{y\},\Theta^p)$ by (24), $\boldsymbol{E}_z(\lambda_1 E_1)$ can be simplified to

$$\begin{aligned} \boldsymbol{E}_z(\lambda_1 E_1) &= \int \lambda_1 E_1 P(\{\vec{z}\}|\{y\},\{\vec{x}\},\Theta^p)\, d(\{\vec{z}\}) \\ &= \lambda_1 \sum_{n=2}^{N+1}\sum_{j=1}^{L}\int (z_j(n) - h_j(n))^2 \\ &\quad \cdot \prod_{k=2}^{k=n} P(\vec{z}(k)|y(k),\vec{z}(k-1),\vec{x}(k-1),\Theta^p) \end{aligned} \tag{69}$$

$$\begin{aligned} &\quad \cdot d(\{\vec{z}(k)\}_{k=2}^{k=n}) \\ &\approx \lambda_1 \sum_{n=2}^{N+1}\sum_{j=1}^{L}\int (z_j(n)^2 - 2z_j(n)\tilde{h}_j(n) + \tilde{h}_j(n)^2) \\ &\quad \cdot P(\vec{z}(n)|y(n),\tilde{\vec{z}}(n-1),\vec{x}(k-1),\Theta)\, d(\vec{z}(n)) \end{aligned} \tag{70}$$

$$\begin{aligned} &= \lambda_1 \sum_{n=2}^{N+1}\sum_{j=1}^{L}(\sigma_{z_j(n)}^2 + (\tilde{z}_j(n) - \tilde{h}_(n))^2) \\ &= \lambda_1 \sum_{n=2}^{N+1}\sum_{j=1}^{L}\sigma_{z_j(n)}^2 + E_h. \end{aligned} \tag{71}$$

In the above derivation, (69) is obtained through using (27) and integrating over all $\vec{z}(k)$ for $n < k \le N+1$. Equation (70) is the mean-field approximation of (69)–(32). $\tilde{z}_j(n)$ is defined in (35) and $\sigma_{z_j(n)}^2 = \int (z_j(n) - \tilde{z}_j(n))^2 P(\vec{z}(n)|y(n+1),\tilde{\vec{z}}(n-1),\vec{x}(n),\Theta^p)\, d(\vec{z}(n))$. Since $\sigma_{z_j(n)}^2$ only depends on Θ^p which is the set of previous parameters, it is a constant in terms of Θ.

Similarly, using the mean-field approximation, and substituting E_2 (14) and $P(\{\vec{z}\}|\{\vec{x}\},\{y\},\Theta^p)$ (27), $\boldsymbol{E}_z(\lambda_2 E_2)$ can be computed as

$$\begin{aligned} \boldsymbol{E}_z(\lambda_2 E_2) &= \int \lambda_2 E_2 P(\{\vec{z}\}|\{y\},\{\vec{x}\},\Theta^p)\, d(\{\vec{z}\}) \\ &= \lambda_2 \sum_{n=2}^{N+1}\int (\vec{z}(n)\cdot\vec{w}^{(2)} - y(n))^2 \\ &\quad \cdot \prod_{k=2}^{k=n} P(\vec{z}(k)|y(k),\vec{z}(k-1),\vec{x}(k-1),\Theta^p) \\ &\quad \cdot d(\{\vec{z}(k)\}_{k=2}^{k=n}) \end{aligned} \tag{72}$$

$$\begin{aligned} &\approx \lambda_2 \sum_{n=2}^{N+1}\int (\vec{w}^{(2)^T}(\vec{z}(n) - \tilde{\vec{z}}(n)) \\ &\quad + (\vec{w}^{(2)}\cdot\tilde{\vec{z}}(n) - y(n)))^2 P(\vec{z}(n)|y(n) \\ &\qquad \tilde{\vec{z}}(n-1),\vec{x}(k-1),\Theta)\, d(\vec{z}(n)) \end{aligned} \tag{73}$$

$$\begin{aligned} &= \lambda_2 \sum_{n=2}^{N+1}\vec{w}^{(2)^T}\Sigma\vec{w}^{(2)} + \lambda_2\sum_{n=1}^{N}(\vec{w}^{(2)^T} \\ &\quad \cdot \tilde{\vec{z}}(n) - y(n))^2 \\ &= E_o + E_p. \end{aligned} \tag{74}$$

Σ, a $L \times L$ covariance matrix, is equal to $\int (\vec{z}(n) - \tilde{\vec{z}}(n))(\vec{z}(n) - \tilde{\vec{z}}(n))^T P(\vec{z}(n)|y(n+1),\tilde{\vec{z}}(n-1),\vec{x}(n),\Theta^p)\, d(\vec{z}(n))$. It turns out that Σ is independent of n and only depends on previous parameters.

Putting (68), (71), and (74) together into (67), we obtain $Q(\Theta|\Theta^p)$ as

$$\begin{aligned} Q(\Theta|\Theta^p &= \boldsymbol{E}_z(\ln A_{yz}^N) - \boldsymbol{E}_z(\lambda_1 E_1) - \boldsymbol{E}_z(\lambda_2 E_2) \\ &\quad + \ln\left(P(\{\vec{x}\}|\Theta)\right) \\ &= 0.5N\ln\left(\frac{\lambda_2}{\pi}\left(\frac{\lambda_1}{\pi}\right)^L\right) + \lambda_1\sum_{n=1}^{N}\sum_{j=1}^{L}\sigma_{z_j(n)}^2 \\ &\quad - E_h - E_o - E_p + \ln P(\{\vec{x}\}|\Theta) \\ &= E_c - E_h - E_o - E_p. \end{aligned} \tag{75}$$

ACKNOWLEDGMENT

The authors would like to thank T. Grossman, B. Pearlmutter, and James Modestino for suggesting the related references and many helpful discussions. The authors would like to thank anonymous reviews for their helpful comments and suggestions.

REFERENCES

[1] S. Amari, K. Kurata, and H. Nagaoka, "Information geometry of Boltzmann machines," *IEEE Trans. Neural Networks*, vol. 3, pp. 260–271, 1992.

[2] R. Andrews, J. Diederich, and A. B. Tickle, "A survey and crique of techniques for extracting rules from trained artificial neural networks," to appear in *Knowledge-Based Systems*, available FTP:ftp.qut.edu.au//pub/NRC/ps/QUTNRC-95-01-02.ps.Z

[3] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Annu. Math. Statist.*, vol. 41, pp. 164–171, 1970.

[4] Y. Bengio, "Credit assignment through time: Alternative to backpropagation," *Advances in Neural Inform. Processing Syst.*, vol. 6, pp. 75–82, 1994.

[5] Y. Bengio, P. Frasconi, and P. Simard, "The problem of learning long-term dependencies in recurrent networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 157–166, 1993.

[6] L. E. Breiman and J. H. Friedman, "Function approximation using ramps," *Neural Networks for Computing*. Snowbird, UT: Snowbird, 1993.

[7] W. Byne, "Alternating minimization and Boltzmann machine learning," *IEEE Trans. Neural Networks*, vol. 3, 1992.

[8] T. M. Cover and J. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[9] S. Das and M. C. Mozor, "A unified gradient descent clustering architecture for finite-state machine induction," *Advances in Neural Inform. Processing Syst.*, vol. 6, 1994.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via EM algorithm," *J. Roy. Statist. Soc.*, vol. 39, pp. 1–33, 1977.

[11] S. E. Fahlman, "The recurrent cascade-correlation architecture," *Advances in Neural Inform. Processing Syst.*, vol. 3, pp. 190–196, 1991.

[12] T. Grossman, "Learning, capacity and implementation of neural-network models," Ph.D. dissertation, Weizmann Institute of Science, 1992.

[13] A. Herz, A. Krough, and P. G. Palmer, *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley.

[14] B. G. Horne and C. L. Giles, "An experimental comparison of recurrent neural networks," *Advances in Neural Inform. Processing Syst.*, vol. 7, 1995.

[15] M. Jordan and R. A. Jacobs, "Hierarchical mixture of experts and the EM algorithm," *Neural Computa.*, vol. 6, pp. 181–214, 1994.

[16] A. Krough, G. I. Thorberaso, and J. A. Hertz, "A cost function for internal representations," *Advances in Neural Inform. Processing Syst.*, vol. 2, pp. 733–745, 1990.

[17] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A time-delay neural-network architecture for isolated word recognition," *Neural Networks*, vol. 3, pp. 23–44, 1990.

[18] E. Levin, N. Tishby, and S. A. Solla, "A statistical approach to learning and generalization in layered neural network," *Proc. IEEE*, vol. 78, no. 10, pp. 1568–1574, 1990.

[19] S. Ma, C. Ji, and J. Farmer, "An efficient EM-based training algorithm for feedforward neural networks," *Neural Networks*, to be pulished.

[20] D. J. Mackey, "A practical Bayesian framework for backpropagation networks," *Neural Computa.*, vol. 4, pp. 448–472, 1992.

[21] K. S. Narendra, "Adaptive control of dynamical systems using neural networks," in *Handbook of Intelligent Control*, D. A. White and D. A. Sofge, Eds. NY: Van Nostrand Reinhold, 1992.

[22] C. Omlin and C. L. Giles, "Extraction of rules from discrete-time recurrent neural networks," *Neural Networks*, vol. 9, no. 1, p. 41, 1996.

[23] J. B. Pollack, "The induction of dynamical recognizers," *Machine Learning*, vol. 7, no. 23, 1991.

[24] R. Rohwer, "The moving target training algorithm," *Advances in Neural Inform. Processing Syst.*, vol. 2, pp. 558–565, 1990.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, vol. 1, *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, ch. 8.

[26] D. Saad and E. Marom, "Learning by choice of internal representations: An energy minimization approach," *Complex Syst.*, vol. 4, pp. 107–118, 1990.

[27] L. K. Saul, T. Jaakkola, and M. I. Jordan, "Mean field theory for sigmoid belief networks," *J. Artificial Intell. Res.*, vol. 4, pp. 61–76, 1996.

[28] J. W. Shavlik, "Combining symbolic and neural learning," *Machine Learning*, vol. 14, no. 3, 1994.

[29] R. S. Shadafan and M. Niranjan, "A dynamic neural network architecture by sequential partitioning of the input space," *Neural Computa.*, vol. 6, pp. 1203–1222, 1994.

[30] E. Sontag, "Systems combining linearity and saturations and relations to neural networks," Rutgers Center for Syst. and Contr., Tech. Rep. SYCON92-01, 1992.

[31] P. Tino and J. Sajda, "Learning and extracting initial mealy machines with a modular neural network model," *Neural Computa.*, vol. 7, no. 4, 1995.

[32] P. Werbos, "Beyond regression," Ph.D dissertation, Harvard Univ., Cambridge, MA, 1974.

[33] R. J. Williams and J. Peng, "An efficient gradient-based algorithm of on-line training of recurrent network trajectories," *Neural Computa.*, vol. 2, pp. 491–501, 1990.

[34] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computa.*, vol. 1, pp. 270–280, 1989.

[35] J. Zhang, "The mean-field theory in EM procedures for Markov random fields," *IEEE Trans. Signal Processing*, vol. 40, pp. 2570–2583, 1992.

Sheng Ma received the B.S. degree from Tsinghua University, Beijing, China, in 1992 and the M.S. degree from Rensselaer Polytechnic Institute, Troy, NY, in 1995, both in electrical engineering. He is currently a Ph.D. candidate in electrical engineering at Rensselaer Polytechnic Institute.

His current research interests are pattern recognition, time series modeling, and computer communication networks.

Chuanyi Ji (M'97) received the B.S. degree (with honors) from Tsinghua University, Beijing, China, in 1983, the M.S. degree from University of Pennsylvania, Philadelphia, in 1986, and the Ph.D. degree from California Institute of Technology, Pasadena, in 1992, all in electrical engineering.

Since November 1991, she has been an Assistant Professor of Electrical, Computer, and Systems Engineering at Renselaer Polytechnic Institute, Troy. Her research areas include computer communication networks with a focus on traffic modeling, analysis, and network management, statistical learning theory and algorithms, neural networks, and applications in communication networks.

Dr. Ji was a recipient of the NSF CAREER award in 1995.