HWR No. 96-030

# A SUPERIOR TRAINING STRATEGY FOR THREE-LAYER FEEDFORWARD ARTIFICIAL NEURAL NETWORKS

by

Kuo-lin Hsu

Hoshin Vijai Gupta

and

Soroosh Sorooshian

Department of Hydrology and Water Resources

University of Arizona

Tucson, Arizona 85721

# A SUPERIOR TRAINING STRATEGY FOR THREE-LAYER FEEDFORWARD ARTIFICIAL NEURAL NETWORKS

by

**Kuo-lin Hsu**

**Hoshin Vijai Gupta**

**and**

**Soroosh Sorooshian**

**Department of Hydrology and Water Resources**
**University of Arizona**
**Tucson, Arizona 85721**

March 1996

# TABLE OF CONTENTS

## ACKNOWLEDGMENTS

This research was partially supported by grants from the Hydrologic Research Laboratory of the U.S. National Weather Service (Grant no. NA37WH0385), the NASA-EOS Interdisciplinary Research Program (IDP-88-086), and the NOAA Research Program (NA16RC0119-0). The first author greatly appreciates the fellowship support provided by the NASA Global Change Program (Grant No. NGT-30045).

## ABSTRACT

A new algorithm is proposed for the identification of three-layer feedforward artificial neural networks. The algorithm, entitled LLSSIM, partitions the weight space into two major groups: the input-hidden and hidden-output weights. The input-hidden weights are trained using a multi-start SIMPLEX algorithm and the hidden-output weights are identified using a conditional linear-least-square estimation approach. Architectural design is accomplished by progressive addition of nodes to the hidden layer. The LLSSIM approach provides globally superior weight estimates with fewer function evaluations than the conventional back propagation (BPA) and adaptive back propagation (ABPA) strategies. Monte-carlo testing on the XOR problem, two function approximation problems, and a rainfall-runoff modeling problem show LLSSIM to be more effective, efficient and stable than BPA and ABPA.

# I. INTRODUCTION

IN the identification of artificial neural network (ANN) models, certain characteristic problems have been observed. Firstly, the error surface exhibits extensive regions that have very little sensitivity to variations in the weights, and which contain numerous multi-local minima. Secondly, the error surface is generally of very high dimension. General optimization strategies based on gradient search algorithms seem to have a difficult time locating the optimal weights under such conditions, and the training time required can be very long. The backpropagation error algorithm (BPA) [1], a gradient search algorithm, is the method most widely used in ANN training. The training speed of the BPA is notoriously slow and has been found to significantly depend on the network weights used to initialize the search. Methods to improve the convergence speed of the BPA that have been investigated include modifications based on the use of momentum factors [2], [3], adaptive learning rates (eg. Adaptive Back Propagation Algorithm (ABPA)) [4], second order strategies (eg. Second Order Back Propagation Algorithm) [5], [6], and conjugate gradient strategies [7], [8]. However such enhancements seem to provide only marginal improvements in training time, and do not deal with the problem of sensitivity to initial weights. Recent research has focused on the implementation of stochastic global optimization methods; such methods have the potential to provide improved performance based on their ability to elude local minima. One strategy that has been tested is the Simulated Annealing algorithm [9], [10]; however the training time can still be extremely long especially for large networks. Other strategies, notably the genetic algorithm (GA), combined GA with BPA, and simulated evolutionary optimization (SEO) have recently been shown to provide improved performance [11], [12], [13].

In this paper we present an effective network training algorithm, entitled LLSSIM, designed

for the training of three-layer feedforward ANN's. The LLSSIM algorithm uses a partition of the weight space to implement an optimal synthesis of two training strategies. The input-hidden layer weights are estimated using the SIMPLEX non-linear optimization algorithm [14], while the hidden-output layer weights are estimated using optimal linear least square estimation (LLS) [15]. The algorithm takes advantage of this weight-space partition to conduct the non-linear portion of the search in a reduced dimensional space, resulting in an acceleration of the training process. The SIMPLEX search algorithm provides improved local search characteristics due to its ability to not be trapped by minor local optima, and improved global search characteristics through the use of multiple starts initiated randomly in the search space. Identification of the structure of the ANN is done using a strategy of progressively adding nodes to the hidden layer until a structure appropriate to the complexity of the problem is achieved. The algorithm has been tested on four problems (the XOR problem, two function approximation problems, and a rainfall-runoff relationship modeling problem) and the performance found to be significantly superior to that of the classical BPA and ABPA methods.

This paper is organized as follows. Section 2 reviews the fundamentals of the three-layer feedforward ANN structure, and presents the new algorithm, LLSSIM. Section 3 discusses a strategy for structure identification of ANNs using LLSSIM. Section 4 presents the test results. Section 5 contains concluding remarks. The appendix contains a brief review of the BPA, ABPA, and SIMPLEX training procedures.

## II. THE LLSSIM TRAINING ALGORITHM

This paper is concerned with the three-layer feedforward Neural Network. A graphical representation is presented in figure 1. We will use the following notation:

$t_k(p)$: target value of output node k of training pattern p

$x_i(p)$: input value of input node i of training pattern p

$y_j(p)$: output value of hidden node j of training pattern p

$z_k(p)$: output value of output node k of training pattern p

$s_j^h(p)$: weighted sum of inputs to hidden node j of pattern p

$s_k^o(p)$: weighted sum of hidden outputs to output nodes k of pattern p

$w_{ji}^h$: connection weight from input node i to hidden node j

$w_{kj}^o$: connection weight from hidden node i to output node k

Each processing element (figure 2) receives one or more input signals from the elements of the previous layer and each of these signals, $x_i$, is assigned a relative weight, $w_{ji}$. The effective input, $S_j$, to a processing element, is computed as the weighted sum of all the relevant input signals,

$$s_j = \sum_{i=0}^{n} w_{ji} x_i \qquad (1)$$

where $x_0$ is the bias input and $w_{j0}$ is its associated weight. The effective input, $S_j$, is passed through a nonlinear logistic activation function (figure 3), bounded between 0.0 and 1.0, to produce an output

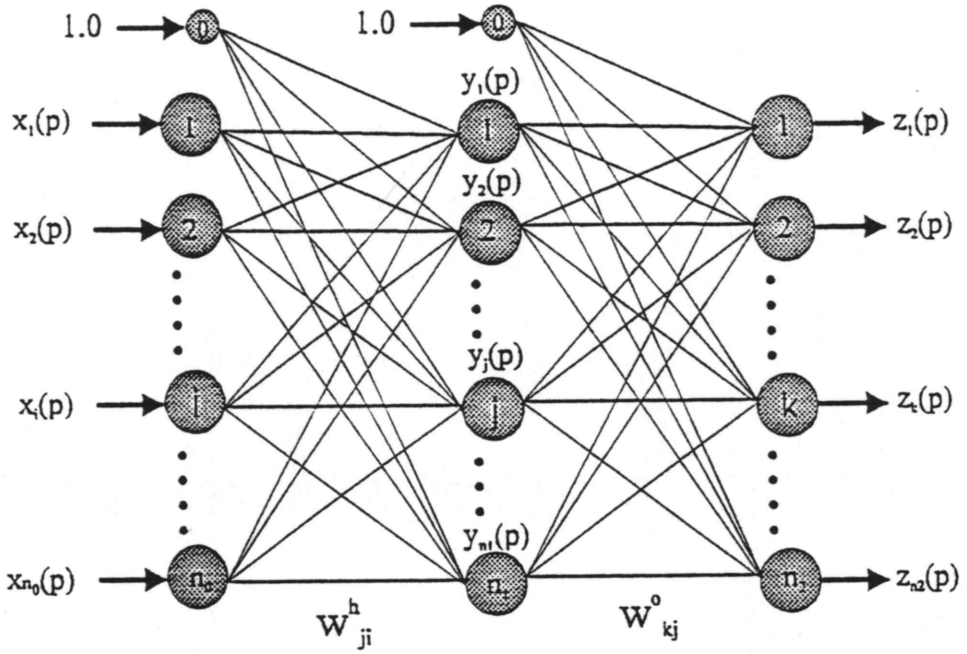


Fig. 1. A typical three layer neural network

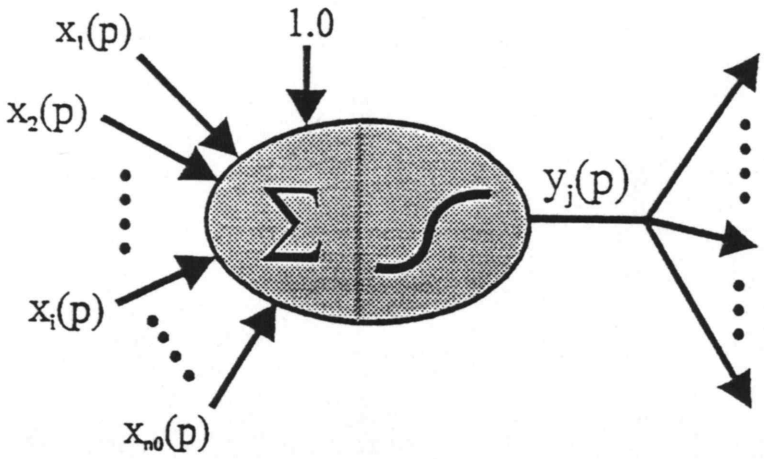


Fig. 2. A process element

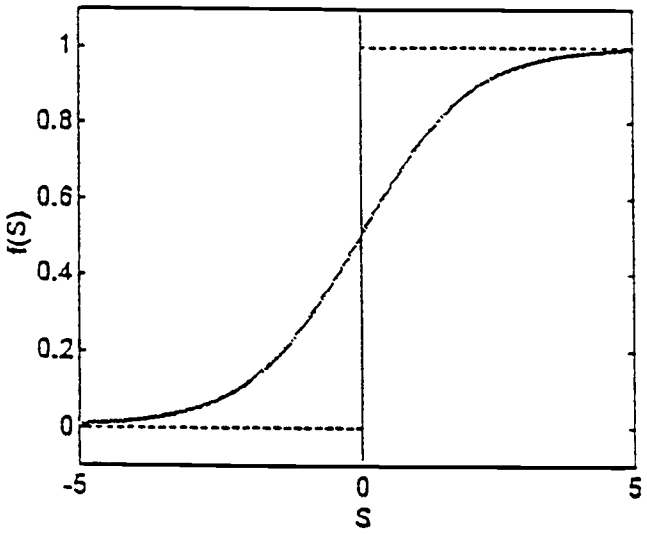


Fig. 3. The sigmoid function

value $y_j$ ,

$$y_j = f(s_j) = \frac{1}{1+\exp(-s_j)} \tag{2}$$

The derivative of the logistic function is given by:

$$f'(s_j) = f(s_j)\,(1-f(s_j)) \tag{3}$$

In this formulation, there are m pairs of input-output patterns with $n_0$ inputs and $n_2$ output variables. A specific pattern of inputs and model outputs is described as $[x_1(p), x_2(p), \ldots, x_{n0}(p)]^T$ and $[z_1(p), z_2(p), \ldots, z_{n2}(p)]^T$ and the targets of outputs are given as $[t_1(p), t_2(p), \ldots, t_{n2}(p)]^T$. The cost function can be expressed as:

$$\begin{aligned} F[w] &= \frac{1}{2}\sum_{p=1}^{m}\sum_{k=1}^{n_2} (t_k(p)-z_k(p))^2 \\ &= \frac{1}{2}\sum_{p=1}^{m}\sum_{k=1}^{n_2} [t_k(p)-f(\sum_{j=0}^{n_1} w_{kj}^{o} f(\sum_{i=0}^{n_0} w_{ji}^{h} x_i(p)))]^2 \end{aligned} \tag{4}$$

where f is the transfer function indicated in equation 2..

The most popular method that has been applied to search for the hidden-output weights and input-hidden weights is the Back Propagation Algorithm (BPA). Because the BPA is a local search algorithm, its performance is heavily dependent on the starting weights and is poor in the flat regions of the cost function. It is also incapable of eluding local minima. Adaptive versions of the BPA based on the addition of a momentum factor and/or an adaptable learning step size still result in local search performance. The BPA and adaptive BPA (ABPA) are briefly presented in the appendices.

We present here a new training algorithm, LLSSIM. In this approach, the weight space is

partitioned two groups, the first consisting of the input-hidden weights and the second consisting of the hidden-output weights. We define $TS_k(p)$ to be the value of the target p transformed backward through the logistic function of output node k,

$$TS_k(p) = \ln\left(\frac{t_k(p)}{1-t_k(p)}\right) \tag{5}$$

and define a new error function $F_1$ in terms of the transformed target values:

$$\begin{aligned} F_1[w] &= \frac{1}{2}\sum_{p=1}^{m}\sum_{k=1}^{n_2}\left(TS_k(p) - S_k^o(p)\right)^2 \\ &= \frac{1}{2}\sum_{p=1}^{m}\sum_{k=1}^{n_2}\left(TS_k(p) - \sum_{j=0}^{n_1} w_{kj}^o f\left(\sum_{i=0}^{n_0} w_{ji}^h x_i(p)\right)\right)^2 \end{aligned} \tag{6}$$

In this representation we see that transformed targets $TS_k(p)$ are linear in the hidden-output weights, $w_{kj}^o$, while still being nonlinear in the input-hidden weights, $w_{ji}^h$. Let us assume that the values of the input-hidden weights are known. In this case, the optimal hidden-output weights (conditioned on the values of the input-hidden weights) can be computed explicitly using the method of linear least squares (LLS) by setting the derivative of function $F_1$ relative to the hidden-output weights to zero:

$$\frac{\partial F_1}{\partial w_{kj}^o} = -\sum_{p=1}^{m}\left(TS_k(p) - S_k^o(p)\right) y_j(p) = 0 \tag{7}$$

The above equation can be rewritten as:

$$\begin{aligned} \sum_{p=1}^{m} TS_k(p)\, y_j(p) &= \sum_{p=1}^{m}\sum_{l=0}^{n_1} w_{kl}^o y_l(p)\, y_j(p) \\ &= \sum_{p=1}^{m} y_j(p)\sum_{l=0}^{n_1} y_l(p)\, w_{kl}^o \end{aligned} \tag{8}$$

Define

$$R_{jl} = \sum_{p=1}^{m} \sum_{i=0}^{n_1} y_j(p)\, y_l(p) \tag{9}$$

$$Q_j = \sum_{p=1}^{m} TS_k(p)\, y_j(p) \tag{10}$$

Now the hidden-output weights, $w_k^o$, can be found by solving the equation:

$$w_k^o = R^{-1} Q \tag{11}$$

where $w_k^o = [w_{k0}^o, w_{k1}^o, \ldots, w_{km1}^o]^T$. We call these the conditionally optimal hidden-output weights, because their values depend on the values selected for the input-hidden weights.

The formulation presented above permits us to reduce the dimension of the search space in which a non-linear training algorithm need be applied, to that of the input-hidden weight space. However, rather than employ the local search BPA or ABPA training strategies, we employ a global strategy based on a multi-start version of the non-linear "Simplex" algorithm [14], [16]. The multi-start Simplex algorithm is listed in the Appendix C. At each step in the evolution strategy of the algorithm, the hidden-output weights are estimated using the LLS and the value of the error function F[w] is computed. The LLSSIM algorithm is summarized below:

step 0) specify number of restarts of the multi-start Simplex algorithm:

Assign nSIM=0 and nSTART

step 1) initialize the simplex:

Select a simplex of size m=n+1 points, where $n = (n_0+1)*(n_1)$ is the dimension of the input-hidden weight space, and randomly assign all weights within $[w_{max}, w_{min}]$ to

these m points.

step 2) evolve the simplex:

At each evolution step of the Simplex algorithm (reflection, extension, and contraction), estimate the conditionally optimal hidden-output weights using LLS and compute the error function F[w].

step 3) test for convergence:

If the group convergence volume (GCV) measure (of the size of the Simplex) becomes smaller than the group convergence criterion (GCC), let nSIM=nSIM+1 and go to step 4. Otherwise, go to step 2.

step 4) test for stopping criterion:

If nSIM=nSTART, then go to step 5; otherwise go to step 1.

step 5) terminate the search:

Stop the training and output the results.

Note that, unlike the BPA which starts the search from a single point in the feasible weight space, the Simplex algorithm starts with n+1 points selected randomly in the n-dimensional input-hidden weight space. The algorithm has global search ability and is well known to not be easily trapped by minor local minima on the cost function surface. Restarts of the Simplex procedure each time the n+1 points converge into a very small cluster guarantee that the procedure will find the globally optimal solution with probability approaching 1.0 [17].

## III. SELECTION OF THE NUMBER OF HIDDEN LAYER NODES

One approach to ANN structural design is to begin with a very large number of hidden nodes and to gradually prune the structure by removing the connection weights that are found to be unimportant [18]. However, this method may not result in an economical solution. An alternative approach is to begin with a simple network and to systematically augment it by increasing the number of hidden nodes until the network outputs satisfactorily match the targets [19], [20]. In general, one or more hidden nodes are added to the network each time the existing network converges to a solution but the output errors are still unacceptable. The training procedure is then continued in the extended weight space. We have chosen to adopt this latter strategy and incorporate it into the LLSSIM network training procedure. The entire procedure is summarized below:

step 1) choose an initial structure:

Assign an initial ANN structure, an acceptable minimum reference training error (RTE), and a reference augmentation error (RAE).

step 2) train the ANN:

Use the LLSSIM algorithm to train the network until nSIM=nSTART. Record the RMSE of the best point, BF.

step 3) test for termination of training:

Compare RTE and BF. If BF is less than RTE, assign RTE=BF and go to step 2; else go to step 4.

step 4) test for termination of structure augmentation:

Compare RAE and BF, if BF is less than RAE, assign RAE=BF, augment the network with one or more hidden nodes, and go to step 2; else to go step 5.

step 5) termination stage:

Stop training the ANN and record the final results.

In the procedure outlined above, RTE and RAE control the training and augmentation of the ANN. Their values are automatically updated during the training process, and so should be initialized to some large values at the outset.

## IV. TEST EXAMPLES AND DISCUSSION

Four test problems were used to evaluate the LLSSIM training algorithm. The first is the "exclusive or" (XOR) pattern recognition problem. The second and third are function approximation problems. The fourth is a rainfall-runoff simulation modeling problem. The performance of the LLSSIM approach is compared with that of the BPA and ABPA approaches. Each method is run 100 times from randomly initiated starting points and the statistical behavior (in terms of the average and standard deviation of RMSE) of the three methods are compared.

### A. The XOR Problem

The exclusive or (XOR) pattern recognition problem consists of the four input-target mapping: $([x_1,x_2],t) = ([0,0],0)$, $([0,1],1)$, $([1,0],1)$, and $([1,1],0)$. In the actual practice of training, the desired targets 0 and 1 are represented by 0.1 and 0.9 respectively. Based on reports published in the literature, a fixed three-layer ANN(2,2,1) structure (2 input nodes, 2 hidden nodes, and 1

output node, giving rise to 6 input-hidden weights and 3 hidden-output weights) can be used to solve this problem [1]. In the LLSSIM algorithm, seven points formed the simplex used to train the six input-hidden weights, $w^h_{ji}$. The three hidden-output weights were trained using the linear least square estimation procedure. Each evaluation of the cost function is counted as an iteration. The group convergence criterion GCC was set to 1.0e-4, nSTART was set to 10 (ten restarts of the simplex procedure), and the initial random weights were bounded within [-5,5].

Based on some experimentation, the BPA algorithm learning rate, $\eta$, was set to 0.15 and the momentum factor, $\alpha$, was set to 0.15. The ABPA algorithm had the same initial learning rate, but had a higher initial momentum factor, 0.9, and then updated these factors by the parameters $\phi$=1.2 and $\beta$=0.5 respectively.

The final training results are shown in figures 4a-d. Each line on each figure represents the average value from 100 independent runs. Figure 4a compares the average RMSEs and figures 4b-d show the standard deviations of the RMSEs as a function of the number of function evaluations for the three algorithms. Clearly the performance of the LLSSIM method is superior to that of BPA and ABPA. Figures 4a and 4d indicate that the LLSSIM algorithm is effective, efficient and stable on this problem, converging within 300 iterations for all 100 runs. In contrast, the variability in performance of both BPA and ABPA is quite high, as measured by the large standard deviation of RMSE about its mean value (figures 4b-c), indicating that many of the ABPA and BPA runs did not converge even within 1000 iterations; in fact many did not converge even when the methods were run to 3000 iterations (not shown).

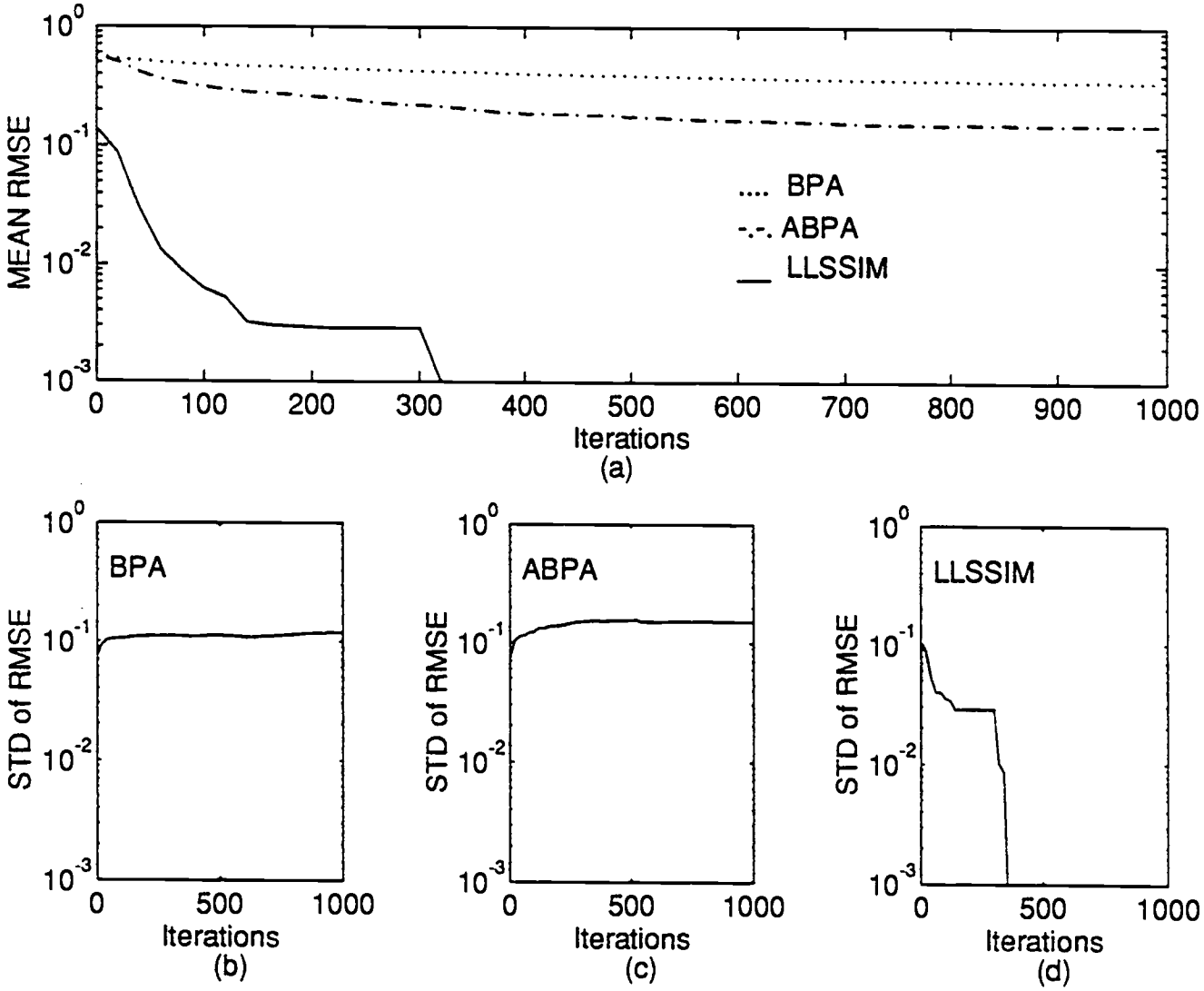


Fig. 4. Simulation results of XOR problem (100 independent runs): (a) Average performance of BPA, ABPA, and LLSSIM.(b) One standard deviation of RMSE of BPA. (c) One standard deviation of RMSE of ABPA. (d) One standard deviation of RMSE of LLSSIM.

## B. Function Approximation I

In this test case, an ANN was developed to approximate the simple input-output relation shown in figure 5. The parameters $\eta$, $\alpha$, $\beta$, and $\phi$, used in the BPA and ABPA methods remained the same as in the XOR test problem. For LLSSIM the GCC criterion was $10^{-4}$ and nSTART was 1. The ANN was trained using the automatic structure training procedure starting from a single hidden node, and gradually increasing the number of hidden nodes. For BPA and ABPA, if the error function did not improve by $10^{-4}$ within 100 iterations of training, network training was stopped and the test for whether to augment the structure with an additional hidden node was applied. Network training and augmentation were terminated when the cost function did not improve by more than $10^{-4}$ from one structure to the next.

Figures 6a-d compare the average performance of 100 runs of each training algorithm. Note that the BPA runs appear to have reached a region of poor function sensitivity at the end of 2000 iterations; although the STD of the RMSE is small, the mean RMSE is still large. The LLSSIM converged faster and to a lower average RMSE than the BPA and ABPA. The LLSSIM algorithm was also the most stable, especially at early stages of the search. Note also that the periodic increase in STD of RMSE during the search (eg. figure 4d) correlates with periods of more rapidly improving function value.

## C. Function Approximation II

In this test case, an ANN was developed to approximate the more complex input-output relation shown in figure 7. The algorithm parameters, and stopping and augmentation criteria remained the same as in test case B. Figure 8 compares the average performance of 100 runs of

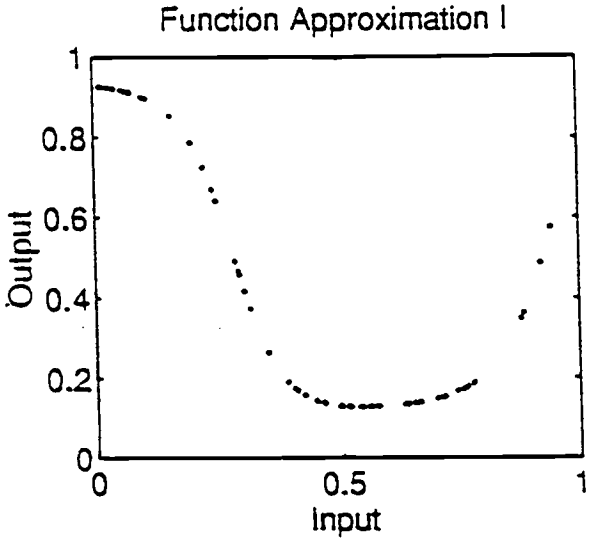


Fig. 5. Input and output relationship of function approximation problem I.

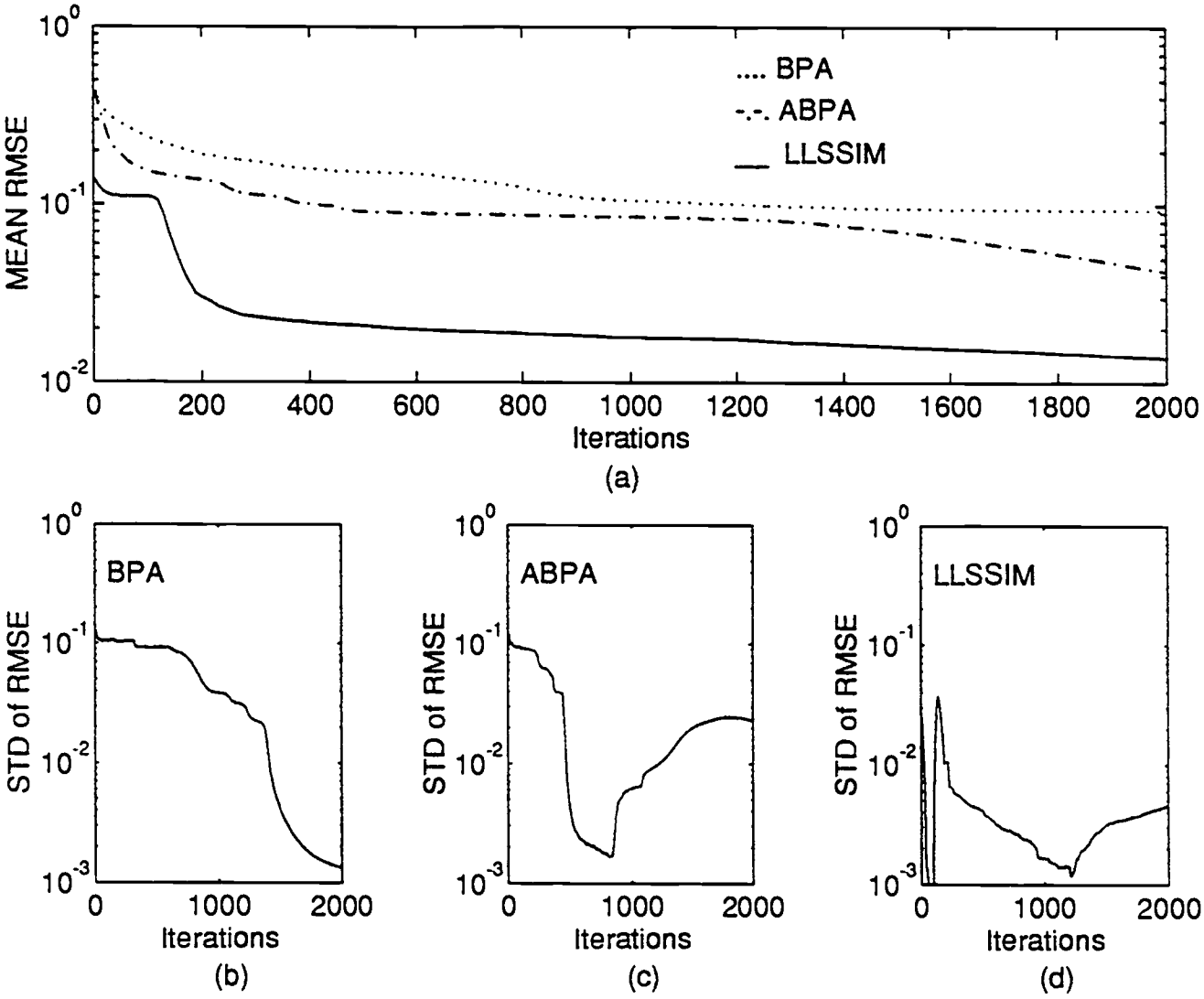


Fig. 6. Simulation results of function approximation I (100 independent runs): (a) Average performance of BPA, ABPA, and LLSSIM.(b) One standard deviation of RMSE of BPA. (c) One standard deviation of RMSE of ABPA. (d) One standard deviation of RMSE of LLSSIM.

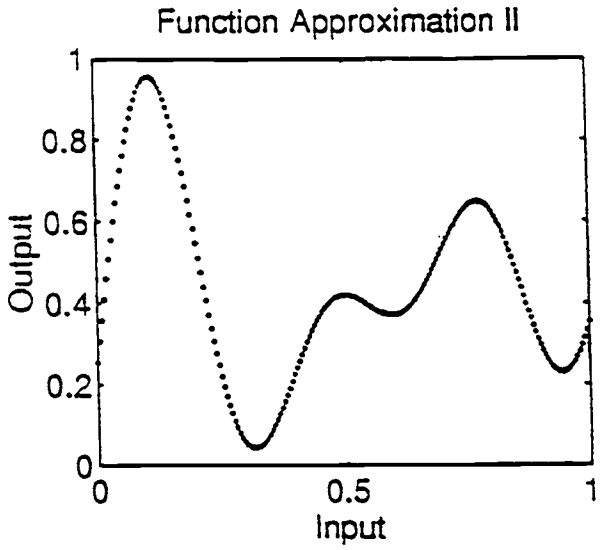


Fig. 7. Input and output relationship of function approximation problem II.

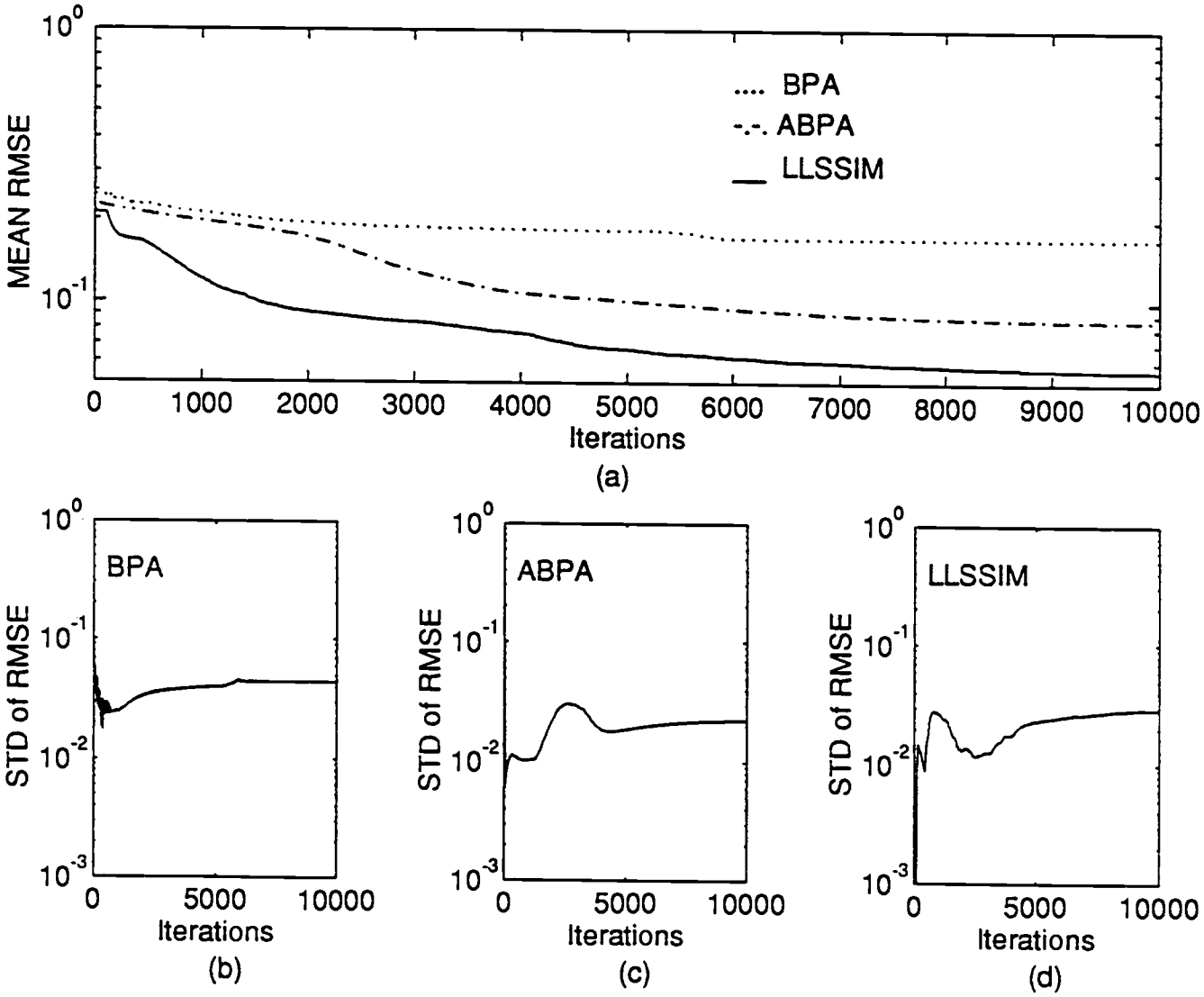


Fig. 8. Simulation results of function approximation II (100 independent runs): (a) Average performance of BPA, ABPA, and LLSSIM.(b) One standard deviation of RMSE of BPA. (c) One standard deviation of RMSE of ABPA. (d) One standard deviation of RMSE of LLSSIM.

each training algorithm. As in the previous cases, the LLSSIM method obtains significantly better RMSEs than BPA and ABPA, although the respective STD of RMSE values are similar.

**D. Rainfall Runoff (R-R) Modeling**

The rainfall-runoff process is extremely complex. It is influenced by several factors, including the temporal and spatial distribution of rainfall, the topographic and soil characteristics of the watershed, and the mechanisms by which water enters into long-term groundwater storage. Rain falling over a watershed may travel through many alternative paths en-route to becoming flow in a river. If the rainfall intensity is strong enough part of it may give rise to overland flow which travels quickly to the river. Other portions, however, may be detained on the surface or may infiltrate into the ground, taking substantially longer to reach the river.

The overall mechanism transforming rainfall into runoff is highly nonlinear, time-varying, and spatially distributed and is therefore not easily described by simple models [21]-[25]. However, many simplified conceptual and systems-theoretic rainfall-runoff models have been developed and used for several decades in order to aid in water resources planning and management processes [26]-[34]. In the example presented here, an ANN model was developed to represent the rainfall-runoff behavior of the Leaf River Basin (1949 $km^2$) near Collins, Mississippi. Six consecutive water years of areal-average daily rainfall and runoff records (see figure 9) from the basin were selected for model identification. One water year, WY 1982 (10-1-82 to 9-30-83) of data was used for network training, and five water years, WY 1983 to 1987, were used for network validation. The rainfall and runoff data were normalized to the range $[T_{min}, T_{max}]$, using the equation:

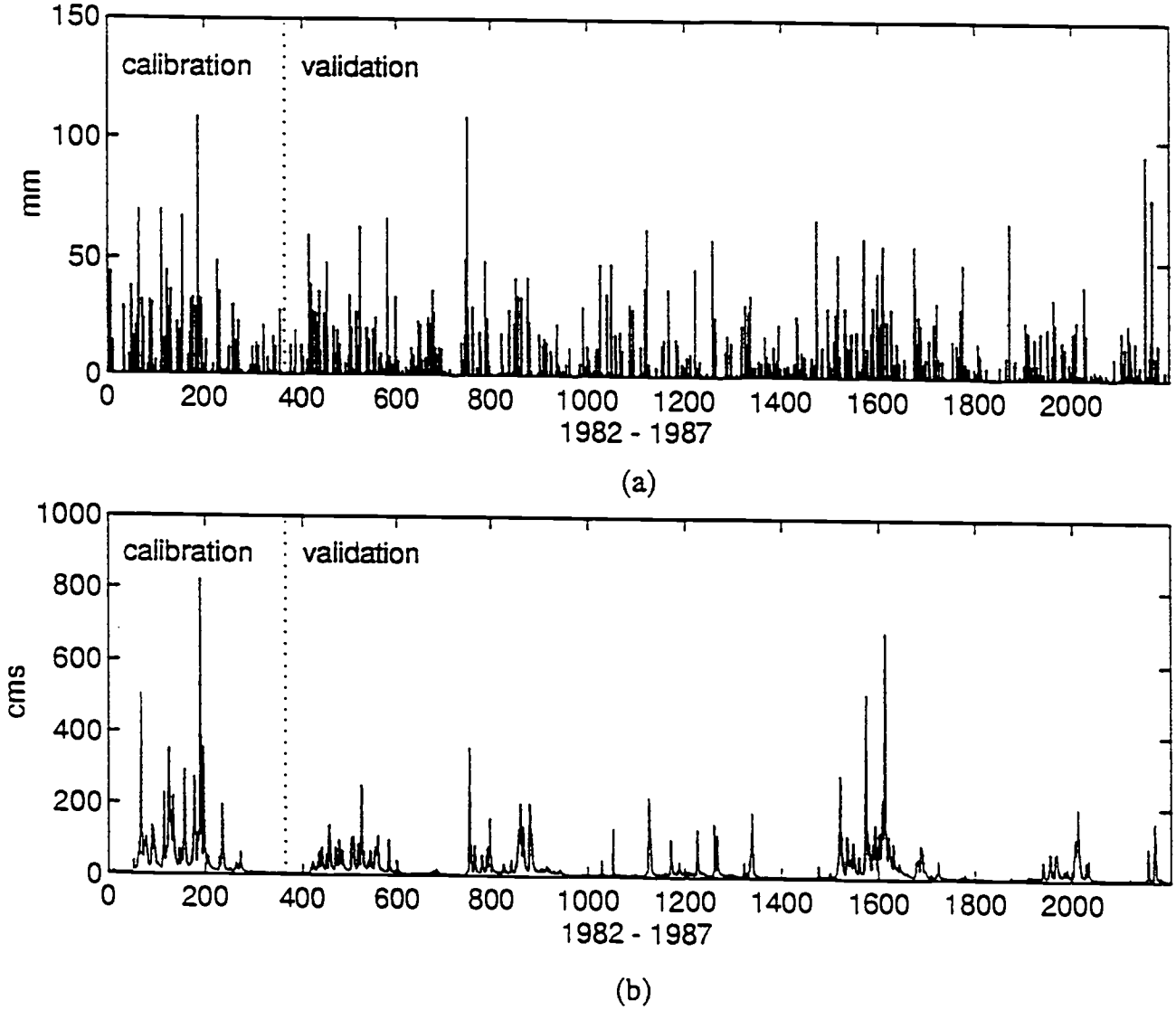


Fig. 9. (a) Daily rainfall record of Leaf River basin (b) Daily runoff record of Leaf River basin

$$N_i = T_{min} + \frac{X_i - X_{min}}{X_{max} - X_{min}} (T_{max} - T_{min}) \quad (12)$$

where $N_i$ is the normalized data. $X_i$ is the original data

$X_{min}$ is the minima of the original data

$X_{max}$ is the maxima of the original data

$T_{min}$ is the lower bound of the normalized data

$T_{max}$ is the upper bound of the normalized data

The rainfall series was normalized to the range [0,1] and the runoff series was normalized to the range [0.1,0.9].

A nonlinear auto-regressive moving average (NARMAX) model was considered to describe the runoff output z(t) as a nonlinear function of past inputs x(t-j) and outputs z(t-j) using the relationship:

$$y(t) = f_{non}(y(t-1), \ldots, y(t-n_a), u(t-1), \ldots, u(t-n_b)) + e(t) \quad (13)$$

where $f_{non}(.)$ is a nonlinear mapping function, $n_a$ and $n_b$ are the number of time lags of the past inputs and outputs contributing to the present output, and e(t) is the model error. An ANN model was used to identify the nonlinear mapping function $f_{non}(.)$.

The model lag parameters were set to $n_a$=3 and $n_b$=3, and no attempt was made in this study to find the optimal values for these parameters. The algorithm parameters, and stopping and augmentation criteria remained the same as in test case B. Figures 10a-d show the average performance of 100 runs of each training algorithm. As with the previous test cases, the LLSSIM

converges faster, to lower RMSE and is more stable than BPA and ABPA. For completeness, the ability of the best LLSSIM model to reproduce the training and validation hydrographs is shown in figure 11; this model had the structure ANN(6,8,1).

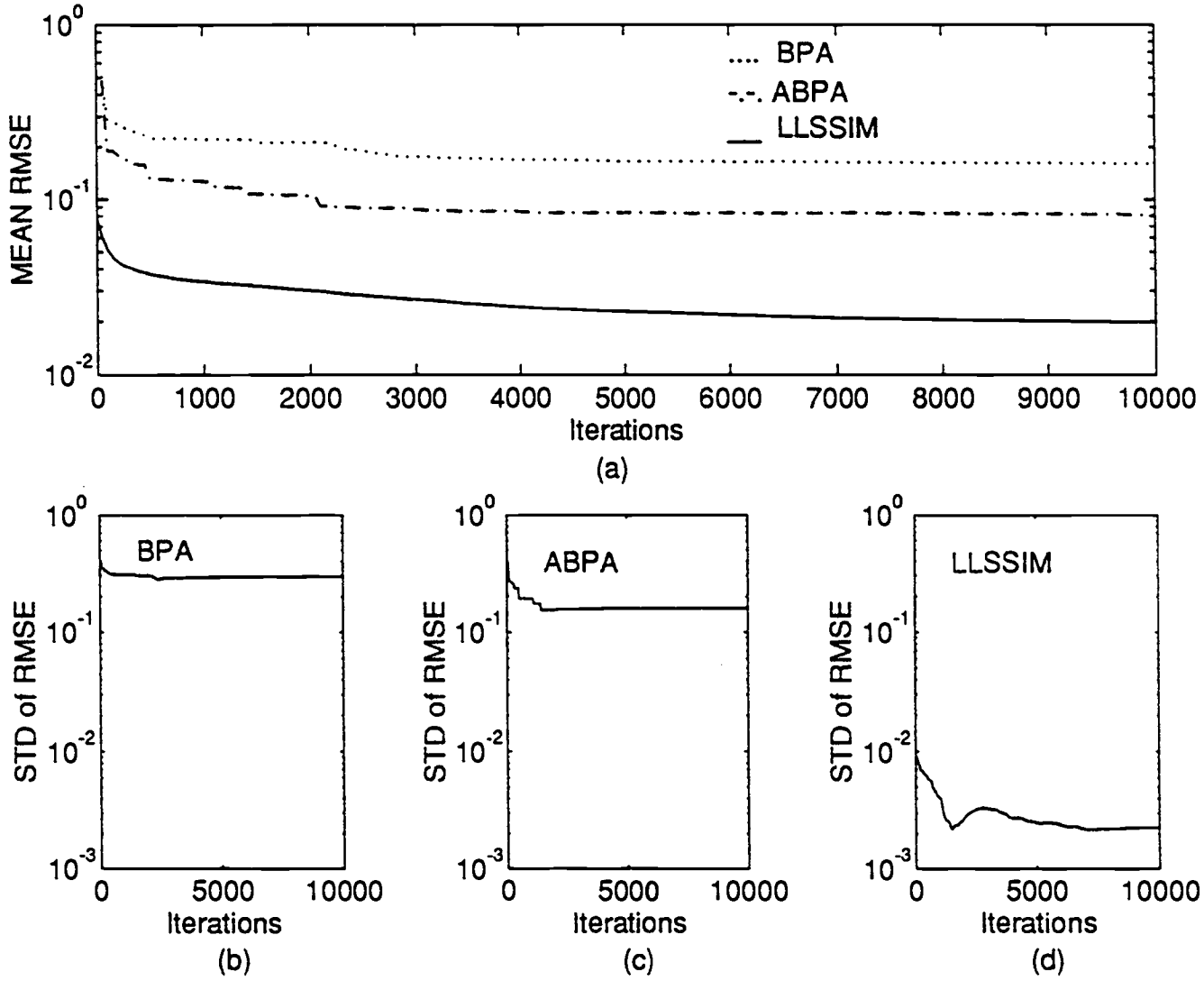


Fig. 10. Calibration results of rainfall runoff problem (100 independent runs): (a) Average performance of BPA, ABPA, and LLSSIM.(b) One standard deviation of RMSE of BPA. (c) One standard deviation of RMSE of ABPA. (d) One standard deviation of RMSE of LLSSIM.

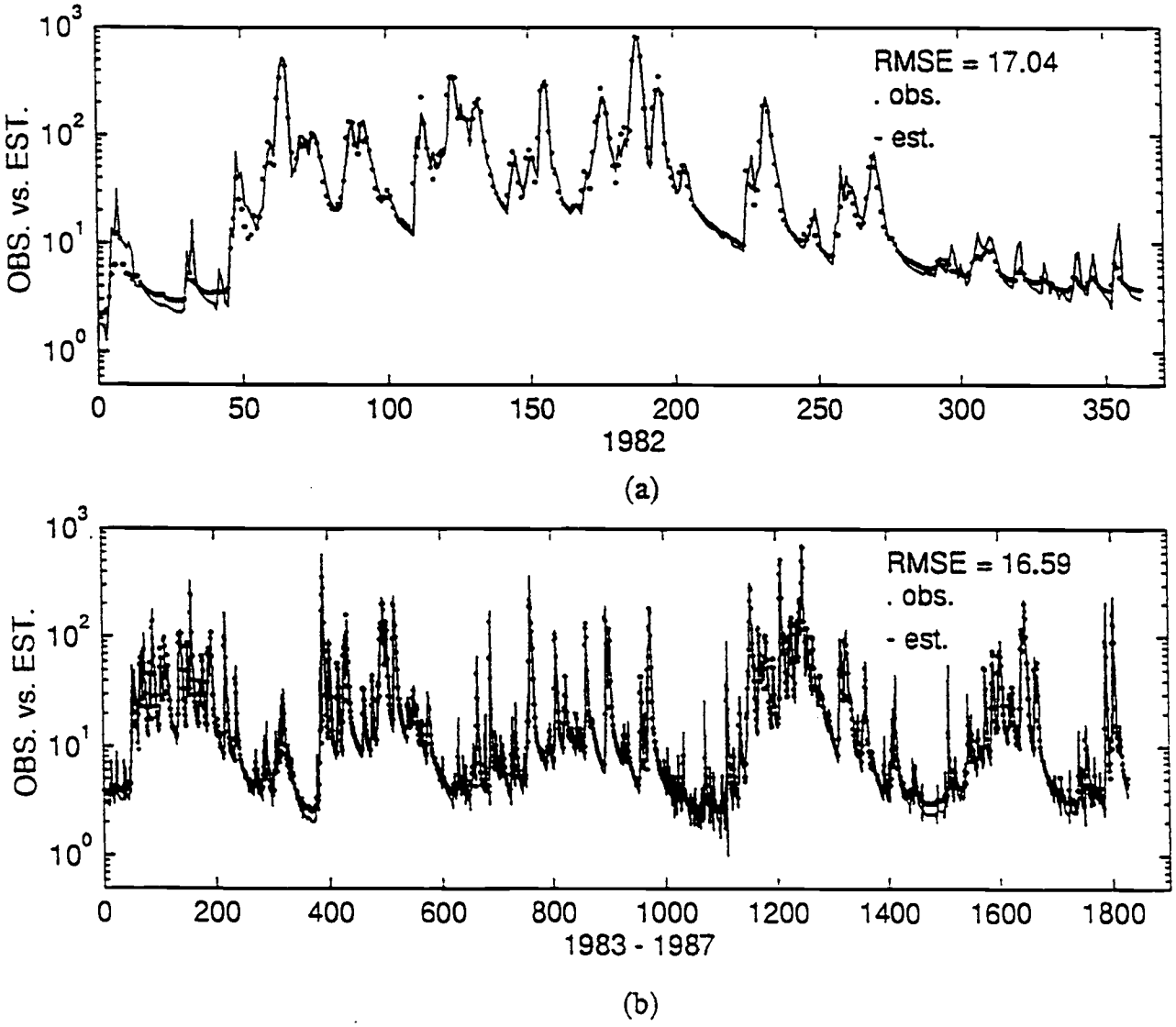


Fig. 11. (a) The calibration hydrograph (1982) (b) The validation hydrograph (1983-1987)

## V. SUMMARY AND CONCLUSIONS

This paper has presented a new effective and efficient network training algorithm (LLSSIM) suitable for identification of three-layer feedforward artificial neural network models. The method was coupled with a general ANN structural identification procedure, and tested on four problems, including a rainfall-runoff transfer function identification problem. The LLSSIM algorithm appears to converge more quickly and to lower RMSE values than the classical BPA and ABPA methods. It also appears to be more stable, being less affected by the initialization conditions. The reason for this is, we believe, the fact that the LLSSIM takes advantage of a weight space partition to conduct the non-linear input-hidden weight search in a reduced dimensional space, while the conditionally optimal hidden-output weights are computed using linear least squares. These results suggest that the average cost for training a three-layer feedforward network using the LLSSIM algorithm may be 5-10 times less than using the conventional BPA or ABPA methods. Further testing of the algorithm on problems requiring larger network sizes will be reported in a future paper.

# VI. APPENDIX

## A. Backpropagation Algorithm (BPA)

The equation used to update the weights from one iteration to the next is:

$$w(t+1)=w(t)+\eta\Delta w(t+1)+\alpha\Delta w(t) \qquad \text{(A1)}$$

where $\eta$ is the learning rate parameter, and $\alpha$ is a momentum factor ($0 < \alpha < 1$). The hidden-output layer weights updates, $\Delta w_{kj}^{o}(t+1)$, are updated using the equation:

$$\Delta w_{kj}^{o} = -\frac{\partial F}{\partial w_{kj}^{o}} = \sum_{p=1}^{m} \delta_{k}^{o}(p)\, y_{j}(p) \qquad \text{(A2)}$$

where $\delta_{k}^{o}(p) = (t_k(p) - z_k(p))\, f'(s_{k}^{o}(p))$ (A3)

and $f'(s_{k}^{o}(p))=(1-z_k(p))z_k(p)$ (A4)

The input-hidden layer weights, $\Delta w_{ji}^{h}(t+1)$, are updated using the equation:

$$\Delta w_{ji}^{h} = -\frac{\partial F}{\partial w_{ji}^{h}} = \sum_{p=1}^{m} \delta_{j}^{h}(p)\, x_{i}(p) \qquad \text{(A5)}$$

where

$$\delta_{j}^{h}(p) = f'(s_{j}^{h}(p)) \sum_{k=1}^{n_2} w_{kj}^{h}\delta_{k}^{o}(p) \qquad \text{(A6)}$$

and $f'(s_{j}^{h}(p))=(1-y_j(p))y_j(p)$ (A7)

## B. Adaptive Stepsize Backpropagation Algorithm (ABPA)

The learning rate ($\eta$) and momentum ($\alpha$) parameters determine the training speed of BPA. If the learning rate is too high, the result can be damping or divergence, while if the learning rate is too small, the algorithm can take a very long training time to converge (particularly if the training point reaches a flat region of the cost function). Strategies using learning rates and momentum factors that adapt during the search have been reported [4]. The procedures used in this paper were:

1) Learning rate: If the new iteration decreases the error, the learning rate is increased for the next iteration by multiplying it by a factor $\phi > 1.0$. If the error increases in the new iteration by more than 1% of the previous error, the new weights are discarded and the learning rate is decreased by multiplying it by a factor $\beta < 1.0$.

2) Momentum factor: Use the momentum factor $\alpha = 0.9$ if the previous iteration decreased the total error (this preserves a tendency to move in the same direction during the next iteration). Otherwise, set the momentum factor to zero.

## C. Multi-start Simplex Search Algorithm

The Simplex algorithm was developed by Nelder and Mead [14]. It begins with a geometric simplex consisting of n+1 points in an n-dimensional parameter space, and evolves the simplex in the direction of improving function value by the procedures of reflection, extension, contraction, and shrinking until the stopping criteria has been reached. This algorithm has both local and global search characteristics, and is not easily trapped by minor optima. By implementation of the Simplex algorithm in the context of a multi-start strategy, convergence to the global optimum with probability 1.0 can be guaranteed [16]. The multi-start algorithm used in this paper is outlined below:

step 0) initialize the multi-start counter and the termination criteria

Let nSIM=0 and assign the total number of independent simplex starts (nSTART).

Select a value for GCC (eg. GCC = $10^{-4}$ ).

step 1) initialize the "simplex":

Randomly select m=n+1 points in the allowable n-dimensional weight space and estimate the cost function value $F_i$ at each point $\mathbf{x}_i$, $\{(\mathbf{x}_i,F_i),i=1..m \text{ and } \mathbf{x}_i \subset \mathbf{R}^n\}$. The allowable weight space is defined by $w_{min} < \mathbf{x}_i < w_{max}$, for i = 1..m, where $w_{max}$ and $w_{min}$ are the upper and lower bounds on the weights.

step 2) sort the simplex and find the worst point:

Sort the m points in order of increasing function value, such that j=m has worst error function value, into array $\mathbf{D}=\{(\mathbf{x}_j,F_j),j=1..m\}$.

step 3) find the centroid:

Find the centroid, **g**, of the first n points.

step 4) test the reflection point:

Find the reflection point $\mathbf{r}=2\mathbf{g}-\mathbf{x}_m$. If the function value $F(\mathbf{r})$ less than $F(\mathbf{x}_m)$, then go to step 5; otherwise, go to step 6.

step 5) test the extension point:

Find the extension point $\mathbf{e}=3\mathbf{g}-2\mathbf{x}_m$. If $F(\mathbf{e}) < F(\mathbf{r})$, let $\mathbf{x}_m=\mathbf{e}$; otherwise, let $\mathbf{x}_m=\mathbf{r}$. Go to step 8.

step 6) test the contraction point:

Find the contraction point $\mathbf{c}=(\mathbf{g}+\mathbf{x}_m)/2$. If $F(\mathbf{c}) < F(\mathbf{x}_m)$, let $\mathbf{x}_m=\mathbf{c}$ and go to step 8; otherwise go to step 7.

step 7) shrink the simplex:

Shrink the simplex by $\mathbf{x}_j^{new}=(\mathbf{g}+\mathbf{x}_j^{old})/2, j=2..m$. Go to step 8.

step 8) test for convergence:

If the group convergence value (GCV) of the simplex is smaller than the group convergence criterion (GCC), let nSIM=nSIM+1 and go to step 9. If not, go to step 2. The GCV measures the size of the region spanned by the m points of the simplex and is defined as:

$$GCV=\left[\prod_{k=1}^{n}\frac{x_{max}^{k}-x_{min}^{k}}{BD}\right]^{1/n} \qquad (A8)$$

where $x_{max}^{k}$ is the maxima of all m points at parameter k

$x_{min}^{k}$ is the minima of all m points at parameter k

$BD=w_{max}-w_{min}$

step 9) test for termination of training:

If nSIM=nSTART, then stop; else save the best point and go to step 1.

## REFERENCES


[1] D. E. Rumelharts, E. Hinton, and J. Williams, "Learning internal representation by error propagation," in Parallel Distributed Processing, vol. 1, pp318-362. Cambridge, MA: MIT Press, 1986.

[2] R. A. Jacob, "Increased rates of convergence through learning rate adaptation," Neural Networks, vol. 1, pp. 295-307, 1988.

[3] T. Tollenaere, "SuperSAB: fast adaptive back propagation with good scaling properties," Neural Networks, Vol. 3, pp. 561-573, 1990.

[4] T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, "Accelerating the convergence of the back-propagation method," Biological Cybernetics, vol. 59, pp. 257-263, 1988.

[5] D. B. Parker, "Optimal algorithms for adaptive networks: second-order backpropagation, second-order direct propagation, and second order Hebbian learning," in Proc. 1st Int. Conf. Neural Networks, vol. 2, pp. 593-600, 1987.

[6] R. Battiti, "First and second order methods for learning: between steepest decent and newton's method," Neural Computation, Vol. 4, no.2, pp. 141-166, 1992.

[7] C. Charalambous, "Conjugate gradient algorithm for efficient training of artificial neural networks," IEE Proceedings, Vol. 139, no. 3, 1992.

[8] T. Masters, "Practical neural network recipes in C++," Academic Press, Inc., pp. 105-116, 1993.

[9] M. A. Styblinski, and T. S. Tang, "Experiments in nonconvex optimization: stochastic

approximation with function smoothing and simulated annealing." Neural Networks, Vol. 3, pp. 467-483, 1990.

[10] E. Aarts and J. Korst, "Simulated annealing and boltzmann machines," Wiley, New York, 1989.

[11] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," Proceedings of the Eleven International Joint Conference on Artificial Intelligence, Vol. 1, pp.762-767, 1989.

[12] M. McInerney and A. P. Dhawan, "Use of genetic algorithms with back propagation in training of feed-forward neural networks," in proc. Int. Joint Conf. Neural Networks, vol. I, pp. 203-208, 1993

[13] D. B. Fogel, "An introduction to evolutionary Optimization, IEEE Trans. on Neural Networks, Vol 5, no. 1, pp.3-14, 1994.

[14] A. J. Nelder and R. Mead, "A simplex method for function minimization," Compu. J., Vol. 7, no. 4, pp. 308-313, 1965.

[15] R. S. Scalero and N. Tepedelenlioglu, "A fast new algorithm for training feedforward neural networks", IEEE Trans. on Signal Processing, Vol. 40, no. 1, 1992.

[16] Q. Duan, S. Sorooshian, and V. K. Gupta, "Effective and efficient global optimization for conceptual rainfall runoff models," Water Resour. Res., Vol. 28, no. 4, pp.1015-1031, 1992.

[17] Q. Duan, V. K. Gupta, and S. Sorooshian, "A shuffled complex evolution approach for effective and efficient global minimization, J. Optimiz. Theory Appl., Vol. 73, no. 3, pp. 501-521, 1993.

[18] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," IEEE Trans. Neural Networks, Vol. 1, no. 2, pp. 239-242, 1990.

[19] T. Chang, K. A. S. Abdel-Ghaffar, "A universal neural net with guaranteed convergence to zero system error," IEEE Trans. on Signal Processing, Vol. 40, no. 12, pp. 3022-3031, 1992.

[20] J. Hertz, A. Krogh, and R. G. Palmer, "Introduction to the theory of neural computation," Addison-Wesley, Inc., pp. 156-162, 1991.

[21] V. P. Singh, "Nonlinear instantaneous unit hydrolograph theory," J. of the Hydraulics Devision, Proc. of American. Society of Civ. Eng., 90(HY2), pp. 313-347, 1964.

[22] J. Amorocho, "The nonlinear prediction problem in the study of runoff cycle," Water Resour. Res., Vol. 3, no. 3, pp. 861-880, 1967.

[23] X. C. Kulandaiswamy, and C. V. Subramanian, "A nonlinear approach to runoff studies," Proc. of the Int. Hydrology Symposium, Fort Collins, Colo, 1, pp. 92-79, 1967.

[24] C. L. Chiu, and J. T. Huang, "Nonlinear time-varying model of rainfall-runoff relation", Water Resour. Res., Vol. 6, no. 1, pp. 1277-1286, 1970.

[25] D. H. Pilgrim, "Travel times and nonlinearity of flood runoff from tracer measurements on a small watershed," Water Resour. Res., Vol 12, no. 3, pp. 487-496, 1976.

[26] R. J. E. Burnash,, R. L. Ferral, and R. A. McGuire, "A generalized streamflow simulation system," report 220, Joint Fed.-State River Forecasting Center, Sacramento, Calif., 1973.

[27] I. E. Brazil, and M. D. Hudlow, "Calibration procedures used with the National Weather Service Forecast System," Proc. of the IFAC Symposium -- Water and Related Land Resource Systems, Cleveland, Ohio, 1980.

[28] U. S. Army, Crops of Engineers, "HEC-1 flood hydrograph package," User programmers

manuals, HEC Program 723-X6-L2010, 1973

[29] N. H. Crawford and R. K. Linsley, "Digital simulation in Hydrology: Stanford watershed model IV," Tech. Rep. 39, Dept. of Civ. Eng., Stanford, Calif., 1966.

[30] J. Amorocho and W. E. Hart, "A critique of current methods of hydrologic systems investigation," Trans. of the American Geophysical Union, Vol. 45, pp. 307-321, 1964.

[31] S. Sorooshian, Q. Duan, and V. K. Gupta, "Calibration of rainfall-runoff models: Application of global optimization to the Sacramento Soil Moisture Accounting model," Water Resour. Res., Vol. 29, no. 4, 1993.

[32] V. K. Gupta, "The identification of conceptual watershed models," Ph.D. Dissertation, Case Western Reserve University, Cleveland, Ohio, 1984.

[33] V. P. Singh, "Hydrologic systems: rainfall-runoff modeling," Prentice Hall Press, 1988.

[34] Duan, Q., A global optimization strategy for efficient and effective calibration of hydrologic models, Ph.D. Dissertation, University of Arizona, Tucson, Arizona, 1991.