Turkish Journal of Electrical Engineering and Computer Sciences

Volume 20 | Number 4 Article 10

1-1-2012

Simulated annealing algorithm-based Elman network for dynamic system identification

ADEM KALINLI

TÜBİTAK

Turk J Elec Eng & Comp Sci, Vol.20, No.4, 2012, © TÜBİTAK
doi:10.3906/elk-1012-942

Simulated annealing algorithm-based Elman network for dynamic system identification

Adem KALINLI
Kayseri Vocational High School, Computer Division, Erciyes University, 38039 Kayseri-TURKEY
e-mail: kalinlia@erciyes.edu.tr

Received: 27.12.2010

Abstract

One of the well-known recurrent neural networks is the Elman network. Recently, it has been used in applications of system identification. The network has feedforward and feedback connections. It can be trained essentially as a feedforward network by means of the basic backpropagation algorithm, but its feedback connections have to be kept constant. For training success, it is important to select the correct values for the feedback connections. However, finding these values manually can be a lengthy trial-and-error process. This paper investigates the use of the simulated annealing (SA) algorithm to obtain the weight values of both the feedforward and feedback connections of Elman networks used for dynamic system identification. The SA algorithm is an efficient random search procedure, which can simultaneously obtain the optimal weight values of both the feedforward and feedback connections.

Key Words: *Elman network, simulated annealing, tabu search, dynamic system identification*

1. Introduction

The use of artificial neural networks (ANNs) to identify or model dynamic inputs is a topic of much research interest. The advantage of neural networks for these types of applications is their ability to learn the behaviour of a system without much a priori knowledge about it. From a structural point of view, there are 2 main types of neural networks: feedforward neural networks (FNNs) and recurrent neural networks (RNNs) [1]. When the connections in a network allow the flow of information from the input layer to the output layer in one direction only, it is called a FNN. Connections that allow information to loop back to the same processing element are called recursive, and neural networks with these types of connections are known as RNNs. Feedforward networks can implement static input-output mapping. However, networks with a dynamic memory or recurrent networks are more suitable for representing a dynamic system, which has dynamic mapping between its input(s) and output(s). FNNs generally require a large number of input neurons and thus necessitate a long computation time as well as having a high probability of being affected by external noise. Due to their structure, RNNs do not suffer from these drawbacks. RNNs have attracted the attention of researchers in the field of dynamic system identification [1-9].

Although gradient-based search techniques such as backpropagation (BP) are currently the most widely used optimisation techniques for training neural networks, it has been shown that these techniques are severely limited in their ability to find global solutions. Global search techniques such as the genetic algorithm (GA), simulated annealing (SA), tabu search (TS), and particle swarm optimisation (PSO) have been identified as potential solutions to this problem. Although the use of GAs for ANN training has mainly focused on FNNs [10-12], there are several works on training RNNs using GAs in the literature [8,13-15]. However, SA, TS, and PSO have not found as many applications for the training purposes of ANNs [16-22].

A special type of RNN is the Elman network [23]. The Elman network and its modified models have been used in applications of system identification. These networks have feedforward and feedback connections. However, so that they can be trained essentially as feedforward networks by means of the simple BP algorithm, their feedback connections have to be kept constant. For the training to converge, it is important to select the correct values for the feedback connections. However, finding these values manually can be a lengthy trial-and-error process. In order to improve the performance of Elman networks, Kalinli and Karaboga introduced an approach to train some network models for dynamic system identification based on the TS algorithm [19]. Furthermore, feedforward connections of a modified Elman network were trained using the SA algorithm [18]. They showed that the performances of the TS and SA algorithms were better than that of the BP algorithm as introduced in [3,4].

In order to improve the performance of the Elman networks introduced in [3,5,19], in this paper, the SA algorithm is also used to obtain the weight values of both the feedforward and feedback connections of the Elman networks. The performance of the SA algorithm was also compared with the results of the BP and TS algorithms taken from [5,19]. Section 2 gives the basic principles of the SA and TS algorithms. Section 3 reviews the dynamic system identification and the structure of the Elman network and explains how SA was used to train the given networks to identify a dynamic system. The results obtained for 9 different single-input single-output (SISO) systems are covered in Section 4. The work is concluded in Section 5.

2. Simulated annealing and tabu search algorithms

The great difficulty encountered by optimisation problems in practical areas such as production, control, communication, and transportation has motivated researchers to develop new powerful algorithms. The most popular of these new algorithms include GAs, SA, ant colony optimisation (ACO), and TS. Although all of these algorithms have a convergence property to the global optimum, they cannot always guarantee the optimum solutions for the problem. Therefore, they are called approximate or heuristic algorithms.

One group of approximate algorithms for optimisation problems is known as the iterative improvement (local search) method. An iterative improvement algorithm starts from a feasible solution and repeatedly seeks to improve it by altering the solution through the application of search mechanisms until no further improvements are possible. At this stage, the algorithm stops at a local optimum solution. The local optimum depends heavily on the starting feasible solution and the search mechanisms. These local optima are often of low quality. Recent advances in the design of sophisticated approximate methods have resulted in the development of innovative approaches called metaheuristic algorithms that overcome some of the disadvantages and limitations of local search methods. Two of these metaheuristic algorithms are the SA and TS algorithms [24,25].

Both SA and TS algorithms are devised so as to avoid being trapped in poor local optima. They allow the local search method to be continued after a local minimum is detected at which no further search for a

global minimum can be performed. Hence, they can be viewed as enhanced versions of local search techniques.

The SA algorithm is particularly effective for the optimisation of nonlinear and multimodal functions. In comparison with other optimisation techniques, the SA algorithm has 2 unique features. First, it is not easily fooled by the quick payoff achieved by falling into unfavourable local minima. Second, configuration decisions tend to proceed in a global order. The algorithm initially explores the objective function with coarse detail over a wide region, progressing to finer detail in regions where global optima are likely to exist. The algorithm employs a random search that not only accepts the changes that reduce the objective function, but also some changes that increase it. Although SA has found many applications in both combinatorial and numerical optimisation areas, the TS algorithm has been generally applied to solve combinatorial-type optimisation problems rather than numerical ones.

2.1. Simulated annealing algorithm

SA is a random-search technique that is motivated by an analogy to annealing in solids. The algorithm employs a random search that not only accepts changes that reduce the objective function (assuming a minimisation problem), but also some changes that increase it, which is accepted according to the Metropolis criterion [1]. The Metropolis criterion always accepts the perturbed solution as the next current solution if its cost is lower than that of the current solution. The criterion also allows for the probabilistic acceptance of higher-cost perturbed solutions as the next current solutions. Probabilistic acceptance enables the SA algorithm to avoid becoming trapped in local minima. According to the Metropolis criterion, if the difference between the cost function values of the current and the newly produced solutions, Δ, is equal to or larger than zero, a random number δ in [0,1] is generated from a uniform distribution, and if $\delta \leq e^{\Delta/T}$, then the newly produced solution is accepted as the current solution. Otherwise, the current solution remains in its current state. The main steps of the basic SA algorithm are shown in Figure 1.

Step 1. Generate an initial solution, S
Step 2. Choose a solution $S' \in N(S)$ and compute the difference in the objective values,
$\Delta = C(S) - C(S')$
Step 3. **If**:
(i) S' is better than S ($\Delta > 0$), **or**
(ii) $\delta \leq e^{\Delta/T}$
Then replace S by S' ($S \leftarrow S'$)
Else retain the current solution
Step 4. Update the temperature
Step 5. **If** a "stopping criterion" is satisfied STOP, **else** GO TO step 2

Figure 1. Main steps of the basic SA algorithm.

In order to use the algorithm for a problem, important factors that must be taken into consideration when making choices fall into 2 classes [26]:

a) Problem-specific choices: representation of possible solutions, definition of the cost function, and the generation mechanism for the neighbours.

b) Generic choices for cooling schedules: the initial value of the temperature, the cooling rate and the temperature update rule, the number of iterations to be performed at each temperature, and the stopping criterion.

The performance of the SA algorithm depends strongly on the chosen cooling schedule. A great variety of cooling schedules have been suggested by many authors in the literature. One of them is the geometric cooling rule. This rule updates the temperature sequence in the following way: $T(t+1) = r.T(t)$, $t = 0, 1, \ldots$, where r is a temperature factor that is a constant smaller than 1.0 but close to 1.0. Experience has shown that r should be between 0.8 and 0.99 [1].

2.2. Tabu search algorithm

The TS is a general heuristic search procedure devised for finding a global minimum of the function $f(\boldsymbol{x})$, $\boldsymbol{x} \in \hat{\mathrm{I}}\mathrm{X}$. The function $f(\boldsymbol{x})$ may be linear or nonlinear and the condition $\boldsymbol{x} \in \hat{\mathrm{I}}\mathrm{X}$ describes the constraints on the solution $\boldsymbol{x}$. The first step of the TS starts with the present solution $\boldsymbol{x}_{now}$. $\boldsymbol{x}_{now} \in \hat{\mathrm{I}}\mathrm{X}$ has an associated set of feasible solutions, Q, which can be obtained by applying a simple modification to $\boldsymbol{x}_{now}$. This modification is called a move. In order to be able to get rid of local minima, a move to the neighbour, $\boldsymbol{x}^*$, is created even if $\boldsymbol{x}^*$ is worse than $\boldsymbol{x}_{now}$. This causes the cycling of the search. In order to avoid the cycling problem, a tabu list, T, is introduced. The tabu list stores all of the tabu moves that cannot be applied to the present solution, $\boldsymbol{x}_{now}$. The moves stored in the tabu list are those carried out most frequently and recently according to some criteria called tabu restrictions. The use of the tabu list reduces the possibility of cycling because it prevents returning within a certain number of iterations to a solution visited recently. After a subset of feasible solutions, Q*, is produced according to the tabu list and evaluated for $f(\boldsymbol{x})$, the next solution is selected from it. The highest evaluation solution is selected as the next solution, $\boldsymbol{x}_{next}$. This loop is repeated until one of the stopping criteria is satisfied.

The TS algorithm usually uses the following 2 constraints, which are based on recency and frequency memories: $recency(\boldsymbol{x}^*) \geq restriction\_period$ and $frequency\_ratio(\boldsymbol{x}^*) \leq frequency\_limit$. The recency of a move is the difference between the current iteration count and the last iteration count at which the move was created. The frequency of a move is the count of its changes. The frequency measure is the frequency ratio, whose numerator represents the count of the number of occurrences of a specific move and whose denominator represents the average numerator value over all possible moves. A tabu restriction is activated when the reverse of a move recorded in the tabu list occurs within a predetermined number of iterations or with a certain frequency over a longer range of iterations. The former produces a recency-based restriction and the latter a frequency-based restriction. Tabu restrictions may sometimes prevent the search from finding solutions that have not yet been visited or even cause all available moves to be classified as the tabu. Therefore, it should be possible to forget the tabu constraints when a freedom is required for the search. A criterion called the aspiration criterion is employed to determine which moves should be freed in such cases.

The flowchart of a basic TS is presented in Figure 2. In the initialisation unit, a random feasible solution, $\boldsymbol{x}_{initial} \in \hat{\mathrm{I}}\mathrm{X}$, for the problem is generated, and the tabu list and other parameters are initialised. In the neighbour production unit, a feasible set of solutions is produced from the present solution according to the tabu list and aspiration criteria. The evaluation unit evaluates each solution, $\boldsymbol{x}^*$, produced from the present $\boldsymbol{x}_{now}$ solution. After the next solution, $\boldsymbol{x}_{next}$, is determined by the selection unit, in the last unit the history record of the search is modified. If the next determined solution is better than the best solution found so far, $\boldsymbol{x}_{best}$, the next solution is replaced with the present best solution.

In the algorithm used, the highest evaluation move is selected as the next solution. The aspiration by default is employed as the aspiration criterion. According to this criterion, the least tabu solution is selected as

the next solution. This is the solution that loses its tabu classification by the least increase in the value of the present iteration number.

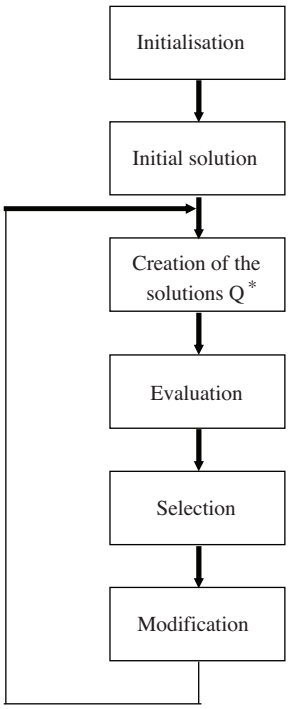

Figure 2. Flowchart of the basic tabu search.

3. Training Elman network using SA algorithm for system identification

3.1. Dynamic system identification

System identification is the process of constructing a model for an unknown dynamic system and estimating its parameters from experimental data. The behaviours of the systems can be linear or nonlinear. The objective of system identification techniques is to establish a suitable model and adjust it to approximate the system under investigation. The adjustment of the model is usually based on the error between the system output and model output [3].

The area is already fairly mature, with various conventional techniques being developed and implemented in practical applications. However, most of the identification methods, such as those based on least mean squares or maximum likelihood estimates, are in essence gradient-guided local search techniques. They require a smooth search space or a differentiable error energy function. These conventional approaches can thus easily fail in obtaining the global optimum if the multimodal search space is not differentiable or the performance index is not well-behaved in practice. Furthermore, these conventional identification methods cannot easily be applied to nonlinear systems [27,28]. Therefore, more research is needed in order to have systematic and generally applicable approaches for system identification.

Neural networks have proven themselves to be useful in many application areas, because they possesses many advantages, such as a general-purpose nature, massive parallelism, nonlinear properties, adaptive learning

capabilities, and powerful fault tolerant capabilities, and they are suitable for simple very-large-scale integration implementations. The ability of neural networks to deal with nonlinear systems is of prime interest in control engineering. Therefore, neural networks constitute a promising tool for system identification and control. Compared to FNNs, the advantage of using RNNs for identification is that fewer processing elements are required and the identifiers are less noise-sensitive. The identification structure based on RNNs is parallel, as shown in Figure 3.

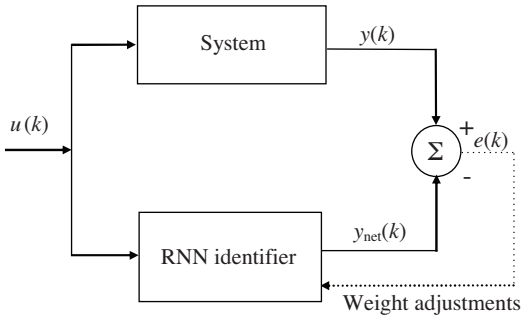

Figure 3. A RNN identifier for system identification.

3.2. Training Elman network using SA algorithm

The Elman network is a special type of RNN. Figure 4 depicts the original Elman network with 3 layers of neurons [23]. U(.), X(.), C(.), Y(.), and W in Figure 4 represent the inputs to the network, the outputs of the hidden units, the outputs of the context units, the outputs of the network, and the weight vector, respectively. The first layer of this network consists of 2 different groups of neurons. These are the group of external input neurons and the group of internal input neurons, also called context units. The inputs to the context units are the outputs of the hidden neurons. The outputs of the context units and the external input neurons are fed to the hidden neurons. Context units are also known as memory units, as they store the previous output of the hidden neurons. In the Elman network, the values of the feedback connection weights have to be fixed by the user if the basic BP algorithm is employed to train the network; usually, their strengths are fixed at 1.0 [7].

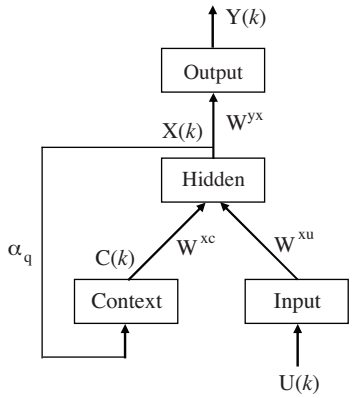

Figure 4. Structure of the Elman network.

The following equations are generated from this representation:

$$X(k) = F\left\{W^{xc}C(k), W^{xu}U(k)\right\}, \tag{1}$$

$$C(k) = X(k-1), \tag{2}$$

$$Y(k) = W^{yx}X(k), \tag{3}$$

where W^{xc}, W^{xu}, and W^{yx} are weight matrices and F is a function. In particular, when linear hidden units are adopted and the biases of the hidden and output units are zero, Eqs. (1) and (3) become:

$$X(k) = W^{xc}X(k-1) + W^{xu}U(k), \tag{4}$$

$$Y(k) = W^{yx}X(k). \tag{5}$$

If inputs U(k) are delayed by one step before they are sent to the input units in Eq. (4), the following equation is obtained:

$$X(k) = W^{xc}X(k-1) + W^{xu}U(k-1). \tag{6}$$

Eqs. (5) and (6) are standard state-space descriptions of dynamic systems. The order of the model depends on the number of states, which is also the number of hidden units. Therefore, theoretically, the Elman network is able to model an nth-order dynamic system [3]. When a neural model is used to model a SISO system, only one unit is needed in the input and output layers.

In this work, the SA algorithm is used to train the weights of the Elman network, assuming that the structure of the network has been decided. That is, the number of layers, the type of activation functions and number of neurons in each layer, the pattern of connections, the permissible ranges of trainable connection weights, and the values of constant connection weights are all known. The SA algorithm used in this work is the one described in [1]. An initial solution is produced with a random number generator. Using this solution representing a possible weight set of the Elman network, the SA algorithm searches for the best weight set by means of some strategies.

A solution to the problem is a string comprising n elements, where n is the number of trainable connections (Figure 5). When all connections are trainable, feedback connections have weight values in the range of 0.0 to 1.0, while feedforward connections can have positive or negative weights between –1.0 and 1.0. Note that from the point of view of the SA algorithm, there is no difference between feedforward and feedback connections, and training one type of connection is carried out identically to training the other, unlike the case of the commonly used BP training algorithm.

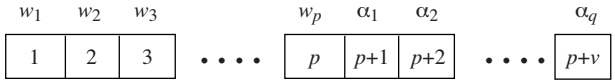

Figure 5. Representation of the trainable weights of a RNN in string form.

The use of SA to train a RNN to identify a dynamic system is illustrated in Figure 6. Here, $y_m(k)$ and $y_p(k)$ are the outputs of the network and the system at time k, respectively. The training of the Elman network can be considered as a minimisation problem defined by:

$$\min_{w \in W} J(w), \tag{7}$$

where $W = [w_{11}w_{12}w_{13}...w_{pq}]^T$ is the weight vector of the RNN. The time-averaged cost function $J(\mathbf{w})$ to be minimised by adaptively adjusting $\mathbf{w}$ can be expressed as:

$$\min J(w) = \left(\frac{1}{M} \sum_{k=1}^{M} \left(y_p(k) - y_m(k) \right)^2 \right)^{1/2}, \tag{8}$$

where M is the number of samples used for the calculation of the cost function.

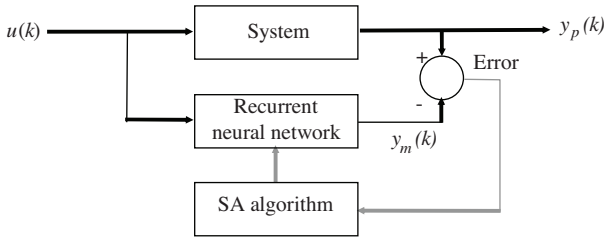

Figure 6. Scheme for training a RNN to identify a system using the SA algorithm.

In the training process, first of all, a sequence of the input signals $u(k)$, (k=0,1,...) is fed to both the system and the RNN, which is designed with the weights obtained from the current solution. After that, the root mean square (rms) error value between the system and network outputs is computed using Eq. (8). According to the rms error values computed for the current solution and the new solution in the neighbourhood of the current solution, the next solution is then selected. If the new solution has less error, the new solution is taken directly as the current solution. Otherwise, it is accepted according to the Metropolis criterion.

4. Simulations results

The structure of the network employed in this work was selected from [4,5,19] to compare the results. Since the systems to be identified are SISO, a number of the external input and output neurons will have linear activations. The number of neurons in the hidden layer is equal to 6. In cases where only feedforward connections are trainable, a solution is represented as a string of 48 weights. When all connections have trainable weights, then the string consists of 54 weights, of which 6 are feedback connection weights. Simulations were conducted to study the ability of the RNN trained by the SA algorithm to model 1 linear and 2 nonlinear systems. A sampling period of 0.1 s was assumed in all cases.

System 1. This is a third-order linear system described with the following discrete-time equation:

$$y(k) = A_1 y(k-1) + A_2 y(k-2) + A_3 y(k-3) + B_1 u(k-1) + B_2 u(k-2) + B_3 u(k-3), \tag{9}$$

where $A_1 = 2.627771$, $A_2 = -2.333261$, $A_3 = 0.697676$, $B_1 = 0.017203$, $B_2 = -0.030862$, and $B_3 = 0.014086$.

The Elman network with all linear neurons was tested. The training input signal, u(k), k=0,1,...99, was randomly produced and varied between –2.0 and 2.0. First, the results were obtained by assuming that only the feedforward connection weights were trainable. Next, the results were obtained by considering that all of the connection weights of the Elman network were trainable. For each case, experiments were repeated 6 times for different initial solutions. The results obtained using the BP, SA, and TS algorithms are given in Figure 7. The performance of SA was also compared with the results of the basic TS algorithm taken from [19]. The average rms error values and the improvement percentages for this system were obtained using the BP, SA, and TS algorithms presented in Table 1. As an example, the responses of the system and the network designed by SA are illustrated in Figure 8.

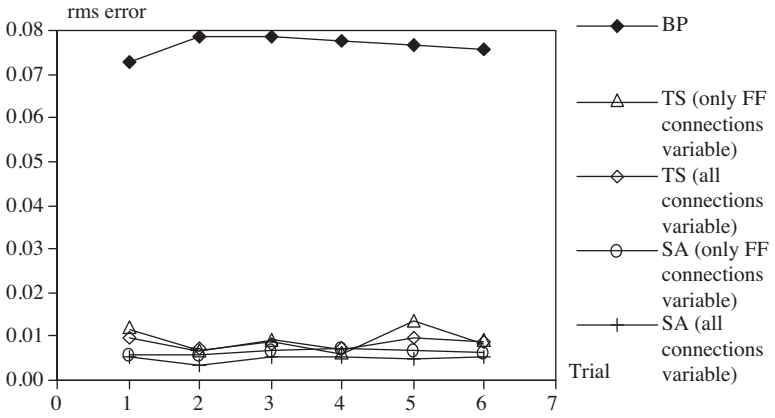

Figure 7. The rms error values obtained for System 1 for 6 different runs.

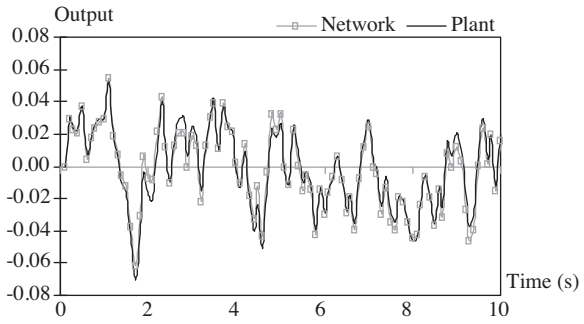

Figure 8. Responses of the system and the network trained by the SA algorithm (rms error = 5.22282E-03).

Table 1. Comparison of the results for System 1.

Model	Average rms error	Improvement (%)
BP	7.6754E-02	-
TS ($\alpha = 1$)	9.6077E-03	87.48
TS (all weights trainable)	8.1664E-03	89.36
SA ($\alpha = 1$)	6.5135E-03	91.51
SA (all weights trainable)	4.9653E-03	93.53

System 2. The second system model adopted for simulations was that of a simple pendulum swinging through small angles [3]. This is a second-order nonlinear system and the discrete time description of the system is:

$$y(k) = A_1 y(k-1) + A_2 y(k-2) + A_3 y^3(k-2) + B_1 u(k-2), \tag{10}$$

where $A_1 = 1.040000$, $A_2 = -0.824000$, $A_3 = 0.130667$, and $B_1 = -0.160000$. The Elman network with nonlinear neurons in the hidden layer was employed. The hyperbolic tangent function was adopted as the activation function of nonlinear neurons. The networks were trained using the same sequence of random input signals as mentioned above.

As in the case of System 1, the results were obtained for 6 different initial solutions. The rms error values obtained by using the BP, SA, and TS algorithms are presented in Figure 9. The time response obtained using the SA algorithm for the Elman network, assuming that all of the connections were trainable, is presented in

Figure 10. The average rms error values and the improvement percentages for the system obtained by using the BP, SA, and TS algorithms are presented in Table 2.

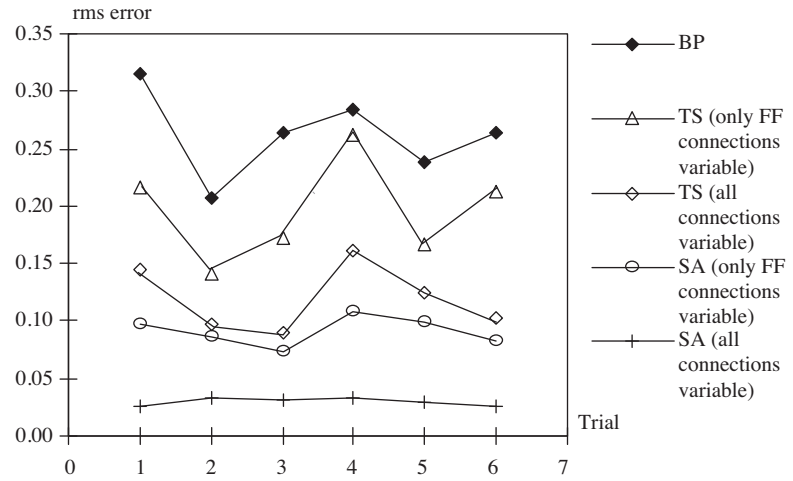

Figure 9. The rms error values obtained for System 2 for 6 different runs.

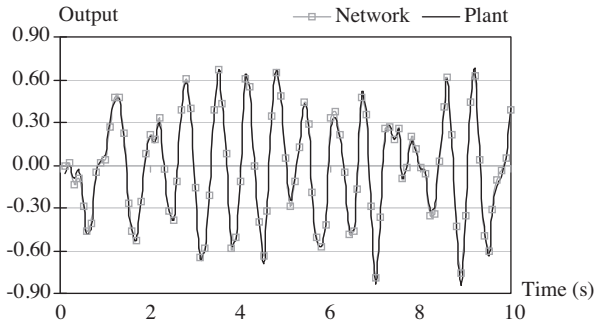

Figure 10. Responses of the system and the network trained by the SA algorithm (rms error = 2.57737E-02).

Table 2. Comparison of the results for System 2.

Model	Average rms error	Improvement (%)
BP	2.6182E-01	-
TS ($\alpha = 1$)	1.9589E-01	25.18
TS (all weights trainable)	1.1990E-01	38.79
SA ($\alpha = 1$)	9.1310E-02	65.12
SA (all weights trainable)	2.9430E-02	88.76

System 3. This is a nonlinear system with the following discrete time equation:

$$y(k) = \frac{A_1 y(k-1) + A_2 y(k-2) + B_1 u(k-1) + B_2 u(k-2)}{1 + y^2(k-2)}, \tag{11}$$

where $A_1 = 1.752821$, $A_2 = -0.818731$, $B_1 = 0.011698$, and $B_2 = 0.010942$. The random training input sequence $u(k)$, k=0,1,...99 had values of $|u(k)| \leq (a^2 + w^2)/w$ where $a = 1$ and $w = 2\pi/2.5$ [3]. The rms error values obtained for 6 different initial solutions by using the BP, SA, and TS algorithms are presented in Figure

11. The responses of the system and the network designed by the SA algorithm are presented in Figure 12. The average rms error values and the improvement percentages for this system, which were obtained by using the 3 algorithms, are presented in Table 3.

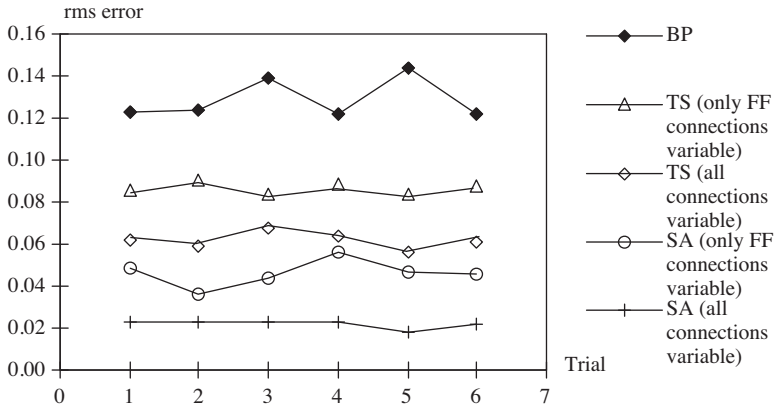

Figure 11. The rms error values obtained for System 3 for 6 different runs.

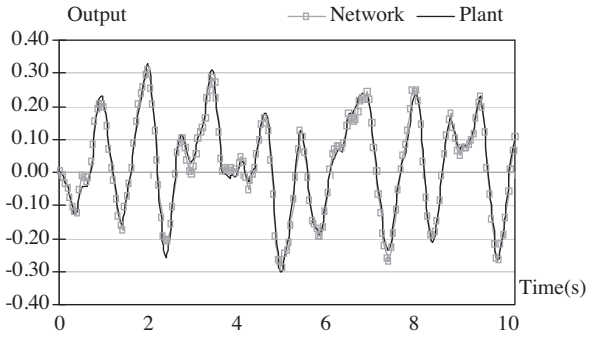

Figure 12. Responses of the system and the network trained by the SA algorithm (rms error = 0.017705).

Table 3. Comparison of the results for System 3.

Model	Average rms error	Improvement (%)
BP	1.2863E-01	-
TS ($\alpha = 1$)	8.6820E-02	32.50
TS (all weights trainable)	6.1720E-02	52.01
SA ($\alpha = 1$)	4.6370E-02	63.95
SA (all weights trainable)	2.1760E-02	83.08

Systems 4-9. In addition to the systems given above, a number of computer simulations were carried out to test the proposed identification approach. A total of 6 systems, given in Table 4, were used for testing. Systems 4 and 5, Systems 6 and 7, and Systems 8 and 9 represent first-, second-, and third-order systems, respectively. The training and test data files with 200 data points were obtained by applying uniform sequences $u(k) \in [-1.0,1.0]$, $k = 1,...,199,200$ [5]. The average rms error values and the improvement percentages obtained for 6 different initial solutions by using the BP and SA algorithms are presented in Table 5.

In this work, the parameters of the SA algorithm are as follows: the number of temperature points is 15, the number of points with constant temperature is 15, and the temperature decrease factor is 0.9. These parameters were determined after many trials (at least 20 trials).

Table 4. Linear systems used for the proposed identification scheme.

System	Orders	Discrete-time domain representation of the systems
4	First-order	$y(k) = 0.5y(k-1) + 0.1u(k-1)$
5		$y(k) = 0.4y(k-1) + 0.3u(k-1)$
6	Second-order	$y(k) = 1.1953y(k-1) - 0.4317y(k-2) + 0.1348u(k-1) + 0.1017u(k-2)$
7		$y(k) = 1.8y(k-1) - 0.837y(k-2) + 0.019u(k-1) + 0.018u(k-2)$
8	Third-order	$y(k) = 2.038y(k-1) - 1.366y(k-2) + 0.301y(k-3) + 0.0059u(k-1) - 0.018u(k-2) + 0.0033u(k-3)$
9		$y(k) = 2.0549y(k-1) - 1.3524y(k-2) + 0.2894y(k-3) + 0.0049u(k-1) + 0.0032u(k-2)$

Table 5. Comparison of the results for Systems 4-9.

System	Model	Average rms error	Improvement (%)
System 4	BP	5.4280E-03	-
	SA ($\alpha = 1$)	2.2908E-04	95.78
	SA (all weights trainable)	5.3355E-05	99.02
System 5	BP	7.0190E-03	-
	SA ($\alpha = 1$)	5.0187E-04	92.85
	SA (all weights trainable)	3.2348E-04	95.39
System 6	BP	2.8961E-02	-
	SA ($\alpha = 1$)	2.3984E-03	91.72
	SA (all weights trainable)	8.0753E-04	97.21
System 7	BP	3.2905E-02	-
	SA ($\alpha = 1$)	3.7838E-03	88.50
	SA (all weights trainable)	2.2652E-03	93.12
System 8	BP	2.8389E-02	-
	SA ($\alpha = 1$)	1.5146E-03	94.67
	SA (all weights trainable)	9.7394E-04	96.57
System 9	BP	2.8919E-02	-
	SA ($\alpha = 1$)	1.1816E-03	95.92
	SA (all weights trainable)	6.6473E-04	97.70

The original Elman network could identify the third-order linear system successfully. Note that the original Elman network, which had an identical structure to that adopted for the original Elman network employed in this work and was trained using the basic BP algorithm, failed to identify even second-order linear systems [3]. Moreover, when the original Elman network was trained by the basic GA, the third-order system could not be identified, although the second-order system was identified successfully [8]. It can be clearly seen that for the 9 systems, the training was significantly more successful when all connection weights of the network were trainable than when only the feedforward connection weights could be changed. Training of the feedback connection weights was simple using the SA algorithm. It is clearly seen from Figures 7-12, Tables 1-3, and Table 5 that, for both of the network structures (with all connection weights variable and with only feedforward connection weights trainable), SA trained the networks better than the BP and TS algorithms. The performance of SA is more effective for training nonlinear systems, although the performance of the TS is similar for training linear systems.

5. Conclusion

This paper investigated the use of the SA algorithm to train the original Elman network for the identification of linear and nonlinear dynamic systems. The main conclusion is that the SA algorithm was successful and produced superior results compared to the BP and TS algorithms. As a result, the original Elman network can be trained successfully using a more efficient algorithm instead of using more complex network structures. The obtained results show that the SA algorithm can be used for this purpose.

It can be seen from the simulation results that the SA algorithm is quite versatile since it does not rely on problem constraints. Furthermore, the SA algorithm can be easily tuned for any given optimisation problem. Finally, it can also be concluded that the proposed approach is applicable for training other recurrent neural network models.

References

[1] D.T. Pham, D. Karaboga, Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks, London, Springer-Verlag, 2000.

[2] D.T. Pham, S.J. Oh, "A recurrent backpropagation neural network for dynamic system identification", Journal of Systems Engineering, Vol. 2, pp. 213-223, 1992.

[3] X. Liu, Modelling and Prediction Using Neural Networks, PhD Thesis, University of Wales College of Cardiff, Cardiff, UK, 1993.

[4] D.T. Pham, X. Liu, "Identification of linear and nonlinear dynamic systems using recurrent neural networks", Artificial Intelligence in Engineering, Vol. 8, pp. 67-75, 1993.

[5] A. Kalinli, S. Sagiroglu, "Elman network with embedded memory for system identification", Journal of Information Science and Engineering, Vol. 22, pp. 1555-1568, 1996.

[6] D. Karaboga, A. Kalinli, "Training recurrent neural networks using tabu search algorithm", 5th Turkish Symposium on Artificial Intelligence and Neural Networks, pp. 293-298, 1996.

[7] D.T. Pham, X. Liu, Neural Networks for Identification, Prediction and Control, 4th ed., London, Springer-Verlag, 1999.

[8] D.T. Pham, D. Karaboga, "Training Elman and Jordan networks for system identification using genetic algorithms", Artificial Intelligence in Engineering, Vol. 13, pp. 107-117, 1999.

[9] A. Thammano, P. Ruxpakawong, "Nonlinear dynamic system identification using recurrent neural network with multi-segment piecewise-linear connection weight", Memetic Computing, Vol. 2, pp. 273-282, 2010.

[10] R.S. Sexton, J.N.D. Gupta, "Comparative evaluation of genetic algorithm and backpropagation for training neural networks", Information Sciences, Vol. 129, pp. 45-59, 2000.

[11] P.A. Castillo, J.J. Merelo, A. Prieto, V. Rivas, G. Romero, "G-Prop: Global optimization of multilayer perceptrons using GAs", Neurocomputing, Vol. 35, pp. 149-163, 2000.

[12] J Arifovic, R. Gençay, "Using genetic algorithms to select architecture of a feedforward artificial neural network", Physica A, Vol. 289, pp. 574-594, 2001.

[13] K.W. Ku, M.W. Mak, W.C. Siu, "Adding learning to cellular genetic algorithms for training recurrent neural networks", IEEE Transactions on Neural Networks, Vol. 10, pp. 239-252, 1999.

[14] A. Blanco, M. Delgado, M.C. Pegalajar, "A genetic algorithm to obtain the optimal recurrent neural network", International Journal of Approximate Reasoning, Vol. 23, pp. 67-83, 2000.

[15] A. Blanco, M. Delgado, M.C. Pegalajar, "A real-coded genetic algorithm for training recurrent neural networks", Neural Networks, Vol. 14, pp. 93-105, 2001.

[16] R. Battiti, G. Tecchiolli, "Training neural nets with the reactive tabu search", IEEE Transactions on Neural Networks, Vol. 6, pp. 1185-1200, 1995.

[17] P.A. Castillo, J. Gonzalez Peñalver, J.J. Merelo, A. Prieto, V. Rivas, G. Romero, "SA-Prop: optimization of multilayer perceptron parameters using simulated annealing", Lecture Notes in Computer Science, Vol. 1606, pp. 661-670, 1999.

[18] A. Kalinli, "Training Elman network using simulated annealing algorithm", Journal of the Institute of Science and Technology of Erciyes University, Vol. 19, pp. 28-37, 2003 (in Turkish).

[19] A. Kalinli, D. Karaboga, "Training recurrent neural networks by using parallel tabu search algorithm based on crossover operation", Engineering Applications of Artificial Intelligence, Vol. 17, pp. 529-542, 2004.

[20] R.S. Sexton, B. Alidaee, R.E. Dorsey, J.D. Johnson, "Global optimization for artificial neural networks: a tabu search application", European Journal of Operational Research, Vol. 106, pp. 570-584, 1998.

[21] B. Dengiz, C. Alabas-Uslu, O. Dengiz, "A tabu search algorithm for the training of neural networks", Journal of the Operational Research Society, Vol. 60, pp. 282-291, 2009.

[22] H.W. Ge, Y.C. Liang, M. Marchese, "A modified particle swarm optimization-based dynamic recurrent neural network for identifying and controlling nonlinear systems", Computers & Structures, Vol. 85, pp. 1611-1622, 2007.

[23] J.L. Elman, "Finding structure in time", Cognitive Science, Vol. 14, pp. 179-211, 1990.

[24] S. Kirkpatrick, C.D. Gelatt Jr, M.P. Vecchi Jr, "Optimization by simulated annealing", Science, Vol. 220, pp. 671-680, 1983.

[25] F. Glover, "Future paths for integer programming and links to artificial intelligence", Computers and Operations Research, Vol. 5, pp. 533-549, 1986.

[26] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, "Optimization by simulated annealing: an experimental evaluation. Part I, graph partitioning", Operations Research, Vol. 37, pp. 865-892, 1989.

[27] K.C. Tan, Y. Li, D.J. Murray-Smith, K.C. Sharman, "System identification and linearization using genetic algorithms with simulated annealing", First IEE/IEEE International Conference on GA in Engineering Systems: Innovations and Applications, pp. 164-169, 1995.

[28] İ. Eksin, O.K. Erol, "A fuzzy identification method for nonlinear systems", Turkish Journal of Electrical Engineering and Computer Sciences, Vol. 8, pp. 125-135, 2000.