Paper

Two-Phase Pattern Search-based Learning Method for Multi-layer Neural Network

Xugang Wang Non-member
Zheng Tang Non-member
Hiroki Tamura Member
Masahiro Ishii Non-member

A new multi-layer artificial neural network learning algorithm based on pattern search method is proposed. The learning model has two phases-a pattern search phase, and a local minimum—escaping phase. In the pattern search phase, our method performs local search iteratively and minimize the error measure function along with the set of descent directions of the error measure directly and finds the nearest minima efficiently. When the network gets stuck in local minima, the local minimum-escaping phase attempts to fill up the valley by modifying temperature parameters in ascent direction of the error measure. Thus, the two phases are repeated until the network gets out of local minima. The learning model is designed to provide a very simple and effective means of searching the minima of objective function directly without any knowledge of its derivatives. We test this algorithm on benchmark problems, such as *exclusive-or* (XOR), parity, Arabic numerals recognition, function approximation problems and a real world classification task. For all problems, the systems are shown be trained efficiently by our method. As a simple direct search method, it can be applied in hardware implementations easily.

Keywords: multi-layer neural networks, pattern search method, local minimum, Backpropagation, learning

1. Introduction

It is well known that the training of multi-layer feed-forward neural networks [1] can be viewed as the optimization of a criterion function from a set of input-output pairs with respect to a set of parameters-the weights and thresholds [2]. In other words, it is a kind of multidimensional minimization of the error measure function. Minimizing the multi-layer neural network error measure function in realistic problems is a difficult task since many layers, the multitude of training patterns and the variety of categories cast a very complex landscape with wide plateaus and narrow valleys [3].

Since the solution to this problem is NP (Nondeterministic Polynomial)-complete, no direct algorithm is possible [4] [5]. Therefore, gradient descent approaches, such as the Backpropagation model are most widely used and effective algorithms for training feed-forward neural networks [1] [6]. The Backpropagation algorithm iteratively adjusts the network parameters (all weights and thresholds) to minimize the error measure function using a gradient descent technique. Generally speaking, methods that use derivatives are efficient. However some problems correspond to the error measure functions that by nature are non-differentiable or difficult to compute. Furthermore, they are usually difficult for hardware implementations for they need analog multipliers and other analog computations. Hence these methods will not work in hardware manner [7]. In addition, due to the highly nonlinear modeling power of such networks, the learned function may oscillate abruptly between various training data. This is clearly undesirable for function approximation [8].

Pattern search methods are a class of direct search methods for nonlinear optimization proposed by Hooke and Jeeves [9]. Since the introduction of the original pattern search methods, they have remained popular with users due to their simplicity and the fact that they work well in practice on a variety of problems [10]. Although the pattern search method has been an established approach in the field of optimization, it has never been employed in MLP (Multi-Layer Perceptron) training before. Based on the pattern search method, we proposed a learning method for multi-layer artificial neural networks [11]. The pattern search based training procedure could render the procedure efficient and robust and provided a very simple and effective means of searching the minima of objective function directly without any oscillation between training data. However, as a local search method, the pattern search method may often converge to a local minimum. In this paper, in order to avoid the local minimum problem, we propose an efficient means of helping the network escape from local minima by modifying temperature parameters to make the error measure ascend in the temperature parameter space after the pattern search descends into a minimum that achieves insufficient accuracy in weight space. We

* Faculty of Engineering, Toyama University, Toyama-shi, 930–8555 Japan.

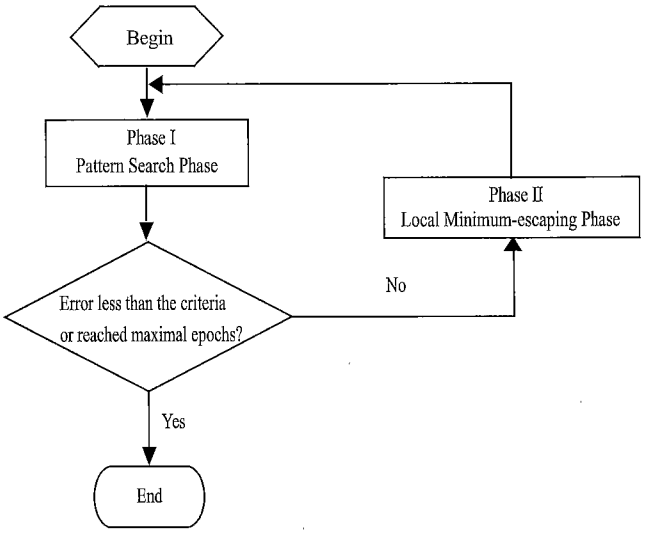


Fig. 1. Flowchart of the proposed algorithm.

apply the algorithm to a diverse set of problems, including *exclusive-or* (XOR), parity problems, Arabic numerals recognition, function approximation problems and classification of radar returns from the ionosphere task. The simulation results show that our method can train multi-layer neural networks effectively and improve the pattern search method much.

2. Learning Algorithm

Figure 1 is a flowchart of the proposed learning algorithm. We can see from the flowchart that our algorithm consists of two phases. Phase I is the pattern search phase in weight space and Phase II is the local minimum-escaping phase in the temperature parameter space.

2.1 Pattern Search Phase A multi-layer feed-forward artificial neural network [1] usually has one output layer and one input layer with one or more hidden layers. Each layer has a set of units, nodes or neurons. Each neuron has a threshold. It is usually assumed that each layer is fully connected with an adjacent layer without direct connections between layers that are not consecutive. Each connection has a weight.

The input of each unit in a layer (except input layer) is given by

$$net_{pj} = \sum_i w_{ji} o_{pi} + \theta_j \quad \cdots\cdots (1)$$

where net_{pj} is the net input to unit j produced by the presentation of pattern p, w_{ji} is is the weight from unit i to unit j, θ_j is a threshold of the unit j and $o_j$is the output value of unit i for pattern p. The output of unit j for pattern p is specified by

$$o_{pj} = f_j(net_{pj}) \quad \cdots\cdots (2)$$

where $f(x)$ is a semilinear activation function that is differentiable and nondecreasing.

The Backpropagation algorithm [1] tries to find a set of weights and thresholds that minimizes an overall error measure E,

$$E = \sum_p E_p(W, \Theta) \quad \cdots\cdots (3)$$

where p indexes over all the patterns in the training set. E_p is defined by

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2 \quad \cdots\cdots (4)$$

where t_{pj} is target value (desired output) of the j-th component of the output pattern for pattern p and o_{pj} is the actual output of the j-th unit produced by the presentation of input pattern p, and j indexes over the output units.

The Backpropagtion algorithm iteratively adjusts the network parameters (weights and thresholds) to minimize the error measure function (Eq.4) using a gradient descent technique. Therefore it needs to compute the differentials of the error measure, activation function and their analog multiplications.

Since the objective of training is to adjust all parameters so as to minimize the sum of squared residuals, we consider this unconstrained optimization problem of minimizing a continuously differentiable function $E : R^n \longrightarrow R$, without any knowledge of its derivatives, and without any means of approximating them.

So for the multi-layer neural networks, we can define a vector X whose elements include all adaptable parameters (weights between all layers and thresholds of every neuron):

$$X = [w_{11}, w_{12}, \ldots, w_{ij} \ldots, \theta_1, \theta_2, \ldots, \theta_i \ldots]^T \quad \cdots\cdots (5)$$

Then, the error measure E can be written as

$$E = E(X) \quad \cdots\cdots (6)$$

Then we can iteratively adjust X to minimize the function $E(X)$.

First, the search starts at an initial point X_0 and moves along n directions. n denotes the number of elements of vector X. Then, we can define the h-th direction vector at iteration k:

$$e_k^h = [\underset{1}{0}, \ldots, 0, \underset{h}{1}, 0, \ldots, \underset{n}{0}]^T \quad \cdots\cdots (7)$$

The sequence of iterations $X_0, X_1, \ldots, X_k \ldots$in R^n is produced as follows: for $k \geq 0$, iteration k is initiated with $X_k \in R^n$, and aims to find a new point along the direction e_k^h, $(h = 1, \ldots, n)$, such that $E(X_k + \triangle_k e_k^h) < E(X_k)$ or $E(X_k - \triangle_k e_k^h) < E(X_k)$, $(h = 1, \ldots, n)$, where Δ_k is the step size at iteration k.

If such a point is found, then the iteration is declared successful, and the next iterate is $X_{k+1} = X_k + \triangle_k e_k^h$ or $X_{k+1} = X_k - \triangle_k e_k^h$, $(h = 1, \ldots, n)$. If no such point is found, then the iteration is declared unsuccessful, and the next iteration is initiated at the same point$X_{k+1} = X_k$. The step size parameter $\triangle_{k+1}$ is reduced to $b\triangle_k$, where $0 < b < 1$ is a constant over all iterations. The initial vector X_0 and the step size $\triangle_0$ are given.

The detailed learning algorithm can be described as following rules:

Step 1. Initialize the vector X (weights and thresholds) and other parameters:
Generate a vector X_0 (the weight and threshold vector) randomly and set temperature parameters to 1.0.
Set $\delta_{L1}, \delta_{L2}, \ldots, \delta_{Ln}, \delta_{R1}, \delta_{R2}, \ldots, \delta_{Rn} = \triangle_0 > 0$, where δ is the move step associated with each weight and threshold.
Set constant $a \geq 1$, constant $b \in (0, 1)$, and small constant $\varepsilon > 0$.
Let $Y_1 = X_0, k = 0, h = 1$.

Step 2. Search a new vector with lower error function:
For h=1 to n (that is for all weights and thresholds), carry out:
If $E(Y_h + \delta_{Rh} e_h) < E(Y_h)$, then
$Y_{h+1} = Y_h + \delta_{Rh} e_h$,
else {
$\delta_{Rh} = b\delta_{Rh}$,
if $E(Y_h - \delta_{Lh} e_h) < E(Y_h)$, then
$Y_{h+1} = Y_h - \delta_{Lh} e_h$,
else $Y_{h+1} = Y_h$, $\delta_{Lh} = b\delta_{Lh}$.
}

Step 3. Adapt the vector X (weights and thresholds):
If $E(Y_{n+1}) < E(X_k)$, then {
set the new vector $X_{k+1} = Y_{n+1}$,
$Y_1 = X_{k+1} + a(X_{k+1} - X_k)$,
}
else{
set the new vector $X_{k+1} = X_k$,
$Y_1 = X_k$,
}
$k = k + 1, h = 1$.

Step 4. Estimate stop conditions of pattern search:
If $E(X_{k+1}) < \xi$, then {
stop the pattern search at point X_{k+1} (the training is considered to be successful.)
}
else if training reaches the upper limit epochs, then
stop search at point X_k
(the training is considered to be unsuccessful);
else if all δ (that is $\delta_{L1}, \delta_{L2}, \ldots, \delta_{Ln}, \delta_{R1}, \delta_{R2}, \ldots, \delta_{Rn}$) $< \varepsilon$ then
go to the local minimum-escaping phase
(the training is considered to get trapped in a local minimum).
else go to Step 2.
}

where ξ is a preselect error criterion; $\varepsilon > 0$ is a preselect small constant, and one epoch corresponds to one modification of all weights and thresholds.

In this phase, our method performs local search iteratively and minimize the error measure function along with the set of descent directions directly. Each adaptive parameter has its own learning rate. If it finds a new minimum, then it moves to the new point with lower error. If it fails to produce a decrease, the search step is reduced by a certain proportion. This search continues until the search step δ gets sufficiently small, thus ensuring convergence to a minimum [12] [13], whether local or global. In our method, if all search steps $\delta_{L1}, \delta_{L2}, \ldots, \delta_{Ln}, \delta_{R1}, \delta_{R2}, \ldots, \delta_{Rn} < \varepsilon$, where ε is a small constant, for example, 0.01, and the resulting error measure is larger than the error criteria, the training is considered to get trapped in a local minimum. Weights and thresholds cannot be updated and training is stopped. To help it escape from the local minimum and converge to a global minimum, the method will start the local minimum-escaping phase.

2.2 Local Minimum-escaping Phase Usually, the activation function of unit j is given by a sigmoid function $f_j(x)$ with the "temperature" parameter T_j

$$f_j(x) = \frac{1}{1 + e^{-\frac{x}{T_j}}} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (8)$$

where T_j is the temperature parameter of neuron j. The "temperature" parameter T_j is usually set to 1.0 and not changed by the learning rule. However, there is evidence suggesting that the nervous system has mechanisms that modulate the neural response function in a manner similar to "temperature" parameter or "gain" (the reciprocal of "temperature" parameter). Servan-Schreiber, Printz and Cohen [14] proposed that a "temperature" parameter can help explain the effect of biogenic monoamines, a type of neurotransmitter associated with neural responsiveness modulation. They used this parameter in Backpropagation networks to simulate phenomena associated with catecholamine manipulations. There is also evidence that these animes play an important role in the modulation of learning [15].

In our method, the error measure is also considered as a function of the "temperature" parameters of neurons. The "temperature" parameter T_j of each node is updated in a manner analogous to the way weights and thresholds using to make the error measure increase. Therefore we can increase intentionally the error measure by modifying the temperature parameters in the ascent directions of the error measure in the temperature parameter space and drive the network out of the local minimum.

Here suppose that a vector $V = [T_1, \ldots, T_j \ldots, T_m]^T$ whose elements correspond to the temperature parameters of neurons, where m denotes the number of elements of vector V. Similarly, the search starts at an initial point and moves along m directions. Then the j-th search direction is:

$$e_j = [\underset{1}{0}, \ldots, \underset{j}{0}, 1, \underset{m}{0}, \ldots, 0]^T \cdots\cdots\cdots\cdots\cdots\cdots (9)$$

Since for the temperature vector V, the modification requires the change of the temperature to be in the direction opposite to the weights and thresholds, we take following rules:

Step 1. Initialize the vector V (temperature parameters) and other Parameters:
Set base vector V_0 to current values of temperature parameters.
Set change step $\Delta T = \Delta T_0 > 0$.
Set constant $\beta > 1.0$, and constant $\gamma > \triangle T_0$.
Set $U_1 = V_0$, $k = 0$, $j = 1$.

Step 2. Search new temperature vector with larger error function:
For j=1 to m, do:
If $E(U_j + \Delta Te_j) > E(U_j)$, then
$U_{j+1} = U_j + \Delta Te_j$,
else {
if $E(U_j - \Delta Te_j) > E(U_j)$, then
$U_{j+1} = U_j - \Delta Te_j$,
else $U_{j+1} = U_j$.
}

Step 3. Adapt the vector V (temperature parameters):
If $E(U_{m+1}) > E(V_k)$, then {
set the new vector $V_{k+1} = U_{m+1}$,
$U_1 = V_{k+1}$,
}
else{
set the new vector $V_{k+1} = V_k$,
increase the change step $\Delta T = \beta \Delta T$,
$U_1 = V_k$.
}
$k = k + 1$, $j = 1$.

Step 4. Estimate Stop Conditions of Local Minimum-Escaping:
If $\Delta T > \gamma$, then {
set $\delta_{L1}, \delta_{L2}, \ldots, \delta_{Ln}, \delta_{R1}, \delta_{R2}, \ldots, \delta_{Rn} = \Delta_0$,
return to pattern search phase with new temperature parameters and start pattern search with current X vector.
}
else go to Step 2.

In the local minimum-escaping phase, we iteratively adjust the temperature parameters to increase the error measure using a simple direct search technique to help the network escape from local minima.

In order to explain simply how the proposed algorithm helps a network escape from a local minimum, we use a two-dimensional graph of an error measure of a neural network with a local minimum and a global minimum, as shown in Fig.2 (a). The vertical axis represents the error produced by the network corresponding to some states in weight space on the horizontal axis (to facilitate understanding, it is expressed in one dimension). The initial network defines a point (e.g., point A) on the slope of a specific "valley", the bottom of this valley (point B) is found by the pattern search based error minimization in the weight space (Fig.2 (b))—Phase I. Then the error measure is evaluated. If the error measure is less than an error criterion, then stop. If the network has larger error values than the error criteria, go to Phase II. Phase II tends to increase the error measure in the temperature space. After the local minimum-escaping phase (Phase II), the error measure at point B is raised, and then point B becomes a point at the slope of the "valley" in weight landscape again (Fig.2 (c)). Then the pattern search phase seeks a new minimum (Point C) in weight landscape (Fig.2 (d)).

Thus, the repeats of the Phase I in weight space and the Phase II in the temperature space may result in a movement out of a local minimum as shown in Fig. 2 (e)(f).

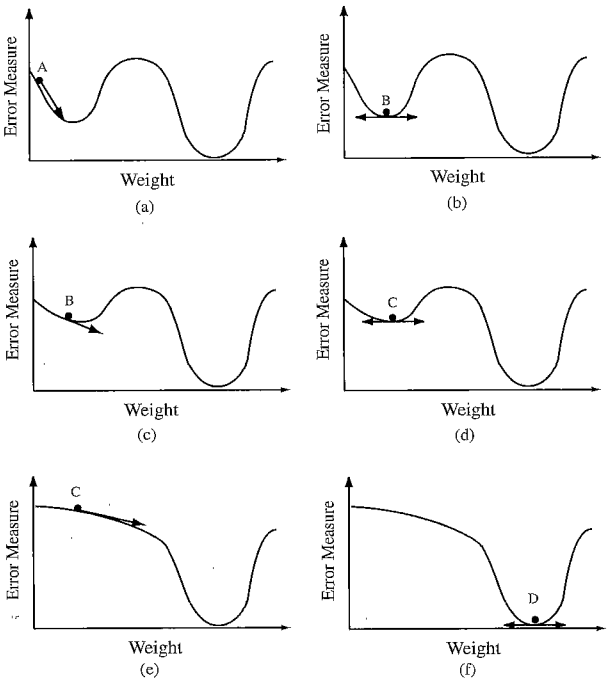


Fig. 2. The concept figure of the relationship between the error measure and the weight space.

As can be seen, our method performs local search iteratively and minimizes the error measure function along with the set of descent directions directly. With the local searching, it can find the nearest minima efficiently. Once the network becomes trapped in a local minimum, the local minimum-escaping phase can help the network escape from this "valley", and converge to the global minimum.

As we all know, the local search methods often suffer from the local minima problem. Since they try to iteratively update the weights for decreasing the error function, they cannot escape from the local minima by themselves. Numerous approaches [4] [16] have been proposed, such as the simulated annealing method, modify the learning model in order to help the network escape from local minima. The basic idea of these methods is adding a random factor to the model. Therefore, occasional increase of error function is allowed during the training process. However, the random perturbations of the search direction and 'various kinds of stochastic adjustment to the current set of weights' are not effective at enabling network to escape from local minima and they may take so much time to make the network fail to converge to a global minimum within a reasonable number of iterations [17]. To help the network escape from the local minima, our proposed method also increases the error function intentionally by a deterministic method. If error function is increased enough, the local minimum valley will be filled and leveled up. The network is expected to escape from the local minima and converge to the global minimum eventually.

The analysis of convergence of the pattern search method has been given by R.M.Lewis *et al* [10]. The

convergence is guaranteed. Both phases of our proposed method are based on the pattern search method. Additionally, since the temperature parameter update causes only the changing of slope of activation function, the convergence in weight space of our method can be guaranteed.

Moreover, neither explicit estimate of the derivative nor anything like a Taylor's series appears in our proposed method. This makes algorithm useful in situations where derivatives are not available or difficult to get. Furthermore, the learning is performed by simply changing vector by a small positive or negative constant, and accepting the change if it produces a desired result. Therefore, the algorithm is extremely simple to be specified and be implemented in hardware applications.

3. Simulations

In order to demonstrate the effectiveness of the proposed learning algorithm, a large number of simulations were performed for experimental purposes. We compared the results to the original pattern search method without local minimum-escaping phase and those of the typical gradient descent method - the Backpropagation algorithm and the global optimization method - the simulated annealing method, because they are the most popular and effective learning methods for neural networks [1] [4]. In our simulations, the weight update rule typically used in the Backpropagation algorithm is given by

$$\Delta_p w_{ji}(n+1) = -\eta \frac{\partial E_p}{\partial w_{ji}} + \alpha \Delta_p w_{ji}(n) \cdots\cdots (10)$$

where η is called the learning rate, and α is called the momentum term parameter that determines the effect of past weight changes on the current weight changes. Both two parameters were chosen properly based on many trials and differently according to different complexities of problems. The process of the simulated annealing method was based on Ref.(4) and the same parameters as in Ref.(4) were used in our simulations.

Since the proposed method and the pattern search method have a number of parameters in common (such as the constants a, b and initial change step Δ_0), these parameters were all set to the identical value for all examples. Since neural network training is a rather complex task, the increase constant a and decrease constant b were all set to be around 1.0 to avoid any violent oscillation during training. For our proposed method and the original pattern search method, the constants a, b and Δ_0 were set to 1.0, 0.99 and 0.1, respectively. Small additional parameters were introduced in our proposed method. To make our proposed method be a general training method, these parameters of our method should not be affected by individual problem too much. Therefore, the same values of these parameters were used for all examples, although there may be more suitable values for each problem. Since it is difficult to set the values theoretically, we have chosen these parameters by trials as following: small constant $\varepsilon = 0.01$ and the initial temperature parameters were all set to 1.0; for local minimum-escaping phase, $\Delta T_0 = 0.1$, $\beta = 1.5$ and $\gamma = 1.0$. As will be shown in following simulation results that these parameters were well suited for the different types of problems.

3.1 *Exclusive-or* (XOR) Problem It is useful to begin with the *exclusive-or* problem since it is the classic problem requiring hidden units and since many other difficult problems involve an *exclusive-or* as a subproblem [1]. A simple architecture (2-2-1 network) with one hidden layer containing two hidden neurons and no direct connections from input to output was used to learn the *exclusive-or* problem.

To investigate the training processing of a neural network by our method, a typical learning curve of *exclusive-or* problem obtained from our method is shown in Fig.3, in which one epoch corresponds to one modification to all parameters (all weights and thresholds). Initially the weight vector was set randomly from -1 to 1 (time=t_0). After the first pattern search learning for 72 epochs (time=t_1), the error measure E decreased from 0.517 to 0.252 and got stuck in a local minimum. Then the local minimum-escaping learning (Phase II) was performed and the error measure was increased from 0.252 to 0.263 (time=t_2). Then the network escaped from the local minimum and converged to the global minimum (time=t_3) after the second pattern search. To explain how the network escaped from local minima more clearly, it is useful to plot a scatter map of 2-class discriminant function obtained by the network training. Fig. 4 shows the simulation results that illustrate a typical progressive intermediate decision region during the pattern search phase and the local minimum-escaping phase. The two axes correspond to the two inputs to the network. Samples from class 1 are represented by black dots and samples from class 0 by white dots. As can been seen, the correct decision regions for the *exclusive-or* problem were formed from the initial one (Fig. 4 (a) time=t_0), to a local minimum convergence (Fig. 4 (b) time=t_1) and finally the global convergence (Fig. 4 (d) time=t_3) after the local minimum-escaping learning at point of time t_2 (Fig. 4(c)).

As we all know that the initial values of weights and thresholds affect the convergence of a learning algorithm strongly. Therefore, to see the sensitivity of our

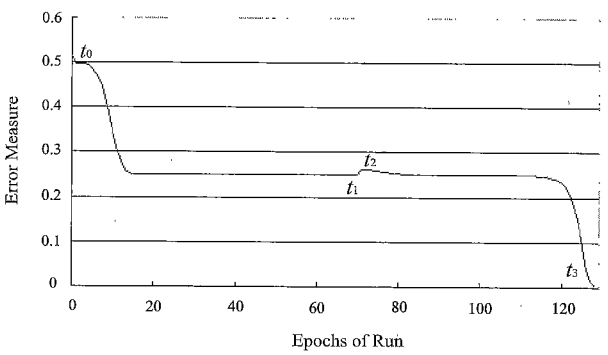


Fig. 3. The typical learning curves of our proposed method for *exclusive-or* problem.

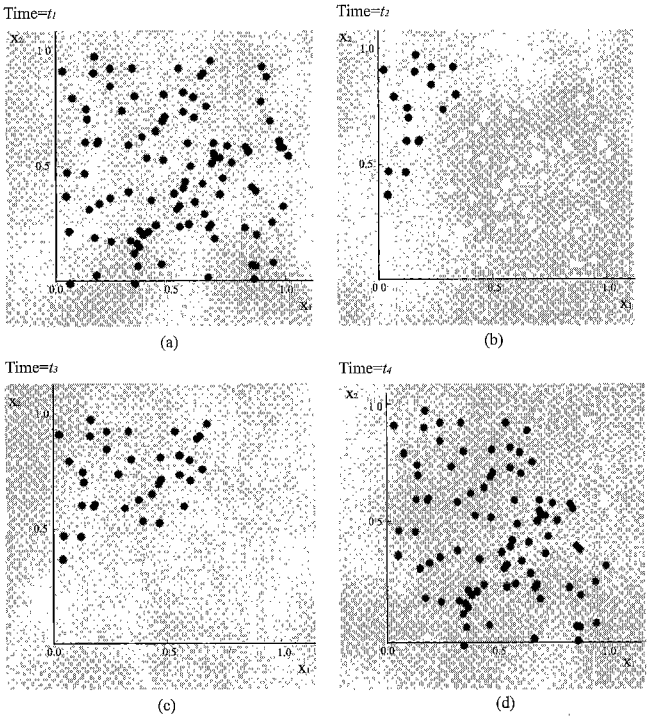


Fig. 4. A typical progressive intermediate decision region during the pattern search phase and the local minimum-escaping phase. Samples from class 1 are represented by black dots and samples from class 0 by white dots.

method to initial settings, we generated initial weight and threshold vectors randomly for the exclusive-or problem in (-1,1), (-2,2),..., (-10,10)—10 ranges respectively. Every range includes 100 different weight vectors. We performed the learning with the Backpropagation algorithm, and our method, using these weights and thresholds respectively. For the Backpropagation algorithm, η was set to 1.0, and α was set to 0.8. To give enough time for training, in all cases, training was allowed for up to 5000 epochs. For all methods, one epoch corresponds to one modification to all weights and thresholds. The error criteria used for all tests was 0.01. The success rates with different initial setting gained by the Backpropagation algorithm, the original pattern search method and our method are showed in Fig. 5. The comparison results revealed that as the range of initial vectors grew larger, the success rate of the Backpropagation algorithm became worse and decreased drastically. However, the effect on our method was not so severe—the success rates of our method does not change significantly and keeps higher than the original pattern search method. So it is clear that our method is more robust than the Backpropagation algorithm and the local minimum-escaping phase in our method can greatly improve the success rate in convergence to the global minimum.

3.2 Parity Problem Another one of the most popular tasks given a good deal of discussion is parity problem, in which the output required 1 if the input pattern contains an odd number of 1s and 0 otherwise. The exclusive-or problem is a parity problem with input

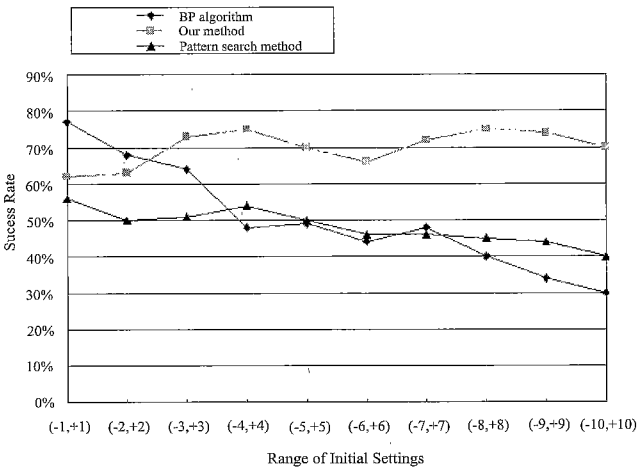


Fig. 5. Comparison of networks with various initial settings.

patterns of size two [1]. The parity problem is a very demanding classification task for neural networks to solve, because the target-output changes whenever a single bit in the input vector changes [18] and N-parity training set consists of 2^N training pairs. We have tried a number of parity problems with input patterns ranging from size four to seven for experimental purposes. An N-M-1 (N-input, M-hidden neurons and 1 output) architecture was used for the N-bit parity problem. The numbers of hidden units were selected from [19].

Since the initial values of weights affect the convergence of a learning algorithm, it is reasonable to judge each algorithm by the statistics obtained from multiple runs. In our experiments, for each simulation of the three algorithms on 4-bit, 5-bit, 6-bit and 7-bit parity problems, 100 sets of different initial weights were randomly generated from -1 to 1 to be used for training. The Backpropagation algorithm, the simulated annealing method, the original pattern search method and our method were used to train a network on a given problem, with all networks having identical starting weight values. For the Backpropagation algorithm, η was set to 1.0, and α was set to 0.8. The error criteria used for all parity problems was set to 0.01. The maximal number of epochs is 30000 epochs.

Table 1 shows the simulation results of the three algorithms on these parity problems. The success rate of convergence and average CPU time entries of the table represent the rate and average CPU time of the convergence runs. Because the training procedures of these algorithms are different, it is difficult to give an identical definition of the iteration or training cycles. It is more reasonable to use the uniform criterion—CPU time instead of iteration number or the number of training cycles. The average CPU times are only the average time for successful runs and the times for unsuccessful convergence are not counted. This table shows that the proposed learning algorithm is more reliable for difficult problems than both the Backpropagation algorithm and the simulated annealing algorithm in terms of faster

Table 1. Simulation results for the parity problems.

N-bit	Network	Algorithm	Success Rate	Average CPU Time (Millisecond)
4-bit	4-6-1	Backpropagation algorithm	88%	7754.8
		SA method	97%	293965.9
		Pattern search method	62%	676.8
		Our method	84%	3440.7
5-bit	5-10-1	Backpropagation algorithm	74%	53704.2
		SA method	95%	2862956.3
		Pattern search method	83%	4909.9
		Our method	98%	14371.9
6-bit	6-12-1	Backpropagation algorithm	10%	27555.1
		SA method	100%	30105990.0
		Pattern search method	64%	14904.0
		Our method	99%	17217.0
7-bit	7-14-1	Backpropagation algorithm	3%	1058061.0
		SA method	0%	
		Pattern search method	42%	121779.2
		Our method	75%	1272801.6

learning and higher successful learning rates. Especially, our algorithm worked much better than the Backpropagation for larger problems. Although the simulated annealing method could do 100% at parity 6, it needed more epochs than 30000 to converge for parity 7. Then the slow convergence resulted in the low success rate.

Furthermore, the training time of our method is longer than that of the pattern search method because it has the local minimum-escaping phase in order to escape from a local minimum. But with our method the systems are shown to be capable of escaping from the pattern search local minima and getting much better global convergence.

3.3 Arabic Numerals Recognition Problem

Moreover, to show the effectiveness of our proposed algorithm for some high-dimensional and practical problems, we applied our algorithm to a larger network that is set up for a more artificial task—Arabic numerals recognition. This task is a classical pattern classification problem. Our simulation involved recognizing Arabic numerals 0-9 in an 8 by 8 pixel input field (Fig. 6). In order to solve this problem, a relatively complex architecture was employed. Figure. 7 shows the basic structure of the network we employed. Input layer consisted of 64 units that were conceptualized as two-dimensional patterns corresponding to 8 by 8 pixel numeral array. Hidden layer had 4 units, each of which was fully connected to the 64 inputs. All hidden units were then fully connected to all output units. The number of output units was set to 10. Therefore, each of the 10 output units corresponded to one of these characters. That's, 0000000001 → 0,0000000010 → 1, 0000000100 → 2,...,1000000000 → 9.

The error criteria used for all tests was 0.01. For the Backpropagation algorithm, η=1.0, and α=0.8 were used. All weights and thresholds were also initialized from (-1.0, +1.0). Results of these simulations are presented in Table 2. The statistics in this table are based on 50 trials of simulations. It can be seen from the table that our method significantly outperformed the Backpropagation algorithm and the simulated annealing method in both global optimization and convergence speed.

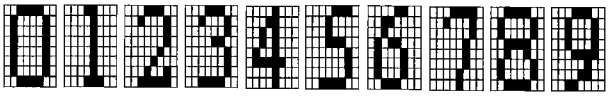


Fig. 6. Input patterns of Arabic numerals recognition.

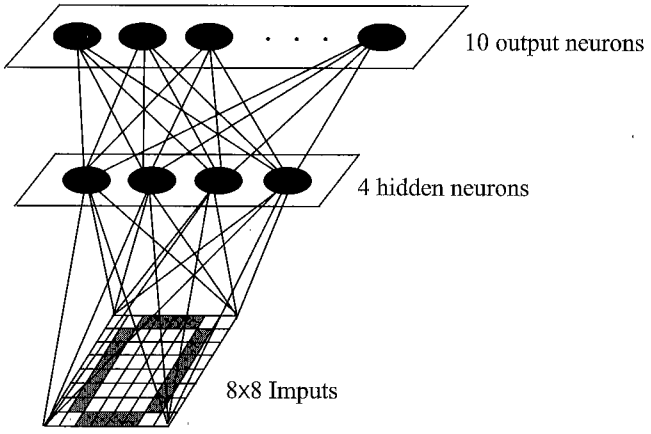


Fig. 7. The architecture of the network to solve the Arabic numerals recognition problem.

Table 2. Simulation results for the Arabic numerals recognition problem.

Algorithm	Success Rate	Average CPU Time (Millisecond)
Backpropagation algorithm	90%	28493.6
SA method	0%	
Pattern search method	96%	17078.0
Our method	98%	18884.7

3.4 Function Approximation Problem

To test the performance of the proposed method for some problems whose outputs take real values such as function approximation or time-series prediction, we applied our method to approximate a simple trigonometric function $y = \cos(x)$. Because output range of network is between 0 and 1, we changed the target function to $y = (cos(2x) + 1)/3$. The cosine function training set adopted here following Ref.(20)—requires network to approximate the cosine function for a sample of 64 input points chosen uniformly in the interval $[0, \pi]$. The training set is illustrated in Fig.8. The minimal "standard"

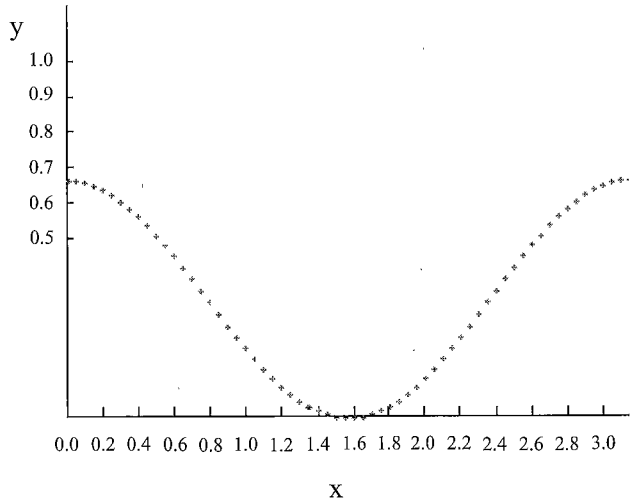


Fig. 8. The training set with 64 samples of function $y = (cos(2x) + 1)/3$.

architecture capable of learning the cosine(x) training set given in Fig.8 is 1-2-1 [20] [21].

For evaluating and investigating network training, a testing set including 128 samples was created. It is obtained by sampling from the input range $[0, \pi]$ at equal interval. When testing network's performance, the testing sample was fed to network; the corresponding network output can be compared with the desired output. In order to demonstrate the behavior of network trained by our method, we gave the results of approximating during a typical training process as shown in Fig. 9. In this figure, "target function" represents the original underlying function to be approximated from the training data provided and "approximated function" represents the actual outputs of network produced by the testing data. The network started with the pattern search phase at an initial state (Fig. 9 (a)) until network found a local minimum (Fig. 9(b)). It is obviously that the result of approximation was far from the target function at this state. After the local minimum-escaping phase (Fig. 9(c)), network converged to the global minimum (Fig. 9(d)) and achieved satisfied results.

We also generated statistic based on 100 trials in the same way and compared results with those of the Backpropagation algorithm and the simulated annealing method. All weights and thresholds were initialized from (-1.0, +1.0). The error criteria used for this problem was 0.01. The maximal number of learning epochs was set to 5000 epochs for the Backpropagation algorithm, the patter search method and our method, 30000 epochs for the simulated annealing method. For the Backpropagation algorithm η=0.5, α=0.8 was used. It can be seen from Table. 3, our method could gain more satisfied approximation performance than the original Backpropagation algorithm, the simulated annealing method and the pattern search method.

3.5 Ionosphere Data Finally we applied our method to a realistic "real-world" problem: classification of radar returns from the ionosphere. The data set was created by the Johns Hopkins University and obtained from [22]. We can refer the usage of this database from Sigillito, V. G. *et al* [23].

Table 3. Simulation results for the function approximation problem.

Algorithm	Success Rate	Average CPU Time (Millisecond)
Backpropagation algorithm	75%	68791.3
SA method	0%	
Pattern search method	97%	5101.6
Our method	100%	5945.7

Table 4. Simulation results for the ionosphere data classification problem.

Algorithm	Success Rate	Average CPU Time (Millisecond)
Backpropagation algorithm	44%	117110.0
SA method	0%	
Pattern search method	14%	241312.0
Our method	88%	321985.0

This radar data was collected by a system in Goose Bay, Labrador. This system consisted of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. There were 34 attributes used to represent the pattern. Each pattern belonged to two classes: "Good" or "Bad" radar returns that were free electrons in the ionosphere. "Good" radar returns that were represented by "0" in the experiment were those showing evidence of some type of structure in the ionosphere. "Bad" returns that were represented by "1" were those that do not.

The database consisted of 351 data patterns. We used first 200 instances for training, which were carefully split almost 50% "Good" and 50% "Bad". We used a 34-3-1 neural network model to solve this problem. The error criteria used for this problem was 0.1. η=0.1, α=0.8 were set for the Backpropagation algorithm. We also performed 100 trails for the backpropagation algorithm, the simulated annealing method, pattern search method and the proposed method. All weights and thresholds were also initialized from (-1.0, +1.0). The simulation results are shown in Table 4. The results demonstrated that the proposed method was superior to the all other three methods in network training. The results of the simulated annealing method were obtained by performing simulation on computers for 24 hours.

4. Conclusions

This paper presented a new simple learning method for multi-layer artificial neural networks based on the pattern search method. The method consisted of two phases: the pattern search phase and the local minimum-escaping phase. The former performed local search iteratively and minimize the error measure function along with the set of descent directions of the error measure directly and found the nearest minima efficiently. When the network got stuck in local minima, the local minimum-escaping phase attempted to fill up the valley by modifying temperature parameters in ascent direction of the error measure in order to help the network escape from local minima and gain higher

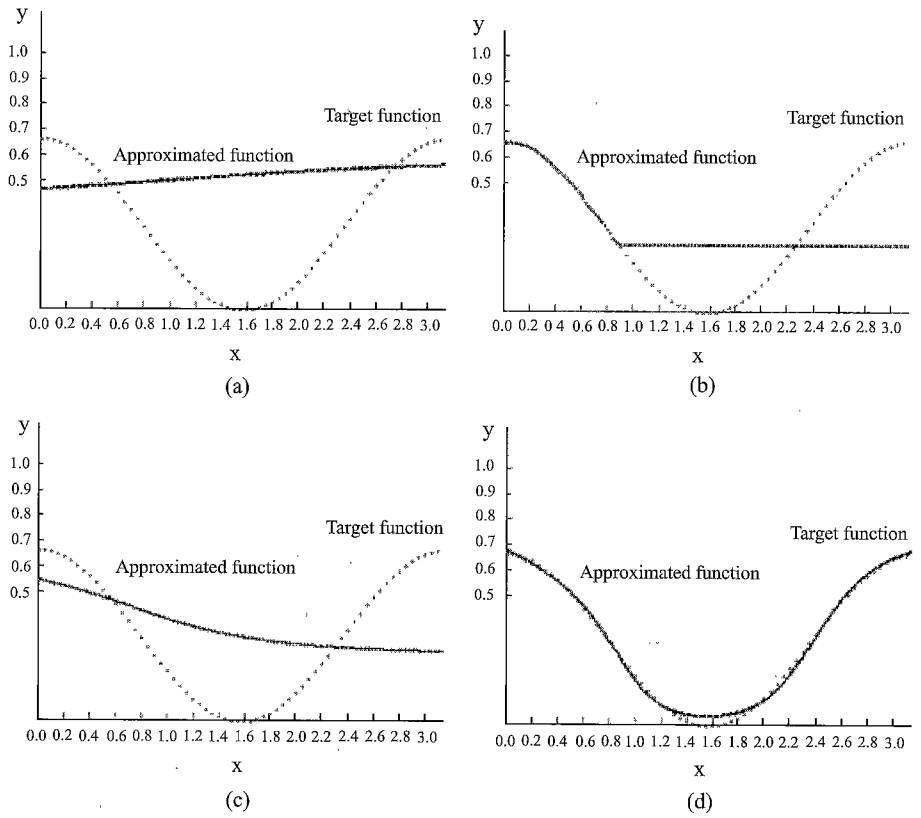


Fig. 9. Approximation results of function $y = (cos(2x) + 1)/3$ with our method. (a) Initial state of network. (b) The network got struck into a local minimum. (c) After the local minimum-escaping phase. (d) The network converged into the global minimum.

convergence to the global minimum. Because this approach was designed as a simple direct search method, it could be applied in hardware implementations easily. Finally experimental results from an implementation of the model on *exclusive-or*, parity, Arabic numerals learning, function approximation and the classification of radar returns from the ionosphere problems showed that our algorithm could train the network effectively and significantly outperformed the original pattern search method, Backpropagation algorithm, and the simulated annealing method.

(Manuscript received July 2, 2002,
revised Sep. 26, 2003)

References

(1) D.E. Rumelhart, G.E. Hinton, and R.J. Williams: "Learning Internal Representations by Back-propagating Errors", In Parallel Distributed Processing: Explorations in the Microstructure of Cognition, pp.318–362, MIT Press, Cambridge, MA (1986)

(2) L.F.A. Wessels and E. Barnad: "Avoiding false local minima by proper initialization of connections", *IEEE Trans. Neural Networks*, Vol.3, No.6, pp.899–905 (1992)

(3) S. Haykin: Neural Networks: A Comprehensive Foundation, Macmillan Publishing Company (1994)

(4) C.B. Owen and A.M. Abunawass: "Application of simulated annealing to the backpropagation model improves convergence", Proc. of the SPIE conference on the Science of Artificial Neural Networks II, Vol.1966, pp.269–276, Orlando, FL, USA (1993-8)

(5) J.F. Kolen: "Faster learning through a probabilistic approximation algorithm", Proc. of IEEE International Conference on Neural Networks, pp.335–341, San Diego, California (1988-7)

(6) S. Amari: "Theory of adaptive pattern classifiers", *IEEE Trans. Electronic Computer*, Vol.16, No.3, pp.299–307 (1967)

(7) A. Likas, D.A. Karras, and I.E. Lagaris: "Neural network training and simulation using a multidimensional optimization system", *Int. J. of Computer Mathematics*, Vol.67, pp.33–46 (1998)

(8) D.S. Chen and R.C. Jain: "A robust back propagation learning algorithm for function approximation", *IEEE Trans. Neural Networks*, Vol.5, No.3, pp.467–479 (1994)

(9) R. Hooke and T.A. Jeeves: "Direct search: Solution of numerical and statistical problems", *Journal of the Association of Computing Machinery*, Mach. Vol.8, pp.212–229 (1961)

(10) R.M. Lewis, V. Torczon, and M.W. Trosset: "Why pattern search works", *OPTIMA*, Vol.59, pp.1–7 (1998)

(11) X.G. Wang, Z. Tang, H. Tamura, and M. Ishii: "Multilayer Network Learning Algorithm Based on Pattern Search Method", *IEICE Transaction on Fundamentals of Electronics, Commun and Computer Sciences*, Vol.E86-A, No.7, pp.1869–1875 (2003)

(12) M. Momma, K.P. Bennett: "A Pattern Search Method for Model Selection of Support Vector Regression", *Proc. SDM02*, pp.11–13 (2002-4)

(13) R.M. Lewis, V. Torczon, and M.W. Trosset: "Direct search methods: then and now", *Journal of Computational and Applied Mathematics*, Vol.124, pp.191–207 (2000)

(14) C. Servan-Schreiber, H. Printz, and J.D. Cohen: "A network model of neuromodulatory effects: Gain, signal-to-noise ratio, and behavior", *Science*, Vol.249, pp.892–895 (1990)

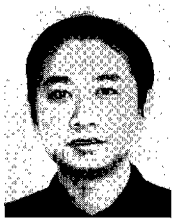
(15) T. Kamatsu and J.D. Pettigrew: "Depletion of brain catecholamines: Failure of ocular dominance shift after neurocular occlusion in kittens", *Science*, Vol.194, pp.206–208 (1976)

(16) A. Von Lehmen, E.G. Paek, P.F. Liao, A. Marrakchi, and J.S. Patel.: "Factors influencing learning by backpropagation," *Proc. IICNN*, Vol.I, pp.335–341, San Diego, California (1988)

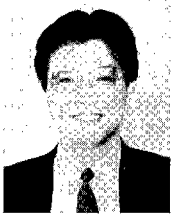
(17) C. Wang and J.C. Principe: "Training neural networks with additive noise in the desired signal," *IEEE Trans. Neural Net-*

works, Vol.10, No.6, pp.1511–1517 (1999)
(18) E. Hjelmås and P.W. Munro: "A comment on the Parity problem", *Report nr 7*, Gjøvik College (1999)
(19) M. Riedmiller: "Advanced supervised learning in multi-layer perceptrons - from backpropagation to adaptive learning algorithms", *Int. Journal of Computer Standards and Interfaces, Special Issue on Neural Networks*, Vol.16, pp.265–278 (1994)
(20) J.M. McInerney, K.G. Haines, S. Biafore, & R. Hecht-Nielsen: "Error surfaces of multi-layer networks can have local minima", Technical Report No.CS89-157, Department of Computer Science and Engineering, University of California (1989)
(21) A.J. Shepherd: Second-order methods for neural networks: fast and reliable training methods for multi-layer perceptrons, Springer, London, New York (1997)
(22) C.L. Blake and C.J. Merz: UCI Repository of machine learning databases [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science (1998)
(23) V.G. Sigillito , S.P. Wing, L.V. Hutton, and K.B. Baker: "Classification of radar returns from the ionosphere using neural networks", *Johns Hopkins APL Technical Digest*, Vol.10, pp.262–266 (1989)

Xugang Wang (Non-member) received his M.S. degree from Shandong University, Shandong, China in 2001. He is currently working for his Ph.D. degree at Toyama University, Japan. His main research interests are neural networks and optimizations.



Zheng Tang (Non-member) received his D.E. degree from Tsinghua University, Beijing, China in 1988. He is currently a professor in the Department of Intellectual Information Systems,Toyama University. His current research interests include intellectual information technology, neural networks, and optimizations. He is a member of IEICE and IEEE



Hiroki Tamura (Member) received his M.E degree from Miyazaki University in 2000. He currently works as a technical official in the Department of Intellectual Information Systems, Toyama University. His main research interests are neural networks and optimization. He is a member of IEEJ and JNNS

Masahiro Ishii (Non-member) obtained a doctorate degree from Tokyo Institute of Technology in 1995. He is now an associate professor of the Department of Intellectual Information Systems, Toyama University. He is a member of IEICE and ARVO