Evolutionary Learning-Rate Selection for BPNN with Window Control Scheme

Sung Hoon Jung
School of Information and Computer Engineering, Hansung Univ.
389 SamsunDong-2-Ga, SungBook-Gu, Seoul, 136-792, Korea
Tel: 02-760-4344
Email: shjung@ice.hansung.ac.kr

Abstract

The learning speed of the neural networks, the most important factor in applying to real problems, greatly depends on the learning rate of the networks. Three approaches—empirical, deterministic, and stochastic ones—have been proposed to date. We proposed a new learning-rate selection algorithm using an evolutionary programming search scheme. Even though the performance of our method showed better than those of the other methods, it was found that taking much time for selecting evolutionary learning rates made the performance of our method degrade. This was caused by using static intervals (called static windows) in order to update learning rates. Our algorithm with static windows updated the learning rates even though previously updated learning rates showed good performance or didn't update the learning rates even though previously updated learning rates showed bad performance. This paper introduce a window control scheme to avoid such problems. With the window control scheme, our algorithm try to update the learning rates only when the learning performance is continuously bad during a specified interval. If previously selected learning rates show good performance, new algorithm will not update the learning rates. This diminish the updating time of learning rates greatly. As a result, our algorithm with the window control scheme show better performance than that with static windows. In this paper, we will describe the previous and new algorithm and experimental results.

1 Introduction

Back-propagation neural networks (BPNNs), a kind of feed-forward artificial neural network, have been widely used in many application areas, such as associative mapping, classification, and system identification [1]. The most important factor in applying back-propagation neural networks to real problems is the learning speed of the networks. The learning speed greatly depends on the learning rate of the networks. Several approaches have been researched for improving the learning speed. These approaches can be classified into three methods—empirical [2, 3], deterministic [4, 5, 6], and stochastic [7, 8].

We proposed a new learning-rate selection algorithm using an evolutionary programming search scheme [9, 10]. Experimental results in [9, 10] showed that the learning speed of proposed algorithm was better than those of other methods [9, 10]. It was also found that the main factor degrading the performance of proposed algorithm was that selecting evolutionary learning rates was to take much time. This was because the selection of new learning rates occurred at static iteration intervals (the other word, static windows). The proposed algorithm with static windows updated the learning rates even though previously updated learning rates showed good performance or didn't update the

learning rates even though previously updated learning rates showed bad performance.

To solve this problem, this paper introduces a window control scheme. With the window control scheme, our algorithm try to update the learning rates only when the learning performance is continuously bad during a specified interval. This will reduce learning time by avoiding unnecessary update of learning rates.

Section 2 presents the existing learning-rate selection algorithms including evolutionary learning-rate selection algorithm. Proposed window control scheme is given in section 3. Experimental results with three applications are in section 4. Section 5 concludes this paper.

2 Learning-Rate Selection Algorithms

This section provides the previous algorithms and our evolutionary selection algorithm.

2.1 Previous Algorithms

A lot of learning-rate selection algorithms have been proposed to date. The methods can be divided into three techniques: empirical, deterministic, and stochastic methods. The empirical methods considered by [2] [3] are hard to apply to real applications because it is difficult or even impossible to pick out a proper leaning rate according to application domains.

Deterministic methods have also been introduced [4, 5, 6]. Nachtsheim derives the leaning rate by expanding the error function E, in Taylor's series about the current weight w_{ji} [5], so as to obtain the following relation

$$E(w_{ji}+\Delta w_{ji}) = E(w_{ji})+\Delta w_{ji}\cdot\partial E/\partial w_{ji}+\ldots. \quad (1)$$

The displacement Δw_{ji} is made so as to make $E(w_{ji}+\Delta w_{ji})$ zero within the second order derivation. This yields the learning rate

$$\eta = \frac{E}{\partial E/\partial w_{ji}\cdot\partial E/\partial w_{ji}} = \frac{E}{(\partial E/\partial w_{ji})^2}. \quad (2)$$

The weights of the Parlos method, a kind of steepest-descent technique, are iterated as follows [11]:

$$w_{ji}(t) = w_{ji}(t-1) - \rho(E)\frac{\partial E}{\partial w_{ji}}/\| \frac{\partial E}{\partial w_{ji}} \|^2, \quad (3)$$

where $\rho(E)$ is some function of the error E. There are several choices for the form of $\rho(E)$.

Weir determines the step length that is optimal in the sense of being the largest length avoiding the problem of overshooting the goal [4]. Given the current height (i.e., the error E) and the extreme gradient of error, the minimum distance of the goal weight from the current weight can be computed.

$$\begin{aligned}\eta &= \frac{height}{(extreme\ gradient)\cdot(actual\ gradient)} \\ &= \frac{E}{extreme(\partial E/\partial w_{ji})\cdot\partial E/\partial w_{ji}},\end{aligned} \quad (4)$$

where $extreme(\partial E/\partial w_{ji})$ is a positive upper bound or negative lower bound on the possible values of the error-weight gradient over the next weight change. There is the bound within the interval (0,1).

Hsin, et al. dynamically modified the learning rate, as a weighted average of direction cosines of successive incremental weight vectors at several steps [12]. The direction cosines of previous iterations contain information about the local error surface. Thus, the learning rate is set to be a weighted average of the $L+1$ direction cosine of successive incremental weight vectors for the purpose of providing further improvement in the training process. The modified learning rate is given by

$$\begin{aligned}\eta(k) &= \alpha_0\frac{\Delta w(k)\cdot\Delta w(k-1)}{\|\Delta w(k)\|\|\Delta w(k-1)\|}+\ldots \\ &\quad +\alpha_L\frac{\Delta w(k-L)\cdot\Delta w(k-L-1)}{\|\Delta w(k-L)\|\|\Delta w(k-L-1)\|}\end{aligned} \quad (5)$$

where $\alpha_0+\alpha_1+\ldots+\alpha_L = 1$ and $\alpha_0 \geq \alpha_1 \geq \ldots \geq \alpha_L$. It adaptively scales the descent step according to the local information of the error surface reflected in the direction cosines of the successive incremental weight vectors. As another approach, Eaton proposes a method of learning-rate selection using only the number of sample patterns [13]. The learning

rate is represented by 1.5 divided by the square root of the sum of the squares of the number of each input pattern type.

It is given as

$$\eta = \frac{1.5}{\sqrt{N_1^2 + N_2^2 + \ldots + N_m^2}}, \quad (6)$$

where N is the number of patterns and m is the number of pattern types.

The second order approaches require much memory and time. Let W be the vector of all weights for the connections of the units. The overall error E is a function of W. If we change W, say ΔW, the value of E will also change. The relation of the change ΔE to ΔW can be obtained simply by truncating to the second order terms the Taylor expansion, if ΔW is sufficiently small. In this case we have

$$\Delta E = G^t \Delta W + \frac{1}{2} \Delta W^t H \Delta W, \quad (7)$$

where G is the gradient of E in the space of the weights, and H is the corresponding Hessian matrix [14] [15] [16]. If the matrix H is definite positive and invertible, the optimal weight change ΔW is $\Delta W = -H^{-1}G$. The computation of the inverse of the Hessian matrix is excessively time consuming.

The training method using a genetic algorithm is a kind of stochastic approach because of the embedding stochastic process. Genetic algorithms have been used by Montana and Davis [7] to train back-propagation neural networks. Kitano [17] and Belew, et al. [18] developed hybrids in which genetic algorithms are used to search the weight space for the best parameters to be used in the back-propagation algorithm [17].

2.2 Evolutionary Selection Algorithm

The overall block diagram for the operation of the adaptive learning-rate selection algorithm is shown in Figure 1. In the figure, evolutionary programming iteratively applies three operations, i.e., selects parents, generates offspring, and assigns costs to the population, to each vector (learning rate) at every training step. The best learning rates selected from

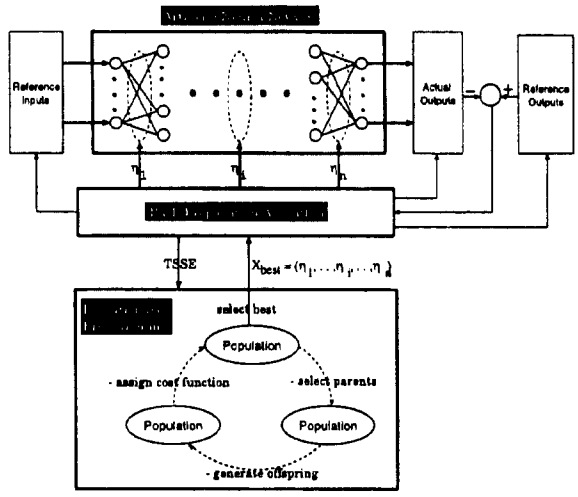

Figure 1: Overall Block Diagram for Learning-Rate Selection

evolutionary programming are offered to the back-propagation algorithm at every iteration step. The back-propagation network receives the best learning rate, updates its weights, and sends the newly generated TSSE to evolutionary programming. With this TSSE, evolutionary programming evaluates the fitness of the best learning rate using the objective function (equation 8) and generates offspring using the perturbation factor (equation 9). As these processes progress, the selected learning rates improve.

We define the objective function as the inverse value of the total sum squared error (TSSE) of the network. The objective function is as follows.

Definition 1: *Objective Function*

Let X_i be a solution (learning rate) of population P. Then an *objective function* is defined as:

$$F(X_i) = \frac{1}{TSSE} = \frac{1}{\sum_{p=1}^{R} \sum_{k=1}^{K} (t_{pk} - o_{pk}^{X_i})^2}, \quad (8)$$

where d_{pk} and $o_{pk}^{X_i}$ are the target output and the actual output of a pattern p in an output neuron k on a vector X_i, respectively; R and K are the number of patterns and output neurons, respectively.

The perturbation factor in this application is

closely related to the fitness and the number of patterns and output neurons. The perturbation must be inversely proportional to the fitness and the two numbers, R and K. This is because if a vector has a small fitness, its offspring must be largely mutated to search broadly for better solutions; the searching of a new solution should be scrutinized more when the number of patterns and output neurons is large.

Definition 2: *Perturbation Factor (Standard Deviation)*

Let X_i be a solution of the population. Then a *perturbation factor* is defined as:

$$\sigma_{X_i} = \frac{1}{\sqrt{F(X_i) * R * K}} \qquad (9)$$

where R and K are the same as in the above definition.

We can summarize all addressed processes as an algorithm. Algorithm 1 shows our adaptive learning-rate selection algorithm with Definitions 1 and 2.

Algorithm 1 Adaptive Learning-Rate Selection Algorithm with Evolutionary Programming()

// $2m$: the number of the population, (m parents + m offspring) //
// ϕ : the back-propagation network //
// n : the number of layer-1 in ϕ //
// C : the number of competition //
// X_i : the ith vector of the population P, $X_i = [x_{(0,i)}, \cdots, x_{(n-1,i)}]$ //
// $F(X_i)$: the objective function of vector X_i //
// J_i : the fitness score of vector X_i, $J_i \leftarrow F(X_i)$ //
// W_i : the winning number of X_i in competition //
// σ_i : the perturbation factor of vector X_i, $\sigma_i \propto \frac{1}{J_i}$ //

1 Initialize a population $P = [X_0, X_1, \ldots, X_{2m-1}]$, where $x_i \in [x_{min}, x_{max}]^n$
2 Randomly generate weight of ϕ
3 **while** (not termination-condition)
4 Assign a score J_i to each solution vector X_i by ϕ
5 **for** k=0 to C
Process competition until the count of competition
6 Select competitors (X_j) at random from the population (P), ($X_i \neq X_j$)
7 J_i is less than or equal to J_j, increment W_i
8 **end for**
9 Reorder the population P in descending order based on number of wins (W_i)
10 Select the m parents $P = [X_0, X_1, \ldots, X_{m-1}]$ in the population that have the highest ranked learning rates
Generate offsprings $(X_m, \ldots, X_{2m-1})$ from parents
11 Create the elements of an offspring X_{i+m} from each parent X_i. Modify the elements jth of each offspring X_{i+m} by a random perturbation $\delta_{x_{(i,j)}} \sim N(0, \sigma_{x_{(i,j)}})$, $x_{(i+m,j)} = x_{(i,j)} + \delta_{x_{(i,j)}}$
12 **end while**

---

3 Window Control Scheme

In previous work, the evolutionary adaptation of learning rates occured only when a predefined intervals (window size) was passed. More specifically, the window size is defined as follows.

Definition 3: *Window Size (S)*

Window Size (S) is an interval for selecting the learning rate with evolutionary programming. It has an integer value of from 1 to ∞.
$S = 1$; selection at every iteration (on-line).
$S = 2$; selection at every other iteration (on-line).
$S = i$; selection at every ith times iteration (on-line).
$S = \infty$; selection before training (off-line).

Algorithm 2 shows our modified algorithm with a window technique.

Algorithm 2 Adaptive Learning-Rate Selection Algorithm with Window Technique()

// N : the number of the iteration //
// S : the number of the window size//

1 **for** l=0 to N
2 **if** (not true of l *modulus* S)
3 perform algorithm 1
4 select a best solution
5 **end if**
6 apply the best solution to the neural network
7 **end for**

Using these static windows made the performance of our algorithm degrade because the evolutionary selection of new learning rates occured even though the current learning rates were good for learning. To solve this problem, we introduce a window control scheme. In this window control scheme, update learning rates occur only when the learning performance is continuously bad during a specified interval. That is, to speed up the training and avoid local minima, the adaptation of learning rates with evolutionary programming are applied only if a performance measure indicates that the training neural network may fall in the local minima. Otherwise, the neural network is trained with a previously selected best solution. The TSSE can be used for the performance measure. That is, if the TSSE is increased or not diminished during n integration steps, then the performance measure decides that adaptations are needed. As a result, the size of the window in a window control algorithm is dynamically changed according to a performance measure.

Algorithm 3 shows the learning-rate selection algorithm with window control algorithm.

Algorithm 3 Learning Rate Selection with Window Control Algorithm()

```
    // N : the number of the iteration //
    // P_l : the performance measure (in this paper
    TSSE) in lth iteration //
    // α : the predefined interval //
    // γ : the counter //
1   for l=0 to N
2       if (P_l > P_{l-1})
3           γ = γ + 1
4       else
5           γ = 0
6       end if
7       if (γ >= α)
8           perform algorithm 1
9           select a best solution
10      end if
11      apply the best solution to the neural network
12  end for
```

4 Experimental Results

We experimented with a sine function problem using three fixed window sizes and three intervals for window control scheme. In all experiments, CPU time is measured on an UltraSPARC workstation. We used 40 populations and three competitions for evolutionary programming in all experiments.

4.1 Sine Function Problem

We select sine function for the second experiment. The training set consists of 100 samples of the function from -2π to 2π [11]. Input sample patterns are provided for the 1-15-1 network.

Figure 2 shows the performances using our methods with fixed windows and the other adaptive methods. As you can see in figure 2, the performance

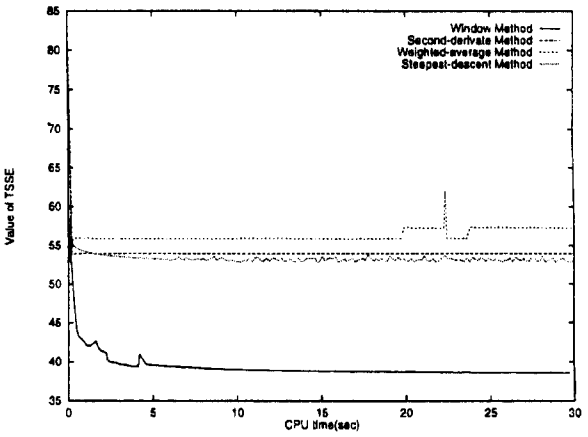

Figure 2: Results of a Sine Function in the 1-15-1 Network

Table 1: Averaged Value of The Smallest TSSE for Sine Function Problems

EPM1	EPM2	STM	SDM	WAM
40.3	40.0	49.6	53.4	54.5

of window method outperforms than those of three adaptive methods. Table 1 shows averaged value of the smallest TSSE for ten runs. In order to compare the fixed window method and window control method, we take four values, i.e., 1, 2, 4, and 8. In

fixed window method, the values are window sizes while the values are predefined intervals (α in algorithm 3). Figure 3 $\sim$ 6 shows the result of each value. In values 1, 2, and 4, the window con-

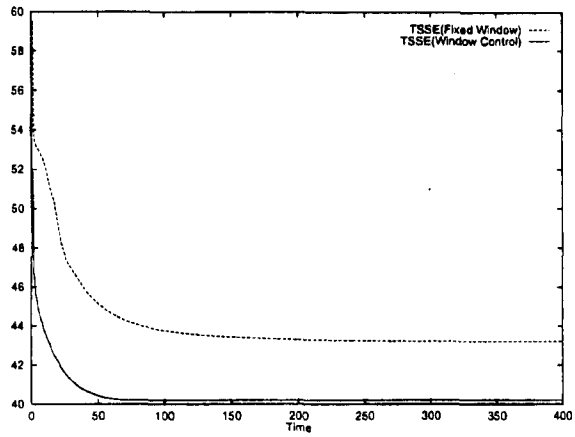

Figure 3: Results of a Sine Function in the 1-15-1 Network (Value: 1)

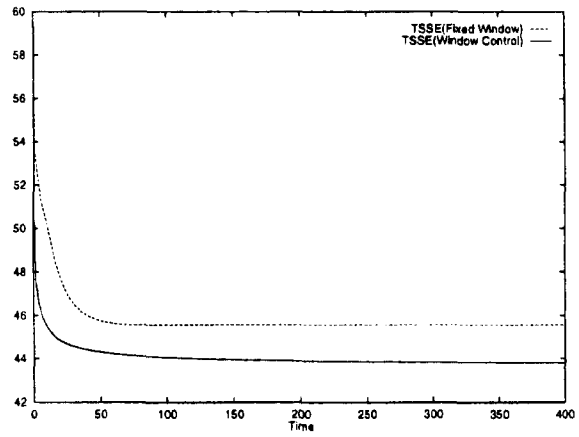

Figure 4: Results of a Sine Function in the 1-15-1 Network (Value: 2)

trol methods show better performance than fixed window methods. However, the performance is diminished according as the value is larger and larger. This is because the chance of updating learning rates is more and more diminished according as the value is larger and larger. As a result, in value 8, the performance of window control method becomes worse

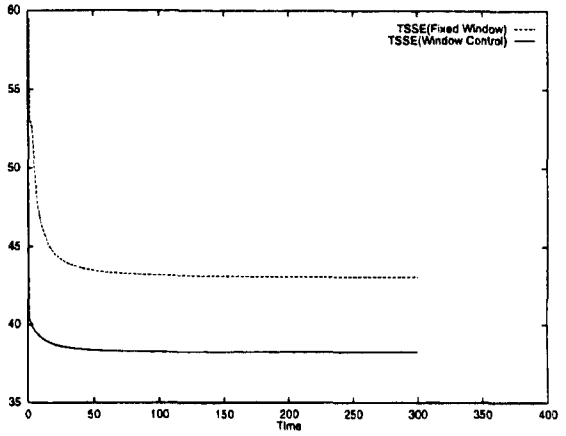

Figure 5: Results of a Sine Function in the 1-15-1 Network (Value: 4)

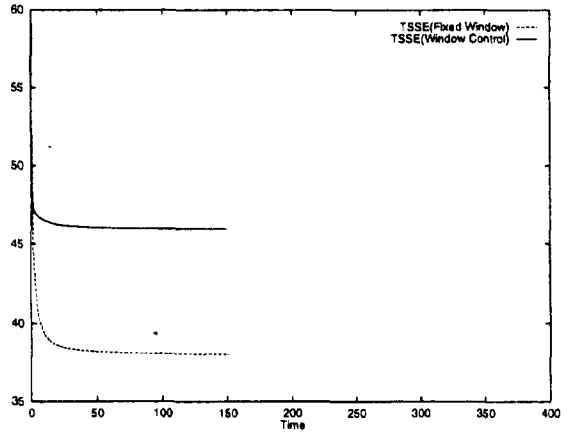

Figure 6: Results of a Sine Function in the 1-15-1 Network (Value: 8)

than the fixed window method. In the algorithm 3, if the value α does not become zero when the new performance measure P_l is less than or equal to P_{l-1}, then the performance may be increased. Table 2 shows the results in these experiments.

Table 2: Comparison of Fixed Window and Window Control Method for Sine Function Problems

	Fixed Window		Window Control	
value	TSSE	Time	TSSE	Time
1	43.23	1205.55	40.20	1197.80
2	45.54	606.33	43.77	598.20
4	43.05	300.70	38.25	299.81
8	38.06	150.60	45.95	149.60

5 Conclusion

In this paper, we introduced a window control scheme to enhance the performance of evolutionary learning-rate selection. The window control algorithm reduced the number of evolutionary selections by updating learning-rate only when the learning performance became bad compared with prior iterations. This made it possible that the unnecessary learning-rate selections didn't occur. Using this scheme, the main drawback of evolutionary selection method could be somewhat reduced. However, when the value of interval α was large, the effect of window control algorithm was more and more decreased. Thus, according as the learning proceeds, it is necessary that the value is also reduced like a simulated annealing algorithm. We will experiment about that as a future work.

References

[1] A. T. Bahill, *Verifying and Validating Personal Computer-based Expert Systems.* Prentice Hall New Jersey, 1991.

[2] J. McClelland and D. Rummelhart, *Parallel Distributed Processing, Volumes 1 and 2.* MIT press, Cambridge, MA, 1986.

[3] Watrous, "Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization," *Proceedings of International Joint Conference on Neural Networks*, pp. 619–627, 1986.

[4] M. K. Weir, "A Method for Self-Determination of Adaptive Learning Rates in Back Propagation," *Neural Networks*, vol. 4, pp. 371–379, 1991.

[5] P. R. Nachtsheim, "A First Order Adaptive Learning Rate Algorithm For Back Propagation Networks," *International Conference on Neural Network*, pp. 257–262, 1994.

[6] A. Zaghw and W. M. Dong, "An Automated Approach for Selecting the Learning Rate and Momentum in Back-Propagation Networks," *International Conference on Neural Network*, pp. 464–469, 1994.

[7] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," *International Joint Conference on Artificial Intelligence*, pp. 762–767, 1989.

[8] M. McInerney and A. P. Dhawan, "Use of Genetic Algorithms with Back Propagation in Training of Feed-Forward Neural Networks," *International Conference on Neural Network*, pp. 203–208, 1993.

[9] H. B. Kim, S. H. Jung, T. G. Kim, and K. H. Park, "A Fast Learning Method for Back-Propagation Neural Network by Evolutionary Adaptation of Learning Rates," *Neurocomputing*, vol. 11, pp. 101–106, May 1996.

[10] H. B. Kim, S. H. Jung, and K. H. Park, "Adaptive Learning-Rate Selection for BPNN Using Evolutionary Programming ," *IEEE Trans. on Neural Networks.* submitted to IEEE NN.

[11] A. F. A. A. G. P. Benito Fernandez and W. K. Tsai, "An accelerated learning algorithm for multilayer perceptron networks," *IEEE Transactions on Neural Networks*, vol. 5, pp. 493–497, May 1994.

[12] M. S. H.-C. H. Ching-Chung Li and R. J. Sclabassi, "An adaptive training algorithm for back-propagation neural networks," *IEEE Transactions on System, Man, and Cybernetics*, vol. 25, pp. 512–514, Mar. 1995.

[13] H. A. C. Eaton and T. L. Olivier, "Learning Coefficient Dependence on Training Set Size," *Neural Networks*, vol. 5, pp. 283–288, 1992.

[14] L. P. R. S. Ragazzini and G. Martinelli, "Learning of word stress in a sub-optimal second order back-propagation neural network," *IEEE International Conference of Neural Networks*, pp. I–355–I–361, 1988.

[15] W. L. Buntine and A. S. Weigen, "Computing second derivatives in feed-forward networks: A review," *IEEE Transactions on Neural Networks*, vol. 5, pp. 480–488, May 1994.

[16] A. El-Jaroudi and J. Makhoul, "A new error criterion for posterior probability estimation with neural nets," *International Joint Conference on Neural Networks*, pp. III–185–192, 1990.

[17] H. Kitano, "Emprical studies on the speed of convergence of neural network training using genetic algorithms," *Proceedings 8th JMIT national conference in artifical intelligence*, pp. 789–796, 1990.

[18] J. M. R. K. Belew and N. N. Schraudolph, "Evolving networks: Using the genetic algorithm with connectionist learning," *Artificial life II, SFI studies in the science of complexity, vol X*, pp. 511–547, 1991.

[19] M. G. Bello, "Enhanced training algorithms, and integrated training/architecture selection for multilayer perceptron networks," *IEEE Transactions on Neural Networks*, vol. 3, pp. 864–875, May 1992.