Adding Learning to Cellular Genetic Algorithms for Training Recurrent Neural Networks

Kim Wing C. Ku, *Student Member, IEEE,* Man Wai Mak, *Member, IEEE,* and Wan Chi Siu, *Senior Member, IEEE*

***Abstract*—This paper proposes a hybrid optimization algorithm which combines the efforts of local search (individual learning) and cellular genetic algorithms (GA's) for training recurrent neural networks (RNN's). Each weight of an RNN is encoded as a floating point number, and a concatenation of the numbers forms a chromosome. Reproduction takes place locally in a square grid with each grid point representing a chromosome. Two approaches, Lamarckian and Baldwinian mechanisms, for combining cellular GA's and learning have been compared. Different hill-climbing algorithms are incorporated into the cellular GA's as learning methods. These include the real-time recurrent learning (RTRL) and its simplified versions, and the delta rule. The RTRL algorithm has been successively simplified by freezing some of the weights to form simplified versions. The delta rule, which is the simplest form of learning, has been implemented by considering the RNN's as feedforward networks during learning. The hybrid algorithms are used to train the RNN's to solve a long-term dependency problem. The results show that Baldwinian learning is inefficient in assisting the cellular GA. It is conjectured that the more difficult it is for genetic operations to produce the genotypic changes that match the phenotypic changes due to learning, the poorer is the convergence of Baldwinian learning. Most of the combinations using the Lamarckian mechanism show an improvement in reducing the number of generations required for an optimum network; however, only a few can reduce the actual time taken. Embedding the delta rule in the cellular GA's has been found to be the fastest method. It is also concluded that learning should not be too extensive if the hybrid algorithm is to be benefit from learning.**

***Index Terms*—Baldwin effect, genetic algorithms, Lamarckian learning, real-time recurrent learning, recurrent neural networks.**

I. Introduction

NEURAL networks with closed paths[1] in their topology are known as recurrent neural networks (RNN's). The architecture of RNN's enables them to preserve past states of the networks. Therefore, RNN's have the capability of dealing with spatio-temporal problems which have been found to be difficult for feedforward networks [36]. In order to determine the weights of RNN's, a number of training algorithms have been proposed [38], [44], [48]. These algorithms are based on some gradient descent approaches where the weights in the RNN's are adjusted continually in order to minimize an error function.

Genetic algorithms (GA's) [12], [16], [33], [34], in contrast, are stochastic search algorithms based on the mechanics of natural selection and natural genetics. GA's can be and have been used in training neural networks[2] (for a review, see [46]). In this respect, the GA's are used to minimize the network error function which is typically defined as the mean squared error (MSE) between the actual outputs and the desired outputs for the whole training set. We have previously [25], [27] demonstrated that using cellular GA's [45] to train RNN's requires a long time to evolve acceptable solutions. One possible way to reduce the time taken is to add a learning mechanism to the cellular GA's. This leads to a hybrid optimization algorithm in which the effort of local search (individual learning) and GA's is combined.

In biological systems, learning occurs during the life-span of an individual, and it is a process that involves the interaction between an individual and its environment. Through the experience of this interaction, the behavior (expressed by the phenotype) of an individual is adapted accordingly such that it will be better at achieving its goals. This behavioral adaptation is achieved by modifying the "inborn" phenotype to the "learned" phenotype via learning. The motivation of adding a learning mechanism to GA's is that if each chromosome acquires knowledge about the environment through learning, it is possible to accelerate evolutionary adaptation.

There are two possible forms of embedding learning in GA's. In the first form, the change in the phenotype by learning is transformed to the corresponding change in the genotype. This is known as Lamarckian learning [2], [47] through which the acquired experience is passed to the offspring. The acquired information (observed in the phenotype) through learning is directly coded into the genotype.

In the second form of embedding learning in GA's, the learned behavior affects the genotypes indirectly. This is known as Baldwinian learning (based on the Baldwin effect [5], [43]).[3] Unlike Lamarckian learning, the genotypes after Baldwinian learning remain unchanged (i.e., the changes in phenotypes by learning cannot be transformed to genotypic changes). Only the fitness will be replaced by the "learned" fitness (i.e., fitness after learning). A chromosome will survive

Manuscript received February 4, 1998; revised September 28, 1998 and November 23, 1998. This work was supported by The Hong Kong Polytechnic University under Grant A/C 350/412.

The authors are with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong.

Publisher Item Identifier S 1045-9227(99)02030-5.

[1] Each node in an RNN is fully connected to all other nodes, and it has a feedback loop connecting itself.

[2] GA's may have difficulty in training neural networks due to the competing conventions problem [4].

[3] Hereafter, we denote the learning mechanism based on the Baldwin effect as "Baldwinian learning."

longer if its "learned" fitness is better, resulting in a smaller chance of being replaced in the next generation. If it can survive for a sufficient number of generations, then it is possible to evolve, by genetic operations, into the right genotype corresponding to the "learned" fitness. Although Baldwinian learning cannot change genotypes instantly, there is evidence [1], [15], [21] that it can direct the genotypic changes.

In other words, with Baldwinian learning, even if a chromosome has an undesirable "inborn" fitness (i.e., fitness before learning), it may still have a high chance (provided that its "learned" fitness is better) of being selected to evolve into a better chromosome by genetic operations. Baldwinian learning can be regarded as a kind of phenotypic variability; consequently, learning increases the variance of the selection (i.e., the effect of learning is to weaken selection and to increase genetic polymorphism) [3]. Results of previous research [19], [47] showed that incorporating Baldwinian learning into GA's has the effect of altering the fitness landscape such that it would become flatter around each local optimum. This phenomenon leads to an enlargement of the basin of attraction such that more chromosomes will be allocated around each local optimum. The overall effect of Baldwinian learning is that it can help to find the global optimum [37], [47], especially in a changing environment [3], [6].

Although both of the above learning mechanisms can be used in GA's, their philosophies are different and the extent to which they can assist GA's is also not clear. This prompts us to explore the effects of using these learning mechanisms in cellular GA's. Our finding is that Baldwinian learning cannot be better than Lamarckian learning in evolving neural networks, especially when the learning method can change a large number of weights in the networks and the changes are too large for genetic operations to cope with. Furthermore, it is found that the learning methods need not be sophisticated in order to gain the benefit of combining GA's and learning.

The paper is organized as follows. In Section II, we compare various approaches (including ours) to embedding learning in GA's for the optimization of the weights and/or the topologies of neural networks. A long-term dependency problem, to be tackled by the recurrent neural networks in the experiments, is described in Section III. Section IV introduces the cellular GA. Section V describes the learning methods that we have used in our experiments. Sections VI and VII compare and discuss the results of the simulations in which learning is embedded in cellular GA's to optimize the weights of RNN's. Finally, we conclude in Section VIII.

II. Background

Various attempts have been made to combine GA's and learning for the optimization of the weights and/or topologies of neural networks. Some researchers [9], [17], [23], [30] achieved good results while others [24], [35] found that learning could not help much. Their experiments differ in how learning is applied. Some researchers [23] used GA's to find possible regions containing the global optimum, then used learning as a final fine-tuning operator. Good results could possibly be obtained provided that an effective learning method is employed or the best solution found by GA's is already very close to the global optimum. As the learning methods we used are not effective for training RNN's when they are used alone (to be discussed in detail below), we did not consider this approach in our experiments. Other researchers [24] fine-tuned a chromosome when its fitness was good enough, or in other words, when its fitness was greater than a predefined threshold. However, it is difficult to determine the threshold value. Moreover, this approach assumes that greater improvement could be achieved by applying learning to chromosomes with better fitness. We believe that learning should be applied equally and that allowing poorly performed chromosomes to learn could also improve the evolution of the whole population. Therefore, we have adopted the approach similar to that of [9] and [17] where learning was applied to fine-tune every chromosome generated in each cycle of GA's. In our experiments, we have also investigated the effect of varying the learning frequency on the evolution process, as in [18].

Usually, learning methods depend very much on the chromosomal representation. For floating point representation, some researchers [23], [35], [41] used gradient descent algorithms such as backpropagation or its variants as the learning methods. In this case, the gradient in the fitness surface (or error surface) is calculated, and weights are changed accordingly. The gradient information is therefore fully utilized. However, these learning methods are computationally expensive for large networks. Apart from the gradient descent algorithms, the algorithms from Solis and Wets [42] can also be used [32]. On the other hand, if binary representation is used, the learning methods [24], [30] will usually involve flipping some bits in a chromosome randomly in order to obtain a better chromosome. These "bit-flipping" learning methods do not take the gradient information of the error surface into account. The implementation can be very simple as in [30], where learning is based upon the genotypes of parent chromosomes and their corresponding fitness. The time complexities of these "bit-flipping" learning methods can be very high as in [24], where fitness has to be calculated for each flipped bit. There is also a learning method [17], using binary chromosomal representation, that flips bits in a chromosome according to the Hebbian learning on output nodes. An interesting result among these "bit-flipping" learning methods is that even though they are simple and do not guarantee to produce a better chromosome after learning, they can improve the convergence of GA's [17], [30].

In our experiments, chromosomes have been represented as a string of floating point numbers. We are more interested in the learning methods that take the difference between the desired and actual outputs into account. If gradient information about the error surface is available, it is better to make use of it. Therefore, we have tried different gradient descent algorithms with the aim of making a learning method as simple as possible. Most reports did not show the actual time improvement, making the real benefit of combining GA's and learning difficult to observe. We, however, compare the actual

TABLE I
AN EXAMPLE OF TRAINING SEQUENCES FOR THE LONG-TERM DEPENDENCY PROBLEM. (NOTE THAT FOR THE OUTPUT PATTERNS, THE LAST TWO BITS DETERMINE WHETHER THE OUTPUT SYMBOL AFTER FIVE TIME STEPS IS x' OR y'

time step	$t-1$	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$	$t+6$	$t+7$	$t+8$	$t+9$	$t+10$	$t+11$	$t+12$
input symbol	...	x	a_1	a_2	a_3	a_4	a_5	y	a_1	a_2	a_3	a_4	a_5	...
input (binary)	...	001	010	011	100	101	110	000	010	011	100	101	110	...
output symbol	...	a_1	a_2	a_3	a_4	a_5	x'	a_1	a_2	a_3	a_4	a_5	y'	...
output (binary)	...	01000	01100	10000	10100	11000	00010	01000	01100	10000	10100	11000	00001	...

time taken in this paper.

III. THE LONG-TERM DEPENDENCY PROBLEM

Many sequence recognition tasks such as speech recognition, handwriting recognition and grammatical inference involve long-term dependencies—the output depends on inputs occurred long time ago. The sequences involved in these tasks are usually characterized by different time scales. In terms of short time scales, they can be characterized by the dynamics that generates the sequences, while in terms of long time scales, they may have syntactic and semantic structures. For example, speech recognition involves the processing of short-term speech signals as well as the processing of phonemic features spanning a much longer interval. In grammatical inference [28], a single word at the beginning of a sentence may affect the grammatical correctness or alter the interpretation of the sentence. In online handwriting recognition [7], words formed by a pen trajectory may possess sequential structures that spans a long period.

The performance of these applications depends mainly on whether the long-term dependencies can be accurately represented; however, extracting these dependencies from data is not an easy task. While recurrent neural networks provide a promising solution to this problem, previous research [8] has shown that the commonly used gradient descent algorithms have difficulty in learning the long-term dependencies. To overcome this difficulty, we propose to combine GA's and local search algorithms for training RNN's. The hybrid algorithms not only resolve the long-term dependencies problem efficiently, but also provide us an effective means to illustrate the benefit of combining different local search methods and GA's. We emphasis the benefit via the gain in convergence performance when the GA's and local search are combined. Here, the convergence performance is defined as the mean squared error (MSE) attained after a fixed period of time.

The problem we used is defined as follows. It is required to learn a temporal relationship such that the output at time t depends on the inputs from time $t-t'$ to $t-1$. Let us assume that an input sequence contains symbols drawn from a symbol

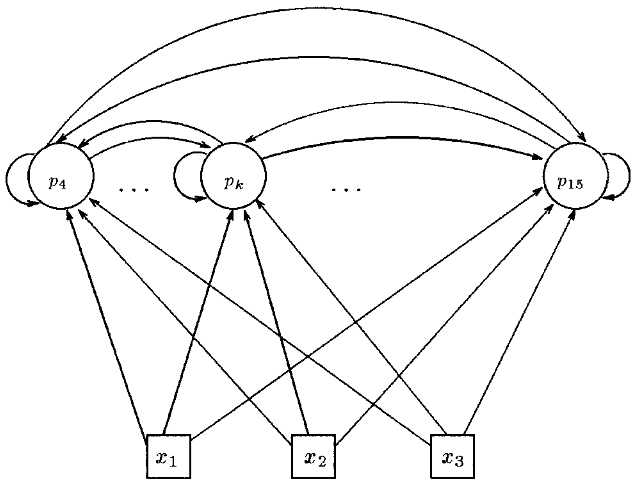


Fig. 1. A fully connected recurrent neural network with three inputs (x_1 to x_3), 12 processing nodes (p_4 to p_{15}), and five outputs (obtained from p_{11} to p_{15}).

set, and that each symbol is represented by a binary number with N bits. There are only two possible input sequences

$$I = \begin{cases} (x, a_1, a_2, a_3, \cdots, a_k) \\ (y, a_1, a_2, a_3, \cdots, a_k) \end{cases}$$

where $x, y,$ and $\{a_i\}_{i=1}^{k}$ are the symbols in the symbol set. The first symbol in the input sequence can be either x or y, but the next k input symbols are fixed. The corresponding output sequences are

$$O = \begin{cases} (a_1, a_2, a_3, \cdots, a_k, x') & \text{if } I = (x, a_1, a_2, a_3, \cdots, a_k) \\ (a_1, a_2, a_3, \cdots, a_k, y') & \text{if } I = (y, a_1, a_2, a_3, \cdots, a_k). \end{cases}$$

In other words, when the first input symbol is x at time t, the output at time $t+k$ is x'; when the first input symbol is y at time t, the output at time $t+k$ is y'. For other time intervals, the output predicts the next input. A training sequence is formed by the concatenation of ten randomly chosen input–output sequences. A test sequence comprising 100 randomly chosen input–output sequences is used to determine the misclassification rate (i.e., the chance of misclassifying an input sequence). Table I shows an example of the training sequences with temporal length $k = 5$. As the problem becomes increasingly difficult when the temporal length increases, we used a length of five time steps which was found to be sufficiently difficult for the gradient descent algorithms.

In this study, RNN's (Fig. 1) with three input nodes and twelve processing nodes (five of them were dedicated as the output nodes) have been used to learn the long-term dependency problem with a temporal length of five time steps. Therefore, there are a total of $12 \times 12 + 12 \times (3+1) = 192$ weights required to be optimized.

Although the long-term dependency problem is a hypothetical problem, it can be used as a framework for more complex sequence recognition tasks where classification decisions must be made at the end of a sequence. For example, in [39], the correct spelling of a sequence of corrupted text can be found by a prediction model, which is trained to predict the next letter from the previous letters. After training, the model is able to generate a large number of possible text sequences. The most probable text is the sequence that has the largest probability of matching the corrupted text, given an estimation of the probability of having incorrect text. Likewise, predictive neural networks which predict the next frame of speech based on several previous frames can be used as speaker models for speaker identification. Given an utterance spoken by an unknown speaker, his/her identity can be found by selecting the speaker model with minimum prediction error at the end of the utterance [20].

While the long-term dependency problem is rather simple when compared to the above real-world problems, it allows us to have a better control of the experimental conditions. For example, the extent of the long-term dependency can be easily controlled by changing the number of time steps between the first input symbol and the last output symbol. In more difficult problems, however, there may be many uncontrollable factors that affect the efficiency of the training process, making the interaction of learning and GA's difficult to observe.

IV. Cellular GA's

The idea of cellular GA's has been introduced by several researchers [10], [11], [45]. It has been used in [2], [17] where learning and GA's were combined to train neural networks. In cellular GA's, the population of chromosomes are organized as a two-dimensional toroidal grid with each grid point representing a chromosome. To use cellular GA's to optimize the weights of RNN's, each weight in the networks is encoded as a gene of a chromosome and in the form of a floating-point number. A chromosome, in which the number of genes is equal to the number of weights, represents an RNN. The fitness of a chromosome is determined by the network error function which is the MSE between the desired outputs and the actual outputs. In this case the better the fitness, the lower is the MSE. The following is the procedure of the cellular GA's used in our experiments:

```
procedure cellularGA
$c_k$        Chromosome at position $(x_k, y_k)$ in the grid.
$c_{\text{new}}$     Newly produced chromosome.
$l$          Length of random walk.
$M$          Total number of chromosomes in the population.
$w_{ij}^{c_k}$      Weights $w_{ij}$ of the network corresponding to $c_k$.
$f(c_k)$     fitness of $c_k$.
$\mathcal{I}$          Set of indexes representing the input nodes
             (including the bias).
$\mathcal{U}$          Set of indexes representing the processing
             nodes.

begin
  Initialize a population of $M$ chromosomes $c_k$, and
  evaluate the corresponding
    fitness $f(c_k)$ where $k = 1, 2, \cdots, M$

  // Generate a new chromosome for each
     reproduction cycle
  repeat
    Randomly select $c_0$ at $(x_0, y_0)$ in the grid

    // Choose parent $c_a$ along a random walk
       originated from $(x_0, y_0)$
    Create a random walk $\{c_1$ at $(x_1, y_1), c_2$
    at $(x_2, y_2), \cdots, c_l$ at $(x_l, y_l)\}$ such that
      $|x_{k+1} - x_k| \le 1$ and $|y_{k+1} - y_k| \le 1$,
      $k = 0, 1, 2, \cdots, l-1$
    Select $c_a$ such that $f(c_a)$ is the best along the
    random walk

    // Choose parent $c_b$ along another random walk
       originated from $(x_0, y_0)$
    Create a random walk set $\{c_1'$ at $(x_1', y_1'), c_2'$ at
    $(x_2', y_2'), \cdots, c_l'$ at $(x_l', y_l')\}$ such that
      $|x_{k+1}' - x_k'| \le 1$ and $|y_{k+1}' - y_k'| \le 1$,
      $k = 1, 2, \cdots, l-1$ and
      $|x_1' - x_0| \le 1$ and $|y_1' - y_0| \le 1$
    Select $c_b$ such that $f(c_b)$ is the best along the
    random walk

    // Apply crossover to $c_a$ and $c_b$ to produce $c_{\text{new}}$
       for all $i \in \mathcal{U}, j \in \mathcal{U} \cup \mathcal{I}$ do

         $w_{ij}^{c_{\text{new}}} := \begin{cases} w_{ij}^{c_a} & \text{with a probability of 0.5} \\ w_{ij}^{c_b} & \text{with a probability of 0.5} \end{cases}$

    endloop

    // Apply mutation to $c_{\text{new}}$ by randomly selecting
    //a processing node in the network, and each
    // weight connected to the input part of the node is
    //changed by exponentially distributed mutation
    Randomly select $i \in \mathcal{U}$
    for all $j \in \mathcal{U} \cup \mathcal{I}$ do

      $w_{ij}^{c_{\text{new}}} := \begin{cases} w_{ij}^{c_{\text{new}}} + \delta & \text{with a probability of 0.5} \\ w_{ij}^{c_{\text{new}}} - \delta & \text{with a probability of 0.5} \end{cases}$
```

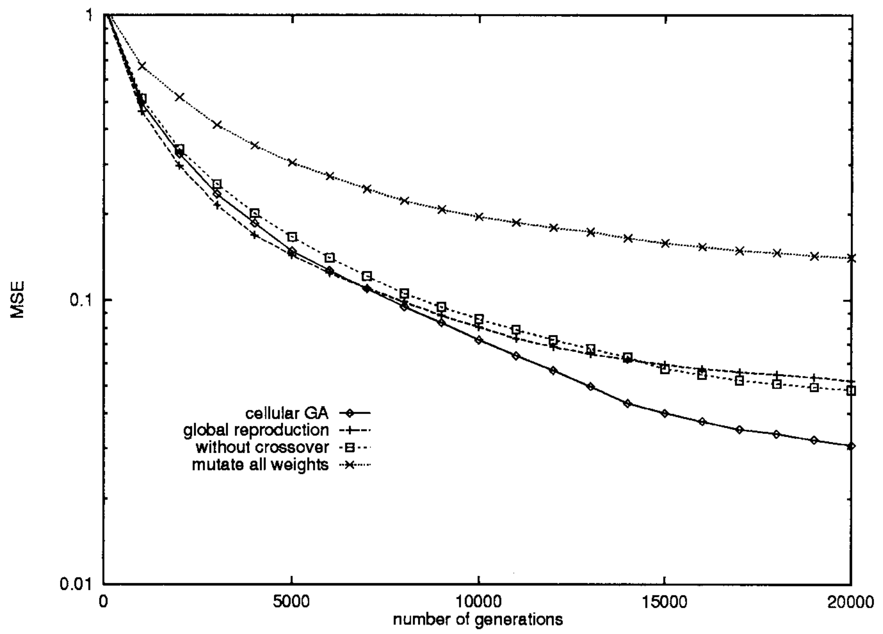


Fig. 2. Comparisons of using different forms of GA's to train RNN's to solve the long-term dependency problem. Results are based on the average of 100 simulations.

```
            // $\delta$ is a positive number randomly generated
            //from an exponential
            // distribution with density function of the form
            //$e^{-x}, x > 0$
        endloop
        // Replace $c_0$ by $c_{\mathrm{new}}$ if the latter has better fitness
        Evaluate $f(c_{\mathrm{new}})$
        if $f(c_{\mathrm{new}}) < f(c_0)$ then $c_0 := c_{\mathrm{new}}$
    until termination condition reached
endproc cellularGA.
```

In each reproduction cycle, every position in the grid has equal opportunity of being selected for starting a random walk. However, as the best chromosome along a random walk is always chosen for crossover, chromosomes with better fitness have a higher probability of being selected. In our experiments, a population size of 100 and a random walk of four steps[4] have been used. We have found that the cellular GA is able to find an acceptable solution for the long-term dependency problem with these parameter settings.

In cellular GA's, the reproduction process takes place "locally" in the grid. The reason for using cellular GA's in our experiments is that bigger variance in genomes is allowed if the population is spatially distributed (i.e., chromosomes are arranged spatially, say in a toroidal grid, and reproduction can only be occurred between neighboring chromosomes). Local reproduction has the effect of reducing selection pressure so that more exploration of the search space can be achieved [29] and the risk of getting stuck in local optima can be reduced, especially in the case where Lamarckian learning is used [2]. The effect of using a spatially distributed population can be assessed by comparing its performance with that of another GA where crossover is allowed between any two parents (i.e., the reproduction process takes place "globally" in the population). Such comparison can be found in Fig. 2 where different GA's were used to train RNN's in solving the long-term dependency problem. It is evident that the cellular GA (-◇- in Fig. 2) outperforms the GA with "global" reproduction (-+- in Fig. 2). We also found that when the GA with "global" reproduction was used, six out of 100 simulation runs were trapped in local optima with MSE being higher than 0.2. However, when the cellular GA was used, none of the simulations was found to be trapped in these local optima. Therefore, it is more appropriate to use cellular GA's in this case.

One may notice that cellular GA's use crossover extensively. It has been criticized [40] that the use of crossover can be detrimental to searching for a good solution in some circumstances. Fogel *et al.* [13], [14] also found that there was no advantage of using crossover in their experiments. To investigate the effectiveness of the crossover operator, we have removed it from the cellular GA, resulting in an evolutionary algorithm with asexual reproduction. The cellular GA as specified in **procedure cellularGA** and the asexual evolutionary algorithm formed by removing the crossover operator from **procedure cellularGA** were used to train RNN's in solving the long-term dependency problem. Fig. 2 illustrates that the evolutionary algorithm without crossover attains a higher MSE. This result prompts us to use crossover in the cellular GA and all hybrid algorithms in this work.

During the mutation, a node in the network is randomly selected, and each weight connected to the input part of that node is changed by a positive or negative offset with

[4]The length of random walk depends on the population size, too long ·□· (in Fig. 3) or too short (· + · in Fig. 3) are not appropriate.

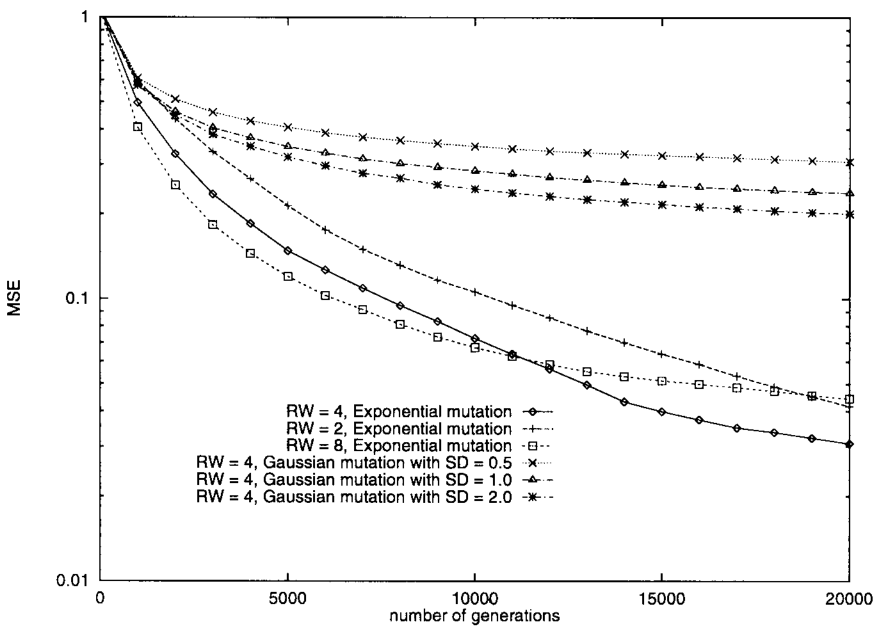


Fig. 3. Comparisons of exponential mutation and Gaussian mutation using different standard deviations (SD) and random walk (RW) length. Results are based on the average of 100 simulations.

exponential distribution (see **procedure cellularGA**). As there are 12 processing nodes, the probability for a weight to be mutated is $\frac{1}{12}$. We have increased this probability to 1 (i.e., all weights will be mutated). However, the result (✕ in Fig. 2) is poor. In this work, the offset due to mutation follows an exponential distribution. Other researchers [35], [41] also made use of exponential distributions rather than Gaussian distributions. Their reasoning is that most of the weights in the optimal solution tend to be small in magnitude but some may have large absolute values. Therefore, exponential distributions which favor small offsets but still allow large offsets to occur were used. This can be justified by a pilot experiment in which the effect of using Gaussian mutation is compared with that of using an exponential one. As shown in Fig. 3, the former leads to a very poor result in the long-term dependency problem. Therefore, exponential distributions have been adopted in this study.

V. Learning Methods

We have used the cellular GA as described above to optimize the weights of an RNN in solving the long-term dependency problem. In order to improve the convergence, we have also incorporated several learning methods into the cellular GA. These learning methods are hill-climbing algorithms, and their aim is to obtain a better set of weights such that a smaller MSE can be achieved when they are incorporated into the cellular GA.

A. Real-Time Recurrent Learning (RTRL)

The real-time recurrent learning (RTRL) [48] algorithm is an on-line training algorithm for RNN's. It is a gradient-based algorithm in which the weights of the network are determined by minimizing the MSE between the desired output and the actual output at the current time step. Given an RNN, the corresponding error gradient at the current time step is calculated, and the weights are changed according to the error gradient to minimize the MSE. The parameters of an RNN are defined as follows:

$x_k(t)$	Signal applied to input node k at time step t.
$y_k(t)$	Actual output of processing node k at time step t.
$\mathcal{I}$	Set of indexes representing the input nodes (including the bias).
$\mathcal{U}$	Set of indexes representing the processing nodes (including the output nodes).
$\mathcal{O}$	Set of indexes representing the output nodes.
$z_k(t) = x_k(t)$	$z_k(t) = x_k(t)$ if $k \in \mathcal{I}, z_k(t) = y_k(t)$ if $k \in \mathcal{U}$.
$d_k(t)$	Target output of processing node k at time step t.
$s_k(t)$	Activation of processing node k at time step t.
w_{ij}	Weight connecting node j to node i.

The dynamics of node $k \in \mathcal{U}$ in the RNN is defined as

$$\begin{aligned} y_k(t+1) &= f_k(s_k(t)) \\ &= f_k\left(\sum_{p\in\mathcal{I}} w_{kp}x_p(t) + \sum_{q\in\mathcal{U}} w_{kq}y_q(t)\right) \end{aligned} \tag{1}$$

where $f_k(\ )$ is a sigmoidal function and $y_k(0) = 0$. The instantaneous squared error at time step t is defined as

$$E(t) = \tfrac{1}{2}\sum_{k\in\mathcal{O}}\{e_k(t)\}^2 = \tfrac{1}{2}\sum_{k\in\mathcal{O}}\{d_k(t) - y_k(t)\}^2. \tag{2}$$

The weights are updated by

$$\Delta w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}} = \alpha \sum_{k \in \mathcal{O}} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}} \tag{3}$$

where $i \in \mathcal{U}, j \in \mathcal{U} \cup \mathcal{I}$, and α is the learning rate.

For notation convenience, we denote $\partial y_k(t)/\partial w_{ij}$ as $p_{ij}^k(t)$. By differentiating (1) with respect to $w_{ij}, p_{ij}^k(t+1)$ can be found by

$$p_{ij}^k(t+1) = f_k'(s_k(t)) \left\{ z_j(t)\delta_{ki} + \sum_{q \in \mathcal{U}} w_{kq} p_{ij}^q(t) \right\} \tag{4}$$

where $i \in \mathcal{U}, j \in \mathcal{U} \cup \mathcal{I}, k \in \mathcal{U}, \delta_{ki}$ is the Kronecker delta, and $p_{ij}^k(0) = 0$.

The RTRL algorithm [represented by (1)–(4)] is computational intensive because it has a time complexity of $O(n^4)$ for each time step, where n is the number of processing nodes. A simple GA-RTRL hybrid approach will lead to unrealistic computation time. This limitation causes us to derive several simplified versions of RTRL. The idea behind the simplified versions is that we aim at reducing the overall computation time of the GA-RTRL hybrid algorithms by reducing the complexity of the learning algorithm.

B. Restricted RTRL

In the original RTRL algorithm, all weights are changed in a direction opposite to the error gradient. Therefore, the amount of computation increases with the number of weights. In order to reduce the complexity of each learning cycle, changes in weights are restricted to those connecting to output nodes where target outputs are given. Therefore, (3) remains the same, but $i \in \mathcal{O}$ instead of $\mathcal{U}$. As a result, we only need to calculate those p_{ij}^k values where $i \in \mathcal{O}, j \in \mathcal{U} \cup \mathcal{I}$, and $k \in \mathcal{O}$. Furthermore, we assume that $p_{ij}^k = 0$ for all $k \neq i$ and for all $k \notin \mathcal{O}$. Therefore, only those p_{kj}^k $(= \partial y_k / \partial w_{kj})$ where $k \in \mathcal{O}$ and $j \in \mathcal{U} \cup \mathcal{I}$ have values other than zero. This means that the changes in w_{kj} will only affect the changes in the output of processing node k. Equation (3) becomes

$$\Delta w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}} = \alpha e_i(t) \frac{\partial y_i(t)}{\partial w_{ij}} \tag{5}$$

where $i \in \mathcal{O}, j \in \mathcal{U} \cup \mathcal{I}$, and α is the learning rate. Hence (4) becomes

$$p_{ij}^i(t+1) = f_i'(s_i(t))\{z_j(t) + w_{ii} p_{ij}^i(t)\} \tag{6}$$

where $i \in \mathcal{O}$ and $j \in \mathcal{U} \cup \mathcal{I}$.

Comparing to the original RTRL algorithm, the restriction on the weight changes in this learning method may cause errors in the gradient computation. However, combining this restricted learning method with the cellular GA is an attractive alternative provided that the combination can shorten the time in finding an acceptable solution. This also applies to the following learning method where the computational complexity is further reduced.

C. Delta Rule for Output Nodes Only (DR)

This approach simplifies the above learning method further. It differs from the restricted RTRL in that $p_{kj}^k(t+1)$ does not depend on $p_{kj}^k(t)$, where $k \in \mathcal{O}$ and $j \in \mathcal{U} \cup \mathcal{I}$. Therefore (6) becomes

$$p_{ij}^i(t+1) = f_i'(s_i(t)) z_j(t) \tag{7}$$

where $i \in \mathcal{O}$ and $j \in \mathcal{U} \cup \mathcal{I}$. Combining (3) and (7), the weights connected to the output nodes are updated by

$$\Delta w_{ij}(t) = \alpha e_i(t) f_i'(s_i(t-1)) z_j(t-1) \tag{8}$$

where $i \in \mathcal{O}$ and $j \in \mathcal{U} \cup \mathcal{I}$.

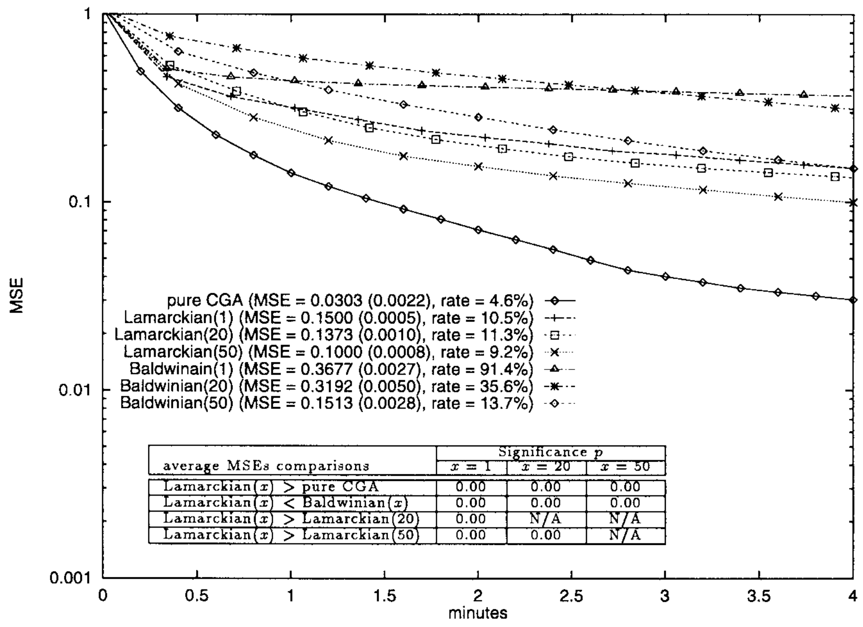
We can see that (8) is the delta rule for the output nodes. During each learning cycle, we consider the fully connected RNN as a feed-forward network. The dynamics of the network is based on a fully connected RNN architecture; however, the updates of weights are based on a feed-forward architecture [i.e., the delta rule given in (8)]. The philosophy behind this approach is to reduce the computational complexity as much as possible by eliminating the term $\Sigma_{q \in \mathcal{U}} w_{kq} p_{ij}^q(t)$ in (4).

VI. Embedding the Learning Methods in Cellular GAs

There are various ways of incorporating the learning methods, as described in the previous section, into the cellular GA. First, different learning methods can be used to learn for one epoch, where an epoch is a complete presentation of all training patterns. Second, the learning frequency can be varied, i.e., learning can take place after every reproduction or at regular generation intervals. Third, we can adopt Lamarckian learning or Baldwinian learning. In this section, different combinations are specified and their results are shown. The average result (averaged over 200 simulations) of each combination is plotted. The time taken for each simulation is based on the CPU time of a Sun Sparc 1000 workstation. The MSE's (together with the variances) attained after 4 min of simulation are also tabulated so that the significance p of the difference between two MSE's can be calculated by Student's t-tests, where $p < 0.05$ implies that the difference is statistically significant.

In all simulations, the reproduction process has been the same as **procedure cellularGA**, but learning was applied to the newly-born offspring at each generation. The learning rate of all learning algorithms was fixed at 0.9. Since RTRL is computational intensive, applying learning after every reproduction results in long computation time. In order to reduce the overall complexity, simulations where the RTRL was applied to a randomly selected chromosome at regular generation intervals have also been performed.

A set of control experiments have been performed. In these experiments, only the learning methods described in Section V were used (i.e., without GA's) to train an RNN in order to solve the long-term dependency problem. It was found that the RTRL algorithm found a solution with MSE being less than 0.01 only in one out of ten simulation runs. For other nine simulation runs, the RTRL algorithm can only reduce the



average MSEs comparisons	Significance p		
	$x = 1$	$x = 20$	$x = 50$
Lamarckian(x) > pure CGA	0.00	0.00	0.00
Lamarckian(x) < Baldwinian(x)	0.00	0.00	0.00
Lamarckian(x) > Lamarckian(20)	0.00	N/A	N/A
Lamarckian(x) > Lamarckian(50)	0.00	0.00	N/A

Fig. 4. Comparisons of average MSE's (with respect to CPU time taken) achieved by embedding RTRL in the cellular GA. The MSE's after 4 min of simulation, their variances (inside brackets), the corresponding misclassification rates, and the significance p (calculated by Student's t-tests) of the difference in MSE's are also shown. Note that Lamarckian(x)/Baldwinian(x) represents Lamarckian/Baldwinian learning applied at every x generations.

MSE's to 0.08. For other learning methods (restricted RTRL and the delta rule), no acceptable solution can be found in all simulation runs, and the MSE's can only be reduced to 0.09. This indicates that using the gradient-based methods alone is not able to solve the long-term dependency problem.

A. Embedding RTRL in Cellular GA's

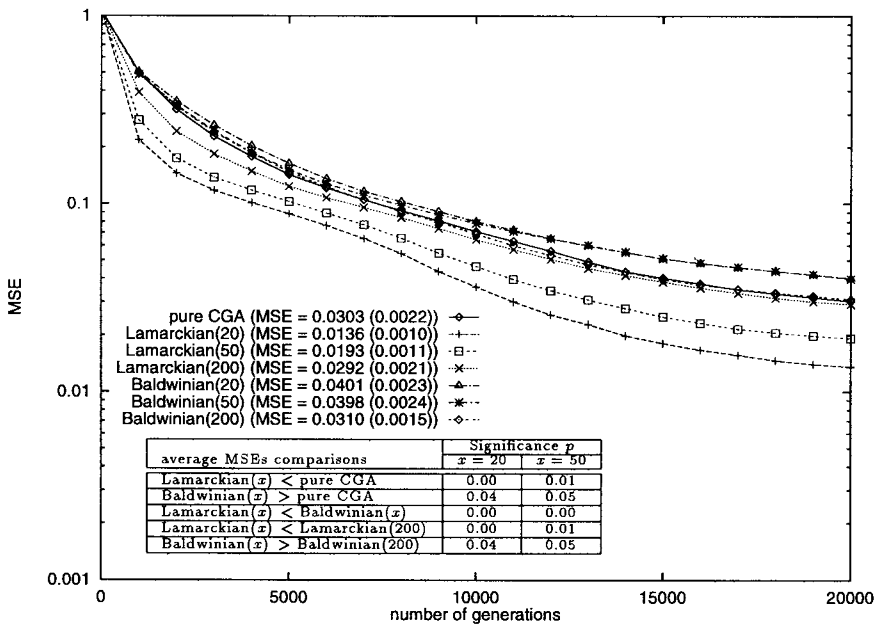
The table in Fig. 4 shows the significance p, computed by Student's t-tests, of the difference in MSE's between any two approaches to embedding RTRL in cellular GA's. The difference is considered to be statistical significant when p is less than 0.05. The results show that the pure cellular GA achieves a statistically lower MSE than all cases of Lamarckian learning. It also shows that when the generation interval between the applications of Lamarckian learning is short, the MSE attained is high. However, Fig. 5 shows that when the time involved in learning is neglected, applying Lamarckian learning at a short generation interval (e.g., 20 or 50) can achieve a statistically lower MSE. These results suggest that although RTRL may provide some benefit, the corresponding increase in computation time may not provide sufficient payoff.

A comparison between the convergence of the Lamarckian learning and the Baldwinian learning applied at the same generation interval (see Fig. 4) reveals that the latter achieves statistically higher MSE's. The inefficiency of Baldwinian learning is clearly shown. Fig. 5 shows that even if the learning time is neglected, the MSE's attained after 20 000 generations are statistically higher when Baldwinian learning is applied at a short generation interval (e.g., 20 or 50). We have the following conjecture for explaining this phenomenon. *The more difficult it is for genetic operations (crossover and mutation) to produce the changes between the genotypes corresponding to the "inborn" fitness and the "learned" fitness, the poorer is the convergence of Baldwinian learning.*

In Baldwinian learning, the "learned" fitness of a chromosome is the fitness obtained after learning. This "learned" fitness is not equal to the "inborn" fitness corresponding to the genotype. Genetic operations are therefore required to produce the change in the genotype, where the change should correspond to the difference between the "inborn" fitness and the "learned" fitness. While these genotypic changes are produced randomly by crossover and mutation, only some of them may match the phenotypic changes caused by learning. If only one gene (or one weight) is allowed to be changed[5] during Baldwinian learning, the genetic operations should have no difficulty in producing this change. However, in the RTRL algorithm, all weights are changed; consequently, it is very difficult for genetic operations to produce the corresponding changes in the weights. It becomes more difficult to produce the changes when the learning frequency is high, since the weights are changed more often. Therefore, according to our conjecture, the results of the Baldwinian learning are poor even if the time spent on learning is neglected. This also explains why the convergence of all cases of Baldwinian learning in Fig. 5 is poorer than that of the pure cellular GA.

French *et al.* [15] did similar investigations on the factors that affect the convergence of Baldwinian learning. They found that when the amount of phenotypic plasticity (difficulty in learning) was either too small or too large, the convergence became poor. In another study, Keesing *et al.* [22] showed that the amount of fitness improvement incurred by learning affects

[5]The "learned" fitness is obtained by changing that gene while keeping other genes fixed.



average MSEs comparisons	Significance p	
	$x = 20$	$x = 50$
Lamarckian(x) < pure CGA	0.00	0.01
Baldwinian(x) > pure CGA	0.04	0.05
Lamarckian(x) < Baldwinian(x)	0.00	0.00
Lamarckian(x) < Lamarckian(200)	0.00	0.01
Baldwinian(x) > Baldwinian(200)	0.04	0.05

Fig. 5. Comparisons of average MSE's (with respect to the number of generations taken) achieved by embedding RTRL in the cellular GA. The MSE's after 20 000 generations, their variances (inside brackets) and the significance p (calculated by Student's t-tests) of the difference in MSE's are also tabulated. Note that Lamarckian(x)/Baldwinian(x) represents Lamarckian/Baldwinian learning applied at every x generations.

the Baldwin effect significantly. In other words, too little or too much improvement could lead to poorer convergence. In addition to these factors, this study suggests that the level of difficulties for subsequent genetic operations to obtain the necessary changes in genotypes is also a significant factor that affects the Baldwin effect.

B. Embedding Restricted RTRL and Delta Rule in Cellular GA's

Figs. 6 and 7 show that when the complexity of the learning method is reduced, the MSE's achieved by Lamarckian learning are statistically lower than that achieved by the pure cellular GA. This indicates that when Lamarckian learning is properly embedded in the cellular GA, better neural networks can be obtained. Another advantage of embedding Lamarckian learning is that the resulting hybrid algorithms save computation time considerably. For example, the pure cellular GA takes 4 min to attain a MSE of 0.0303. To evolve a network to the same accuracy, the hybrid algorithm with restricted RTRL requires 2.0 min and that with the delta rule requires 1.4 min, suggesting that up to 65% of computation time can be saved.

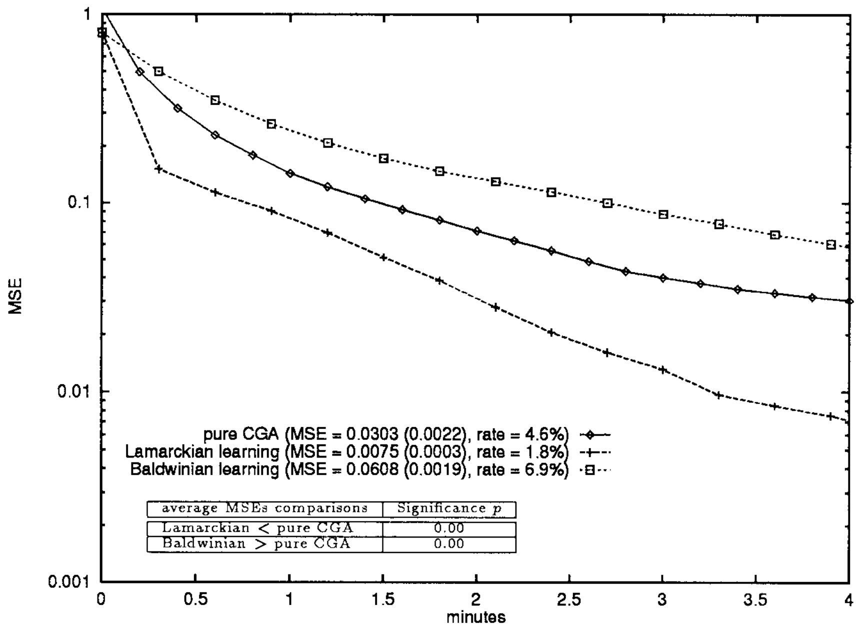
Figs. 6 and 7 show that embedding restricted RTRL or the delta rule in the cellular GA using Baldwinian mechanism performs poorly during the first 4 min. However, these hybrid algorithms achieve a significantly lower (significance $p < 0.01$) MSE after 20 000 generations, as shown in Table II. This indicates that if computation time is not a concern, Baldwinian learning has merits. Of particular interest is that no such situation occurs when RTRL is embedded in the cellular GA using Baldwinian mechanism (see Fig. 4 and Table II where Baldwinian learning performs poorly with respect to both convergence rate and achievable MSE's). Recall that the main difference between RTRL and simplified learning methods such as restricted RTRL and the delta rule is that the latter has a smaller number of changeable weights. Consequently, it is relatively easy for the genetic operations to produce the changes in weights caused by the simplified learning methods. This suggests that Baldwinian learning is able to assist evolutionary search provided that the learning is not excessive.

VII. Discussions

In general, a well-trained network has a low misclassification rate on test data. This can be observed in Figs. 4, 6, and 7. We found that when a network learned well on the training set, then it also performed well on the test set. Therefore, a network that is successfully trained is able to solve the long-term dependency problem.

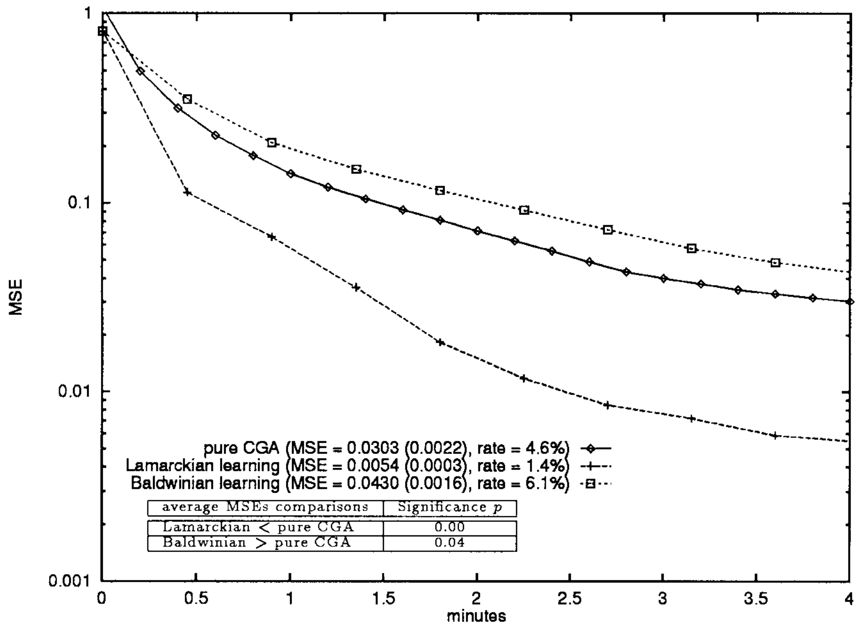
Comparing various implementations of Lamarckian learning, embedding the delta rule in the cellular GA achieves the lowest MSE in a given CPU time. For example, the average MSE attained after 4 min is only 18% of that achieved by the pure cellular GA. Bear in mind that applying the delta rule does not guarantee any improvement in fitness in each learning cycle. Applying this learning method alone is not rewarding because the error gradient computed by this method may differ significantly from the more accurate one [compare (7) and (4)]. To see whether this approach is successful in other GA's, we have tried embedding the delta rule in different GA's. Table III illustrates the improvement obtained when the delta rule was embedded, suggesting that this approach can also be applied to other GA's.

So far we have focused on the convergence performance of the hybrid algorithms by looking at the MSE's after a



average MSEs comparisons	Significance p
Lamarckian < pure CGA	0.00
Baldwinian > pure CGA	0.00

Fig. 6. Comparisons of average MSE's (with respect to CPU time taken) achieved by embedding the restricted RTRL in the cellular GA. The MSE's, their variances (inside brackets), the corresponding misclassification rates, and the significance p (calculated by Student's t-tests) of the difference in final MSE's are also shown.



average MSEs comparisons	Significance p
Lamarckian < pure CGA	0.00
Baldwinian > pure CGA	0.04

Fig. 7. Comparisons of average MSE's (with respect to CPU time taken) achieved by embedding the delta rule (DR) in the cellular GA. The final MSE's, their variances (inside brackets), the corresponding misclassification rates, and the significance p (calculated by Student's t-tests) of the difference in final MSE's are also shown.

given CPU time. It is also interesting to explore the capability of these algorithms without considering the computation time involved. Table II summarizes the MSE's achieved by various hybrid algorithms after 20 000 generations. It shows that combining cellular GA's and RTRL with Lamarckian learning applied at every generation attains the lowest MSE. This approach, however, has limitations as it requires an extremely long computation time. For example, to reach 20 000 generations, this hybrid algorithm requires 11 h, whereas the hybrid algorithm that combines cellular GA's and delta rule requires 9 min only. It is also evident that when the learning frequency decreases (generation interval between learning increases), the MSE achieved by Lamarckian learning increases while that achieved by Baldwinian learning decreases. This phenomenon agrees with our conjecture for Baldwinian learning that learning should not be too extensive; otherwise, the genetic

TABLE II
MSEs AND VARIANCES (INSIDE BRACKETS) ATTAINED AFTER 20 000 GENERATIONS BY EMBEDDING DIFFERENT LEARNING METHODS IN CELLULAR GA'S. ALL RESULTS ARE BASED ON THE AVERAGE OF 200 SIMULATION RUNS, EXCEPT RTRL WITH LEARNING APPLIED BY EVERY GENERATION WHERE THE MSE'S ARE BASED ON THE AVERAGE OF TEN SIMULATION RUNS BECAUSE OF THE LONG COMPUTATION

Learning method	Learning every x generations	Lamarckian	Baldwinian
without learning	N/A	0.0303 (0.0022)	
RTRL	1	0.0001	0.1161
RTRL	20	0.0136 (0.0010)	0.0401 (0.0023)
RTRL	50	0.0193 (0.0011)	0.0398 (0.0024)
RTRL	200	0.0292 (0.0021)	0.0310 (0.0015)
restricted RTRL	1	0.0024 (0.0002)	0.0167 (0.0010)
delta rule	1	0.0047 (0.0003)	0.0196 (0.0013)

TABLE III
MSES ATTAINED BY EMBEDDING THE DELTA RULE IN DIFFERENT GA'S USING THE LAMARCKIAN MECHANISM. ALL FIGURES WERE OBTAINED BY RECORDING THE MSE'S AFTER 4 MIN (CPU TIME) OF SIMULATION

	cellular GA	cellular GA without crossover	cellular GA with 'global' reproduction	cellular GA with all weights mutate
without learning	0.0303	0.0483	0.0521	0.1531
with delta rule	0.0054	0.0081	0.0080	0.1353

operations would not be able to produce the changes in phenotypes caused by learning. Table II also demonstrates the superiority of Lamarckian learning over Baldwinian learning, suggesting that Baldwinian learning may not be appropriate for training RNN's.

To verify the benefit of Lamarckian learning, let us increase the complexity of the long-term dependency problem—the temporal length is doubled to ten time steps. The RNN to be trained has four input nodes and 16 processing nodes, where six of them were dedicated as output nodes.[6] Therefore, there are totally $16 \times 16 + 16 \times 5 = 336$ weights. In our experimental work, we used a population size of 1600 instead of 100, but other parameters remained unchanged. As the problem is more difficult, a large population size is required to increase the chance of finding an acceptable solution. However, a large population size also increases the computation time significantly. This is a typical problem in GA's. Fig. 8 illustrates that combining cellular GA's with the delta rule achieve a better convergence as compared to the pure cellular GA despite the large number of weights.

Our conjecture for Baldwinian learning stated in Section IV-A suggests that if many weights are changed by Baldwinian learning and the changes are large, the hybrid algorithms will not be better than the pure cellular GA. This is because the search space is too large for genetic operations to produce the correct genotype associated with the "learned" fitness. The validity of the conjecture has also been justified in our recent report [26] where further evidence is provided. It is interesting to point out that the results of a recent independent study performed by Mayley [31] also support our conjecture. Mayley [31] suggested that to get the maximum benefit out of the Baldwin effect, the phenotypic distance between two phenotypes has to be correlated with the genotypic distance between the corresponding genotypes. The phenotypic distance is measured by the "ease" of transforming the "inborn" phenotype to the "learned" phenotype by learning, while the genotypic distance is measured by the expected number of genetic operations required to achieve the corresponding transformation in genotype space. When there are many changeable weights, the correlation between the genotypic distance and the phenotypic distance becomes small. As a result, the advantage of Baldwinian learning is lost.

This study found that combining cellular GA's and Lamarckian learning is a promising approach. For a learning method to be efficient, the learning process must not spend too much computation time as compared to the reproduction process so that a net gain could be obtained. Therefore, the criteria for a good learning method are: 1) it should be simple so that computation time taken is short and 2) it should have the capability of moving toward on obtaining a better solution in each learning cycle. Obviously, these two criteria are contradictory. One should choose an algorithm that strikes a balance between these two criteria, although it may be difficult to decide which criterion is more important. Comparing to the delta rule, the restricted RTRL might be more capable of improving the fitness; but we have found that its computation time is 40% longer. In the long-term dependency problem, it is the hybrid algorithm that uses the delta rule has better convergence. However, it is possible in other problems that the restricted RTRL is more capable of improving the fitness than the delta rule, and this improvement could be so significant that it can compensate for the cost of longer computation time. In

[6]Note that the number of input–output pattern pairs increases with the temporal length. In order to represent the additional patterns, we increased the number of input nodes as well as the number of output nodes. We found that an RNN with 16 processing nodes was sufficient.

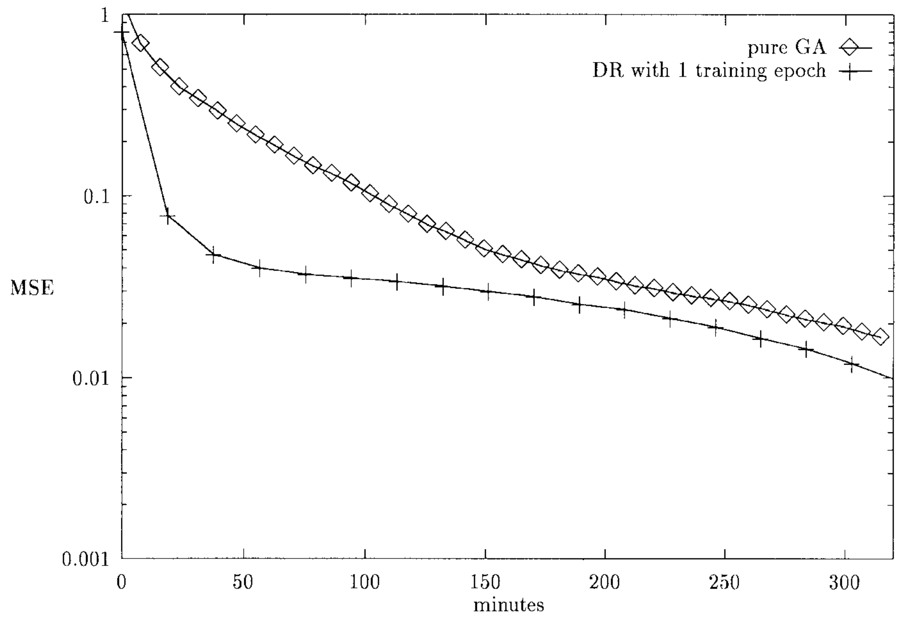


Fig. 8. Average MSE's (based on the average of 30 simulations) achieved by embedding the delta rule in the cellular GA using Lamarckian learning for solving the long-term dependency problem with a temporal length of ten time steps.

this case, the hybrid algorithm that uses the restricted RTRL may converge better.

There is a dilemma in adding learning to GAs: the more frequent and the larger extent (e.g., more epochs for each learning cycle) we apply learning, the more improvement can be achieved in each generation; however, this can only be achieved at the expense of more computation time. As a result, these parameters have to be chosen carefully such that the combination of learning and GA's is better (in terms of computation time) than the pure GA's. The selection of these parameters may not be difficult. This is because in our experiments, the convergence of combining cellular GA's and the delta rule is better than the pure cellular GA even for the simplest case in which minimum amount of learning (one epoch per learning cycle) is applied.

VIII. Conclusions

This study has found that embedding simple learning methods in the cellular GA using the Lamarckian mechanism can improve the prediction and classification capability of RNN's. This suggests that the learning methods need not be sophisticated in order to get the benefit of combining GA's and learning. It is commonly believed that using GA's to train RNN's is a slow approach. However, our study suggests a way to speed up and to improve the accuracy of the training process. Our experiments also show that Baldwinian learning cannot be better than Lamarckian learning. We postulate that Baldwinian learning is not suitable for evolving RNN's, especially when the learning method can change a large number of weights in the networks and the changes are too large for genetic operations to cope with. Our findings are based on the experimental results obtained by embedding various learning methods in the cellular GA. The resulting hybrid algorithms were used to train the RNN's in order to solve the long-term dependency problem. Further investigations are required to see whether this approach will be successful in other problems, and to provide a more critical comparison between Lamarckian learning and Baldwinian learning.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

[1] D. H. Ackley and M. L. Littman, "Interactions between learning and evolution," in C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds., *Artificial Life 2*. Reading, MA: Addison-Wesley, 1992, pp. 487–509.

[2] ——, "A case for Lamarckian evolution," in C. G. Langton, Ed., *Artificial Life 3*. Reading, MA: Addison-Wesley, 1994, pp. 3–10.

[3] R. W. Anderson, "Learning and evolution: A quantitative genetics approach," *J. Theoretical Biol.*, vol. 175, pp. 89–101, 1995.

[4] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 54–65, 1994.

[5] J. M. Baldwin, "A new factor in evolution," *Amer. Naturalist*, vol. 30, pp. 441–451, 1896.

[6] R. K. Belew, "Evolution, learning, and culture: Computational metaphors for adaptive algorithms," *Complex Syst.*, vol. 4, pp. 11–49, 1990.

[7] Y. Bengio, Y. LeCun, C. Nohl, and C. Burges, "Lerec: A NN/HMM hybrid for on-line handwriting recognition," *Neural Comput.*, vol. 7, pp. 1289–1303, 1995.

[8] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, pp. 157–166, 1994.

[9] H. Braun and P. Zagorski, "ENZO-M—A hybrid approach for optimizing neural networks by evolution and learning," in Y. Davidor, H.-P. Schwefel, and R. Manner, Eds., *Parallel Problem Solving from Nature—PPSN III*, Springer-Verlag, pp. 440–451, 1994.

[10] R. J. Collins and D. R. Jefferson, "Selection in massively parallel genetic algorithms," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 249–256.

[11] Y. Davidor, "A naturally occuring niche & species phenomenon: The model and first results," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 257–262.

[12] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1995.

[13] D. B. Fogel and J. W. Atmar, "Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems," *Biol. Cybern.*, vol. 63, pp. 111–114, 1990.

[14] D. B. Fogel and L. C. Stayton, "On the effectiveness of crossover in simulated evolutionary optimization," *BioSyst.*, vol. 32, no. 3, pp. 171–182, 1994.

[15] R. M. French and A. Messinger, "Genes, phenes and the Baldwin effect: Learning and evolution in a simulated population," in A. B. Rodeny and M. Pattie, Eds., *Artificial Life 4*. Cambridge, MA: MIT Press, 1994, pp. 277–282.

[16] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[17] F. Gruau and D. Whitley, "Adding learing to the cellular development of neural networks: Evolution and the Baldwin effect," *Evolutionary Comput.*, vol. 1, no. 3, pp. 213–233, 1993.

[18] W. E. Hart, "Adaptive global optimization with local search," Ph.D. dissertation, Dept. Comput. Sci. Eng., Univ. California, San Diego, 1994.

[19] W. E. Hart, T. E. Kammeyer, and R. K. Belew, "The role of development in genetic algorithms," in L. D. Whitley and M. D. Vose, Eds., *Foundations of Genetic Algorithms 3*. San Mateo, CA: Morgan Kaufmann, 1995, pp. 315–332.

[20] H. Hattori, "Text-independent speaker recognition using neural networks," in *Proc. ICASSP*, 1992, pp. 153–156.

[21] G. E. Hinton and S. J. Nowlan, "How learning can guide evolution," *Comput. Syst.*, vol. 1, pp. 495–502, 1987.

[22] R. Keesing and D. G. Stork, "Evolution and learning in neural networks: The number and distribution of learning trial affect the rate of evolution," in R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds., *Advances in Neural Information Processing Systems 3*. San Mateo, CA: Morgan Kaufmann, 1991, pp. 804–810.

[23] H. Kitano, "Empirical studies on the speed of convergence of neural network training using genetic algorithms," in *Proc. 8th Nat. Conf. Artificial Intell.*, 1990, pp. 789–795.

[24] P. G. Korning, "Training neural networks by means of genetic algorithms working on very long chromosomes," *Int. J. Neural Syst.*, vol. 6, no. 3, pp. 299–316, 1995.

[25] K. W. C. Ku and M. W. Kak, "Exploring the effects of Lamarckian and Baldwinian learing in evolving recurrent neural networks," in *Proc. IEEE Int. Conf. Evolutionary Comput.*, 1997, pp. 617–621.

[26] ——, "Empirical analysis of the factors that affect the Baldwin effect," in *Parallel Problem Solving from Nature—PPSN V.*. New York: Springer-Verlag, 1998.

[27] K. W. C. Ku, M. W. Mak, and W. C. Siu, "A cellular genetic algorithm for training recurrent neural networks," in *Proc. Int. Conf. Neural Networks Signal Processing*, 1995, pp. 140–143.

[28] T. Lin, B. G. Horne, P. Tiňo, and C. L. Giles, "Learning long-term dependencies in NARX recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 7, pp. 1329–1338, 1996.

[29] B. Manderick and P. Spiessens, "Fine-grained parallel genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 428–433.

[30] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 39–53, 1994.

[31] G. Mayley, "Landscapes, learning costs, and genetic assimilation," *Evolutionary Comput.*, vol. 4, no. 3, pp. 213–234, 1997.

[32] J. R. McDonnel and D. Waagen, "Evolving recurrent perceptrons for time-series modeling," *IEEE Trans. Neural Networks*, vol. 5, pp. 24–28, 1994.

[33] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1996.

[34] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.

[35] D. J. Montana and L. Davis, "Training feedforward neural network using genetic algorithms," in *Proc. 11th Int. Joint Conf. Artificial Intell.*, 1989, pp. 762–767.

[36] M. C. Mozer, "A focus backpropagation algorithm for temporal pattern recognition," *Complex Syst.*, vol. 3, pp. 349–381, 1989.

[37] S. Nolfi, J. L. Elman, and D. Parisi, "Learning and evolution in neural networks," *Adaptive Behavior*, vol. 3, pp. 5–28, 1994.

[38] B. A. Pearlmutter, "Learning state-space trajectories in recurrent neural networks," *Neural Comput.*, vol. 1, pp. 263–269, 1989.

[39] D. Ron, Y. Singer, and N. Tishby, "The power of amnesia," in J. Cowan, G. Tesauro, and J. Alspector, Eds., *Advances in Neural Information Processing Systems 6*. San Mateo, CA: Morgan Kaufmann, 1994, pp. 176–183.

[40] J. D. Schaffer and L. J. Eshleman, "On crossover as an evolutionary viable strategy," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 61–68.

[41] A. J. Skinner and J. Q. Broughton, "Neural networks in computational materials science: Training algorithms," *Modeling Simulation Materials Sci. Eng.*, vol. 3, pp. 371–389, 1995.

[42] F. J. Solis and R. J.-B. Wets, "Minimization by random search techniques," *Math. Operations Res.*, vol. 6, no. 1, pp. 19–30, 1981.

[43] P. Turney, "Myths and legends of the Baldwin effect," in *Proc. Workshop Evolutionary Comput. Machine Learning 13th Int. Conf. Machine Learning*, 1996, pp. 135–142.

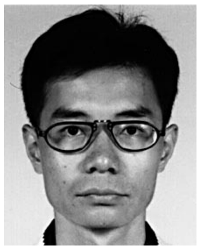
[44] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, pp. 1550–1560, 1990.

[45] D. Whitley, "A genetic algorithm tutorial," *Statist. Comput.*, vol. 4, no. 2, pp. 65–85, 1994.

[46] ——, "Genetic algorithms and neural networks," in G. Winter, J. Periaux, M. Galan, and P. Cuesta, Eds., *Genetic Algorithms Engineering and Computer Science*. New York: Wiley, 1995, pp. 191–201.

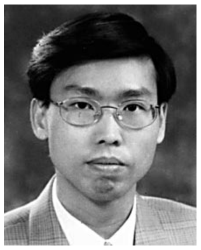
[47] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization," in Y. Davidor, H.-P. Schwefel, and R. Manner, Eds., *Parallel Problem Solving from Nature—PPSN III*, Springer-Verlag, 1994, pp. 6–15.

[48] R. J. Williams and D. Zipser, "Experimental analysis of the real-time recurrent learning algorithm," *Connection Sci.*, vol. 1, pp. 87–111, 1989.



Kim-Wing C. Ku (S'98) received the B.Sc.(Hons) degree in computer studies from the City Polytechnic of Hong Kong in 1993 and the M.Sc. degree in information technology (knowledge-based systems) from the University of Edinburgh, U.K., in 1994. Presently, he is a Ph.D. student at the Department of Electronic and Information Engineering of the Hong Kong Polytechnic University.

His research interests inlude recurrent neural networks and evolutionary algorithms.



Man-Wai Mak (S'91–M'93) received the B.Eng.(Hons) degree in electronic engineering from Newcastle Upon Tyne Polytechnic, U.K., in 1989 and the Ph.D. degree in electronic engineering from the University of Northumbria at Newcastle, U.K., in 1993.

He was a Research Assistant at the University of Northumbria at Newcastle from 1990 to 1993. He joined the Department of Electronic Engineering at the Hong Kong Polytechnic University as a Lecturer in 1993 and as an Assistant Professor in 1995. His research interests include recurrent neural networks, genetic algorithms, and speaker recognition.

Since 1995, Dr. Mak has been an executive committee member of the IEEE Hong Kong Section Computer Chapter. He is currently the Secretary of the IEEE Hong Kong Section Computer Chapter.

Wan-Chi Siu (S'77–M'77–SM'90) received the Associateship degree from the Hong Kong Polytechnic University (formerly called Hong Kong Polytechnic), the M.Phil. degree from the Chinese University of Hong Kong, and the Ph.D. degree from the Imperial College of Science, Technology & Medicine, London, U.K., in 1975, 1977, and 1984, respectively.

He was with the Chinese University of Hong Kong between 1975 and 1980. He then joined the Hong Kong Polytechnic University as a Lecturer in 1980 and became Chair Professor and Associate Dean of Engineering Faculty in 1992. He has been Chair Professor and Head of Department of Electronic Engineering of the same university since 1994. He has published more than 180 research papers. His research interests include digital signal processing, fast computational algorithms, transforms, video coding, computational aspects of image processing and pattern recognition, and neural networks.

Dr. Siu is a Member of the Editorial Board of the JOURNAL OF VLSI SIGNAL PROCESSING SYSTEMS for SIGNAL, IMAGE AND VIDEO TECHNOLOGY, and an overseas member of the Editorial Board of the *IEE Review*. He is also a Guest Editor of the Special Issue on ISCAS'97 of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, Pt. II, published in May 1998. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, Pt. II from 1995 to 1997. He was the General Chairman of the International Symposium on Neural Networks, Image and Speech Processing (ISSIPNN'94), and a Cochair of the Technical Program Committee of the IEEE International Symposium on Circuits and Systems (ISCAS'97) which were held in Hong Kong in April 1994 and June 1997, respectively. He is now also chairing the tentative organizing committee of the 2003 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) which is to be held in Hong Kong. From 1991 to 1995, he was a member of the Physical Sciences and Engineering Panel of the Research Grants Council (RGC), Hong Kong Government, and in 1994 he chaired the first Engineering and Information Technology Panel to assess the research quality of 19 Cost Centers (departments) from all universities in Hong Kong. He is a Chartered Engineer, a Fellow of the Institute of Electrical Engineers and the HKIE, and has also been listed in *Marquis Who's Who in the World*, *Marquis Who's Who in Science and Engineering*, and other citation biographies.