01 Jun 2009

Effects of Learning Rate on the Performance of the Population Based Incremental Learning Algorithm

Ganesh K. Venayagamoorthy
Missouri University of Science and Technology

K. A. Folly

Recommended Citation

G. K. Venayagamoorthy and K. A. Folly, "Effects of Learning Rate on the Performance of the Population Based Incremental Learning Algorithm," *Proceedings of the International Joint Conference on Neural Networks, 2009. IJCNN 2009*, Institute of Electrical and Electronics Engineers (IEEE), Jun 2009.
The definitive version is available at https://doi.org/10.1109/IJCNN.2009.5179080

Effects of Learning Rate on the Performance of the Population Based Incremental Learning Algorithm

Komla A. Folly and Ganesh K. Venayagamoorthy

Abstract— **The effect of learning rate (LR) on the performance of a newly introduced evolutionary algorithm called population-based incremental learning (PBIL) is investigated in this paper. PBIL is a technique that combines a simple genetic algorithm (GA) with competitive learning (CL). Although CL is often studied in the context of artificial neural networks (ANNs), it plays a vital role in PBIL in that the idea of creating a prototype vector in learning vector quantization (LVQ) is central to PBIL. In PBIL, the crossover operator of GAs is abstracted away and the role of population is redefined. PBIL maintains a real-valued probability vector (PV) or prototype vector from which solutions are generated. The probability vector controls the random bitstrings generated by PBIL and is used to create other individuals through learning. The setting of the learning rate (LR) can greatly affect the performance of PBIL. However, the effect of the learning rate in PBIL is not yet fully understood. In this paper, PBIL is used to design power system stabilizers (PSSs) for a multi-machine power system. Four cases studies with different learning rate patterns are investigated. These include fixed LR; purely adaptive LR; fixed LR followed by adaptive LR; and adaptive LR followed by fixed LR. It is shown that a smaller learning rate leads to more exploration of the algorithm which introduces more diversity in the population at the cost of slower convergence. On the other hand, a higher learning rate means more exploitation of the algorithm and hence, this could lead to a premature convergence in the case of fixed LR. Therefore, in setting the LR, a trade-off is needed between exploitation and exploration.**

I. Introduction

THE past decade has witnessed a flurry of interest in the application of Genetic Algorithms (GAs) to design power system controllers [1]- [2]. Genetic algorithms are randomized parallel search method modeled on natural selection [3]-[4]. GAs have recently seen extensive applications in solving global optimization problems [4]. They operate on a population of potential solutions (chromosomes) applying a sequence of operators to the population based on the relative fitness of the members [5]. The operators typically involve selection, crossover, and mutation. The goal of the evolutionary process is to continually improve the fitness of the best solution vector, as well as the average population fitness, until some termination criteria is satisfied. One important feature of GAs is their implicit parallelism, i.e., the ability to search the function space from multiple points in parallel [3]-[5]. Therefore, GAs are more likely to locate the global optima than traditional techniques, because they are much less likely to get stuck at the local optima [5]. However, GAs have several shortcomings. For example, the convergence of a GA is usually slower than traditional optimization techniques. Furthermore, the problem of genetic drift can lead to the lost of diversity in the population. Once the diversity is lost, the crossover operator becomes ineffective in exploring the search space. Although mutation can be used to introduce diversity in the population, its effect is limited.

To cope with GAs' limitations, several researchers have recently proposed a family of new algorithms called Estimation of Distribution Algorithms (EDA) [6]-[9]. Like GAs, EDA work with a population of individuals. However, one of the important features of EDA is that they avoid the 'blindness' of crossover by finding how the problem space distributes and use this information to guide individuals to explore better space areas during the search. One of the algorithms that belong to the family of EDA is the so called Population-Based Incremental Learning (PBIL) which was originally proposed by Baluja [7]-[8]. PBIL is a technique that combines simple GAs with competitive learning. In PBIL, the crossover operator of GAs is abstracted away and the role of population is redefined. PBIL maintains a real-valued probability vector from which solutions are generated. The probability vector controls the random bitstrings generated by PBIL and is used to create other individuals through learning. Learning in PBIL consists of using the current probability distribution to create N individuals. These individuals are evaluated according to the objective function. The best individual is used to update the probability vector, increasing the probability of producing solutions similar to the current best individuals.

Many authors have shown the effectiveness of PBIL in solving many difficult optimization problems [6]-[12]. One of the key aspects of PBIL is the learning. The learning process in PBIL ensures that individuals can adapt easily to a new environment. In PBIL, the learning rate plays a crucial role in finding the optimal solution of a problem. However, the effect of the learning rate in PBIL is not well understood.

Manuscript received January 15, 2008. This work was supported in part by the TESP, the THRIP and the NSF EFRI grants.

K. A. Folly is with the University of Cape Town, Department of Electrical Engineering, Private Bag, Rondebosch 7701, Cape Town, SA phone: 27-21-650-4490; fax: 27-21-650-3465; (e-mail: Komla.Folly@uct.ac.za.). He is currently on Sabbatical leave and is Fulbright Scholar with the Real-Time Power and Intelligent Systems Laboratory, Missouri University of Science and Technology, Rolla, MO 65409, USA.

G. K. Venayagamoorthy is with the Real-Time Power and Intelligent Systems Laboratory, Missouri University of Science and Technology, Rolla, MO 65409, USA, phone: 1-573-341-6641; fax: 1-573-341-4532; (e-mail: gkumar@ieee.org.)

978-1-4244-3553-1/09/$25.00 ©2009 IEEE

The effect of the learning rate (LR) on the performance of a PBIL used to design power system stabilizers (PSSs) for a multi-machine power system is investigated in this paper. Four types of learning rates are investigated. These includes the fixed learning rate where the learning rate is fixed and does not change during the run; the purely adaptive learning rate where the learning rate varies with the generations, and a combination of fixed and adaptive learning rate where the learning rate is fixed for the first portion of generations and becomes adaptive for the other portion and vice-versa. It is shown that a smaller learning rate leads to more exploration of the algorithm which introduces more diversity in the population at the cost of slower convergence. On the other hand, a higher learning rate means more exploitation of the algorithm and hence, this could lead to a premature convergence in the case of fixed learning rate. Therefore, a trade-off is needed between exploitation and exploration when setting the learning rate.

II. Competitive Learning in Ann

Adaptation involves a progressive modification of some structure or structures [1]. Without adaptation, no human or animal species can survive. In PBIL, adaptation is provided to the evolving chromosomes through competitive learning [6]. Competitive learning (CL) is often studied in the context of Artificial Neural Networks (ANNs) [7]. A common goal in competitive learning is to distribute a certain number of vectors in a possibly high-dimensional space. Competitive learning is often used to cluster a number of unlabeled data into distinct groups. The objective is to group the data such that the inputs in the same cluster are in some sense similar [13]. A basic competitive learning network is shown in Figure 1. It has one layer of input neurons and one layer of output neurons. It consists of the feedforward excitatory network(s) and the lateral inhibitory network(s). There are as many output nodes as there are classes and each output node represents a pattern category.

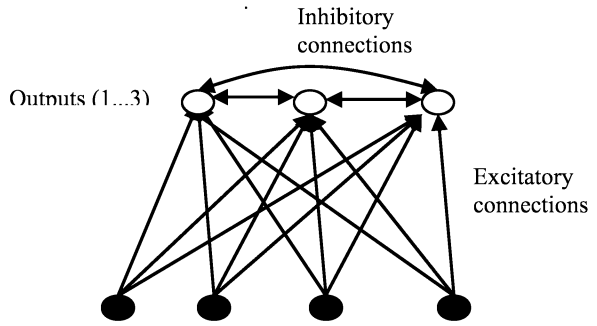

Fig. 1 A basic competitive learning network

The inhibitory between output units ensure that only one output is turned on at a time. The output unit that is turned on is the one which has the largest net input. The feedforward network usually implements an excitatory Hebbian learning rule. This consists of increasing the influence of the input cell that persistently participates in firing an output cell [13]. An example of supervised competitive learning is Learning Vector Quantization (LVQ) proposed in [14]. In learning vector quantization, it is assumed that there is a set of reference vectors represented by the number of weights w_{ij} connecting input i to output j. These weights are initially chosen randomly by the user. The activation of the output units is calculated by the following formula [7]

$$output_j = \sum_i (w_{ij}) \times input_i \qquad (1)$$

Since the output units compete with one another to turn on, they are called "winner-take all" units. During training, the weights of the winning output unit are moved closer to the presented point by adjusting the weights according to the following rule [7]

$$\Delta w_{ij} = LR \times (input_i - w_{ij}) \qquad (2)$$

where:
LR: the learning rate parameter
Δ: small variation

After the training, the weights are considered as prototype vectors. The idea of creating a prototype vector is central to PBIL as will be discussed in the next section.

III. Overview of Population-Based Incremental Learning

PBIL is a technique that combines aspects of genetic algorithms and simple competitive learning from ANN. PBIL has the following features [9]-[12]:

- It has no crossover and fitness proportional operators.
- Instead of representing the entire genetic population using myriads of chromosomes, the population is represented through a probability vector (number in range 0-1). The probability vector can be considered as a prototype for high valuation vectors for the function space being explored. This probability vector controls the random bitstrings generated by PBIL and is used to create other individuals through learning.
- In PBIL, there is no need to store all solutions in the population. Only two solutions are stored: the current best solution and the solution being evaluated.

The three main operators of PBIL used in this paper are: probability vector (PV), Learning rate (LR) and the mutation (i.e., forgetting factor, FF).

Each probability value in the sequence represents the probability that 1 or 0 can be generated at the gene position. The learning rate is used in the updating rule of the probability vector. It affects both the speed at which the probability vector is shifted to resemble the best solution vector and the portion of the search space that will be explored. Like in GAs, the mutation is used to maintain the diversity in the population. In [9]-[10], Baluja employed a "mutation" operator similar to that used in GAs. In this paper, a slightly different "mutation" operator is used. That is, a forgetting factor is used to relax the probability vector toward neutral value of 0.5 [11]-[12].

A summary of the PBIL algorithm used in the paper is given below [9]-[12], [15]-[16]:

Step 1. Initialize elements of the probability vector (PV) to 0.5 to ensure uniformly-random bitstrings.
Step 2. Generate a population (i.e., sample solutions) of uniformly-random bitstrings and comparing it element-by-element with the PV. Wherever an element of the PV is greater than the corresponding random element, a "1' is generated, otherwise a '0' is generated.
Step 3. Interpret each bitstring as a solution to the problem and evaluate its merit in order to identify the "Best".
Step 4. Adjust *PV* by slightly increasing *PV* (*i*) to favor the generation of bitstrings which resemble "Best", if Best (*i*) = 1 and decrease PV(*i*) if Best(*i*) = 0.
Step 5. Perform mutation on the probability vector PV
Step 6. Generate a new population reflecting the modified distribution. Stop if satisfactory solution is found. Otherwise, go to step 2.

This algorithm is much easier to implement than the conventional GAs (which involve crossovers, mutations, reproductions, etc.), and yet effective. It has been shown that PBIL outperforms standard GAs approaches on a variety of optimization problems including commonly used benchmark problems [7]-[9]. Experience in executing GAs and PBIL shows that the overhead for GA operations is significantly higher than for PBIL [12].

IV. Operators of PBIL

A. Probability Vector (PV)

One important feature of GAs is their implicit parallelism, i.e., the ability to search the function space from multiple points in parallel [7]. However, as the search progresses, this parallelism is not easily maintained in the latter generations of GAs. Therefore, the idea behind PBIL is to represent the entire genetic population through a probability vector rather than a myriad of chromosomes. This probability vector should be considered as a prototype for high evaluation vectors for the function space being explored [7]-[8]. This concept is central to PBIL. Each probability value in the sequence represents the probability that a 1 or 0 can be generated at the gene position. For example, the probability vector [0.5, 0.5, 1, 1] can be represented by the following population of 4 bits

0, 0, 1, 1
0, 1, 1, 1
1, 0, 1, 1
1, 1, 1, 1

Note that the size of the above population is 4.

The probability that a 1 or 0 will be generated in the first two positions is equal (i.e., 50/50). The probability of generating 1 in the 3rd and 4th positions is 1.

Unlike the mechanisms inherent to GAs, where operations are defined on the population; in BPIL, the operations take place directly on the probability vector. During the search, the values in the probability vector are updated to represent those in high evaluation vectors. It should also be noted that besides from specifying the prototype based upon the high evaluations of the sample solutions, the probability vector also guides the search, which produces the next sample point from which learning take place [7].

Initially, the values of the probability vector are set to 0.5 to ensure that the probability of generating 0 or 1 is equal. As the search progresses, the values in the probability vector move away from 0.5, towards either 0.0 or 1.0.

It has been argued that because PBIL uses a single probability vector, it may be less powerful than GAs because a large number of points cannot be represented simultaneously. However, this argument is only true at the beginning of the search space. Because of the sampling errors, the population will converge to one point at the latter portion of the search and GA will not be able to maintain multiple dissimilar points [7]-[8].

B. Learning Rate (LR)

The probability update rule is similar to the weight update rule in a competitive learning of ANN as given in (2). The following probability update rule based on the competitive learning is used:

$$PV(i) = PV(i) \times (1.0 - LR) + (LR \times V(i)) \qquad (3)$$

where
PV(i): the probability of generating 1 in bit position *i*.
V(i): the *i*-th position in the solution vector towards which the probability vector is moved.

The learning rate has a greater effect in PBIL as compared to the standard competitive learning. This is because the probability vector is used to generate future sample solutions. Like in competitive learning, the learning rate affects the speed at which the probability vector is shifted to resemble the best solution vector. It also affects the portion

of the search space that will be explored. Therefore, it has a direct impact on the trade-off between exploration (i.e., the ability of the algorithm to search the function space thoroughly) and exploitation (i.e., the ability of the algorithm to use the information it has gained about the function space to narrow its future search) of the search space [7].

As shown later in the simulations, a small learning rate means less exploitation of the information gained through previous search and more exploration of the search space to search for diverse and possibly better solutions. As the learning rate is increased, the amount of exploitation increases, and the ability to search large portions of the search space diminishes. That is, the exploration capability becomes less.

Two types of learning are used in PBIL algorithm[7]-[8]. The positive learning which was discussed previously, and the negative learning. For the negative learning, the probability vector is shifted away from the worst vector. In this paper, only positive learning is used.

C. *Mutation Operator: Forgetting factor (FF)*

To maintain diversity in PBIL, a "mutation" operator similar to that used in GAs was employed by Baluja [7]-[8]. In this paper, a slightly different "mutation" operator is used. That is, a forgetting factor is used to relax the probability vector toward neutral value of 0.5 [9]-[11]. There are two methods that can be used to perform mutation. The first method is to perform mutation directly on the sample vectors generated (i.e., population). The second method is to perform mutation on the probability vector. The PBIL used in this paper adopted the latter method. The formula used for the implementation of the mutation is given by

$$PV(i) = PV(i) - FF \times (PV(i) - 0.5) \qquad (4)$$

where
FF: the forgetting factor

V. System Model and Operating Conditions

The power system model considered in this paper is a three-machine nine-bus power system as shown in Fig. 2 [15]. Each machine is represented by the two-axis model (fourth order). The machines are equipped with a simple AVR, which is modeled by a second order differential equation [11]. The dynamics of the system are described by a set of nonlinear differential equations. However, for the purpose of controller design these equations are linearized around the nominal operating conditions.

The linearized equation of the system is given by [2]-[3]

$$\begin{aligned} \dot{x} &= A_o x + B_o u \\ y &= C_o x + D_o u \end{aligned} \qquad (5)$$

where the state variables are x, the system output is y and the signal u represents the control input. A_o, B_o, C_o, D_o are constant matrices of appropriate dimensions.

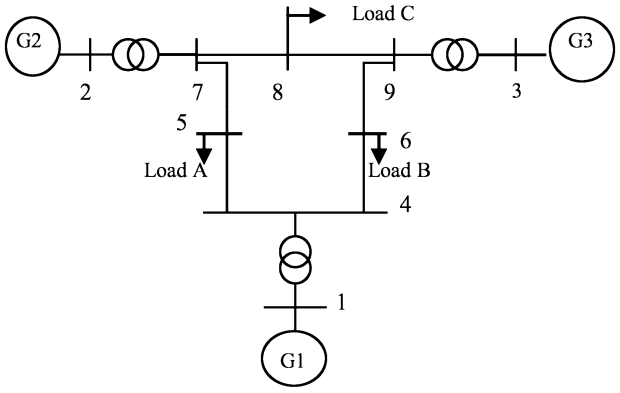

Fig. 2 Power system configuration

Several operating conditions have been considered during the design of the PSSs, this includes, the nominal operating condition, light and heavy load conditions. In addition, the load conditions mentioned above are considered under weak transmission system. Note that weak transmission system in our contest means that the nominal transmission line data are doubled. In this paper, the PSSs are designed using fives operating conditions. The nominal operating condition (case 1), the light load condition under nominal transmission line system (case 2), the heavy load condition under nominal transmission line system (case 3), the light load condition under weak transmission line (case 4) and the heavy load condition under weak transmission system (case 5).

These cases are listed in Table I (generation) and Table II (loads). The eigenvalues of the open-loop nominal system (without PSS) are listed in Table III.

TABLE I
Possible Operating Conditions

Gen. No.	G 1		G2		G 3	
Case	P_e	Q_e	P_e	Q_e	P_e	Q_e
1	0.716	0.321	1.630	-0.001	0.850	-0.118
2	0.505	-0.005	1.100	-0.241	0.300	-0.307
3	2.115	0.877	1.900	0.392	1.240	0.281
4	0.523	-0.674	1.100	-0.637	0.300	0.523
5	2.193	0.500	1.900	0.123	1.240	-0.086

All the values are given in per-unit

TABLE II
LOAD PARAMETERS

Load.	A		B		C	
Case	P_L	Q_L	P_L	Q_L	P_L	Q_L
1	1.250	0.500	0.900	0.300	1.000	0.350
2 & 4	0.750	0.300	0.540	0.180	0.600	0.210
3 & 5	1.750	0700	1.620	0.540	1.800	0.630

All the values are given in per-unit.

TABLE III
OPEN-LOOP EIGENVALUES (WITHOUT PSS)

Case	λ	ζ (%)
1	-0.345±*j*9.787	3.5
	-0.440±*j*13.822	3.2
2	-1.006±*j*9.620	10.4
	-1.786±*j*13.856	12.8
3	-0.277±*j*9.544	2.9
	-0.053±*j*13.621	0.4
4	-0.368±*j*8.691	4.2
	-0.355±*j*12.523	2.8
5	-0.053±*j*8.317	0.6
	+0.053±*j*12.329	-4.3

It can be seen from Table III that the system has two electromechanical oscillation modes [16]-[17]. It should be mentioned that these oscillation modes change as the operating conditions are varied. The effects of learning rate on the performance of the PBIL-PSSs eigenvalues for the nominal operating condition are illustrated in the simulation results section.

VI. PROBLEM FORMULATION

The objective of the study is to optimize the parameters of the PSSs such that controllers simultaneously stabilize the family of the power system models as described previously. It was found that a double stage lead-lag network with time constants T_1-T_4 and gain K_p is sufficient to provide adequate damping to the multi-machine system shown in Fig.2 [11]. The speed input PBIL-PSS is of the form given in (6)

$$K(s) = K_p \left(\frac{T_w s}{1 + T_w s} \right) \left(\frac{1 + T_1 s}{1 + T_2 s} \right) \left(\frac{1 + T_3 s}{1 + T_4 s} \right) \qquad (6)$$

where, K_p is the gain, T_1-T_4 represent suitable time constants. T_w is the washout time constant needed to prevent steady-state offset of the voltage.

Since the electromechanical modes are generally poorly damped and dominate the time response of the system, it is expected that by maximizing the minimum damping ratio over the entire family of the system models, the closed-loop systems could be simultaneously stabilized over a wide range of operating conditions [11],[15]. The following objective function is used in PBIL to achieve the above requirements:

$$J = \max\left(\min\left(\zeta_{i,j} \right) \right) \qquad (7)$$

where $i = 1, 2 \ldots n$, and $j = 1, 2, \ldots m$

and $\zeta_{i,j} = \dfrac{-\sigma_{i,j}}{\sqrt{\sigma_{i,j}^{\,2} + \omega_{i,j}^{\,2}}}$ is the damping ratio of the i-th eigenvalue in the j-th operating condition. σ_{ij} is the real part of the eigenvalue and the ω_{ij} is the frequency. n denotes the total number eigenvalues and m denotes the number of operating conditions.

VII. PSS DESIGN FOR A MULTIMACHINE POWER SYSTEM

There are in total 15 PSS parameters (five for each generator) that need to be optimized. It should be noted that the reset time constant T_w shown in (2) was not considered in the optimization process. This is because T_w is not critical. Its value was fixed to 10 sec.

The configuration of the PBIL is as follows:

- Length of chromosome: 15 bits
- Population: 10
- Generations: 200
- Learning rate (LR): 0.1 (default)
- Forgetting factor (mutation): 0.005

The parameter domain for the PBIL-PSS was set to:

$$0 \le K_p \le 20$$
$$0 \le T_1, T_3 \le 1$$
$$0.010 \le T_2, T_4 \le 0.5$$

A modified MATLAB software [9], [11] was used for the design.

The challenge in PSS design problem (as opposed to other problems) is to guaranty adequate stability of the closed-loop system despite uncertainties in the system (parameter variations, load change, faults, etc). This task is complicated by the fact that power systems are nonlinear, complex, and highly multivariable. Therefore, the search space is typically multi-modal and uncertain. There is a potential danger that the algorithm will be stuck at suboptimal solutions. The controller is designed using a linear model but it expected to give good performance under nonlinear conditions.

VIII. SIMULATION RESULTS

A. Test cases

Fixed LR: The learning rates considered are: LR=0.01, LR =0.0125, LR = 0.167, LR=0.05, LR= 0.1 (default), LR = 0.2, LR = 0.4, LR = 0.6, LR=0.8 and LR =1.

Purely Adaptive LR: The simulation is started with a very small value of LR (LR≈ 0), and this value is increased linearly with the generation until the final value of LR is reached. For example, if the final value learning rate is LR = 0.2, the run is started with LR≈ 0, and after 50 generation, LR = 0.05, after 100 generations, LR = 0.1 and so on.

Combined Fixed LR and Adaptive LR (Adaptive to Fixed LR and Fixed to Adaptive LR) :

- Adaptive to Fixed LR

  For the first 100 generations, adaptive learning rate is used until LR = 0.1 and for the second half of the generations (100 generations), fixed learning rate of LR =0.1 is used.

- Fixed to Adaptive LR:

  In the first half of the generations (100 generations), a fixed learning rate of LR = 0.1 is used and an adaptive learning rate is used for the second half of the generation.

B. Convergence rate

Figs. 3-6 show the convergence rate of the average of the best fitness values over 50 trials for different learning rates.

Fig. 3 shows the average of fitness values over 50 trials, for selected fixed learning rates (LR= 0.01, LR = 0.1, LR = 0.2, and LR = 0.8, LR = 1). It can be seen that LR = 1 gives consistently the lowest fitness value over the 200 generations. This is because of the premature convergence problem. By setting a high value of the learning rate, the ability of the algorithm to explore the search space is reduced. As a result, the algorithm converges to a local maximum instead of a global one. Setting LR to be small (i.e., LR = 0.01) means that we put much weight on exploration than exploitation. The algorithm will need more time to explore the search space and it will take longer time to converge to a good solution and more generations than the 200 generations used here. From our experience, we have observed that the larger the learning rate the quicker the convergence (i.e., less time is used) and the higher the possibility to converge to local maxima because of the loss of this diversity in the population. It can be said that diversity in the population is maintained longer if the learning rate is sufficiently small. Diversity is very critical for adaptation. If there is no diversity, natural selection cannot take place. On the other hand, unnecessary exploration of the search space when the optimal solution has already been found will waste valuable time. What is needed is a trade-off between exploitation and exploration. The results in Figure 3 show that LR = 0.1 provides the best trade-off followed by LR = 0.2.

Fig. 4 shows the average of fitness value over 50 trials, for adaptive learning rates (LR= 0.01, LR = 0.1, LR = 0.2, and LR = 0.8, LR = 1). It can be seen that LR = 1 is now among the best three learning rate to provide a good average fitness value. In fact, the fitness value of LR = 1.0 was the best for the first 80 generations. The reason why LR = 1.0 for adaptive LR is better than fixed LR is because of the adaption that was introduced. In fact the run was started with smaller learning rate, which mean the algorithm has time to explore the search space before exploiting it at the latter stage. As a result, this produces a better solution as compared to the fixed learning rate. On the other hand, LR = 0.01, consistently gives the worse fitness value starting from 50 generations up to 200 generations. In both the fixed and adaptive cases, LR = 0.01 did not perform well. The main reason for this is because there was virtually no exploitation with this learning rate and the algorithm was busy exploring the search space before the run was stopped. It can be seen that the final best fitness value comes from LR = 0.6. The second best is LR = 0.2. LR = 0.2 produces the fastest increase in fitness during the run. It seems from Fig. 4, that if we were to increase the number of generations, LR = 0.2 may eventually outperform LR = 0.6.

It can be seen from Fig. 5 (Adaptive to fixed LR) that for the first 100 generations, when the learning rate is adaptive, the fitness values are much more spread. This suggests that the different learning rates were exploring different locations of the search space. LR =1.0, consistently give the worst solution until about 150 generations, when it starts to improve. Starting from about 25 generations, LR= 0.2 gives the best fitness value and for the rest of the run.

Fig. 6 shows the average fitness curves when the learning rate was fixed to LR = 0.1 for the first 100 generations and then changes to adaptive in the second half of the generation. It can be seen that in contrast to Fig. 5 the fitness values for all the learning rate are very similar from the beginning until the end of the run. This suggests a lack of diversity in the population.

C. Fitness value

Tables IV and V show the learning rate and the average of the best fitness values for each learning rate under the various test cases.

It can be seen from Table IV that the fixed learning rate (column 2 of Table IV), LR = 0.1 gives the 'best' average value of the fitness, the second best is LR = 0.2. The worst value comes from LR = 1, followed by LR = 0.01. The worst performance for LR=1 arises because of a premature convergence problem. Diversity is lost earlier in the run. For LR= 0.01, the algorithm was still exploring the search space when it was stopped. So, it could not found a good solution.

This problem could be solved by increasing the number of generations.

For the adaptive learning rate (column 3 of Table IV), LR= 0.6 gives the 'best' average value of the fitness, the second best is LR = 0.4. The worst value comes from LR = 0.01, followed by LR = 0.0125. Again here, the worst performance for LR = 0.01 arise because the algorithm is still exploring the search space and has not yet converged. Compared to the fixed learning rate, the adaptive scheme performs poorly for LR < 0.2. However, for LR>0.2, the adaptive scheme outperforms the fixed scheme. In particular, for LR =1, which gave the worse fitness value for the fixed learning rate. Furthermore, all the fitness values for LR>0.2 all bigger than 0.15. This suggests that in the case of adaptive learning rate, premature convergence did not occur, this is because adaptation was included. Therefore, the algorithm was able to explore the search space at the beginning of the run before exploiting it at the later stage. As a result, this leads to a better solution compared to the fixed learning rate. The results suggest that for low learning rate (i.e., LR<0.2) it is better to use fixed learning rate. However, for higher learning rates, adaptive learning rate is preferred. The best fitness values are obtained for LR between 0.2 and 0.8.

Table V shows the learning rate and the average of the best fitness values over 50 trials when both fixed and adaptive learning rate are combined.

Column 2 of Table V shows the fitness values when adaptive learning rate is first used at the start of the run until 100 generations, after this a fixed learning rate (LR = 0.1) is used. It can be seen that the best average fitness value is obtained when LR = 0.2. As the learning rate is increasing from LR = 0.2 to LR = 0.8, the fitness value decreases, and increases again slightly at LR =1.0. Compared to the case where only fixed learning rate is used, the fitness values for this case are slightly higher. Compared to the case where purely adaptive learning rate is used, most of the fitness values are smaller except for LR =0.2.

Column 3 of the Table V shows the fitness values when fixed learning rate of LR = 0.1 is first used at the start of the run until 100 generations, and then it was switched to adaptive learning rate. It can be seen that all of the fitness values are higher than the case where adaptive learning rate is first used and then it was switched to fixed learning rate (except for LR = 0.2).

The best overall fitness value for the four test cases is provided by the adaptive learning rate of LR = 0.6 (0.1691) and the second best by LR = 0.4 (0.1659). The third best was provided by the fixed learning rate of LR =0.1 (0.1652).

TABLE IV
BEST FITNESS VALUES OVER 50 TRIALS (FIXED & ADAPTIVE)

LR	$\zeta_{avg\text{-}best}$ (%) - Fixed	$\zeta_{avg\text{-}best}$ (%) – Adaptive
0.0 l00	0.1071	0.0923
0.0125	0.1095	0.0920
0.0167	0.1151	0.0967
0.0250	0.1274	0.1069
0.0500	0.1489	0.1258
0.1000	0.1652	0.1443
0.2000	0.1626	0.1635
0.4000	0.1536	0.1659
0.6000	0.1247	0.1691
0.8000	0.1114	0.1637
1.0000	0.1007	0.1569

TABLE V
LEARNING RATE AND BEST FITNESS VALUES OVER 50 TRIALS

LR	$\zeta_{avg\text{-}best}$ (%) – Adaptive-Fixed	$\zeta_{avg\text{-}best}$ (%) – Fixed-Adaptive
0.2000	0.1641	0.1595
0.4000	0.1447	0.1528
0.6000	0.1430	0.1531
0.8000	0.1304	0.1536
1.0000	0.1339	0.1499

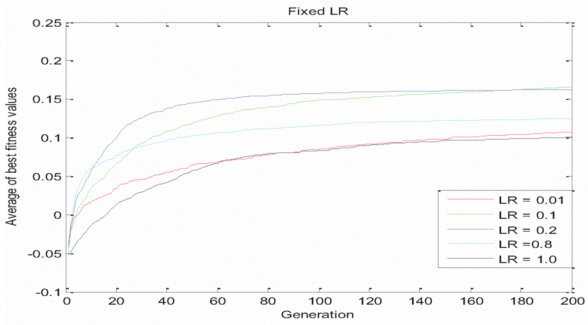

Fig. 3 Fixed learning rate

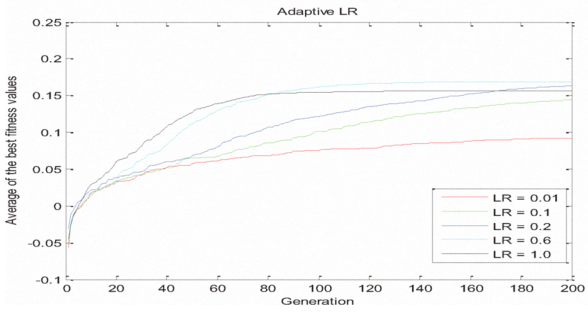

Fig. 4 Adaptive learning rate

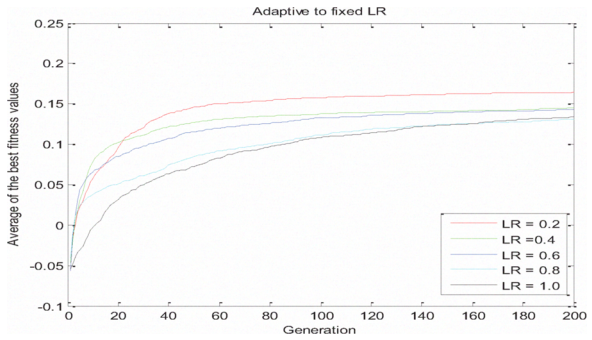

Fig.5 Adaptive to fixed learning rate

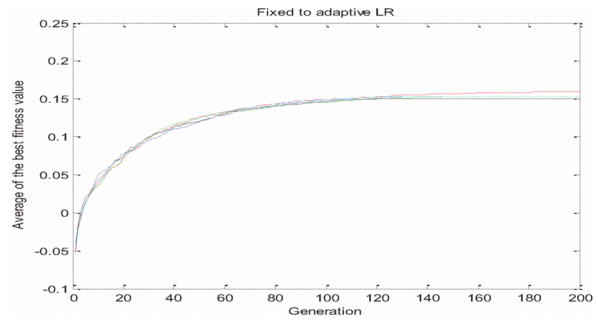

Fig. 6 Fixed to adaptive learning rate

VII. Conclusion

The effect of the learning rate on the performance of PBIL-algorithm has been investigated. The simulations results clearly show that the learning rate has a significant impact on the performance of the algorithm. When the learning rate was set to very high values (i.e., LR = 08-1) for fixed learning rate, diversity was lost in the population and the algorithm converges prematurely in less than 80 generations. It was found that for the same high values, the adaptive learning rate performs better. Smaller learning rates lead to more exploration of the algorithm and hence this introduces more diversity in the population than high learning rates. However, it takes longer for the algorithms to converge. Therefore, a trade-off between exploitation and exploration is needed to obtain the desired results within a reasonable time. At present, this trade-off is not straightforward. It requires the designer to experiment on several values of the learning rate before choosing a suitable one. The simulations with the adaptive learning rate yields more or less consistent fitness of higher values for LR greater than 0.2. Test cases 3 and 4 with combinations of adaptive and fixed learning rates show that setting the learning rate to be initially adaptive introduces diversity. This is not case when the learning rate was initially fixed before switching to adaptive learning rate.

However, these results are true for our particular problem of power system stabilizer design where the search space is multi-modal. In order to generalize this statement, more work still needs to be done on wide range of problems. Also it is necessary to investigate the effect of generation and population on the learning rate. These issues will be looked at in the future.

References

[1] Y.L. Abdel M. A. Abido, and a. H. Mantawy, Simultaneous Stabilization of Multimachine Power Systems via Genetic Algorithms, *IEEE Trans on Power Systems, 14* (4), 1999, 1428-1438.

[2] K. Sundareswaran and S. Razia Begum, Genetic Tuning of a Power System Stabilizer, *Euro. Trans. Electr 14* , 2004, 151-160.

[3] J. H., Holland, *Adaptation in nature and artificial systems*. The University of Michigan Press, 1975.

[4] M. Mitchell, *An Introduction to genetic Algorithms* (The MIT Press 1996).

[5] Z. Michalewicz, *Genetic Algorithms+Data Structure = Evolution Programs* (3rd Ed., Springer-Velag, 1996).

[6] C. Gonz'alez J.A. Lozano P. Larranaga, The convergence behavior of PBIL algorithm: a preliminary approach, *Intelligent Systems Group Computer Science and Artificial Intelligence. The University of the Basque County*, available at http://www.sc.ehu.es/ccwbayes/postscript/kzaa-ik-03-99.ps

[7] S. Baluja, Population-Based Incremental Learning: A method for Integrating Genetic Search Based Function Optimization and Competitive Learning. *Technical Report CMU-CS*-94-163, Carnegie Mellon University, 1994.

[8] S. Baluja and R. Caruana, "Removing the Genetics from the Standard Genetic Algorithm", *Technical Report CMU-CS*-95-141, Carnegie Mellon University, 1995.

[9] J. R. Greene. Simulated and Adaptive Search in Engineering Design-Experience at the University of Cape Town, Invited key paper *at world Conference on soft Computing WSC3*, Springer Verlag, 1997.

[10] K.A. Folly, Stability Enhancement of Power Systems Using a PBIL Based Power System Stabilizer, *6th Africon Conferencein Africa*, 2002, 953-956

[11] K. A. Folly, "Design of Power System: A Comparison Between Genetic Algorithms (GAs) and Population-Based Incremental Learning (PBIL)", to be presented *at IEEE, PES 2006, General Meeting*, Canada.

[12] F. Southey and K Karray, "Approaching Evolutionary Robotics Through Population-Based Incremental Learning, in *Proc. of the 1999 International Conference on Systems, Man, and Cybernetics (SMC'99)*.

[13] D. Andina and I. A. Vega, Artificial Neural Network, available at http://www.gc.ssr.upm.es/inves/neural/ann1/anntutor.htm

[14] T. Kohonen, Improved version of learning vector quantization, *Proc. of the International Joint Conf. on Neural Networks*, 1990 vol 1, 545-550.

[15] KA Folly, "Robust Controller Based on a Combination of Genetic Algorithms and Competitive Learning", IJCNN 2007, Orlando, Florida, USA, No. 1793.ISBN: 1-4244-1380-X, August 12-17, 2007.