Use of Genetic Algorithms with Back Propagation in Training of Feed-Forward Neural Networks.

Michael McInerney
Department of Physics, Rose-Hulman Institute
Terre Haute, Indiana 47803
mcinerney@rosevc.rose-hulman.edu

Atam P. Dhawan
Department of Electrical and Computer Engineering
University of Cincinnati, Cincinnati, Ohio 45221
adhawan@ece.uc.edu

Abstract - The back-propagation learning algorithm is a well known training method for feedforward neural networks. This method is based on the gradient descent principle which suffers from the problem of getting stuck at local minima. In addition, the back-propagation algorithm is sensitive to parameters such as learning rate and momentum. Genetic algorithms are searching strategies suitable for finding the globally optimal solution. The main advantage of using genetic algorithms for the training of feedforward neural networks is that they can find global minima without getting stuck at local minima. In addition, the performance of the genetic algorithms does not depend on parameters such as the learning rate or momentum. The problem of genetic algorithms is that they are inherently slow. In this paper we present a hybrid of genetic and back-propagation algorithms (GA-BP) which should always find the correct global minima without getting stuck at local minima. Various versions of the GA-BP method are presented and experimental results show that GA-BP algorithms are as fast as the back-propagation algorithm and do not get stuck at local minima. The proposed GA-BP algorithms are also not sensitive to the values of momentum and learning rate used in back-propagation and can be made independent of the learning rate and momentum. It is shown that the adaptive GA-BP algorithm can provide the optimal learning rate and better performance than simple back-propagation.

I. INTRODUCTION

The back-propagation algorithm (BP) is a well known method of training a multilayer, feed forward network. Successful though it is, the algorithm does have some shortcomings. Since it is a gradient descent method it has a tendency to get stuck in local minima of the error surface and thus not find the global minimum. Further, the algorithm uses parameters, such as learning rate and momentum, during training. The performance of the algorithm depends on the selection of initial weights and on the parameters used.

Genetic Algorithms (GA), have recently emerged from the study of the mechanics of evolution and are methods of determining the global minimum in a large parameter space [1]. GA can be used to train a multilayer perceptron in which the weights are seen to form a parameter space [2]. This training procedure does not involve gradient descent and usually has fewer, less sensitive, parameters than BP suggesting that GA based training may be more robust.

While Genetic Algorithms have the advantage of not getting stuck in local optima, they do have other problems. When the search space is large, as is usually the case in training a mulltilayer feed-forward network, the GA method takes a long time to converge. The length of search is due to the optimal generalization of the training process with no a-priori knowledge about the parameter space.

The basic difference between BP and GA based training mechanisms is that, unlike BP, GA does not make use of local knowledge of the parameter space. This difference usually results in BP outperforming GA in the training of feedforward networks. However, BP often gets stuck in a local minimum and either does not converge to train the network, or performs very poorly. It is apparent that in such cases, knowledge of the local parameter space is a disadvantage and hence the GA method is superior. A hybrid training algorithm making use of both GA and BP methods may thus be very useful. In the hybrid algorithm, BP can be used as the training method with GA used to escape from local optima. This paper describes the development of such hybrid algorithms, and compares their performance with that of standard BP on two well known test problems: the three-to-one parity problem, and the four-to-four encoding problem.

Genetic Algorithms have been used to train feedforward networks by Montana and Davis [2] who showed that GA could be used by itself to train a perceptron based network and suggested that BP might be usefully incorporated into the GA. Kitano [3] and Belew et al [4] developed hybrids in which GA are used to search the weight space for the best parameters to be used in the BP algorithm. Kitano used the GA techniques to search for a location in the parameter space at which the error function had a value below an arbitrarily selected threshold. The values at this location were used to initialize a perceptron network that was trained to completion with BP. In this method the chromosome used by the GA algorithm was a string of real values representing the weights of the network and the worth, or fitness, of each chromosome was determined directly from the total square error without any BP training. Belew et al

0-7803-0999-5/93/$03.00 ©1993 IEEE

were more interested in the existence of a preferred set of initial weights for the BP algorithm. The chromosomes were set up in the same manner as Kitano's were, although Belew et al did not use real numbers and determined the worth of each chromosome by training the network for 200 cycles using BP and finding the total square error at the end.

The hybrid GA-BP, presented in this paper, differs from those of Kitano and Belew et al in its use of GA. In our method we do not search for a good launching location for BP as was done by Kitano, instead we use GA to escape from any local minima that BP may fall into. The GA-BP algorithm is presented in two forms; GA-BPbasic, which requires the momentum and learning rate parameters for BP, and GA-BPadap, which adapts the learning rate as training continues and does not require the momentum or learning rate be specified. A hybrid, GA-BPthresh, that implements Kitano's general idea is used for comparison. The algorithms are tested on two standard examples; the three bit parity example and the four-two-four encoding example.

We begin with a brief description of the common terms and methods used in applying genetic algorithms and go on to explain how they can be incorporated with BP into a class of hybrid algorithms for the training of feedforward neural networks.

II. Genetic Algorithms

II.1 Chromosomes

A chromosome must be defined to apply genetic algorithms in optimization or search problems. A chromosome represents the necessary information about the problem by encoding the problem parameters in the form of a string. Each element of the string is called a 'gene' each expression of which is referred to as an 'allele'. At any specific moment, the chromosome represents the state of the system at that time. Thus, for example, the chromosome representing a perceptron in this work consists of a string of real numbers representing its weights and thresholds at a particular moment. In the case that the parameters are also unknown the string may be extended to include them. Expression of this chromosome consists of transferring the string values to a perceptron.

Associated with each chromosome is a 'worth' or fitness function which is the value of the problem encoded by the chromosome. To find the worth associated with a chromosome encoding a network of perceptrons, the chromosome is expressed and the training error determined. This training error is the worth of the chromosome.

II.2 Crossover

In the crossover procedures, two chromosomes with equal numbers of genes are placed next to each other and exchange genes in groups of varying sizes. Both chromosomes are divided into corresponding segments of equal length, although the length of each segment may vary. The first segment of each chromosome is left unchanged; each succeeding segment is exchanged with a predetermined probability, the order of the genes within each segment remaining unchanged. There are many variations of the crossover mechanism depending on the number of segments that the chromosomes are divided into. The extreme cases occur when there are only two segments, 'single point crossover' and when there are as many segments as there are genes, 'random crossover'.

II.3 Mutation

Mutation is the random alteration of gene values in the chromosome according to a predetermined probability. In the case that the genes are real numbers mutation can be divided into two types depending on the change in the gene size that the mutation causes. These two types are 'creep', in which the gene is only changed by a small amount; and 'classic mutation' in which the gene is changed by a large amount. In this context 'small' and 'large' are taken in comparison to the initial maximum allowed value of the gene.

Creep may be thought of as a form of gradient ascent procedure but with random direction. In many applications, including the ones described here, creep gradually cycles between minimum and maximum values. Classic mutation is a drastic operation and usually moves the chromosome to a part of the parameter space where it has a very low value. Occasionally, however, the chromosome is moved to a better position to which it could not have got to by any other method.

II.4 Breeding

New chromosomes are formed from the old by applying the techniques of crossover, creep and mutation described above. In the algorithms used here the 'heavy work' of searching the parameter space is performed by random crossover and creep with classic mutation being reserved for cases when the genetic diversity of the population became too small. One of the advantages of using real number chromosomes is that diversity is easily measured as the standard deviation of the worths of the chromosomes. When the standard deviation falls below a predetermined value (0.000001 in this case) all of the chromosomes except the one with maximum worth are classically mutated.

Occasionally a stable pool of chromosomes is formed which have different worths and a relatively large standard deviation. To avoid this form of stagnation classic mutation of all of the chromosomes except the one with maximum worth is also performed when the standard deviation remain unchanged from one cycle to the next.

The appropriate selection of breeding pairs of chromosomes is important. In this work the pair are selected by a normalized form of the 'roulette wheel' [5] method in which the probability of choosing a chromosome is proportional to its worth relative to the remaining chromosomes in the population. Care is taken to ensure that the breeding pair is made up of two different chromosomes.

III. Hybrid Training Algorithms of GA and BP

A feedforward neural net can be trained either by genetic algorithms or by back-propagation but both methods have their disadvantages. Genetic algorithms are slow while back-propagation is dependent on a correct choice of the learning rate, momentum and the initial values of the net's weights and thresholds. The problem with BP arises because it is quick to find local minima at which it may get stuck while the problem with GA occurs because it rapidly gets into the region of the global minimum but then takes a long time to arrive at it.

In this paper we consider two methods of forming a hybrid of GA with BP that utilize the strengths of one to offset the weaknesses of the other. In the first the GA is used to bring the system near to the global optimum which is then found by BP; in the second, BP is allowed to bring several systems to local minima and then GA used to find the global optimum guided by the coordinates of the local minima. The first method has been used by Belew et al and Kitano and the second is developed here.

The general GA that is used is described in figure 1; the BP algorithm is not listed because it is well known [6]. The chromosomes used by the GA are strings of real numbers representing the weights and thresholds of the neural net; in some cases the string also includes the momentum and learning rate parameters of the BP.

III.1 The GA-BPthresh algorithm.

Kitano assumed that the system was close to global optimum when the mean square error fell below a threshold; he used GA to reduce the total square error below this threshold and then finished the training with BP. The GA-BPthresh algorithm , described in figure 2, implements this idea.

1) Establish a base 'breeding' population of chromosomes.

2) Determine the worth of each chromosome in turn.

3) Sort the chromosomes by worth, calculate standard deviation.

4) If the standard deviation, or the change in the standard deviation, is less than an arbitrary constant (0.00001), then classically mutate all the chromosomes except the one with maximum worth and go to step (2).

5) Use the roulette method with windowing to choose a breeding pair of chromosomes. Make two copies of this pair. Apply random crossover to one pair of copies and creep to the other pair.

6) Repeat (5) as many times as there are chromosomes in the base population.

7) The result of (6) is that there are now five times the base population of chromosomes in the pool. These chromosomes are tested, sorted by worth, and the original number restored by deleting those with lowest worth.

8) Finally the 'best' worth is compared with the desired worth and if it is less the algorithm is repeated from (3) with a reduced creep value. If reduction of the creep brings it below an arbitrary value (0.00001) then the creep value is restored to its original size.

Fig. 1. The basic Genetic Algorithm.

III.3 The GA-BPbasic algorithm.

The structure of the GA-BPbasic algorithm, shown in figure 3, is essentially the GA algorithm described in figure 1 with the BP introduced at the test stage to train the neural net to a local minimum. Once this local minimum is reached, the new values for the weights and thresholds of the neural net are returned to the chromosome. This method is similar in spirit to the 'Lamarkian' variant mentioned by Belew et al..

1) Choose a value, by experimenting on sample problems, for the total square error below which the GA algorithm does not train the network quickly.

2) Use the GA algorithm of figure 1 to train the perceptron network to the value selected in step (1).

3) Continue training the network found in step(2) using BP until the total square error has fallen below an arbitrary value (0.01) or the number of BP_CYCLES has exceeded a fixed number (usually 2,000).

Fig. 2. The GA-BPthresh algorithm.

IV. Dynamic Adaptation of the Learning Parameters

The hybrid algorithm's dependence on the BP momentum and learning rate can be removed by the dynamic adaptation of these parameters in a new version, GA-BPadap. The GA-BPbasic algorithm (figure 3) is modified in the testing stage (step 2). At the beginning of testing for each expression of the neural net, the learning rate of the BP algorithm is set to 0.1 and then, after each training cycle in which the square error declines, is increased by 0.1.

For comparison, a version of standard BP with adaptation, BPadap, was developed. This version , starts with a learning rate and momentum of 0.1; the learning rate is increased by 0.1 while the cost function decreases and then decreased by 30%, to a minimum of 0.1, whenever the cost function increases. A comparison of BPadap and GA-BPadap is shown in table 1 in the results section.

V. Results and Discussion

V.1 Experiments and Notation.

Two examples are used to compare the algorithms; the three bit parity example and the four-to-four encoding example. In the three bit parity example the input consists of three 'bits' that can be either +1 or -1 and the output can be either +1 or -1. An odd number of input bits set to +1 results in a +1 output, otherwise the output is -1. The four-to-four encoding example has four inputs and four outputs, each of which can either be +1 or -1. The bits of the input patterns consist of -1 with a sole +1 and the output is identical.

The perceptrons used have three layers, (n,m,p), representing the number of input. hidden and output nodes respectively. The perceptrons are fully connected and the input nodes simply pass the signal straight through while the hidden and output nodes condition it with the usual sigmoid function. A (3,3,1) perceptron is used with the three bit parity example while a (4,2,4) perceptron is used with the four-to-four encoding example. All of the chromosomes are lists of real numbers that are initially selected, at random and with uniform probability, from a fixed range.

A training pair consists of an input and desired output; the input is applied to the input of the perceptron and the desired output compared with the observed output. A complete BP training cycle, written as BP_CYCLE in the graphs in this paper, involves randomly selecting pairs from the training set and using them in turn to change the weights according to the BP algorithm. We used twenty times the number of pairs in the training set and then found the worth for the

1) Establish a base 'breeding' population of chromosomes the genes of which are real numbers, selected randomly with uniform distribution from a predetermined range (+/-5 for example), representing the weights and thresholds of the perceptron network.

2) Test each chromosome in turn by expressing it as a network and using BP to train it until the total square error ceases to decline between one training cycle and the next, or the total square error becomes less than the minimum desired (0.01). In both cases the new weights are returned to the chromosome; in the former case, the next, untested chromosome is expressed and tested while in the latter case the algorithm goes to step (8). It should be remembered that one training cycle involves the presentation of the complete training set 21 times.

3) Sort the chromosomes by worth, calculate standard deviation.

4) If the standard deviation, or the change in the standard deviation between this cycle and the previous one, is less than an arbitrary constant (0.00001), then classically mutate all the chromosomes except the one with optimum worth and go to step (2).

5) Use the roulette method with windowing to choose a breeding pair of chromosomes. Make two copies of this pair. Apply random crossover to one pair of copies and creep to the other pair.

6) Repeat (5) as many times as there are chromosomes in the base population.

7) The result of (6) is that there are now five times the base population of chromosomes in the pool. These chromosomes are tested using the method described in step 2, sorted by worth, and the original number restored by deleting those with lowest worth.

8) If a chromosome has been found in steps 2 or 7 to represent a trained network by having a sufficiently low worth (0.01), then the algorithm finishes. Otherwise, the algorithm is repeated from (3) with a reduced creep value. If reduction of the creep brings it below an arbitrary value (0.00001) then the creep value is restored to its original size.

Fig. 3 The GA-BPbasic Algorithm.

training cycle by applying each pair of the training set in turn, deterministically, to the perceptron and summing the squares of the difference between the observed and desired outputs (the total square error). Each BP_CYCLE thus involves applying the BP algorithm for twenty one times the number of pairs in the training set. A perceptron is considered to be trained when the mean square error drops below an arbitrary constant (0.01).

The algorithms are tested by using them to train the examples for a series of two hundred times and counting cycles. The BP algorithm sometimes does not converge within a reasonable number of BP_CYCLEs and thus averages cannot be calculated. Instead, the median number is reported and a measure of the spread of the results given by reporting the range within which 30% of the total fell above the median (med + 30%) and 30% fell below the median (med - 30%).

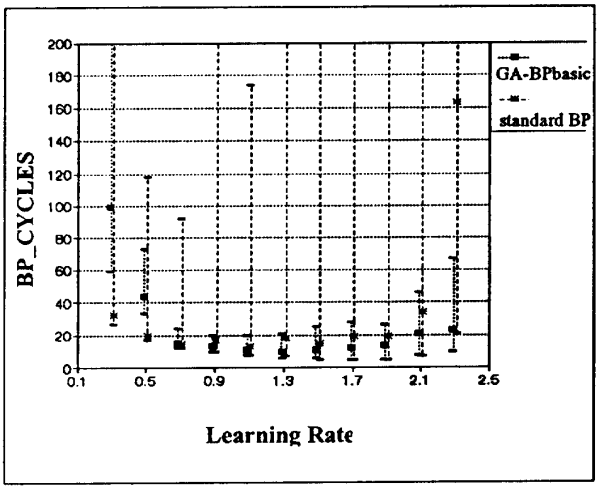

Fig. 4 Variations within 30% of the median for Standard BP and GA-BPbasic for the 3-to-1 problem trained on a (3,3,1) net. The weight space is +/-2.

V.2 A comparison of GA-BPbasic with standard BP.

Figures 4 and 5 show a comparison between the performance of GA-BPbasic and standard BP for different values of the learning rate with constant momentum, 0.1. The overhead of the GA has not been factored in to the BP_CYCLE score of the GA-BPbasic algorithm because the number of cycles of the GA is usually about 1/20th the number of BP_CYCLEs.

It is clear from these results that GA-BPbasic is more robust with respect to the learning rate than is standard BP.

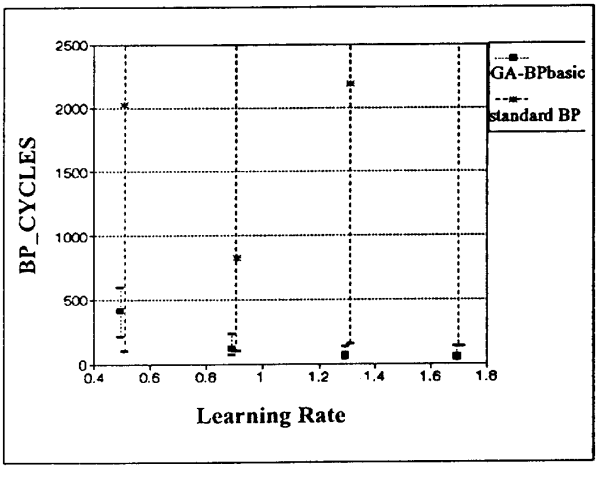

Fig. 5. Variations within 30% of the median for the 4-2-4 problem using a (4,2,4) net when the weight space is +/-5.

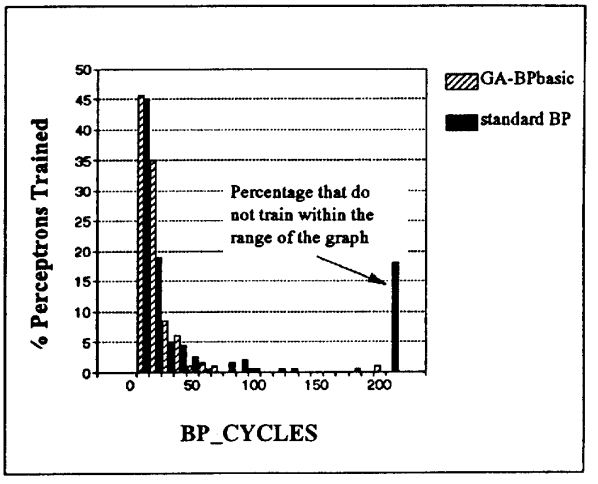

Fig. 6. Histogram of 331 example with GA-BP and BP. Learning rate of 0.7, Momentum 0.1 and the weight space varies +/-2.

When the learning rate is large the BP will train quickly if it is in the correct region of weight space, but these regions become rarer as the learning rate diverges from the optimum. Thus an algorithm, such as GA-BPbasic, that abandons weight sets that BP does not immediately begin to train successfully, might be expected to be less susceptible to the actual value of the learning rate. In fact, not only is GA-BPbasic more robust than BP but it is comparable with, or better than, standard BP at all learning rates. It appears that the cost of keeping several perceptrons training at once is offset by the probability that one of them will train quickly.

Another characteristic of BP is itshighly variable training time, a characteristic not shared with GA-BPbasic. This difference is illustrated by the histogram of figure 7 where we see the difference in distributions between standard BP and GA-BPbasic even when the medians for the two algorithms are comparable.

V.3 Comparison of GA-BPbasic and GA-BPthresh

The GA-BPthresh algorithm, (figure 2), uses GA to get close to the global optimum and then BP to finally reach that optimum. GA-BPbasic, (figure 3) on the other hand, uses GA to escape from the local optima into which it has been placed by BP until BP finds the global optimum. A comparison between GA-BPbasic and GA-BPthresh, with a threshold of 3.5 determined from figure 7, is shown in figure 8. It is clear that GA-BPbasic is faster and more reliable than GA-BPthresh. The assumption in GA-BPthresh is that any point in the weight space that is close to the global optimum, in the sense that the total square error is small, is a

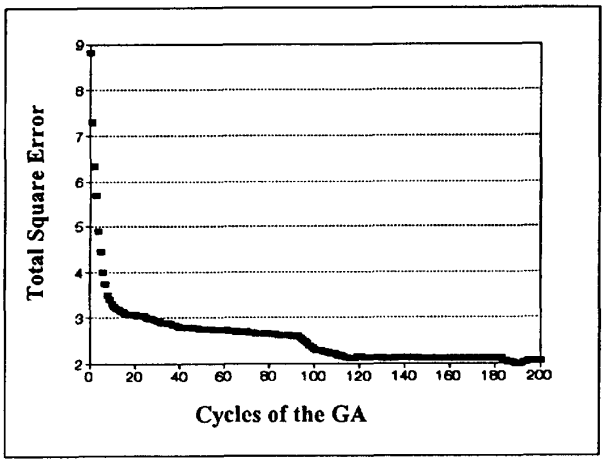

Fig. 7. Average of 100 trials training a (3,3,1) perceptron, with GA techniques only, on the three-to-one parity problem.

good place from which to launch the BP algorithm. It would appear that this assumption is incorrect.

V.4 Dynamic adaptation of learning parameters, GA-BPadap.

The optimum learning rate in BP is usually not known prior to training; an improvement on guessing the learning rate is to dynamically adapt it as learning progresses as described in section III for standard BP and GA-BP resulting in the algorithms BPadap and GA-BPadap respectively. A comparison of these two algorithms is shown in table 1 where it is clear that the adaptation is successful for the hybrid but not for the standard BP. In fact, a comparison of

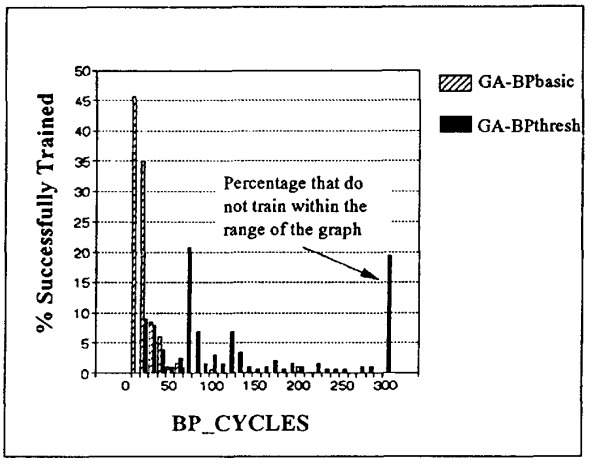

Fig. 8. Comparison of GA-BPbasic with GA-BPthresh; the learning rate is 0.7, the momentum zero and the weight space +/-2.

Test problem:	four-to-four		three-to-one	
	GA-BPadap	BPadap	GA-BPadap	BPadap
Median - 30%	98	154	28	89
Median	166	4,739	97	2,169
Median + 30%	306	>5000	271	>5,000

TABLE I

COMPARISON OF ADAPTIVE LEARNING RATE METHODS. THE NUMBERS REFER TO BP_CYCLES AND THE MOMENTUM IN BP WAS SET AT 0.1. INITIAL VALUES WERE SELECTED FROM THE RANGE +/-5.

these results with those in figure 4 show GA-BPadap to be as good as GA-BPbasic with its best learning rate for the four-to-four example.

VI. CONCLUSION.

A hybrid of genetic algorithms with back-propagation can train a feedforward neural net on the two standard test problems considered here more reliably and just as quickly as standard back-propagation with an optimal initial learning rate. The hybrid has to be one, such as GA-BPbasic, that uses GA to escape from local minima into which BP trains the net and not one, such as GA-BPthresh, that uses GA to find a starting point for BP close to the global optimum.

We were able to successfully extend GA-BPbasic to make it independent of the momentum and learning rate by dynamically adjusting the learning rate and ignoring the momentum. This extension, GA-BPadap, proved to be far superior to a similar modification to standard BP, BPadap, and comparable or better than GA-BPbasic with an optimal learning rate.

REFERENCES.

[1] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

[2] D. J. Montana, L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms", *Proceedings 11th International Joint Conference on Artificial Intelligence, 1989, Detroit MI.*, Morgan Kaufman, San Mateo CA., pp. 762-767.

[3] H. Kitano, "Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms", *Proceedings 8th JMIT National Conference in Artificial Intelligence, 1990, Boston MASS*, Vol2, pp.789-796.

[4] R.K.Belew, J. McInerney, N.N. Schraudolph, "Evolving Networks: Using the Genetic Algorithm with Connectionist Learning", *Artificial Life II, SFI Studies in the Sciences of Complexity, vol. X*, edited by C.G.Langton, C.Taylor, J.D.Farmer, S.Rasmussen, Addison-Wesley Redwood City California, 1991, pp.511-547.

[5] L. Davis (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991. p.13 and p.32

[6] J. M. Zurada *Artificial Neural Systems*, West, 1992, p.187.