Robust Neural Network Training Using Partial Gradient Probing

Milos Manic
Department of Computer Science
University of Idaho
800 Park, Blvd.
Boise, ID 83712, USA
misko@ieee.org

Bogdan Wilamowski
College of Engineering
University of Idaho
800 Park, Blvd.
Boise, ID 83712, USA
wilam@ieee.org

Abstract – Proposed algorithm features fast and robust convergence for one hidden layer neural networks. Search for weights is done only in the input layer i.e. on compressed network. Only forward propagation is performed with second layer trained automatically with Pseudo-Inversion training, for all patterns at once. Last layer training is also modified to handle non-linear problems, not presented here. Through iterations gradient is randomly probed towards each weight set dimension. The algorithm further features serious of modifcations, such as adaptive network parameters that resolve problems like total error fluctuations, slow convergence, etc. For testing of this algorithm one of most popular benchmark tests – parity problems were chosen. Final version of the proposed algorithm typically provides a solution for various tested parity problems in less than ten iterations, regardless of initial weight set. Performance of the algorithm on parity problems is tested and illustrated by figures.

I. INTRODUCTION

The proposed algorithm draws its roots from two distinctive approaches in iterative search. One is iterative gradient search method characteristical for neural networks, such as error back propagation. The other approach considers flexible approximative techniques for neighborhood search.

Neighborhood search methods are iterative procedures where for each feasible solution a neighborhood for next solution is defined. Most popular of those methods are descent method, Simulated Annealing and Tabu search.

Though Tabu search originates from late 1970s, first results were independently presented by [1,2]. It was further formalized by [3,4] and [5]. It is a flexible approximation technique that uses memories or tabu lists to forbid moves that might lead to recently visited, i.e. tabu solutions. Tabu lists can help to intensify the search in 'good' regions or to diversify the search toward unexplored regions by variable tabu list size.

Error Back Propagation (EBP) neural networks and gradient methods generally provide very good results [6]. Though presented a breakthrough in neural network research, backpropagation algorithm has number of disadvantages, such as oscillation, slow convergence, sensitivity to network parameters, etc.

Numerous researchers targeted these problems over the years. In order to smooth the process, Rumelhart et al. [7] have proposed weight adaptation, also formulated by Sejnowski and Rosenberg [8]. To speed up the process, Fahlman [9] proposed a quickprop algorithm. Wilamowski and Torvik [10] proposed solution towards robustness in terms of activation function derivative modification. Still, no universal solution towards robust and fast training algorithm has been found.

The proposed algorithm features fast and robust convergence, achieved through adaptiveness of neural network parameters (learning constant, network gain, and search radius). This way algorithm typically provides a solution for various tested parity problems in less than 10 iterations.

Search for weights is done only in the input layer i.e. not all weights participate in gradient calculation. Algorithm performs only forward propagation with second layer trained automatically with Pseudo-Inversion training for all patterns at once. In each iteration gradient is randomly probed towards each weight set dimension.

Algorithm is further modified to alleviate occasional unwanted algorithm behavior. Those modifications automatize search for most adequate parameters (adaptive search). This way problems such ase slow convergence, fluctuations in gradient search, flat spots, local minima, or sensitivity to initial choice of network parameters, are resolved

The rest of the paper is organized as following. Second section discusses weight space reduction and 2^{nd} layer automatized training. Third section explains steps of proposed algorithm, and is followed by gradient calculation explanation. The fourth section illustrates test examples. The fifth section concludes this paper with directives for future work. Last, sixth section contains references used in this paper.

II. NETWORK COMPRESSION

This algorithm considers compressed neural architecture. Let us consider general 2 layer architecture of $n+1$ neurons, where 1^{st} layer consists of n-neurons and last layer of single neuron (illustrated by Fig.1a).

The reduction of such network starts from initial number of weights, that is:

$$1^{st}\,layer + 2^{nd}\,layer = n*(m+1)+1*n+1 \quad (1)$$

where $n+1$ is number of neurons, and m number of inputs for 1^{st} layer neurons, $m+1$ is because of added bias.

Last layer ($n+1^{st}$ neuron) is trained by Pseudo-Inversion training, reducing the number of weights to $n*(m+1)$. Furthermore, bias weights are fixed to 1 therefore not participating in training. This way, original number of weights is reduced to input weights of the 1^{st} layer, i.e. $n*m$ weights.

0-7803-8200-5/03/$17.00 ©2003 IEEE

For simple 2+1 network architecture (Fig.1b) this compression reduces initial number of 9 weights to only 4 weights (weights of input connections of the first layer).

Second layer training is automatized by Pseudo-Inversion learning rule:

$$W = (X^T X)^{-1} X^T d \quad (2)$$

where $X^{\perp} = (X^T X)^{-1} X^T$ is the pseudoinverse of column matrix X, that exists even when rank is less than the dimension or when the matrix is not square. This algorihtm implements the modification of this rule known as Andersen-Wilamowski rule [11,12] to enable solving non-linear problems:

$$\Delta W = (X^T X)^{-1} X^T \frac{d - o}{f'} \quad (3)$$

Algorithm is further modified to include several steps of this modified rule. This way algorithm is able to solve not only linear parity problems presented in this paper.

II. PROPOSED ALGORITHM (STEPS)

Before going into algorithm steps, certain parameters should be determined. Those are network parameters and training patterns. Network parameters are *initial set of weights*, learning parameter *alpha*, network gain k, and search radius r. The advantage of this algorithm is its robustness with respect to these parameters, therefore all of them can be randomly chosen and algorithm will adaptively adjust them searching for fastest convergence.

For tested parity problems (parity 2, 3, and 4), certain initial values for learning parameter, network gain, and search radius result in faster convergence and higher algorithm precision. Those parameters had been heuristically determined, and some of those tests are illustrated later.

Tests were performed on Intel Pentium III 1GHz, with 512MB SDRAM memory, Windows 2000 Pro machine.

The algorithm steps go as follows.

1. **Assign Input** values. For the one hidden layer network (Fig.1.a), those are $n*m$ weights, where n is number of neurons in 1st layer, and m is number of their input weights (1st layer bias is fixed to 1). In case of simple 3-neuron network (Fig.1b), these are only 4 weights for inputs of first layer neurons set ($w_{k,11}$, $w_{k,21}$, $w_{k,12}$, $w_{k,22}$). Weights for the second layer neuron can be arbitrary chosen and do not influence the calculation.

2. **Initial Pass.** One pass through the network is done. This means *net*, *output*, and finally *total error* (TE) is calculated for all patterns. This is done through forward calculation through the first layer, for each pattern, for all patterns. Now Pseudo inversion takes charge of training of last layer weights. This 2nd layer hypersonic training is done for all patterns at once. Once signals are propagated through the 1st layer (for all patterns), then the 2nd layer can be efficiently trained. The outcome of this step is the initial total error E_0.

3. **Assign input values for next iteration.** These are total error E_0 and the same weight set from the 1st step $n*m$ weights in general case ($w_{k,11}$, $w_{k,21}$, $w_{k,12}$, $w_{k,22}$ in case of 3 neuron network). For this starting weight set, gradient will be estimated in next step. Go to a next step (start with iterations).

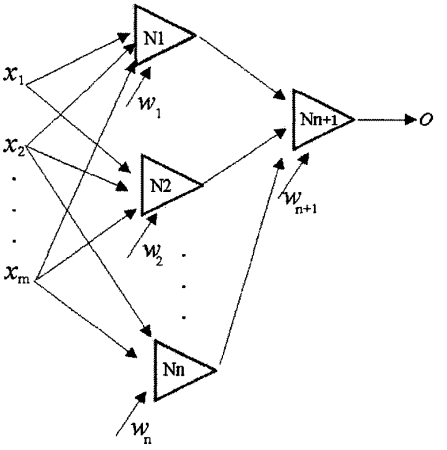


Fig. 1. General cases of typicall one hidden layer neural network architecture used in proposed algorithm.

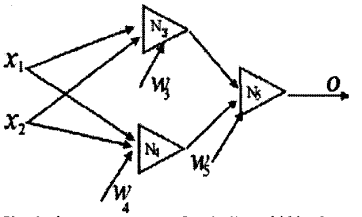


Fig. 1. Simple three neuron case of typicall one hidden layer neural network architecture used in proposed algorithm.

4. **Start iterations. First calculate the gradient.** Gradient is numerically calculated through gradient probing around the weight set from previous step. Such quasi-gradient is calculated in following way. Each of $n*m$ "changeable" weights (4 in case of 3-neuron architecture), is adaptively reproduced, one at the time. Therefore, $n*m$ **cycles** are performed (again in 3-neuron case, 4 cycles). In each cycle, only one of those $n*m$ weights gets changed. The rest of the weight set is being kept the same. Now one feed-forward pass through the whole network is performed, same as in 2nd step. The output of each cycle is the total error. This total error for each cycle (E_1, E_2,..., E_{n*m}) is stored for later gradient evaluation. Gradient in $k+1^{st}$ iteration is calculated according to following formula in vector form (4) where $w_{repk,ij}$ is weight reproduced from weight $w_{k,ij}$, in iteration k, for input i and neuron j. $E_{repk,m}$ is the total error calculated for set of weights from 3rd step where instead of $w_{k,ij}$, reproduced weight $w_{repk,ij}$ was used.

$$grad_k = \begin{bmatrix} \frac{E_{repk,11}-E_{k,0}}{w_{repk,11}-w_{k,11}} \\ \frac{E_{repk,21}-E_{k,0}}{w_{repk,21}-w_{k,21}} \\ \cdots \\ \frac{E_{repk,m1}-E_{k,0}}{w_{repk,m1}-w_{k,m1}} \\ \cdots\cdots\cdots \\ \frac{E_{repk,1n}-E_{k,0}}{w_{repk,1n}-w_{k,1n}} \\ \frac{E_{repk,2n}-E_{k,0}}{w_{repk,2n}-w_{k,2n}} \\ \cdots \\ \frac{E_{repk,mn}-E_{k,0}}{w_{repk,mn}-w_{k,mn}} \end{bmatrix}_{m*n\times 1} = \begin{bmatrix} \frac{\partial E_{k,1}}{\partial w_{k+1,11}} \\ \frac{\partial E_{k,2}}{\partial w_{k,21}} \\ \cdots \\ \frac{\partial E_{k,m}}{\partial w_{k,m1}} \\ \cdots\cdots\cdots \\ \frac{\partial E_{k,1n}}{\partial w_{k,1n}} \\ \frac{\partial E_{k,2n}}{\partial w_{k,2n}} \\ \cdots \\ \frac{\partial E_{k,mn}}{\partial w_{k,mn}} \end{bmatrix}_{m*n\times 1} \quad (4)$$

4.1. Adaptive reproduction of weights goes as follows. Weights are randomly changed in certain radius. The initial radius is specified at the beginning of the algorithm. Algorithm then adapts the radius value after each iteration. This adaptiveness helps algorithm to accelerate through flat spot areas and override eventual local minima problems.

5. Now the **new set of weights** is calculated based on the estimated gradient and set of "changeable" weights. New set of weights is calculated based on previous formula:

$$W_{k+1} = W_k - \alpha * grad_k, \quad (5)$$

or in vector form (6):

$$W_{k+1} = \begin{bmatrix} w_{k+1,11} \\ w_{k+1,21} \\ \cdots \\ w_{k+1,m1} \\ \cdots\cdots \\ w_{k+1,1n} \\ w_{k+1,2n} \\ \cdots \\ w_{k+1,mn} \end{bmatrix}_{m*n\times 1} = \begin{bmatrix} w_{k,11} \\ w_{k,21} \\ \cdots \\ w_{k,m1} \\ \cdots\cdots \\ w_{k,1n} \\ w_{k,2n} \\ \cdots \\ w_{k,mn} \end{bmatrix}_{m*n\times 1} - \alpha * \begin{bmatrix} \frac{E_{repk,11}-E_{k,0}}{(w_{k,11}+\Delta)-w_{k,11}} \\ \frac{E_{repk,21}-E_{k,0}}{(w_{k,21}+\Delta)-w_{k,21}} \\ \cdots \\ \frac{E_{repk,m1}-E_{k,0}}{(w_{k,m1}+\Delta)-w_{k,m1}} \\ \cdots\cdots \\ \frac{E_{repk,1n}-E_{k,0}}{(w_{k,1n}+\Delta)-w_{k,1n}} \\ \frac{E_{repk,2n}-E_{k,0}}{(w_{k,2n}+\Delta)-w_{k,2n}} \\ \cdots \\ \frac{E_{repk,mn}-E_{k,0}}{(w_{k,mn}+\Delta)-w_{k,mn}} \end{bmatrix}_{m*n\times 1} \quad (6)$$

or in different form (7). Each Δ is randomly generated number within search radius. This set of weights is now ready to be used in next iteration. Go to next step where network parameters will be prepared for next iteration.

6. Adapt network parameters: learning parameter *alpha*, network gain *k*, and search radius *r*. These parameters are modified depending on trend of total error (TE) change in previous 4 iterations. Gain and radius are increased if TE is within 10% change, decreased otherwise. Alpha is modified in similar fashion; however, criterion for alpha increase is monotonic TE decrease. In other words, alpha gets decreased in case of TE fluctuations or non-monotonic TE decrease. Intervals of these parameters are experimentally determined, and those are: alpha (0.5,100), gain (1,10), and radius (0.5, 100). This step was introduced after unwanted behavior (stuck in local minima, flat spot/slow convergence problems), has been detected. Related behavior has been illustrated in details in section with experimental results.

$$W_{k+1} = \begin{bmatrix} w_{k,11} \\ w_{k,21} \\ \cdots \\ w_{k,m1} \\ \cdots\cdots \\ w_{k,1n} \\ w_{k,2n} \\ \cdots \\ w_{k,mn} \end{bmatrix}_{m*n\times 1} - \alpha * \begin{bmatrix} \frac{\partial E_{k,11}}{\partial w_{k,11}} \\ \frac{\partial E_{k,21}}{\partial w_{k,21}} \\ \cdots \\ \frac{\partial E_{k,m1}}{\partial w_{k,m1}} \\ \cdots\cdots \\ \frac{\partial E_{k,1n}}{\partial w_{k,1n}} \\ \frac{\partial E_{k,2n}}{\partial w_{k,2n}} \\ \cdots \\ \frac{\partial E_{k,mn}}{\partial w_{k,mn}} \end{bmatrix}_{m*n\times 1} \quad (7)$$

7. End of this iteration. **Start a new iteration** (go to Initial Pass (2nd step). Keep doing this until the criterion for the error is satisfied.

8. **End** of the Algorithm.

Interpretation of learning constant alpha is the following. Learning constant alpha multiplies all "changeable" weights by the same number, so practically accelerates the process jumping over couple of steps that would do the same.

Alpha is experimentally kept in interval [0.5, 100]. If it gets smaller, learning is slowed down too much ("changeable" weights decreased 10 times if alpha=0.1). This means alpha is not initialized in each iteration, it rather keeps changing to accommodate the learning process. Sometimes algorithm stays for 4–8 iterations with the similar total error. Learning constant alpha as well as other adaptive parameters do not change sooner in order to avoid fluctuations. Once the better set of weights is found, algorithm goes rapidly towards the good gradient.

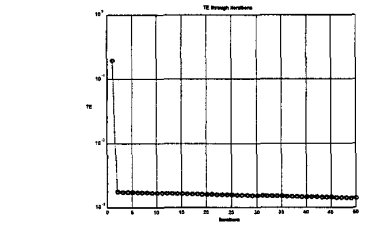
Adaptivity of learning constant is very important, and performance of the algorithm significantly deteriorates without this improvement.

III. EXPERIMENTAL RESULTS

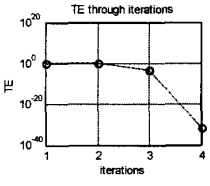
Experimental results were obtained through various parity problems. Activation function was bipolar. The effect of adaptive and network parameters (initial *set of weights,* learning parameter *alpha*, network gain *k*, search radius *r*) will be illustrated on following examples.

First problem was the XOR problem, tested on network architecture from Fig.1a, with required total error of $0.5e10^{-8}$. Problems encountered and effect of introduced modifications will be illustrated.

Though algorithm shows acceptable convergence rate for small gain values (around 0.1), it might experience slow convergence rate (Fig.3a). For large values of gain ($\cong 10$, algorithm exhibits much faster convergence (Fig.3b). Though



a)



a ithm

b dius.

0;

re ; (cpu

exhibiting better behavior, al;
minor convergence slow down (Fig.3c). Therefore, to alleviate this problem, radius needs to probe solutions beyond initially specified fixed radius.

To automate selection of adequate parameter values, modification that adaptively changes radius, gain, and alpha is introduced. These changes are governed by the trend of previous total error behavior.

Fig. 4 illustrates algorithm's behavior with adaptive network parameters. In this case, algorithm detecting a good total error trend, increases all parameters which leads to total error of 10^{-14}. This is the typical algorithm's behavior.

Typically, once the algorithm would experience a good total error trend (TE monotonically decreases), it increases all parameters. This convergence acceleration is supported in 3 ways. First, gradient showing good TE trend is emphasized by enlarged alpha. Second, good TE trend enables allows enforcing

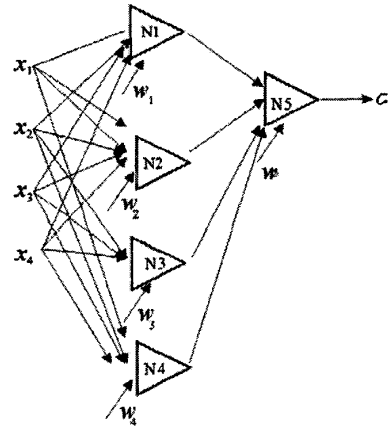


Fig. 5. Neural network architecture used for parity 4 testing.

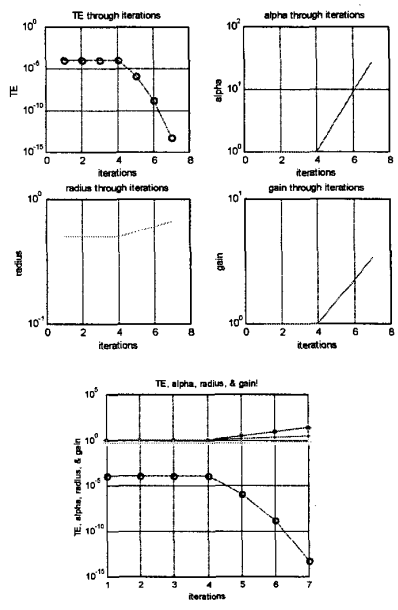


Fig. 4. Parity 2 problem. Total error WITH adaptive alpha, gain, and radius – always converges.

of larger network gain. Third, radius is enlarged so the algorithm can simply jump to even better solution.

This all result in not only accelerated process, but even more important robust convergence. Robustness relies on adaptable parameters that override possible problems of local minima or flat spots.

The architecture used for testing parity 4 problem is given by Fig.5. The algorithm proved all time convergence. However, the convergence rate and therefore minimum achievable total error significantly vary with respect to gain.

Dependence of minimum achievable total error (TE) on network gain is illustrated by Fig.6. Smaller value of gain provides smoother convergence achieving smaller total error. Here, for parity 4, the smaller the gain, the smaller TE can be

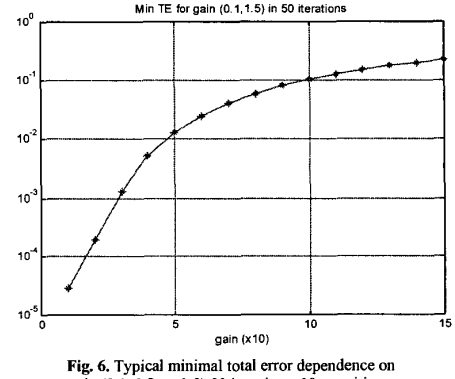


Fig. 6. Typical minimal total error dependence on gain (0.1, 0.2,…,1.5) 50 iterations, 10 repetitions. Parity 4 problem.

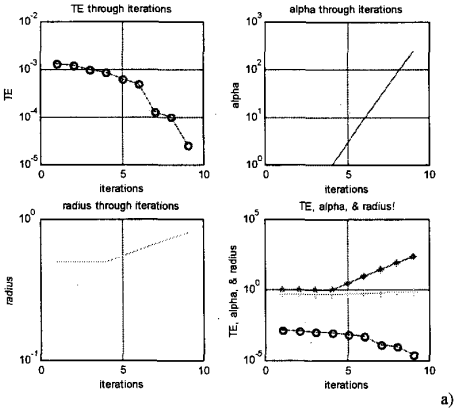
obtained. This graph is obtained by selecting best solution while running 10 repetitions, 50 iterations each. Gain values tested are from interval (0.1,0.2,...,1.5).

Therefore, adaptive gain would not result in better algorithm's accuracy. For higher parity problems, an adaptive gain is excluded from pool of adaptive parameters.

Algorithm shows consistent convergence for smaller gain (gain=0.1), paying the price in small convergence rate. Fig. 7a illustrates typical algorithm behavior without adaptive parameters. Algorithm does converge to total error of order 10^{-5} taking over 35 iterations. With larger gain (1.0) even for smaller radius algorithm can not achieve total error better than 10^{-1}. This undesirable behavior is illustrated by Fig.7b.

As said previously, adaptive gain is excluded from set of adaptive network parameters.

Algorithm exhibits exhibits best behavior with adaptive alpha and radius keeping the gain fixed. This way algorithm achieves robust and smooth convergence in least amount of time possible (Fig.8). The least ammount of fluctuations leads to fastest convergence.



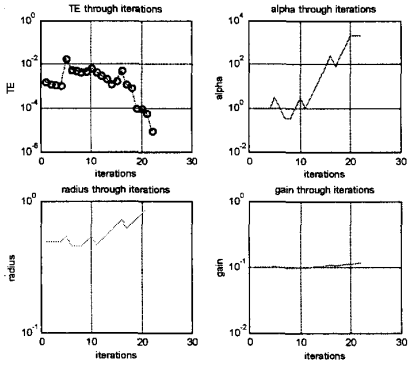


Fig. 8. a) Parity 4 problem. Total error WITH adaptive alpha and radius, but without adaptive gain! b) same WITH adaptive gain.

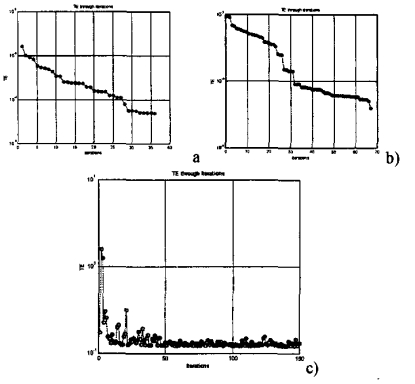


Fig. 7. Parity 4 problem. Typical TE through iterations, alpha=1.0.
a) gain=0.1, radius=0.5 =>CPU time=2.7140e+000 (seconds), TE=4.9501e-005, 36 iterations.
b) gain=0.1, radius=1.5, CPU time=5.2280e+000 (seconds), TE=4.0362e-005, 67 iterations.
c) gain=1.0, radius=0.5, CPU time=1.0836e+001 (seconds), TE=1.3319e-001, 150 iterations.

Demonstration of negative influence of adaptive gain is given at Fig.8b. Fluctuations slow down the learning process, while with fixed gain gradient search is smooth and takes about 3 times less iterations.

Parity 7 algorithm handles in similar, smooth and robust fashion. On specified machine proposed algorithm typically takes only about 2 minutes of CPU time (Fig. 9).

Typical gradient search while solving parity 8 problem is given by Fig.10. Algorithm typically takes 5-6 minuts of CPU time to solve this problem. Even difficult problem of parity 10 converges in smooth and robust fashion. It achieves satisfactory total error in about 76 minutes on specified machine (Fig.11).

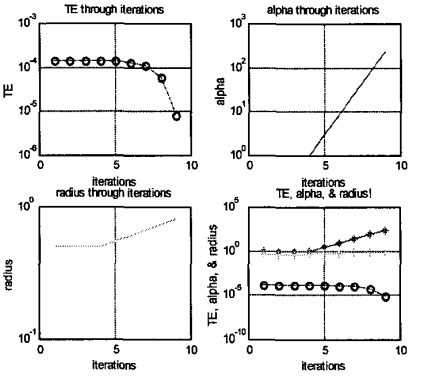


Fig. 9. Parity 7 – with adaptive alpha and radius, without adaptive gain! Always converges smoothly.
Parameters: gain=.1; alpha=1; req_err=.5e-04; TERR=1e3; radius=.5; rad_ww_ini=1.5; Elapsed time=1.0275e+002 (seconds).
Final iteration = 9; TERR=8.5127e-006; alpha=2.4300e+002; radius=8.0526e-001; gain=1.0000e-001

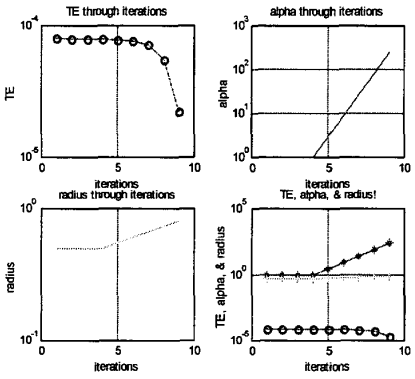


Fig. 10. Parity 8 – with adaptive alpha and radius, without adaptive gain! Always converges smoothly.
Parameters: gain=.1; alpha=1; req_err=.5e-04; TERR=1e3; radius=.5; Elapsed time=3.2061e+002 (seconds).
Final iteration= 9; TERR=2.2358e-005; alpha=2.4300e+002; radius=8.0526e-001; gain=1.0000e-001.

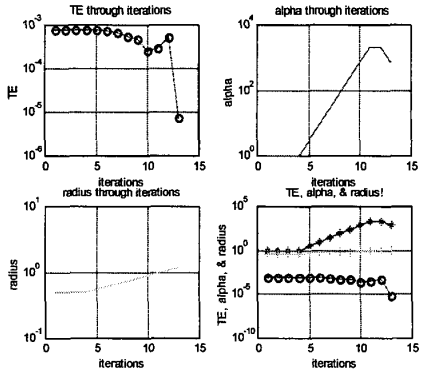


Fig. 11. Parity 10 – with adaptive alpha and radius, without adaptive gain! Always converges smoothly.
Parameters: gain=.1; alpha=1; req_err=.5e-04; TERR=1e3; radius=.5; paty=10; rad_ww_ini=3.5; Elapsed time=4.5846e+003 (seconds).
Final iteration=13; TERR=7.4343e-006; alpha=7.2900e+002; radius=1.1790e+000; gain=1.0000e-001.

IV. CONCLUSION

New gradient search approach is proposed. Further modifications to the approach were made to resolve problems of slow convergence and fluctuations. Those modifications automatize search for most adequate parameters (adaptive search). Final version of the proposed algorithm results in fast (typically less than 10 iterations) and robust (random network parameters) convergence on tested parity problems.

Adaptivity is responsible for two important issues. One is rapid and robust search that avoids flat spots, slow convergence, and local minima. The other is that adaptivity reduces fluctuations during the gradient search leading to faster convergence.

For smaller parity problems (xor2 or xor3), even a very large gain (>10) leads to not only a quick but also a smooth convergence. This algorithm allows use of gain values where other algorithms such as error back propagation even fail at all. However, the more complex the parity problem, the smaller gain needs to be used to maintain the original smoothness. This was incorporated in algorithm's adaptive logic.

Algorithm allows larger random span of initial weight set with higher parity problems, while still maintaining smooth convergence. Simply, even though higher parity is more complex, algorithm has less troubles finding gradient as flat spots and local minima have less influence.

For smaller parity problems larger initial weight set randomness radius can still provoke fluctuations in total error decrease through iterations.

Further research includes different directions. Algorithm will be added to set of online neural network training tools, previously produced by authors and available as freeware at:http://husky.engboi.uidaho.edu/nn/.

Even though highly dependent on type of the problem and network architecture, mutual dependence (relationship) among total error, search radius, learning constant and network gain would be significant step towards ultimate robustness of this learning algorithm. These parameters were heuristically established for illustrated problems. Authors believe that further employment of Genetic Algorithms (GA) might provide satisfactory robustness regardless of problem type.

V. REFERENCES

[1] Glover, F., "Future paths for integer programming and links to artificial intelligence", *Computers & Operations Research* 13, pp.533-549, 1986.

[2] Hansen, P., "*The steepest ascent mildest descent heuristic for combinatorial programming*", Talk presented at the Congresss on Numerical Methods in Combinatorial Optimization, Capri, 1986.

[3] Glover, F., *Tabu Search:part I.* ORSA Journal on Computing 1, pp.190-206, 1989.

[4] Glover, F., *Tabu Search:part II.* ORSA Journal on Computing 2, pp.4-32, 1990.

[5] de Werra, D., Hertz, A., "*Tabu search techniques: a tutorial and an application to neural networks*", OR Spektrum 11, pp.131-141, 1989.

[6] J.M. Zurada, *Artificial Neural Systems*, PWS Publishing Company, St. Paul, MN, 1995.

[7] Rumelhart, D.E., Hinton, G.E., and Williams, R.J., *Learning internal representation by error propagation*, Parallel Distributed Processing, Vol.1, pp.318-362, MIT Press, Cambridge, MA., 1986

[8] Sejnowski T.J., Rosenberg, C.R., *Parallel networks that learn to pronounce English text*, Complex Systems 1:145-168, 1987.

[9] Fahlman S.E., "*Faster-learning variations on backpropagation: An empirical study*", Proceedings of the Connectionist Models Summer School, eds. D.Touretzky, G. Hinton, and T.Sejnowski, Morgan Kaufmann, San Mateo, CA, 1988.

[10] Wilamowski, M., Torvik, L., "*Modification of gradient computation in the back propagation algorithm*", Artificial Neural Network in Engineering, Nov., St.Louis, Missou., 1993.

[11] R.J. Schalkoff, *Artificial Neural Networks*, McGraw-Hill, New York, 1997.

[12] Wilamowski, B. M., "Neural Networks and Fuzzy Systems" chapters 124.1 to 124.8 in *The Electronic Handbook*. CRC Press 1996, pp. 1893-1914.