Training Feedforward Neural Networks with a Modified Genetic Algorithm

D. I. Abu-Al-Nadi
Department of Electrical Engineering
University of Jordan
Amman - Jordan
dnadi@ju.edu.jo

Abstract - **A genetically evolved weight aggregation technique is proposed for training feedforward Neural Network. This technique does not need differentiable performance indices as most numerical optimization techniques do. Hard limiters thresholds can be used instead of the differentiable logsigmoid and tansigmoid functions used in backpropagation. The floating point genetic algorithm needs to specify the solution space in which the genetic search will try to find the optimum solution. This technique doesn't specify the solution space of the weights. It specifies the ranges of the perturbations of the weights which will aggregate to find the optimum weights for the network.**

I. INTRODUCTION

The feedforward multilayer neural network (MNN) trained by the backpropagation algorithm, is currently the most widely used neural network [1]. The research on faster backpropagation algorithms falls roughly into two categories: The first involves the development of heuristic techniques and the second focused on standard numerical optimization techniques [2]. Recently Genetic Algorithms (GA), as unconventional optimization techniques, have been emerged with Neural Networks to be an alternative to backpropagation where no differentiation of the evaluation function index with respect to the weights is needed [3,4]. Moreover, in this technique no restriction should be made on the type of the threshold functions used for the neurons, while for backpropagation the threshold functions have to be differentiable.

The regular floating point GAs need to specify the solution space in which the genetic search will try to find the optimum solution. The range of the solution space is not an easy task, to be done it needs an expert user who knows very well the problem to be solved to decide on this range [5], [6]. The proposed technique does not specify the solution space of the weights of the MNN. It specifies the ranges of the weights perturbations which will aggregate to find the optimum weights of the MNN.

Weights of the MNN will be aggregated in iterative steps. In each iteration a genetic search will be performed to find the fittest perturbation vector which will be aggregated to the weight vector of the previous iteration. The resulted new weight vector will have the best performance index. Iterations will continue until the maximum number of iterations or until the performance index exceed a prespecified tolerance. The rest of the paper is organized as follows: In section 2, the proposed algorithm is presented. In section 3 numerical examples and discussions are included to demonstrate the abilities of proposed algorithm, finally conclusions of this paper are given in section 4.

II. THE PROPOSED ALGORITHM

For multilayer networks the output of one layer becomes the input to the following layer. The equations that describe this operation are:

$$\mathbf{a}^{m+1} = f^{m+1}\left(W^{m+1}\ \mathbf{a}^{m} + \mathbf{b}^{m+1}\right) \quad \text{for } m = 0,1,\ldots, M-1 \tag{1}$$

where M is the number of layers in the network. The neurons in the first layer receive external inputs:

$$\mathbf{a}^{\circ} = P \tag{2}$$

which provides the starting point for Fig. 1. The outputs of the neurons in the last layer are taken on the network outputs:

$$\mathbf{a} = \mathbf{a}^{M} \tag{3}$$

Let us preset the weights and biases of MNN as the following parameter vector:

$$X = [X_1\ \ X_2 \ldots X_N] \tag{4}$$

$$X = \left[W^1_{1,1}\ \ W^1_{1,2}\ \ldots\ W^1_{S^1,R}\ \ b^1_1\ \ldots b^1_{S^1}\ \ W^2_{1,1} \ldots b^M_{S^M}\right] \tag{5}$$

where

$$N = S^1(R+1) + S^2\ (S^1+1) + \ldots + S^M\ (S^{M-1}+1)$$

W^m_{ij} : The weight between the *jth* neuron of the (m − 1) *th* layer and the *ith* neuron of the *mth* layer.

b^m_i : The bias of the *ith* node of the *mth* layer.

S^m : Number of neurons in the *mth* layer.

Training the network is based on aggregations of the parameter vector X as follows:

$$X^k = X^{k-1} + Y^k \tag{6}$$

where

$$X^k = \lfloor x_1^k \ \dots \ x_N^k \rfloor \tag{7}$$

is the parameter vector at the *kth* aggregation step and

$$Y^k = \lfloor y_1^k \ \dots \ y_N^k \rfloor \tag{8}$$

is the parameter vector increments at the *kth* aggregation step.

In this work GA's are used to find the best vector of increments Y such that X will be optimum. This modification gives better rates of convergence and overcomes the problem of restricting the solution space which is inherent in GA's.

The GA technique used floating-point numbers to make up the sequence of genes for each chromosome. For the selection of individuals, normalized geometric ranking was adapted for this application [6]. There are two basic types of operators, crossover and mutation. They create new solution based on existing ones. Operators for real-valued representation have been developed in [5]. Heuristic crossover and multi-non-uniform mutation operators are used. The GA must be provided with an initial population P_o, it is usually done by randomly generating solutions for the entire population within the search space. Termination mechanism for the search of the best individual should be provided, usually the search will be stopped if the maximum number of iterations is reached, or if the solution exceeds some tolerance based on the evaluation function.

The evaluation function or better named the fitness function in GA is a vital part of this type of optimization. Individuals having the highest values of fitness will stay for the next generation and the others will be discarded. GA is usually designed for maximization. If the optimization problem is to minimize a function $f(x)$, it is equivalent to maximize a function g, where $g(x) = -f(x)$, i.e.

$$\min f(x) = \max g(x) = \max \{-f(x)\} \tag{9}$$

The evaluation function is chosen to be the mean square error. The algorithm is provided with a set of examples of proper network behavior:

$$\{P_1, t_1\}, \{P_2, t_2\}, \dots, \{P_Q, t_Q\}$$

$$J(Y^k) = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q(Y^k))(\mathbf{t}_q - \mathbf{a}_q(Y^k)) = \sum_{q=1}^{Q} \mathbf{e}_q^T(Y^k)\,\mathbf{e}_q(Y^k) \tag{10}$$

where $\mathbf{a}_q(Y^k)$ is the actual output of the network at the *kth* iteration when P_q is the input.

One major advantage of GA is that the fitness function does not have to be differentiable. Even the threshold functions for each neuron f^{m+1} do not have to be differentiable in the case GA is used for optimization. The optimization process in this case is to find the best individual Y^k which will be used in Eq. (6) to minimize $J(Y^k)$.

The proposed algorithm will proceed as follows:

```
1) Initialization
     - set $max\_k = K$, $max\_i = I$.
     - set $k = 1$.
     - set $i = 0$.
     - Set $X^0$ = random between (-0.5 , 0.5)
     - Set $Y^0_{best} = [0\ 0\ \dots\ 0]$.
2) Randomly generate an initial population $P_o$ of M individuals.
3) Include the individual $Y^k_{best}$ with the initial population.
4) Evaluate the respective fitness for each individual of the population using Eqs. (6), (5), (2), (1), (3) and (10).
5) Generate an intermediate population $P_r$ by selection mechanism.
6) Generate a new population $P_{new}$ by crossover and mutations operators on $P_r$.
7) If $i < I$, set $i = i + 1$ and go to step 4; otherwise proceed to the next step.
8) Use the best solution $Y^k_{best}$ to find weights using (6) and (5).
9) set $k = k + 1$
     if $k < K$
       - set $i = 0$
       - go to step 2.
     Else
       - Use (2), (1) and (3) to find $\mathbf{a}(Y^k_{best})$.
     End
```

As proposed in the algorithm, the large weights of the NN are calculated by aggregating those small weight increments.

III. NUMERICAL EXAMPLES AND DISCUSSION

The first example is a function approximator. The MNN is supposed to approximate the following:

$$T = \sin(2\pi P) \qquad -1 \le P \le 1 \tag{11}$$

The MNN used in this case has one input, 10 neurons in the hidden layer, and one output. The range of increment vector Y was initially chosen to be $[-10,10]$ and after $K = 50$ it was reduced to $[-5,5]$. The number of iterations used was $(K \times I) = (150 \times 20) = 3000$ iterations. Both the desired output (Solid line) and the MNN output (dashed line) are shown in Fig. 1. The Learning curve, i.e., the evaluation function versus the number of iterations is shown in Fig. 2.

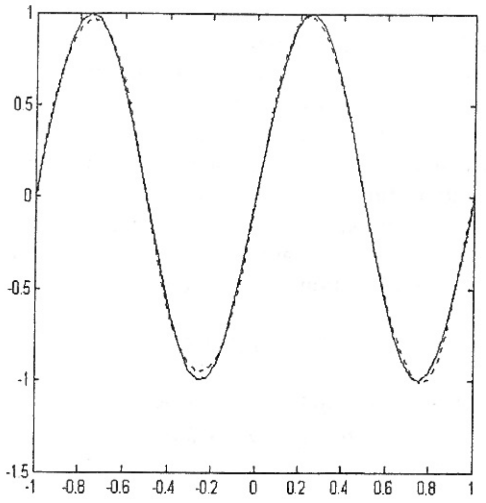

Fig. 1. The desired function (solid Line) and the MNN function approximator (dashed line) for example 1.

In the second example, it is required to identify the dynamical system which is governed by the difference equation:

$$y(n+1) = 0.3\, y(n) + 0.6\, y(n-1) + g[u(n)] \qquad (12)$$

where $g[\cdot]$ is supposed to be unknown function and has the form:

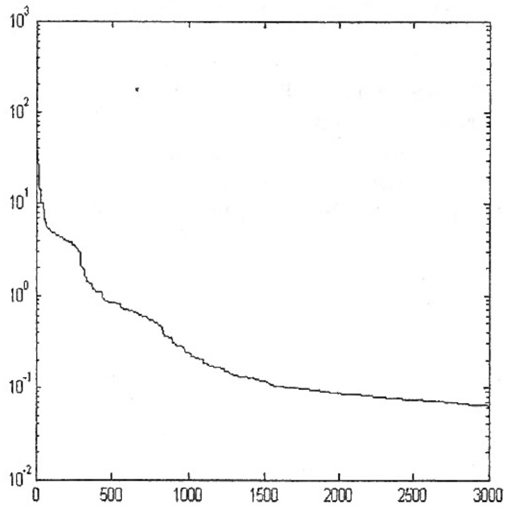

Fig. 2. The learning Curve, i.e., the evaluation function versus the number of iterations for example 1.

$$g(u) = 0.6 \sin(\pi u) + 0.3\ \sin(3\pi u) + 0.1 \sin(5\pi u) \qquad (13)$$

In order to identify the dynamical system, a model governed by the difference equation:

$$\hat{y}(n+1) = 0.3\, \hat{y}(n) + 0.6\, \hat{y}(n-1) + \hat{f}[u(n)] \qquad (14)$$

was used, where $\hat{f}[.]$ is a MNN with 30 neurons in the hidden layer. Fig. 3-5 shows the outputs of the actual system (solid lines) and the identification model (dashed lines) when the training was stopped at $n = 200, 300$ and 400 respectively, where the input $u(n) = \sin(\frac{2\pi n}{250})$. It is noticed that the identification model follows the output of the system more accurately as the training set increases.

The output of the system and the identification model (solid and dashed lines, respectively) is shown in Fig. 6 for the input given in Eq. (15) after the identification model was trained using random input uniformly distributed between $[-1,1]$.

$$u(n) = \begin{cases} \sin(\frac{2\pi n}{250}) & 1 \le n \le 250,\ \textit{and}\ \ 50 \le n \le 700 \\ \\ \sin(\frac{2\pi n}{250}) + 0.5\ \sin(\frac{2\pi n}{25}) & 250 \le n \le 500 \end{cases} \qquad (15)$$

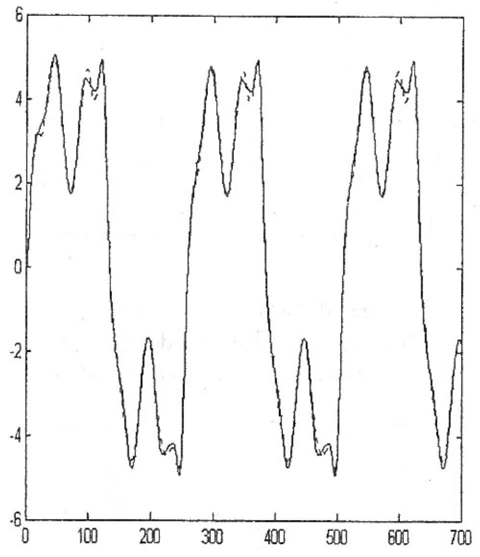

Fig. 3. Outputs of the dynamical system (solid line) and the MNN identification model (dashed line) for example 2 when the training stops at k =200.

The training was carried out for $(k \times I) = (100 \times 20) = 2000$ iterations. The MNN was trained using $[-10,10]$ as the range of the vector of increments Y and then this range was reduced to $[-5,5]$ after $K = 50$. The reduction of the range of the increment vector has the effect of adaptive learning rate in steepest descent techniques.

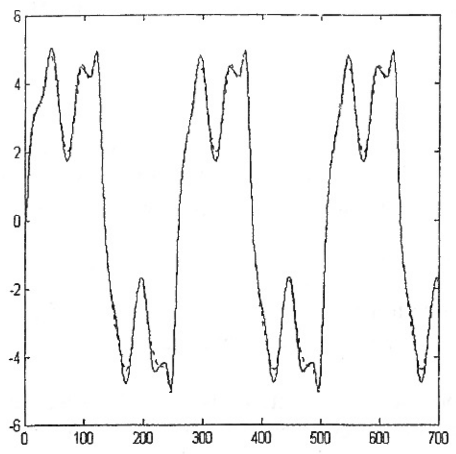

Fig. 4. Outputs of the dynamical system (solid line) and the MNN identification model (dashed line) for example 2 when the training stops at k =300.

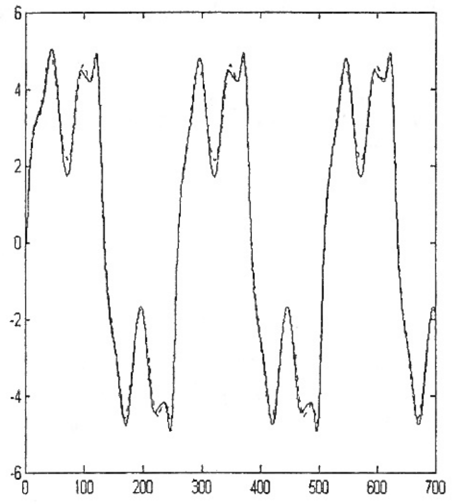

Fig. 5. Outputs of the dynamical system (solid line) and the MNN identification model (dashed line) for example 2 when the training stops at k = 400.

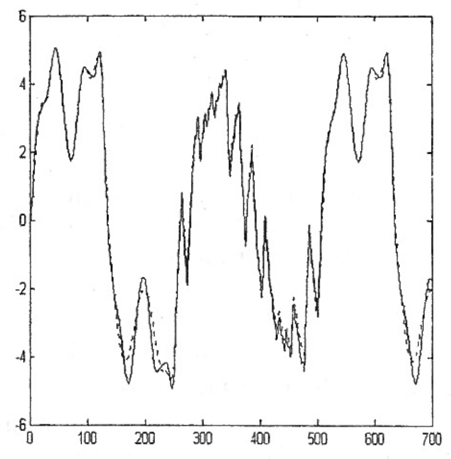

Fig. 6. Outputs of the dynamical system (solid line) and the MNN identification model (dashed line) for example 2 after training of the MNN for 2000 iterations using random input between [-1,1] and for the input given in Eq. (15).

IV. CONCLUSIONS

A modified GA technique used to train MNN has been presented. The technique adopted floating point representation of chromosomes. It does not specify the solution space of the weights, rather it specifies the ranges of the perturbations of the weights which will aggregate to find the optimum weights for the network. This modification gives better rates of convergence than the regular GA and overcomes the problem of restricting the solution space which is inherent in GA's.

V. REFERENCES

[1] F. Ham and I. Kostanic, *Principles of Neurocomputing for Science & Engineering*, McGraw-Hill, 2001.

[2] M. Hagan, H. Demuth, and M. Beale, *Neural network design*. PWS Publishing Company, Boston: MA, 1996.

[3] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks", *IEEE Trans. On Neural Networks*, vol.5, no. 1, Jan. 1994, pp.39-53.

[4] R. S. Sexton and J. N. Gupta, "Comparative evaluation of genetic algorithm and backpropagation for training neural networks", *Information Sciences*, 129, 2000, pp.45-59.

[5] Z. Michalewicz, *Genetic Algorithms+Data Structures=Evolution Programs*, 3rd edition, Springer-Verlag: Berlin Heidelberg, 1996.

[6] J.A. Joines and C.R. Houck," On the Use of Non-Stationary Penalty Functions to solve Nonlinear Constrained Optimization Problems with GA's," *Proceedings of the 1st IEEE Int. Conf. On Evolutionary Computation*, Vol. 1. Orlando, 27-29 June, 1994 pp.579-584.