Wilcoxon Neural Networks Training Using Evolutionary Optimization Methods

Yih-Lon Lin
Department of Information Engineering, I-Shou University, Kaohsiung 840, Taiwan.
Email:yllin@isu.edu.tw

***Abstract*—In this paper, two kinds of evolutionary computations including a genetic algorithm (GA) and a particle swarm optimization (PSO) are used to train the novel Wilcoxon neural networks (WNNs) for function approximation with outliers. Unlike the traditional artificial neural networks (ANNs), the objective function used in the proposed WNNs is the Wilcoxon norm instead of the total sum of squared errors, i.e., L_2-norm. The advantage of using the Wilcoxon norm is to reduce the influence of outliers on overall neural network training. Moreover, to overcome the drawback due to the back-propagation learning algorithm, we utilize the population-based optimization methods containing GA and PSO algorithms to find the suitable weights of WNNs. Finally, some numerical examples, as compared with traditional ANNs, will be provided to verify the robustness against outliers by the proposed methods.**

***Index Terms*—artificial neural networks (ANNs), Wilcoxon neural networks (WNNs), particle swarm optimization (PSO), genetic algorithms (GAs).**

I. INTRODUCTION

It is well known that neural networks have successfully been applied in many branches of science and engineering. The typical architecture of ANNs consists of several layers, i.e., the input layer, one or two hidden layers, and output layer. Each layer includes several neurons, which are usually connected with ones located in another layers by weights. In the back-propagation algorithm [1, 2], the error signal is the feedback layer-by-layer to the input layer to update the connection weights such that the given objective function is minimized. However, the serious drawback of this kind of algorithm is that the solved solution is easy to be trapped at the local minimum around initial values and difficult to escape from there. To accelerate the convergence of the algorithm, many new approaches are presented, such as by adding momentum terms to the updating law [3], or using the adaptive learning rate according to any appropriate step size selection rules, e.g., line minimization, limited line minimization, Armijo, Goldstein, Wolfe, or diminishing step size rules [4].

The genetic algorithm, initially developed by John Holland [5], is a biologically motivated search technique mimicking natural selection and natural genetics. It is a general search method in between the exhaustive search and traditional search. When the fitness landscape of the problem is unclear or riddled with many local optima, the genetic algorithm usually has good searching capability. The GA starts with a population of possible solutions, called chromosomes, to the problem. A prescribed fitness function is firstly defined for the problem, which evaluates the fitness or goodness of each chromosome. Then chromosomes with better fitness are selected for reproduction. The subsequent crossover and mutation operations are performed onto the population in order to generate a new generation of possible solutions. Such a process is repeated until some stopping criterion is met. In recent years, the related researches about GA have been presented and successfully applied in a variety of science and engineering fields [6-10].

Another evolutionary algorithm frequently used is the PSO [11-15]. It is an optimization algorithm having origins from evolutionary computation together with the social psychology principle. Essentially, PSO is dependent on stochastic processes and also uses the concept of fitness as well as GAs. In addition, it provides a mechanism that individuals in the swarm exchange

and communicate information one another, which is similar to the social behavior of insects and human beings. Because of mimicking the social sharing of information, PSO directs the individuals to search the optimal solution more efficiently [12, 13]. Another important feature of PSO is that the paradigm requires only primitive mathematical operations which can easily be implemented to computer programs. Therefore, PSO has been attracted to many real-world applications such as the analysis of human tremor, the reactive power and voltage control, the state-of-charge estimator for a battery pack, the ingredient mix optimization, milling optimization, and improvised music composition [16, 17]. Many of them have been shown that PSO techniques can perform well. To combine PSO algorithm with neural network, some efforts have been made recently. In [11], the author proposed an evolutionary system for evolving artificial neural networks, which is based on the PSO algorithm. A hybrid of GA and PSO was used for recurrent networks design problems in [15].

Robust and non-parametric smoothing is an important idea in statistics that aims to simultaneously estimate and model the underlying structure for given data. The annealing robust back-propagation learning algorithm was proposed to deal with the modeling problem under the existence of outliers in [18]. Based on fuzzy neural network structures, a robust learning algorithm was developed to reduce the outlier effects [19, 20]. In addition, the weighted error back-propagation algorithm was proposed to improve the resistance of multi-layer forward networks training to outliers in [21]. The simulation results of the above-mentioned papers completely demonstrate their satisfactory abilities on dealing with outliers. One principal method belonging to this category is the Wilcoxon approach, which is usually robust against outliers. The concept of Wilcoxon norm and linear Wilcoxon regressors are presented in Hogg [22]. This motivates us to include the Wilcoxon norm concept to the neural networks. The combination is called the Wilcoxon neural networks (WNNs) and this class of learning machines was briefly described in [23]. The contribution of this paper is to apply evolutionary computations of PSO and GA, respectively, as training algorithms for WNNs, and some simulation results, as compared with with ANNs using the traditional back-propagation and adjustable Armijo learning rate algorithms, respectively, are proposed to show the better robustness against outliers.

II. Neural Networks

A. Artificial neural networks

ANNs are a biologically motivated learning machine mimicking the structure and behavior of biological neurons and nervous system. The input-output relationship in each neuron can be described by the following equations

$$net = \sum_{i=1}^{n} w_i x_i + \theta ,$$

$$f(net) = \frac{1 - e^{-net}}{1 + e^{-net}} , \tag{1}$$

where x_i is the input to the neuron and w_i is the weight with respect to x_i, θ is called a threshold, n is the number of inputs, and $f(net)$ is referred to as a nonlinear transfer function and is only used in hidden layers in this study. When the transfer function is used in the output layer, a linear function, $f(net) = net$, is chosen. For training ANNs, a performance index or an object function must be defined previously and will be minimized by means of updating weights and biases. Usually, a total sum of squared errors is defined as an objective function and is given by

$$E_{total} := \frac{1}{2} \sum_{q=1}^{l} \sum_{k=1}^{p} \left(d_{qk} - y_{qk} \right)^2 , \tag{2}$$

where l is the number of training data, p is output number of neural networks, d_{qk} is the kth desired output of the qth training data. The objective function of (2) belongs to the corresponding L_2 norm. This kind of objective function is not a good robustness indictor for outliers. In the following, the concept of Wilcoxon norm [22] is first introduced and we use it as an objective function in training networks for solving outliers problem.

B. Wilcoxon neural networks

To define the Wilcoxon norm of a vector, we first need a score function. A score function is a function $\varphi(u):[0,1]\to\Re$ which is non-decreasing such that

$$\int_0^1 \varphi^2(u)du < \infty .$$

Usually, the score function is standardized so that

$$\int_0^1 \varphi(u)du = 0 \quad \text{and} \quad \int_0^1 \varphi^2(u)du = 1 .$$

The score associated with the score function φ is defined by

$$a_\varphi(i) = \varphi\left(\frac{i}{l+1}\right), \quad i \in \underline{l} .$$

Hence we have

$$a_\varphi(1) \le a_\varphi(2) \le \ldots \le a_\varphi(l).$$

It can be shown that the following function is a pseudo-norm (semi-norm) on $\Re^l$:

$$\|v\|_W := \sum_{i=1}^{l} a(R(v_i))v_i = \sum_{i=1}^{l} a(i)v_{(i)} ,$$

$$v := [v_1 \quad \ldots \quad v_l]^T \in \Re^l , \tag{3}$$

where $R(v_i)$ denotes the rank of v_i among $v_1, \ldots, v_l$, $v_{(1)} \le \ldots \le v_{(l)}$ are the ordered values of $v_1, \ldots, v_l$, $a(i) := \varphi[i/(l+1)]$, and $\varphi(u) := \sqrt{12}(u-0.5)$. We call $\|v\|_W$ defined in (3) the Wilcoxon norm.

There are one input layer with $n+1$ nodes, one hidden layer with $m+1$ nodes, and one output layer with p nodes. We also have p bias terms at the output nodes.

Let the input vector be

$$x := [x_1 \quad \cdots \quad x_n]^T \in \Re^n ,$$

or

$$z := [z_1 \quad \cdots \quad z_n \quad z_{n+1}]^T = [x_1 \quad \cdots \quad x_n \quad 1]^T \in \Re^{n+1} ,$$

i.e.,

$$z_i := x_i , \quad i \in \underline{n} , \quad z_{n+1} := 1 .$$

Let v_{ji} denote the connection weight from ith input node to the input of the jth hidden node. Then the input u_j and output r_j of the jth hidden node are given by, respectively,

$$u_j = \sum_{i=1}^{n+1} v_{ji} z_i , \quad z_{n+1} := 1 , \quad r_j = f_{hj}(u_j), \quad j \in \underline{m} , \tag{4}$$

where f_{hj} is the activation function of the jth hidden node.

Let w_{kj} denote the connection weight from the output of the jth hidden node to the input of the kth output node. Then the input s_k and output t_k of the kth output node are given by, respectively,

$$s_k = \sum_{j=1}^{m+1} w_{kj} r_j , \quad r_{m+1} := 1 , \quad t_k = f_{ok}(s_k), \quad k \in \underline{p} , \tag{5}$$

where f_{ok} is the activation function of the kth output node. The final output y_k of the network is given by

$$y_k = t_k + b_k , \quad k \in \underline{p} ,$$

where b_k is the bias.

For training WNNs, in this study the Wilcoxon norm of the total residuals is taken as an object function, which is given by

$$\Psi_{total} = \sum_{k=1}^{p} \Psi_k = \sum_{k=1}^{p} \sum_{q=1}^{l} a(R(\rho_{qk}))\rho_{qk} = \sum_{k=1}^{p} \sum_{q=1}^{l} a(q)\rho_{(q)k} , \tag{6a}$$

$$\rho_{qk} := d_{qk} - t_{qk} , \quad q \in \underline{l} , \quad k \in \underline{p} . \tag{6b}$$

Here $R(\rho_{qk})$ denotes the rank of the residual ρ_{qk} among $\rho_{1k}, \ldots, \rho_{lk}$, $\rho_{(1)k} \le \ldots \le \rho_{(l)k}$ are the ordered values of $\rho_{1k}, \ldots, \rho_{lk}$, and $a(i) := \varphi[i/(l+1)]$ is a score function with $\varphi(u) := \sqrt{12}(u-0.5)$. The bias term b_k, $k \in \underline{p}$, is given by the median of the residuals at the kth output node, i.e.,

$$b_k = \underset{1\le q\le l}{med}\{d_{qk} - t_{qk}\}.$$

Base on the proposed WNNs, the weights of v_{ji} and w_{kj} here need to be adjusted for minimizing the total residuals (6) according to certain evolutionary algorithms including PSO and GA approaches, respectively. For convenience, we further let Θ represent a parameter vector which contains all adjustable weights v_{ji} and w_{kj} in WNNs.

III. GAS AND PSO

A. Basic concepts of GAs

The GAs begin with generating a population that contains a number of random chromosomes. Each chromosome of the population is to represent a set of possible solution to optimization problem. In the view of using GAs to WNNs training problem, the chromosome here is referred to as the adjustable parameter vector Θ of WNNs. The population is evolved to generate a better offspring according to the size of Wilcoxon norm (6) by means of using genetic operations. To constrain the search interval, a lower bound and upper bound for each gene in the chromosome is given by $k_{\min}$ and $k_{\max}$ during evolutionary procedure. If any resulting gene is outside the defined interval, then the original remains. In addition, let N be population size, i.e., number of chromosomes in the population, l be the number of genes in a chromosome, p_c and p_m be the crossover and mutation probabilities, respectively. The variables will be used in the genetic operations.

A.1 Selection

We first need to evaluate the corresponding Wilcoxon norm for each chromosome. The N_g highly fit chromosomes are directly kept in the next generation. The rest $N - N_g$ chromosomes are then taken into the mating pool to be crossed. This completes the selection operation.

A.2 Crossover

In the mating pool, all of chromosomes are randomly divided into many pairs. Each pair, i.e., Θ_d and Θ_m, proceeds to cross. Moreover, let c be a random number selected from $[0,1]$. If $c \le p_c$, then the following crossover operation for Θ_d and Θ_m are performed:

$$\Theta_b = \Theta_m - \beta \times (\Theta_m - \Theta_d),$$
$$\Theta_s = \Theta_d + \beta \times (\Theta_m - \Theta_d), \qquad (7)$$

else $\Theta_b = \Theta_d$, $\Theta_s = \Theta_m$,

where Θ_b and Θ_s are the offspring chromosomes, $\beta \in [0,1]$ is random numbers.

A.3 Mutation

The number of executing mutation is given by $p_m \times (N - N_g) \times l$. In every mutation, we first randomly select a gene of the chromosome from $N - N_g$ chromosomes and this gene is then replaced by a random number from the search interval between $k_{\min}$ and $k_{\max}$.

The procedure that have run the above selection, crossover, and mutation is called a generation. The algorithm stops when the desired value of Wilcoxon norm is satisfied or pre-specified number of generations is achieved. The overall design steps for training WNNs based on using a GAs can be summarized as follows.

Step 1. Create a population with N chromosomes randomly.

Step 2. Evaluate the value of Wilcoxon norm of (6) for each chromosome.

Step 3. If the pre-specified number of generations G is reached or there is any chromosome with Wilcoxon norm value less then ε, then stop.

Step 4. Perform three genetic operations: selection, crossover, and mutation, respectively. If any of the resulting genes during operations is outside the interval $[k_{\min}, k_{\max}]$, then the original one is retained.

Step 5. Go back to Step 2.

B. Basic concepts of PSO

PSO is another population-based algorithm for searching global optimum. The original idea behind PSO is to simulate a simplified social behavior. It ties to artificial life, like bird flocking or fish schooling, and some common features of evolutionary computation such as fitness evaluation. For example, PSO is like GAs in which the population is initialized with random candidate solutions. The adjustments toward the best individual and the best swarm experiences are basically similar to the crossover operation in GAs. Conversely, the difference between PSO and GAs is that each potential solution, called individual or particle, is “flying” through hyperspace with a

velocity. Moreover, the particles and the swarm in the PSO have the capacity of memory, which does not exist in the population of the GAs.

Let $\Theta_{i,j}(k)$ and $V_{i,j}(k)$ denote the jth dimensional value of the vector of position and velocity of ith particle in the swarm, respectively, at step k. The PSO updating rules can be expressed as

$$\begin{aligned} V_{i,j}(k) = & w \cdot V_{i,j}(k-1) \\ & + c_1 \cdot \varphi_1 \cdot \left(\Theta^*_{i,j} - \Theta_{i,j}(k-1)\right) \\ & + c_2 \cdot \varphi_2 \cdot \left(\Theta^{\#}_{j} - \Theta_{i,j}(k-1)\right), \end{aligned} \quad (8)$$

$$\Theta_{i,j}(k) = \Theta_{i,j}(k-1) + V_{i,j}(k), \quad (9)$$

where w is the inertia weight which controls the effect of the preceding velocity at $k-1$ on the present one, Θ^*_i denotes the best position of ith particle up to step $k-1$ and $\Theta^{\#}$ denotes the best position of the whole swarm up to step $k-1$, φ_1 and φ_2 are random numbers selected from $[0,1]$, and c_1 and c_2 are the positive numbers and represent the individuality and swarm coefficients, respectively.

The PSO algorithm is first to give the swarm size and the position and velocity of each particle are initialized randomly. Each particle moves according to Eqs. (8) and (9), and the fitness function of (6) is then calculated. Meanwhile, the best positions of each particle and the swarm are recorded. Finally, if the stopping criterion is satisfied, the best position of the swarm is the final solution. The main features of PSO algorithm can be outlined as follows.

Step 1. Give the swarm size N and initialize the position and the velocity of each particle randomly.

Step 2. For each particle i, compute the corresponding fitness value of Θ_i and update the individual best position Θ^*_i if better fitness is generated.

Step 3. If the pre-specified number of generations G is reached or the fitness value of the best particle $\Theta^{\#}$ in the swarm is less then ε, then stop.

Step 4. Update the swarm best position $\Theta^{\#}$ if the fitness of the new best position is better than that of the previous one.

Step 5. For each particle, update the velocity and the position according (8) and (9). As well as GAs, if any resulting position during operations is outside the set interval $[k_{\min}, k_{\max}]$, then the original one is retained.

Step 6. Go back to Step 2.

IV. Simulation Resultion

In this section, we will compare the performance of ANNs and WNNs using various updating rules for an illustrative nonlinear regression problem with testing conditions. The updating rules for ANNs are the traditional back-propagation algorithm and the back-propagation algorithm with Armijo rule [4], respectively. Moreover, for training WNNs, the GAs and PSO algorithms introduced above are employed. In order for "fair" comparisons, the simulation machines will use the same set of parameters. For example, for ANNs and WNNs, there are one input neuron, two hidden layers with five and ten neurons, respectively, and one output neuron. The activation function used in the hidden nodes is the bipolar sigmoidal function, and the activation function of the output node is the linear function with unit slope. Besides, for GAs and PSO algorithms, we use the same search space and population size (swarm size). The software to implement the above numerical algorithms is the Visual C++ 6.0 running on Microsoft Windows XP, Pentium IV 2.4 GHz platform.

The true function that will be learned is given by the following complex function

$$\begin{aligned} & y(x) = \sin(\pi x) + 0.5\sin(2\pi x) + 2\cos(3\pi x), \\ & x \in [-1, 1]. \end{aligned}$$

The parameters used in GAs and PSO algorithms are set to $N = 20$, $G = 15000$, $[k_{\min}, k_{\max}] = [-10, 10]$, $N_g = 2$, $P_c = 0.8$, $P_m = 0.005$, $w = 0.45$, and $c_1 = c_2 = 2$. In the simulations, there are fifty training data uniformly generated from the true functions and in which

there are three, five, and eight training samples intentionally replaced by artificial outliers. Figures 1(a)(b)(c) show the simulation results by ANNs with the back-propagation algorithm and back-propagation algorithm with Armijo rule for three, five, and eight artificial outliers, respectively. The simulation results by WNNs with GAs and PSO algorithms are then displayed in Figures 2(a)(b)(c). From these figures, it can easily be seen that WNNs perform with GAs and PSO algorithms better than ANNs with the back-propagation algorithm and Armijo rules for different outliers. The predictive results by WNNs with GAs and PSO algorithms are almost indistinguishable from the true function, and are all robust against outliers.

V. CONCLUSION

In this paper, we have successfully applied GAs and PSO algorithms to train the the Wilcoxon neural networks. These population-based optimization methods have many advantages over the traditional back-propagation learning algorithm, for example, easy to escape from the local minimum around initial values and more efficient to solve the optimal solution for complex functions. The main difference between WNNs and ANNs is the use of the Wilcoxon norm to replace the general the total sum of squared errors. To deal with function approximations with outliers, this change can efficiently reduce the effect of outliers. From simulation results including three, five, and eight artificial outliers for complex function approximations, it is concluded that WNNs with GAs and PSO algorithms proposed in this paper have good robustness against outliers than ANNs with the traditional back-propagation algorithms and Armijo rules.

REFERENCE

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359-366, 1989.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group Eds. MIT Press, Cambridge, Massachusetts, vol. 1, Foundations, 1986, pp. 318-362.

[3] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, no. 4, pp.295-308, 1988.

[4] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, New York, 1999.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[6] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, Massachusetts, 1996.

[7] R. L. Haupt and S. E. Haupt, *Practical Genetic Algorithms*. Wiley, New York, 1998.

[8] D. J. Montana and L. Davis, "Training feedforward neural networks using genetics algorithms," in *Proceedings of the International Joint Conference on Artificial Intelligence*, Morgan Kanufmann, 1989, pp. 762-767.

[9] D. E. Goldberg, *Genetic Algorithm in Search, Optimization and Machine Learning*. Addsion-Wesley, Massachusetts, 1989.

[10] D. A. Coley, *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific, Singapore, 1999.

[11] C. Zhang, H. Shao, and Y. Li, "Particle swarm optimization for evolving artificial neural network," in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, Nashville, Tennessee, 2000, vol. 4, pp. 2487-2490.

[12] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, Perth, Australia, 1995, vol. 4, pp. 1942-1948.

[13] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of IEEE International Symposium on Micro Machine and Human Science*, Nagoya, Japan, 1995, pp. 39-43.

[14] R. C. Eberhart and Y. H. Shi, "Particle swarm optimization: developments, applications and resources," in *Proceedings of IEEE International Conference on Evolutionary Computation*, Seoul, Korea, 2001, vol. 1, pp. 81-86.

[15] C. F. Juang, "A hybrid of genetic algorithm

and particle swarm optimization for recurrent network design," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 34, no 2, pp. 997-1006, 2004.

[16] J. Kennedy, Eberhart R, and Shi Y, *Swarm Intelligence*. Morgan Kaufmann, 2001.

[17] Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, 2006.

[18] C. C. Chuang, S. F. Su, J. T. Jeng, and C. C. Hsiao, "Annealing robust backpropagation (ARBP) learning algorithm," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1067-1077, 2000.

[19] H. H. Tsai and P. T. Yu, "On the optimal design of fuzzy neural networks with robust learning for function approximation," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 30, no. 1, pp. 217-223, 2000.

[20] W. Y. Wang; T. T. Lee, C. L. Liu, C. H. Wang, "Function approximation using fuzzy neural networks with robust learning algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 27, no. 4, pp. 740-747, 1997.

[21] W. Zhao, D. Chen, and S, Hu, "Detection of outlier and a robust BP algorithm against outlier," *Computers and Chemical Engineering*, vol. 28, no. 8, pp. 1403-1408, 2004.

[22] R. V. Hogg, J. W. McKean, and A. T. Craig, *Introduction to Mathematical Statistics*. Prentice-Hall, New Jersey, 2005.

[23] J. G. Hsieh, Y. L. Lin, and J.H. Jeng, "Preliminary study on Wilcoxon learning machines," *IEEE Transactions on Neural Networks*, vol. 19, no. 2, pp. 201-211, 2008.

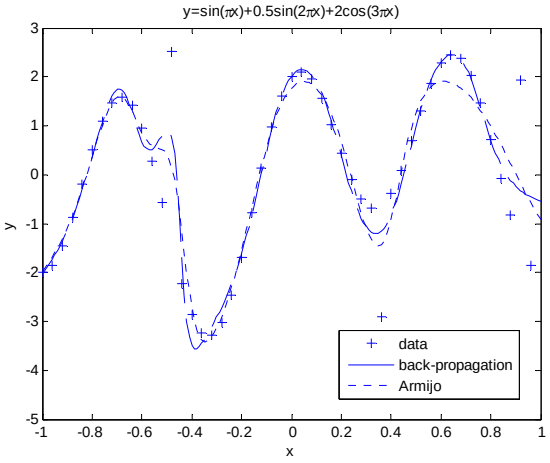

Figure 1(a). Simulation results with three outliers for ANNs.

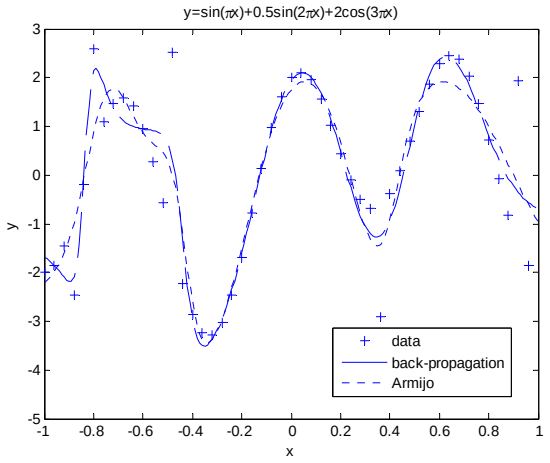

Figure 1(b). Simulation results with five outliers for ANNs.

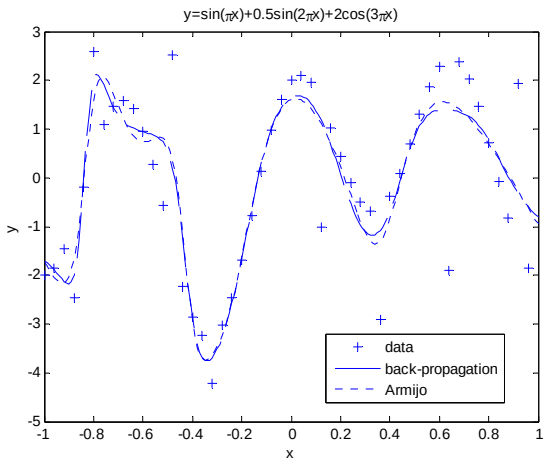

Figure 1(c). Simulation results with eight outliers for ANNs.

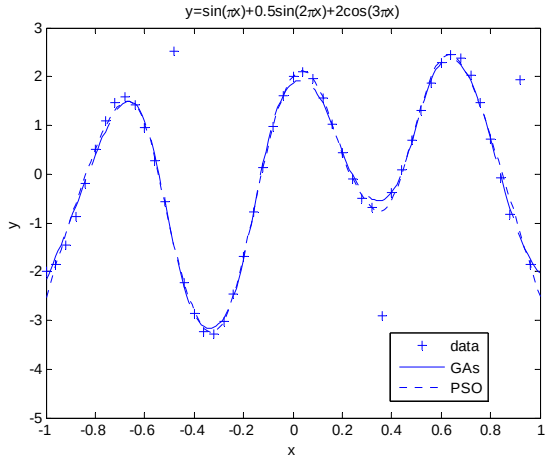

Figure 2(a). Simulation results with three outliers for WNNs.

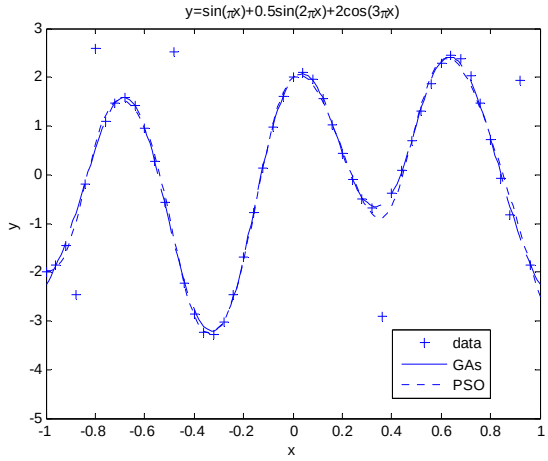

Figure 2(b). Simulation results with five outliers for WNNs.

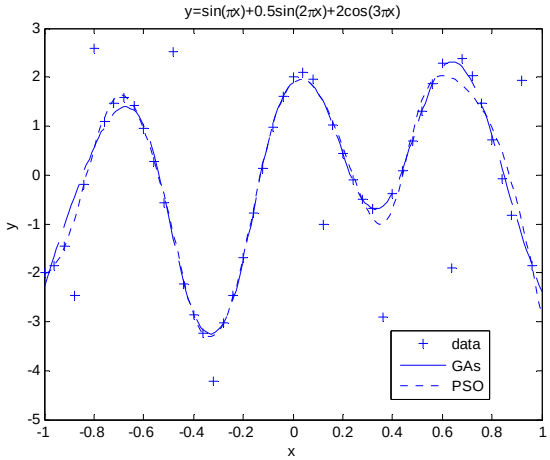

Figure 2(c). Simulation results with eight outliers for WNNs.