International Journal of Innovative
Computing, Information and Control
Volume 7, Number 10, October 2011

ICIC International ©2011 ISSN 1349-4198
pp. 5839–5850

FEED-FORWARD NEURAL NETWORKS TRAINING: A COMPARISON BETWEEN GENETIC ALGORITHM AND BACK-PROPAGATION LEARNING ALGORITHM

Zhen-Guo Che[1], Tzu-An Chiang[2] and Zhen-Hua Che[3,*]

[1]Institute of Information Management
National Chiao Tung University
No. 1001, University Rd., Hsinchu 300, Taiwan
bcc102a@hotmail.com

[2]Department of Business Administration
National Taipei College of Business
No. 321, Sec. 1, Jinan Rd., Taipei 100, Taiwan
phdchiang@gmail.com

[3]Department of Industrial Engineering and Management
National Taipei University of Technology
No. 1, Sec. 3, Chung-Hsiao E. Rd., Taipei 106, Taiwan
*Corresponding author: zhche@ntut.edu.tw

Received March 2010; revised July 2010

Abstract. *This study discusses the advantages and characteristics of the genetic algorithm and back-propagation neural network to train a feed-forward neural network to cope with weighting adjustment problems. We compare the performances of a back-propagation neural network and genetic algorithm in the training outcomes of three examples by referring to the measurement indicators and experiment data. The results show that the back-propagation neural network is superior to the genetic algorithm. Also, the back-propagation neural network has faster training speed than the genetic algorithm. However, the back-propagation neural network has the shortcoming of overtraining, while the genetic algorithm does not. The experiment result proves that the back-propagation neural network yields better outcomes than the genetic algorithm.*

Keywords: Back-propagation neural network, Genetic algorithm, Feed-forward neural network

1. Introduction.

Artificial neural networks are biologically inspired classification algorithms that consist of an input layer of nodes, one or more hidden layers and an output layer. Each node in a layer has one corresponding node in the next layer, thus creating the stacking effect [1]. Artificial neural networks are the very versatile tools and have been widely used to tackle many issues [2-6]. Feed-forward neural networks (FNN) are one of the popular structures among artificial neural networks. These efficient networks are widely used to solve complex problems by modeling complex input-output relationships [7,8].

However, FNNs often end up being over trained. They adopt trials-and-errors to seek possible values of parameters for convergence of the global optimum. The learning process of an FNN cannot guarantee the global optimum, sometimes trapping the network into the local optimum. The back-propagation learning algorithm (BPLA) is a widely used method for FNN learning in many applications. It has the great advantage of simple implementation [9]. In addition, many studies have indicated that genetic algorithms (GA) can be successfully applied to identity global optimizations of multidimensional functions [10,11]. GAs are widely applied in the optimization of the parameters spaces of neural networks. Traditional optimization techniques can also determine the number of network parameters, such as network connection weightings; however, they are not able to control the parameter optimizations in the absence of gradient information. In contrast, GAs are able to resolve this issue. They can be widely used in all aspects of neural networks. Examples include the determination of network structures (the number of input nodes, the number of hidden layers, the number of nodes in the hidden layers, and the number of output nodes), and the optimization of parameters such as the selection of node switch functions, connection weightings and learning speeds [12].

Sexton and Gupta [13] pointed out that GAs are able to process a large variety of data types, as soon as they can be expressed with bits of fixed lengths. GAs also yield good outcomes for optimization. The above study proved that GAs are superior to BPLAs. However, their study does not specify the sources and quantities of their data. In other words, they do not use different data types to train and compare a GA and BPLA.

Based on the above analysis, this study focuses on performances in the training of FNN with BPLAs and GAs. The data types used to compare the performances of these two algorithms are Sin function, Iris plants and Diabetes. This paper is organized as follows: Section 2 introduces the concepts of BPLA and GA; Section 3 expresses the procedures of BPLA and GA for FNN training; Sections 4 and 5 describe the data types, results and analyses; finally, conclusions are given in Section 6.

2. **Literature Review.**

2.1. **Back-propagation neural network.**

BPLAs were proposed by Rumelhart et al. [14]. They have since become famous learning algorithms among ANNs. In the learning process, to reduce the inaccuracy of ANNs, BPLAs use the gradient-decent search method to adjust the connection weights. The structure of a back-propagation ANN is shown in Figure 1. The output of each neuron is the aggregation of the numbers of neurons of the previous level multiplied by its corresponding weights. The input values are converted into output signals with the calculations of activation functions [15]. Back-propagation ANNs have been widely and successfully applied in diverse applications, such as pattern recognition, location selection and performance evaluations.

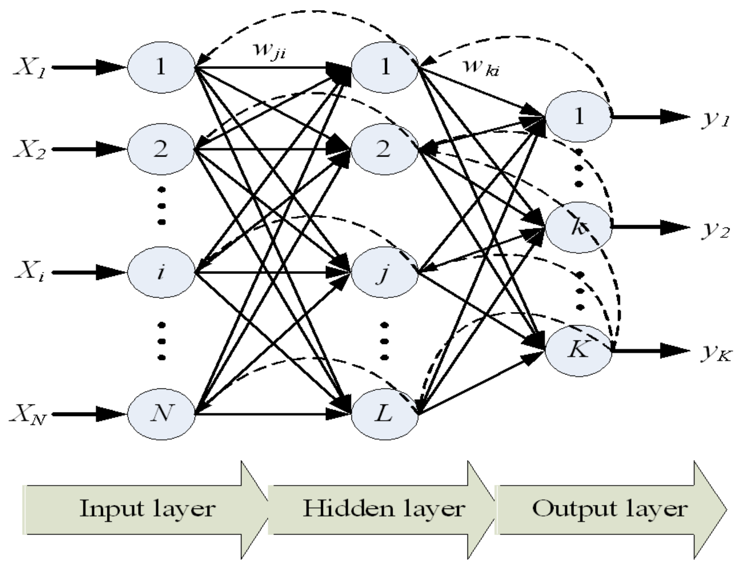

FIGURE 1. Back-propagation ANN

Wang and Bai [16] applied a back-propagation ANN to assess and predict the quality of water by overcoming problems associated with heavy workloads and excessive subjectivity in traditional assessment methods. The results were objective and accurate. There is no need to carry out complex pre-processing in order to monitor data and the workload is light. Wu et al. [17] established a back-propagation ANN for the identification of lifespan distributions of for machinery products. The simulation results show that back-propagation ANNs are suitable for identifying the lifespan distribution models when large ranges of parameters exist. Bongards [18] used back-propagation to predict and control the improvement in density of ammonia and total nitrogen in waste water processing plants. Chou et al. [3] and Xiao and Chandrasekar [19] also successfully applied back-propagation ANNs for patterning of e-commerce customers patterning and rainfall estimation from radar data separately. Che [20] employed a back-propagation ANN for product and mold cost estimation of plastic injection molding.

2.2. **Genetic algorithms.** GAs were proposed by Holland in 1975, as optimization search mechanisms for natural selection. The fundamental principle is to imitate the law of natural selection in nature by choosing parents with better characteristics. Random interactions of bit information are carried out, in the hope of generating offspring superior to parents. The continuous repeats see the birth of optimal species with the strongest adaptability. Chang et al. [21] applied a GA and the testing design principles to develop the optimal design model for underground wells. The outcome shows that the monitoring well should be located in the area adjacent to the water pumping well and the water pumping well should be deployed in the region with smaller transmission. This reduces the uncertainties of estimation parameters. Jiang and Yuan [22] proposed credit valuations based on GAs and neural networks, and controlled the occurrence of Type-2 errors of commercial banks that lead to heavy losses in the design of credit valuations by using GA adaptability functions. The results show that the model is high in reliability and applicability. Jiang and Zhai [23] adopted neural networks and a GA to design thinking patterns of NPC roles in order to establish a self-learning model that is more autonomous and intelligent than the traditional NPC roles. In addition, GAs have been used to solve complicated problems effectively. For example, refer to Wu et al. [24], Che [25-27], Che and Wang [28], Chang [29], Sha and Che [30], Che and Wang [31] and Che and Chaing [32].

3. BPLA and GA for FNN Training.

3.1. **BPLA for FNN training.** The BPLA structure is based mainly on batch learning. Through iterations and modifications, the average square errors of test data are used to determine the optimal weights and biases. The procedure of a BPLA is described step by step as follows:

Step 1: Normalize the data: The obtained data is mapped to the bound $[0, 1]$, so as to adjust the defined range of attributes and avoid the saturation of neurons. The normalization of data is computed by $V_{new} = (V_{old} - MinV) / (MaxV - MinV) \times (D_{\max} - D_{\min}) + D_{\min}$ [33,34]. $MinV$ is the minimal value of the variables, $MaxV$ is the maximal value of the variables, $D_{\max}$ is the maximum value after normalization, $D_{\min}$ is the minimum value after normalization, V_{new} is the new value after normalization, and V_{old} is the old value before normalization.

Step 2: Set the network parameters: Number of hidden layers: In general, one or two hidden layers produce better convergence. Too few or too many hidden layers yield poor results [35]. This study uses one or two hidden layers.
Number of units in hidden layers: The larger the number of units in the hidden

layers, the slower the speed of convergence becomes. Too small of a number of units in the hidden layer is not sufficient to respond to the interactions between input variables. On the other hand, a large number of units generates smaller error but the complexity of the network leads to slow convergence [35,36].

Learning rate (η): In general, too fast or too slow of a learning rate is detrimental to network convergence. This study selects values between 0.1 and 1.0 to test the networks [37].

Momentum coefficient (α): Momentum coefficients also affect network convergence. To avoid error fluctuations in the convergence process, this study selects the values between 0.01 and 1.0 [35] to test the networks.

Transfer function: Sigmoid function $f(x) = 1/(1 + e^{-x})$ is the transfer function [38] in this study and its value is in the interval of [0, 1].

Step 3: Input the data of the training examples: The related values of input and output values of the training examples are inputted into the BPLA.

Step 4: Calculate the neuron's output signal: Each neuron's output signal is calculated by $net_j = \sum_{i=1\sim m} w_{ji}x_i + b_j$, and sigmoidal function is used to convert net_j for each neuron in the hidden layers. net_j is the output of neuron j. w_{ji} is the weight on the connection from neuron i to j; x_i is the input signal of the neuron i; and b_i is the bias of neuron j.

The signal of the output layer can expressed by $net_k = TV_k + \delta_k^L$. TV_k is the target value of the output neuron k and δ_k^L is the error of neuron k. The error of each hidden layer is $\delta_j^l = \sum_{i=1\sim I} \delta_i^{l+1} w_{ji}^l f'(net_j^l)$ and the incremental change for every weight for each learning interaction is computed by $\Delta w_{ji}^l = \eta \delta_j^{l+1} f'(net_j^l) + \alpha \Delta w_{ji}^{l-1}$. $f'(.)$ is the first derivative of the sigmoid function.

Step 5: Calculate the error values: Steps 3-5 are repeated until the network is converged [35]. The error is calculated by $SSE = \sum_{i=1\sim n} (T_i - Y_i)^2$. T_i is the actual value and Y_i is the estimated value.

3.2. **GA for FNN training.** Computation steps of the GA are as follows:

Step 1: Randomly generate initial population: N chromosomes are generated randomly in order to serve as the initial population in the evolutionary process. Their genes are of real numbers in value and are generated randomly, with the values ranging between 0.3 and -0.3 [39].

Step 2: Calculate the fitness values: Fitness function is set up to perform the evolutionary process. The fitness function of the experiment is feed-forward calculation for the computation of SSE.

Step 3: Select the individuals: The smaller the SSE, the higher the probability of the chromosomes being passed down to the next generation [40]. Calculate the fitness function value f_i for each individual in the population and the aggregation of all the fitness function values f_{sum}. A random number r_{rand} is selected in the range of $[0, f_{\min}]$. r_{rand} is deducted by the fitness function value in the population: $r_{rand} = r_{rand} - f_i$. This is repeated until the value of r_{rand} minus f_k is smaller or equal to zero. The k^{th} individual will be duplicated into the mating pool. The selection is repeated until the number of individuals in the mating pool is the same as the number of individuals in the population.

Step 4: Cross individuals: Two individuals are selected randomly from the mating pool as the parent and the random selection of a gene is used as a crossover point. The exchange of the parent genes to the right of the crossover point generates two new individuals, which marks the end of the single-point crossover [41].

Step 5: Mutate gene: The approximate optimal solutions can be found quickly in order to set up the mutation rate as a parameter to control mutation probability. The mutation rate is a real number between 0 and 1 [42].

Step 6: Judge the termination conditions: The stop condition is set as the generation number. If the set generation number of is reached, it will stop.

4. **Data Description.** This study works on three data types, Sin function, Iris and Diabetes. These data types are described as follows:

Sin function: There are a total of 315 Sin function data entries. Kumar [35] divided data into the data type 60%-20%-20%, 189 training entries, 63 test entries and 63 validation entries. The input and output are separate neurons. The network structure is illustrated in Figure 2. The constant modifications of weights in the training of BPLA enable the derived curve to simulate the original lines.

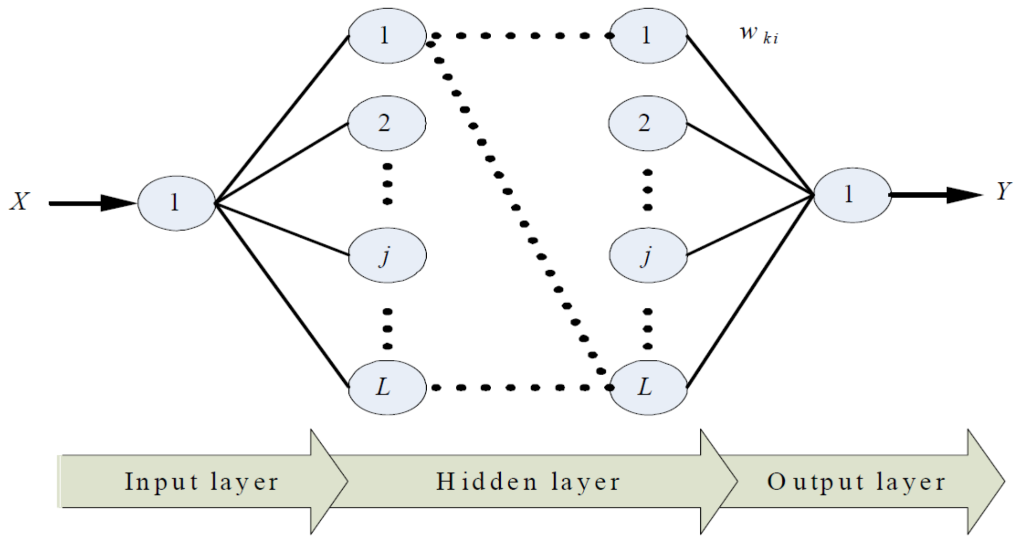

FIGURE 2. Network structure for Sin function

Iris plants: There are a total of 150 data entries. Data is divided into the following types: 60%-20%-20%, 90 training entries, 30 test entries and 30 validation entries. The data records the characteristics of iris flowers. There are 4 attributes and 3 classifications. These 4 attributes are used to predict which of the three types a certain flower belongs to. The network structure is illustrated in Figure 3. The three types of iris flowers are Iris Setosa, Iris Versicolour and Iris Virginica. The four attributes are sepal length, sepal width, petal length and petal width.

Diabetes: Family diabetes is determined according to the *UCI Machine Learning Repository*'s chosen data. A total of 8 attributes (numbers of pregnancy, blood sugar levels, diastolic blood pressure, subcutaneous fat thickness of brachial triceps muscle, serum insulin levels, Body Mass Index, family diabetes history and ages) are used to determine two types. There are a total of 768 data entries, 460 training entries, 154 test entries and 154 validation entries. The network structure is illustrated in Figure 4. Therefore, eight input neurons are designed to represent different attributes and two output neurons. They represent the presence or absence of hereditary diabetes.

The related parameter settings in the BPLA and GA are: learning rate = 0.25, 0.5, 0.9; momentum = 0.01, 0.5, 0.9; population size = 20; crossover rate = 0.7, 0.8, 0.9; mutation rate = 0.01, 0.02, 0.05; iterations/generations = 1000, 2000, 3000, 10000, 20000, 30000.

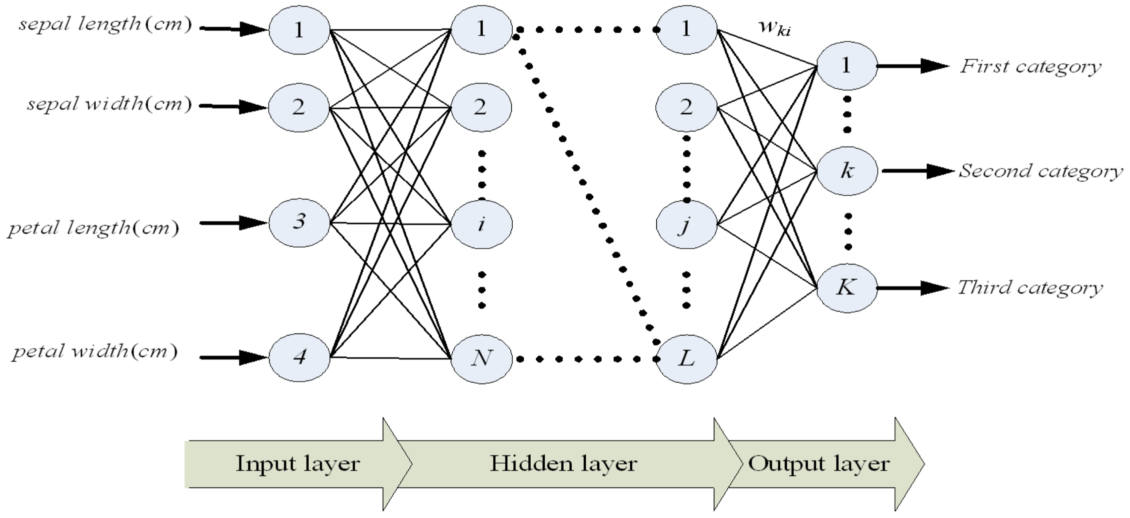

Figure 3. Network structure for Iris plants

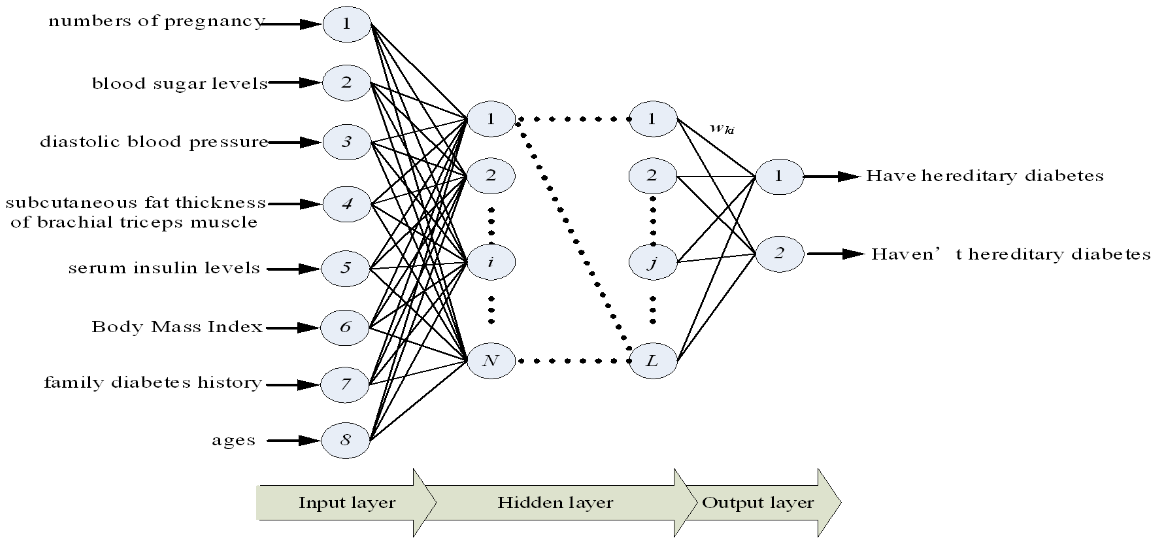

Figure 4. Network structure for Diabetes

5. **Results and Analyses.**

After searching for the best parameters, as described above, this study uses two performance indicators, mean square error (MSE) and CPU time, to verify the performances of the BPLA and GA. MSE is the evaluation index [14,40] and its formula is expressed as $MSE = \sqrt{\sum_{i=1\sim n} (T_i - E_i)^2}/n$. When iterations/generations reach the set maximum iterations/generations, the algorithm terminates the training. Each algorithm is run five times with the same number of hidden layers.

Table 1. Experiments of learning rate and momentum for the BPLA in Sin function

Iterations	Learning rate	0.9			0.5			0.25		
	Momentum	0.01	0.5	0.9	0.01	0.5	0.9	0.01	0.5	0.9
1000	Training MSE	0.0262	0.1397	0.1702	0.0471	0.1653	0.1625	0.1288	0.1641	0.1773
	Testing MSE	0.1000	0.1510	0.1730	0.1005	0.1684	0.1670	0.1455	0.1677	0.1791
2000	Training MSE	0.0235	0.1508	0.1672	0.0500	0.1608	0.1619	0.1283	0.1520	0.1741
	Testing MSE	0.1001	0.1616	0.1704	0.1025	0.1651	0.1659	0.1450	0.1614	0.1763
3000	Training MSE	0.0237	0.1337	0.1611	0.0585	0.1584	0.1559	0.1229	0.1568	0.1663
	Testing MSE	0.1000	0.1474	0.1657	0.1075	0.1635	0.1617	0.1421	0.1624	0.1697

Table 2. Experiments of iterations for the BPLA in Sin function

Iterations	1000	2000	3000	10000	20000	30000
Training MSE	0.0262	0.0235	0.0237	0.0236	0.0235	0.0238
Testing MSE	0.1000	0.1000	0.1000	0.1000	0.1001	0.1000

Table 3. Experiments of crossover rate and mutation rate for the GA in Sin function

Generations	Crossover	0.7			0.8			0.9		
	Mutation	0.01	0.02	0.05	0.01	0.02	0.05	0.01	0.02	0.05
1000	Training MSE	0.0159	0.0245	0.0523	0.0233	0.0261	0.0464	0.0196	0.0253	0.0493
	Testing MSE	0.1479	0.1467	0.1252	0.1473	0.1521	0.1288	0.1476	0.1494	0.1321
2000	Training MSE	0.0155	0.0238	0.0528	0.0217	0.0271	0.0412	0.0188	0.0214	0.0418
	Testing MSE	0.1471	0.1433	0.1255	0.1502	0.1733	0.1206	0.1409	0.1503	0.1433
3000	Training MSE	0.0153	0.0239	0.0644	0.0211	0.0267	0.0432	0.0192	0.0233	0.0455
	Testing MSE	0.1396	0.1477	0.1345	0.1498	0.1676	0.1308	0.1434	0.1501	0.1332

Table 4. Experiments of generations for the GA in Sin function

Generations	1000	2000	3000	10000	20000	30000
Training MSE	0.0158	0.0154	0.0153	0.0159	0.0153	0.0160
Testing MSE	0.1479	0.1470	0.1396	0.1480	0.1415	0.1513

Table 5. Performance comparisons between the BPLA and GA in Sin function

Hidden number	BPLA		GA		Hidden number	BPLA		GA	
S1	MSE	Time (s)	MSE	Time (s)	S1	MSE	Time (s)	MSE	Time (s)
4	0.0262	3.08	0.0159	8.86	10	0.0241	4.57	0.0250	7.88
5	0.0204	3.53	0.0153	8.34	11	0.0212	4.55	0.0221	7.42
6	0.0143	3.96	0.0184	8.53	13	0.0188	6.75	0.0207	8.65
7	0.0192	4.32	0.0192	7.38	15	0.0160	7.83	0.0188	9.78
8	0.0256	6.53	0.0209	6.76	20	0.0182	8.23	0.0182	8.35
9	0.0243	7.78	0.0256	7.42	30	0.0185	8.98	0.0198	11.93

5.1. **Sin function fit.** Tables 1 and 2 show that when learning rate = 0.9, momentum = 0.01 and maximum iterations = 2000, the MSE is minimized for training and testing in the back-propagation ANN. Similarly, in Tables 3 and 4, as the best parameters in the GA reach crossover rate = 0.7, mutation rate = 0.01 and generations = 3000, the MSE is minimized, the averages are aggregated and compared, as shown in Table 5. Table 5 indicates that the BPLA is significantly superior to the GA. Given the same number of neurons, the CPU time to run the BPLA is shorter than that to run the GA. In terms of error performance measurement, the BPLA reports lower errors than the GA.

5.2. **Iris plant classification.** The classification of IRIS applies a Plants Database to train the FNN. The best parameters in the BPLA are: iterations = 1000, learning rate = 0.5, momentum = 0.01 and those in GA are: generations = 2000, crossover rate = 0.7, mutation = 0.02. The results are shown in Tables 6-9. Table 10 shows the comparisons between the BPLA and GA in regard to MSE and CPU time and indicates that the GA is superior to the BPLA in MSE, but the GA requires longer CPU time than the BPLA.

5.3. **Diabetes determination.** The best parameters in that BPLA are as follows: learning rates = 0.9, momentum coefficients = 0.01 and iterations = 20000 (Tables 11 and 12). The best parameters in GA are: crossover rate = 0.7, mutation = 0.02, generations = 3000 (Tables 13 and 14). As shown in Table 15, given the same number of neurons, the

Table 6. Experiments of learning rate and momentum for the BPLA in Iris plants

Iterations	Learning rate	0.9			0.5			0.25		
	Momentum	0.01	0.5	0.9	0.01	0.5	0.9	0.01	0.5	0.9
1000	Training MSE	0.0620	0.0446	0.0566	0.0398	0.0615	0.0532	0.0556	0.0681	0.0544
	Testing MSE	0.0682	0.0568	0.0685	0.0540	0.0748	0.0677	0.0628	0.0756	0.0681
2000	Training MSE	0.0571	0.0501	0.0568	0.0406	0.0616	0.0505	0.0515	0.0596	0.0537
	Testing MSE	0.0681	0.0577	0.0683	0.0549	0.0755	0.0664	0.0606	0.0724	0.0675
3000	Training MSE	0.0566	0.0504	0.0519	0.0470	0.0612	0.0580	0.0574	0.0593	0.0554
	Testing MSE	0.0685	0.0655	0.0688	0.0575	0.0695	0.0641	0.0637	0.0714	0.0665

Table 7. Experiments of iterations for the Iris plants

Iterations		1000	2000	3000	10000	20000	30000
Evaluation indices	Training MSE	0.0398	0.0406	0.0470	0.0453	0.0412	0.0409
	Testing MSE	0.0504	0.0549	0.0575	0.0517	0.0522	0.0523

Table 8. Experiments of crossover rate and mutation rate for the GA in Iris plants

Generations	Crossover	0.7			0.8			0.9		
	Mutation	0.01	0.02	0.05	0.01	0.02	0.05	0.01	0.02	0.05
1000	Training MSE	0.0608	0.0394	0.0554	0.0602	0.0438	0.0509	0.0548	0.0667	0.0533
	Testing MSE	0.0668	0.0529	0.0671	0.0733	0.0558	0.0663	0.0615	0.0741	0.0667
2000	Training MSE	0.0559	0.0309	0.0556	0.0603	0.0491	0.0494	0.0501	0.0584	0.0526
	Testing MSE	0.0665	0.0418	0.0669	0.0735	0.0565	0.0650	0.0593	0.0709	0.0661
3000	Training MSE	0.0556	0.0460	0.0508	0.0599	0.0494	0.0568	0.0563	0.0578	0.0543
	Testing MSE	0.0672	0.0564	0.0666	0.0681	0.0644	0.0628	0.0617	0.0695	0.0652

Table 9. Experiments of generations for the GA in Iris plants

Generations	1000	2000	3000	10000	20000	30000
Training MSE	0.0394	0.0309	0.0388	0.0322	0.0311	0.0330
Testing MSE	0.0528	0.0418	0.0516	0.0467	0.0447	0.0465

Table 10. Performance comparisons between the BPLA and GA in Iris plants

Hidden number	BPLA		GA	
S1	MSE	Time (s)	MSE	Time (s)
4	0.0398	2.30	0.0309	4.40
6	0.0318	3.30	0.0298	6.82
8	0.0311	4.60	0.0293	8.87
10	0.3540	5.71	0.0315	12.32
12	0.3090	16.42	0.0306	18.80
14	0.4050	10.88	0.0319	26.70
16	0.3600	19.10	0.0311	38.30

Table 11. Experiments of learning rate and momentum for the BPLA in Diabetes

Iterations	Learning rate	0.9			0.5			0.25		
	Momentum	0.01	0.5	0.9	0.01	0.5	0.9	0.01	0.5	0.9
1000	Training MSE	0.0401	0.0632	0.0905	0.04165	0.0406	0.0426	0.0432	0.0519	0.0667
	Testing MSE	0.4816	0.7123	0.5143	0.5124	0.4895	0.5390	0.5432	0.6009	0.5266
2000	Training MSE	0.0388	0.0583	0.0760	0.0402	0.0403	0.0463	0.0416	0.0493	0.0611
	Testing MSE	0.4789	0.6834	0.5092	0.5011	0.4823	0.5452	0.5233	0.5829	0.5272
3000	Training MSE	0.0363	0.0443	0.0708	0.0369	0.0399	0.0415	0.0375	0.0421	0.0561
	Testing MSE	0.4673	0.5437	0.5172	0.4830	0.4808	0.5157	0.4987	0.5123	0.5164

TABLE 12. Experiments of iterations for the BPLA in Diabetes

Iterations	1000	2000	3000	10000	20000	30000
Training MSE	0.0401	0.0388	0.0363	0.0319	0.0189	0.0202
Testing MSE	0.4816	0.4789	0.4673	0.4208	0.3192	0.3674

TABLE 13. Experiments of crossover rate and mutation rate for the GA in Diabetes

Generations	Crossover	0.7			0.8			0.9		
	Mutation	0.01	0.02	0.05	0.01	0.02	0.05	0.01	0.02	0.05
1000	Training MSE	0.0471	0.0248	0.0399	0.0557	0.0354	0.0413	0.0424	0.0383	0.0403
	Testing MSE	0.5682	0.3803	0.5511	0.6120	0.5772	0.7123	0.5617	0.5432	0.5524
2000	Training MSE	0.0289	0.0286	0.0339	0.0351	0.0357	0.0343	0.0301	0.0436	0.0369
	Testing MSE	0.5599	0.4272	0.5356	0.5752	0.6106	0.5227	0.5351	0.7321	0.5636
3000	Training MSE	0.0218	0.0229	0.0334	0.0387	0.0404	0.0367	0.0298	0.0405	0.0352
	Testing MSE	0.5363	0.3806	0.0533	0.6431	0.7086	0.6675	0.4636	0.7096	0.5776

TABLE 14. Experiments of generations for the GA in Diabetes

Generations	1000	2000	3000	10000	20000	30000
Training MSE	0.0248	0.0285	0.0229	0.0285	0.0286	0.0304
Testing MSE	0.3802	0.4272	0.3805	0.4234	0.4335	0.4735

TABLE 15. Performance comparisons between the BPLA and GA in Diabetes

Hidden number	BPLA		GA	
S1	MSE	Time (s)	MSE	Time (s)
4	0.33289	20.20000	0.53652	33.70000
5	0.31927	22.10000	0.38058	37.30000
6	0.28892	26.30000	0.37654	40.60000
7	0.32234	27.00000	0.30134	44.70000
8	0.30068	36.80000	0.29842	49.90000

CPU time to run the BPLA is shorter than that to run the GA. In terms of MSE, the BPLA reports fewer errors than the GA.

6. Conclusions.

This study focuses on training a FNN by using a BPLA and GA. To compare the performances of the BPLA and GA, three data sets, Sin function, Iris plants and Diabetes, are employed in this study. According to the measurement indicators, the BPLA is indeed slightly superior to the GA under certain conditions. Also, the BPLA is faster than the GA in terms of training speeds. However, the BPLA has the problem of overtraining, whereas, the GA does not. In terms of CPU time required, the BPLA yields better results than the GA. As far as the influence of different learning rates and momentum coefficients on BPLA convergence, the experiments show that learning rates influence the speed of convergence. Also, when the momentum coefficients are small, they also affect the convergence speed of MSE. When the learning rate is high and the momentum coefficient is large, MSE will overly fluctuate. However, when both parameters are not too high, the MSE convergence does not fluatuate. Neural networks have learning and fault-tolerance characteristics. GAs have the capabilities to find the best solutions in global searches. This study finds that in the process of seeking solutions, back-propagation ANNs report more stable convergence of best solutions, but require a larger number of learning cycles. On the other hand, GAs require a smaller number of learning cycles to find the best solutions, but need longer training time. To enhance the FNN training quality, the current methods, BPLA and GA, will be improved by introducing the local search mechanism in our further studies.

Acknowledgement. This work is partially supported by the National Science Council under projects No. NSC 98-2410-H-027-002-MY2 and No. NSC 99-2622-H-027-001-CC3. The authors also gratefully acknowledge the helpful comments and suggestions of the Editors and reviewers, which have improved the presentation.

REFERENCES

[1] S. Shrivastava and M. P. Singh, Performance evaluation of feed-forward neural network with soft computing techniques for hand written English alphabets, *Applied Soft Computing*, vol.11, no.1, pp.1156-1182, 2011.

[2] Y. E. Shao and B.-S. Hsu, Determining the contributors for a multivariate SPC chart signal using artificial neural networks and support vector machine, *International Journal of Innovative Computing, Information and Control*, vol.5, no.12(B), pp.4899-4906, 2009.

[3] P.-H. Chou, C.-H. Hsu, C.-F. Wu, P.-H. Li and M.-J. Wu, Application of back-propagation neural network for e-commerce customers patterning, *ICIC Express Letters*, vol.3, no.3(B), pp.775-785, 2009.

[4] C. He, H. Li, B. Wang, W. Yu and X. Liang, Prediction of compressive yield load for metal hollow sphere with crack based on artificial neural network, *ICIC Express Letters*, vol.3, no.4(B), pp.1263-1268, 2009.

[5] J. K. Wu, J. Kang, M. H. Chen and G. T. Chen, Fuzzy neural network model based on particle swarm optimization for short-term load forecasting, *Proc. of the CSU-EPSA*, vol.19, no.1, pp.63-67, 2007.

[6] D. K. Li, H. X. Zhang and S. A. Li, Development cost estimation of aircraft frame based on BP neural networks, *Fire Control and Command Control*, vol.31, no.9, pp.27-29, 2006.

[7] B. Karimi, M. B. Menhaj and I. Saboori, Multilayer feed forward neural networks for controlling decentralized large-scale non-affine nonlinear systems with guaranteed stability, *International Journal of Innovative Computing, Information and Control*, vol.6, no.11, pp.4825-4841, 2010.

[8] B. ZareNezhad and A. Aminian, A multi-layer feed forward neural network model for accurate prediction of flue gas sulfuric acid dew points in process industries, *Applied Thermal Engineering*, vol.30, no.6-7, pp.692-696, 2010.

[9] B. Choi, J. H. Lee and D. H. Kim, Solving local minima problem with large number of hidden nodes on two-layered feed-forward artificial neural networks, *Neurocomputing*, vol.71, no.16-18, pp.3640-3643, 2008.

[10] H. S. Chung and J. J. Alonso, Multiobjective optimization using approximation model-based genetic algorithms, *The 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, vol.1, pp.275-291, 2004.

[11] B. Chu, D. Kim, D. Hong, J. Park, J. T. Chung, J.-H. Chung and T. H. Kim, GA-based fuzzy controller design for tunnel ventilation systems, *Automation in Construction*, vol.17, no.2, pp.130-136, 2008.

[12] M. L. Huang and Y. H. Hung, Combining radial basis function neural network and genetic algorithm to improve HDD driver IC chip scale package assembly yield, *Expert Systems with Applications*, vol.34, no.1, pp.588-595, 2008.

[13] R. S. Sexton and J. N. D. Gupta, Comparative evaluation of genetic algorithm and backpropagation for training neural networks, *Information Sciences*, vol.129, no.1-4, pp.45-59, 2000.

[14] D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning representations by back-propagating errors, *Nature*, vol.323, no.6088, pp.533-536, 1986.

[15] M. N. Hajmeer and I. A. Basheer, A hybrid bayesian-neural network approach for probabilistic modeling of bacterial growth/no-growth interface, *International Journal of Food Microbiology*, vol.82, no.3, pp.233-243, 2003.

[16] Y. Q. Wang and X. Q. Bai, Prediction and evaluation of water quality using BP neural network, *Journal of Wuhan Polytechnic University*, vol.26, no.1, pp.64-67, 2007.

[17] Y. M. Wu, Y. Q. Wang and L. Li, Application study on BP network and generalized RBF network in estimating distribution model of mechanical products, *China Mechanical Engineering*, vol.17, no.20, pp.2140-2144, 2006.

[18] M. Bongards, Improving the efficiency of a wastewater treatment plant by fuzzy control and neural networks, *Water Science and Technology*, vol.43, no.11, pp.189-196, 2001.

[19] R. Xiao and V. Chandrasekar, Development of a neural network based algorithm for rainfall estimation from radar observations, *IEEE Transactions on Geoscience and Remote Sensing*, vol.35, pp.160-171, 1997.

[20] Z.-H. Che, PSO-based back-propagation artificial neural network for product and mold cost estimation of plastic injection molding, *Computers and Industrial Engineering*, vol.58, no.4, pp.625-637, 2010.

[21] L. C. Chang, H. J. Chu, Y. W. Chen and H. H. Chen, Application of genetic algorithm and experimental design on designing a groundwater well network, *Journal of Chinese Agricultural Engineering*, vol.52, no.2, pp.1-11, 2006.

[22] M. H. Jiang and X. C. Yuan, Construction and application of personal credit assessment GA neural network model, *Journal of Wuhan University of Science and Technology (Social Science Edition)*, vol.9, no.4, pp.368-372, 2007.

[23] N. Jiang and Y. Q. Zhai, Game design of self automation based on artificial neural nets and genetic algorithms, *Journal of Computer Applications*, vol.27, no.5, pp.1280-1282, 2007.

[24] Z. Y. Wu, Z. Zhang and Q. Hu, The optimization for conceptual design of high-rise buildings based on multi-objective genetic algorithm, *Systems Engineering – Theory and Practice*, vol.25, no.10, pp.121-124, 2005.

[25] Z.-H. Che, Using hybrid genetic algorithms for multi-period product configuration change planning, *International Journal of Innovative Computing, Information and Control*, vol.6, no.6, pp. 2761-2785, 2010.

[26] Z.-H. Che, A two-phase hybrid approach to supplier selection through cluster analysis with multiple dimensions, *International Journal of Innovative Computing, Information and Control*, vol.6, no.9, pp.4093-4111, 2010.

[27] Z.-H. Che, A genetic algorithm-based model for solving multi-period supplier selection problem with assembly sequence, *International Journal of Production Research*, vol.48, no.15, pp.4355-4377, 2010.

[28] Z.-H. Che and H.-S. Wang, Supplier selection and supply quantity allocation of common and non-common parts with multiple criteria under multiple products, *Computers and Industrial Engineering*, vol.55, no.1, pp.110-133, 2008.

[29] J.-F. Chang, A performance comparison between genetic algorithms and particle swarm optimization applied in constructing equity portfolios, *International Journal of Innovative Computing, Information and Control*, vol.5, no.12(B), pp.5069-5079, 2009.

[30] D. Y. Sha and Z.-H. Che, Supply chain network design: partner selection and production/distribution planning using a systematic model, *Journal of the Operational Research Society*, vol.57, no.1, pp.52-62, 2006.

[31] Z.-H. Che and H.-S. Wang, A hybrid approach for supplier cluster analysis, *Computers and Mathematics with Applications*, vol.59, no.2, pp.745-763, 2010.

[32] Z.-H. Che and C.-J. Chiang, A modified pareto genetic algorithm for multi-objective build-to-order supply chain planning with product assembly, *Advances in Engineering Software*, vol.41, no.7-8, pp.1011-1022, 2010.

[33] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*, MIT Press, Cambridge, London, MA, 1995.

[34] T. Masters, *Practical Neural Network Recipes in C++*, Academic Press Professional, Inc., San Diego, CA, 1993.

[35] S. Kumar, *Neural Networks*, McGraw-Hill, New York, 2005.

[36] M. Li, X. Guo, X. D. Tan and J. H. Yuan, Predictive method for power growth based on improved BP neural network and rebuilding of new industry structure, *Journal of Central South University (Science and Technology)*, vol.38, no.1, pp.143-147, 2007.

[37] L. Fu, *Neural Networks in Computer Intelligence*, McGraw-Hill, New York, 1995.

[38] E. E. Tabach, L. Lancelot, I. Shahrour and Y. Najjar, Use of artificial network simulation metamodelling to assess groundwater contamination in a road project, *Mathematical and Computer Modelling*, vol.45, no.7-8, pp.766-776, 2007.

[39] E. Zhou and A. Khotanzad, Fuzzy classifier design using genetic algorithms, *Pattern Recognition*, vol.40, no.12, pp.3401-3414, 2007.

[40] K. W. Chau, Application of PSO-based neural network in analysis of outcomes of construction claims, *Automation in Construction*, vol.16, no.5, pp.642-646, 2006.

[41] A. Meng, L. Ye, D. Roy and P. Padilla, Genetic algorithm based multi-agent system applied to test generation, *Computers and Education*, vol.49, no.4, pp.1205-1223, 2007.

[42] E. Eiben, R. Hinterding and Z. Michalewicz, Parameter control in evolutionary algorithms, *IEEE Transactions on Evolutionary Computation*, vol.3, pp.124-141, 1999.