# Architecture and Weight Optimization of ANN Using Sensitive Analysis and Adaptive Particle Swarm Optimization

**Faisal Muhammad Shah[†] , Md. Khairul Hasan[††] , Mohammad Moinul Hoque[†††] and Suman Ahmmed[††††]**

[†,††,†††] Department of Computer Science and Engineering Ahsanullah University of Science and Technology, Dhaka, Bangladesh
[††††]Department of Computer Science and Engineering, United International University, Dhaka, Bangladesh.

***Abstract***

This paper presents an optimized architecture and weights of three layered ANN designing method using sensitivity analysis and adaptive particle swarm optimization (SA–APSO). The optimized ANN architecture determination means to look for near minimal number of neurons in the ANN and finding the efficient connecting weights of it in such a way so that the ANN can achieve better performance for solving different problems. The proposed algorithm designs the ANN into two phases. In the first phase it tries to prune the neurons from ANN using sensitivity analysis to achieve the near minimal ANN structure and therefore it tries to optimize the weight matrices for further performance enhancement by adaptive particle swarm optimization. In the SA phase the authors use impact factor and correlation coefficients for pruning lower salient neurons. Initially it tries to prune the neurons having less impacts in the performance of ANN based on their impact factor values. Therefore it tries to lessen more neurons through merging the similar neurons in the ANN using correlation coefficient among the neuron pairs. In the optimization part it applied adaptive particle swarm optimization to optimize the connecting weight matrices to attain better performance. In the optimization by APSO, a special type of PSO, the authors' use training and validation fitness functions to emphasis on avoiding overfitting and more adapted with ANN, and to achieve effective weight matrices of ANN. To evaluate SA–APSO, it is applied on the dataset of Regional Power Control Center of Saudi Electricity Company, Western Operation Area (SEC-WOA) to do short term load forecasting (STLF). Results show that the proposed SA-APSO is able to design smaller architecture and attain excellent accuracy.

*Key words:*

*Artificial neural networks, overfitting, correlation coefficients, particle swarm optimization.*

## 1. Introduction

Architecture designing of an artificial neural network (ANN) is a very important area as the performance of an ANN largely depends on its effective structure. When applications become more complex, the structures presumably become larger. Moreover, larger structures increase the numbers of parameters and lose the generalizations ability. The determination of optimized ANN architectures means to decide the number of layers along with their respective neurons and to get the optimized connecting weights among the neurons of consecutive layers. It is well known that a three layered ANN, consists of an input, a hidden, and an output layer, can solve all kinds of linear and non linear problems. Therefore, in this research the number of layers is taken as three and the number of neurons and values of connecting weights will be determined by the sensitivity analysis and adaptive particle swarm optimization (SA–APSO) approach. Usually the numbers of input and output neurons are determined by the sizes of input and output vectors of dataset and architecture designing means to determine the number of hidden neurons and optimization of weights mean to optimize the values of weight matrices.

The problem of designing a near optimal ANN architecture for a given application is a tricky question for the researchers. However, this is an important issue since there are strong biological and engineering evidences to support its' functions. So, the information processing ability of an ANN is majorly depends on its architecture [1-4]. The fact is that both the large and small networks exhibit a number of advantages and disadvantages. On the one hand, a larger-sized network may be trained quickly; it can more easily avoid local minima and more accurately fit the training data. However, it may be inefficient because of its high computational complexity, many degrees of freedom and poor performance in generalization due to over-fitting. On the other hand, a smaller network may save the computational costs and have good performance in generalization. However, it may learn very slowly or may not learn the data set at all. Even it is known, there is no guarantee that the smallest feasible network will converge to the correct weights during training because the network may be sensitive to the initial settings and more likely to be trapped in local minima [5-6]. To design an appropriate architecture for the solution of a given task is always an open challenge [1] [3-4].

Manuscript received August 5, 2010
Manuscript revised August 20, 2010

There have been many attempts to design ANN architectures automatically: such as various pruning [1-2] [4-7], constructive [8-9], and evolutionary (using optimization and their hybrid) [10-14] algorithms. Roughly speaking, a constructive algorithm starts with a minimal sized network (i.e., a network with a minimal number of layers, neurons, and connections) and starts to add layers, neurons, and connections gradually in the training period. In contrast, a pruning algorithm does the opposite, i.e., it starts with larger sized network and gradually deletes unnecessary layers, neurons, and connections during training period.

Sensitivity analysis [1-8] approach is used to identify the effective elements in ANN and discarding the redundant part from ANN. In [4] the authors propose a pruning algorithm using sensitive analysis and cross validation. Here they identified lower salient weights/neurons by their sensitivity and checks cross validation to maintain the generalization property before pruning anything. In [7] the authors propose sensitive analysis using standard deviation and correlation coefficient to determine salient neurons, but they did not apply any optimization approach to attain better weight matrices of ANN.

Adjusting weights to train a feed-forward multilayer ANN has been one of the earliest applications of PSO. The advantage of PSO [10] is it usually computationally inexpensive, easily implementable, and does not require gradient information of an objective function. Besides it uses evolving approach to explore more search spaces for finding better solution. In [11] the authors shown that in ANN, the PSO can make convergence the weight matrices faster with respect to back propagation algorithm. Moreover, the success of back propagation sometimes depends on choosing its initial weights and bias values, where as PSO does not depend on those. It is also stated that the concept of the PSO can be incorporated into back propagation algorithm to improve its global convergence rate. In [12] the authors applied PSO to evolve the structure of an ANN. Both the architecture and the weights of ANNs are adaptively adjusted according to the quality of the neural network. Some hybrid approach of PSO is also used in ANN designing in [13-14].The authors in [16] use ANN and PSO for STLF but there is no architecture determination for ANN.

In this paper the authors propose a hybrid algorithm using sensitivity analysis and optimization technique to design ANN automatically. In the sensitivity analysis (SA) part, impact factor and correlation coefficients among the neurons are used to reduce the hidden neurons from ANN. Therefore to get better weight matrices adaptive particle swarm optimization (APSO) is applied on ANN. Thus the proposed SA-APSO attains optimized ANN architecture. Since impact factor is a good tool to determine the saliency on neurons so it can identify the lower salient neurons to prune, besides correlation coefficient among the neuron pairs indicate the similarity among neurons pair so it is possible to share the load after merging of both. Both pruning steps recheck the performance so that it does not degrade. Since PSO has the property to explore more search spaces in determining the solution and adaptive approach is considered here therefore APSO can determine the optimized weight matrices for attaining better performances and avoiding overfitting.

## 2. PROPOSED ALGORITHM

The proposed SA–APSO algorithm is for designing an optimized three layered ANN. It does it in two steps, initially it looks for pruning the hidden neurons as much as possible to reduce the size of the ANN without degrading performance, and after getting the structure it focuses to optimize its weight matrices to enhance the performances of ANN. In the first phase of pruning it determines the saliency of each hidden neuron using impact factor, therefore it tries to prune less salient neurons. At the time of each neuron pruning, it tries to provide proper replacements to avoid degrading its performance Afterwards SA-APSO tries to find out the similar contributory neuron pairs. Similarity among a pair of neurons is measured by correlation coefficients (positive or negative) among those two neurons. It is obvious that two similar neurons might bear similar properties and their contribution in the ANN might be similar. Therefore it might be possible to share their contribution by a single neuron formed from that pair. This is the main philosophy of merging similar paired neurons. The authors try to put proper replacements at the time of each merging operation. Besides after each pruning or merging operation the performance of the ANN is measured so that the performance of ANN does not degrade. After pruning phase, SA-APSO starts optimization of weight matrices. In this phase it uses PSO approach for exploring more solution spaces so that better candidate solution can be generated and selected gradually. In this phase adaptive approach is applied to update and generate candidate solutions so that only better performed weight matrices will be selected. The detail steps of the proposed SA–APSO algorithm are furnished below:

***Step 1(Initialization):*** Create a fully connected initial ANN architecture. The number of neurons in the input and output layers are same as the size of the input and output vectors of the problem datasets. The numbers of hidden neurons are taken arbitrarily. All the weights

are initialized randomly within a certain range and the biases are assigned to a fixed real value.

***Step 2(Training):*** Train the ANN by using back propagation (BP) algorithm until the error *E* reduces to a certain value. The dataset is divided into two different sets like training and validation datasets. Training dataset is used to train the ANN. Validation dataset is used for determining the interim performance and stopping criteria.The training ends when the training error is still decreasing and the validation error starts to increase. Error *E* is calculated as follows:

$$E = \frac{1}{n_p}\sum_{n=1}^{n_p}\left(\frac{1}{2}\sum_{o=1}^{On}(d_o - a_o)^2\right) \quad (1)$$

where, $n_p$ and $o_n$ are the total number of input patterns and output neurons, respectively. $d_o$ and $a_o$ are the desired and actual outputs, respectively.

***Step 3(Pruning):*** Compute the impact factor (*ImF*) of each hidden neuron using

$$\mathrm{Im}\,F_i = \sum w_{ji}^2 \sigma_i^2 \quad (2)$$

where $w_{ji}^2$ denotes the connecting weight from $i^{th}$ neuron of the hidden layer to the $j^{th}$ neuron in the output layer and $\sigma_i^2$ is the sample variance of the $i^{th}$ hidden neuron's output values for all input patterns. If the impact factor of hidden neurons remains very low i.e., under certain threshold value $(\sigma_{tr})$, identify those as low information bearing neurons and try to prune them on by one with proper replacement, otherwise go to next step 4. SA–APSO always checks the ANN performance after each neuron pruning, if the error remains within acceptable range it moves for more pruning otherwise it sends for retraining to reduce the error under acceptable range. In spite of retraining if the error does not come down within acceptable limit SA–APSO retrieves the latest pruned neuron and goes to next step.

***Step 4(Merging):*** In this step SA–APSO tries to merge similar neuron pairs. Compute correlations among hidden neuron pairs in an ANN. Both positive and negative correlations are considered in similarity finding. The pairs having high correlations and higher than a threshold value $\rho_{tr}$ were marked as the element of set *M* and were selected for merging. Therefore SA–APSO starts to merge paired neurons one by one providing proper replacements. In spite of replacements, if the errors exceed accepted range then the ANN send for retraining. If the retraining becomes successful and error declines within expected range then continue other merging steps in a similar way. But if the retraining could not succeed i.e., error does not reduced under accepted range, then restore the last merge operation and stop further merging and go to next step.

***Step 5(Optimization of connecting weights):*** In this step SA–APSO starts for finding optimized weight matrices using adaptive particle swarm optimization. Here more solution spaces generated using standard PSO approach. Therefore it tried to select the best candidate solution that can produce better performance and can avoid local minima. To update the parameters (*pbest* and *gbest*) of PSO here the authors proposed an adaptive approach so that SA–APSO can attain better solution and maintaining generalization property. To select any updated solution, it uses two different fitness functions to get better solution and avoiding overfitting. Thus this step determines the optimized weight matrices of ANN.

***Step 6(Final ANN architecture):*** Deliver the ANN architecture designed by SA–APSO.

The major components of the SA–APSO algorithm are described in the follows.

### A. Pruning by Impact Factor

After training the ANN, SA–APSO find out the *ImF* value for each hidden neuron to using (2). Therefore it identifies the neurons having lower *ImF* value under certain threshold value. SA–APSO considers those as less salient neurons who have lower *ImF* value. Lower *ImF* value indicates lower and consistent value delivers by that neuron. Therefore the authors attempt to prune the hidden neuron having lowest *ImF* value with proper replacements. Since it deals with fully connected ANN, so SA–APSO adds replacement value to the output neurons biases according to (3) to compensate the contribution of pruned hidden neuron.

$$W^o_{k,0}(t+1) = W^o_{k,0}(t) + W^o_{k,j}\overline{x_j} \quad (3)$$

Here, $W^o_{k,0}(t)$, the biases of the output neurons, where 0 represents biases, *k* represents output neurons i.e., {*k* = 1, 2…On}, $W^o_{k,j}\overline{x}_j$, is the average weighted value feeds from the $j^{th}$ pruned hidden neuron to the

$k^{th}$ output neuron. $W^{o}_{k,0}(t+1)$, denotes updated bias values of the output layered $k^{th}$ neurons.

Generally, a hidden neuron with small *ImF* value indicates that it delivers almost constant output value to the neurons in the succeeding layer. As a result one can easily replace the contributions of those neurons by providing the additional values in the biases of output neurons according to (3). Besides if any pruning causes to raise error above tolerable range then the ANN send for retraining to reduce the errors. If retraining successful then it again go for further pruning otherwise it restore its last pruned neuron and ends this module.

### B. Merging by Correlation Coefficients

Correlation defines a relationship between two given sides, so when it is between two hidden neurons it refers the relationship among them. If any two neurons exhibit correlated responses (either identical or opposite) over the whole input patterns, there is a possibility that these two neurons are closely related in their natures. The idea behind the merging is that since the contribution of each neuron in a correlated pair is similar in nature, so it might be expected that those can be shared the contribution of each other. When the merging of a hidden neuron pair is performed SA–APSO always try to provide proper replacements so that the error remains lower.
Similarities (correlation) among any two neurons are measured as follows:

$$\rho_{H_1,H_2} = \frac{Cov(H_1,H_2)}{\sigma_{x_i}.\sigma_{y_i}} \qquad (4)$$

Here, $\rho_{H_1,H_2}$ represents the correlation among $H_1$ and $H_2$ neurons. $\sigma_{x_i}$ and $\sigma_{y_i}$ are standard deviations of the outcomes of $H_1$ and $H_2$. $Cov(H_1,H_2)$ denotes the covariance among those, which is determined as:

$$Cov(H_1,H_2) = \frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}) \qquad (5)$$

where $x_i$ and $y_i$ are the output values of $H_1$ and $H_2$ respectively. The mean output values of $H_1$ and $H_2$ are expressed by $\bar{x}$ and $\bar{y}$ respectively.
After identifying the similar neuron pair SA–APSO starts merging from the most similar neurons and the connecting weights for the new merged neuron will be changed as follows:

Let $H_m$ be the merged neuron that will be produced by merging $H_1$ and $H_2$ neurons. Thus the connecting weights from input neurons to $H_m$ will be updated as follows:

$$w^{hm}_{m1} = \frac{w^{h1}_{11}+w^{h2}_{21}}{2},\ w^{hm}_{m2} = \frac{w^{h1}_{12}+w^{h2}_{22}}{2},$$
$$\ldots\ldots w^{hm}_{mi} = \frac{w^{h1}_{1i}+w^{h2}_{2i}}{2} \qquad (6)$$

The new weight from input neuron $I_1$ to $H_m$ will be the average of the connecting weights between $I_1$ to $H_1$ and $H_2$. Similarly connecting weights from all input neurons to $H_m$ is updated.

To update the connecting weights from $H_m$ to output neurons will be updated as:

$$w^{o1}_{1m} = (w^{o1}_{11}+w^{o1}_{12}),\ w^{o2}_{2m} = (w^{o2}_{21}+w^{o2}_{22}),\ \ldots$$
$$w^{ok}_{km} = (w^{ok}_{k1}+w^{ok}_{k2}) \qquad (7)$$

where $w^{o1}_{1m}$ is updated connecting weight from $H_m$ to output neuron $O_1$. $w^{o1}_{11}$ and $w^{o1}_{12}$ are the previous connecting weights between $H_1$ and $H_2$ to $O_1$. Similarly the connecting weights from $H_m$ to others output neurons are updated in the same way.
In spite of these replacements of any merge operation raises errors then SA–APSO sends for retraining to decrement the errors. If retraining successful then it looks for more merging to shorten the ANN size. But if the retraining fails to reduce the error increases after merging, SA–APSO restores last merging to maintain better performance. Thus merging operation is applied to prune the hidden neurons without degrading performance.

### C. Optimization of Weights by APSO

Particle Swarm Optimization is a heuristic approach first proposed in 1995 by Kennedy and Eberhart [10] as an evolutionary computational method developed for dealing with the optimization of continuous and discontinuous function decision making. The PSO algorithm is based on the biological and sociological behavior of animals such as schools of fish and flocks of birds searching for their food. PSO imitates this behavior by creating a population with random search solution and each potential solution is represented as a particle in a population (called swarm In standard PSO algorithm, particles are generated and updated following (8)–(9), where each particle tries to adjust its velocity according to best positions ever visited that is stored in its memory called personal best $(pbest)$ and according to the best previous position attained by any particle in its neighborhood called global best $(gbest)$ trying to search for a better position. Thus, particles communicate with each other

and share their information among each other during their searching.

$$v_i(t+1) = w_i(t) + r_1 c_1 [pbest - x_i(t)] + r_2 c_2 [gbest - x_i(t)] \quad (8)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (9)$$

Here $w$ is an inertia weight, which provides a balance between the local and global exploration, $v_i$ and $v_i(t+1)$ are current and modified velocities of that iteration, respectively. $c_1$ and $c_2$ are positive numbers, used to control the particle's movement at each iteration. They represent cognitive and social components, respectively. $r_1$ and $r_2$ are uniform distribution numbers in the range [0, 1]. $x_i(t)$ *and* $x_i(t+1)$ are the current and modified position for each iteration, respectively. *N* denotes the number of particles.

The authors found that an ANN can be represented as the set of four weight matrices. The initial weight matrices set is consider as the initial particle in the optimization process which was provided by the ANN. Thus the initial particle $(x_k)$ can be represented as:

$$x_k = \{w_{ihk}, w_{hok}, b_{hk}, b_{ok}\} \quad (10)$$

Here $w_{hik}, w_{oik}$ matrices are input-hidden and hidden-output layered weight matrix sets. $b_{hk}, b_{ok}$ denote the bias matrices of hidden and output layer neurons respectively. $k$ and $x_k$ denote index of particle and $k^{th}$ particle respectively.

The following steps occurred in APSO weight optimization process:

***Step 1:*** APSO receives the initial particle from ANN in the form of (10). It also assigns the $pbest$ and $gbest$ parameters similar to the initial particle. It initiates the other parameters $w, c_1, c_2$ and $N$, where *N* is the number of candidate particles generated from each particle during exploration. Therefore it generates the number of candidate particles from the initial particle using (8) and (9).

***Step 2:*** Compute the two fitness function (training and validation fitness), $F_{it}, F_{iv}$, for each particle $i$ ,where $i = 1..N$ . The fitness functions are similar to the error *E* defined in (1).

***Step 3:*** To update *pbest* , if a particle ($i^{th}$) can perform better (in terms of both fitness values) with respect to its current $pbest$ value i.e., if $F_{it} < F_{pt}$ and $F_{iv} < F_{pv}$ then update $pbest$ by the new position of $i^{th}$ particle.

***Step 4:*** To update *gbest* , find the $k^{th}$ particle whose validation fitness is best among the candidate particles. Since APSO tries to avoid overfitting, therefore to update *gbest* , it considers $F_v$ as the basic criteria.

***Step 4.1:*** Compare the $k^{th}$ particle with current *gbest* in the same way i.e., if $F_{kt} < F_{gt}$ and $F_{kv} < F_{gv}$ then update *gbest* by the $k^{th}$ particle otherwise *gbest* remains unchanged in this iteration.

***Step 5:*** If stopping criterion meet go to next step otherwise go to *step 1* for further exploration

***Step 6:*** Deliver the final ANN architecture produced by SA–APSO.

## 3. Results

Discussion The simulation program has run on Intel(R) Core(TM) 2 Duo 2.66GHz CPU, 2.96 GB RAM, Microsoft Windows XP OS and MatLab version 7.6. Initially it tries to select a proper ANN structure and then trained by BP. Therefore evaluate the output of ANN for STLF. In the training phase 75% of data are used as training dataset and remaining 25% dataset are used as the validation dataset. Tan-sigmoid function is used as the transfer function for hidden layer nodes and the purelin function is used for output layer nodes. Every time the number of candidate particles generated from each particle is $N$ , which is 20. Initialize the APSO parameters $w$ , $c_1$ and $c_2$ as 1.0, 1.5 and 0.5 respectively. $w$ was started from 1.0 and gradually decreases to 0.4 uniformly. The maximum number of iterations, *m*, is taken as 300; $r_1$ and $r_2$ were generated randomly following distribution ranges between [0, 1]. The initial velocity ranges of particles are assigned between [-1, +1] randomly. The threshold values of *ImF* and correlation vary from 0.10 to 0.25 and 0.65 to 0.90 respectively. The initial hidden neurons are taken as 8.

Since in the benchmark dataset the number of input and output neurons are determined by the vector size of dataset and only hidden neurons need to be determined in designing ANN, therefore in this research the similar approach is applied. To do this the authors initially determine the input and output neurons for the STLF of

SEC-WOA dataset. Since short term load forecasting (STLF) is doing here for a single hour so output neuron becomes one. For determining input neurons, after certain

by *ImF* ANN, after *ImF* and merging ANN, and finally after optimization made by SA–APSO.To demonstrate the architecture determination process of SA–APSO, Table III

**Table I: Sample Data set used in SA_APSO**

| Hours | July 01, 2007 | | | | August 02, 2007 | | | |
|---|---|---|---|---|---|---|---|---|
| | Temp. ($^oC$) | Hum. (%) | LB_24 (MW) | Load (MW) | Temp. ($^oC$) | Hum. (%) | LB_24 (MW) | Load (MW) |
| 1 | 36 | 43 | 8336 | 8249 | 35 | 66 | 8249 | 8122 |
| 2 | 34 | 88 | 8469 | 8377 | 35 | 72 | 8377 | 8261 |
| 3 | 33 | 95 | 8504 | 8492 | 35 | 80 | 8492 | 8312 |
| .. | | | | | | | | .. |
| 21 | 35 | 69 | 8304 | 8331 | 33 | 80 | 8331 | 8332 |
| 22 | 35 | 71 | 8319 | 8338 | 33 | 80 | 8338 | 8254 |
| 23 | 35 | 72 | 8280 | 8249 | 34 | 77 | 8249 | 8123 |
| 24 | 35 | 71 | 8233 | 8092 | 34 | 76 | 8092 | 8015 |

performance analysis the authors found there have been three major criteria those are responsible STLF: Load of same hour of previous day, temperature of same day, and humidity of the same day. Table I shows that the data consists of Temperature (Temp.), Humidity (Hum), load before 24 hours (LB_24), and the actual load value (Load) for different hours. Since this research deals STLF, after some data analysis, it was found that over a period of one year data was sufficient to train the ANN. Dataset from July 2006 to June 2007 was used as training dataset and data of July 2007 was taken as testing dataset. Training dataset was divided into two parts - training and validation. query.

The performance of the ANN is measured in terms of Mean Absolute Percentage Error (MAPE) that can be defined as:

$$MAPE = \frac{100}{D}\sum_{i=1}^{D}(|Actual(i) - Forecast(i)|) / Actual(i) \qquad (11)$$

Where, $D$ is the number of testing dataset, *Actual* and *Forecast* indicates the actual load given in the dataset and forecasted load.

MAPE is considered as the fitness of value of the candidate solutions. Table II shows the performance of proposed SA–APSO. It also compares with ANN in terms of MAPE, minimum error rate (MIN), maximum error rate (MAX), and standard deviation (SD). Tst_01 to Tst_10 denote ten individual test performances for dated July 01, 2007. For all cases SA–APSO performs better than only ANN. Table II shows average MAPE using ANN is 2.849%. However, the average MAPE is only 2.021% for SA-APSO. Average accuracy improved by 0.82%. Table II shows the performances of different stages of SA–APSO. It shows initial performance of ANN, after pruning shows the number of hidden neurons in each stage and their corresponding performances. For all cases SA–APSO becomes successful to reduce the size and improve the performances.

Fig.1 shows graphical depiction of comparison of 24 hours load profile among the actual load and forecasted loads for July 01, 2007 by different techniques. It is clear from the graph that results of SA–APSO are far better than ANN technique; it is clearly visible that the proposed technique has not only captured the trend of the load profile but its forecasted values are very near to the actual load values.

To exhibit the robustness performance of SA–APSO the authors tested to do STLF for the month of July, 2007. Table IV shows the MAPE comparison of day by day forecasted STLF for both ANN and SA–APSO.

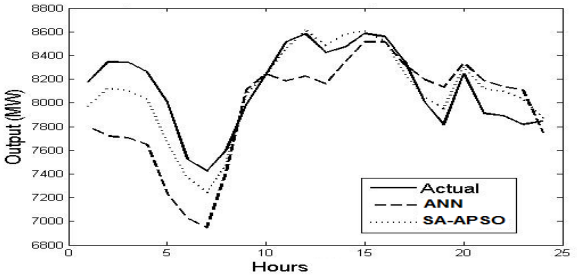


Fig 1: Comparison with actual load with ANN and SA-APSO

**Table II: STLF PERFORMANCE OF ANN AND SA–APSO ON JULY 01, 2007**

| | | Tst_01 | Tst_02 | Tst_03 | Tst_04 | Tst_05 | Tst_06 | Tst_07 | Tst_08 | Tst_09 | Tst_10 | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | |
| MAPE | ANN | 3.612 | 2.637 | 2.084 | 2.712 | 2.866 | 3.640 | 2.650 | 3.040 | 2.735 | 2.515 | 2.849 |
| | Pruned by *ImF* | 3.226 | 1.692 | 1.437 | 1.574 | 2.710 | 1.667 | 1.817 | 2.384 | 2.152 | 2.291 | 2.095 |
| | After Merging | 3.183 | 1.692 | 2.048 | 1.574 | 2.710 | 1.667 | 1.817 | 1.944 | 2.060 | 2.291 | 2.099 |
| | SA–APSO | 3.085 | 1.560 | 1.938 | 1.859 | 2.423 | 1.606 | 1.792 | 1.9079 | 2.051 | 1.989 | **2.021** |
| MIN | ANN | 0.699 | 0.699 | 0.166 | 0.410 | 0.341 | 0.074 | 0.031 | 0.5669 | 0.0549 | 0.796 | 0.384 |
| | Pruned by *ImF* | 0.398 | 0.398 | 0.011 | 0.132 | 0.068 | 0.035 | 0.376 | 0.3932 | 0.110 | 0.075 | 0.200 |
| | After Merging | 0.020 | 0.020 | 0.066 | 0.132 | 0.068 | 0.035 | 0.376 | 0.028 | 0.014 | 0.075 | 0.083 |
| | SA–APSO | 0.158 | 0.158 | 0.124 | 0.240 | 0.319 | 0.079 | 0.407 | 0.001 | 0.037 | 0.231 | 0.175 |
| MAX | ANN | 9.065 | 9.065 | 5.319 | 5.875 | 7.274 | 9.620 | 6.425 | 6.795 | 6.197 | 4.886 | 7.052 |
| | Pruned by *ImF* | 8.795 | 8.795 | 3.68 | 4.271 | 5.921 | 4.774 | 4.622 | 4.933 | 5.277 | 4.535 | 5.560 |
| | After Merging | 9.263 | 9.263 | 5.868 | 4.271 | 5.921 | 4.774 | 4.622 | 5.305 | 3.469 | 4.535 | 5.729 |
| | SA–APSO | 8.495 | 8.495 | 5.711 | 4.108 | 5.008 | 4.281 | 4.482 | 5.194 | 3.451 | 4.292 | 5.352 |
| STD | ANN | 2.458 | 2.458 | 1.459 | 1.638 | 1.962 | 2.678 | 1.789 | 2.024 | 1.840 | 1.135 | 1.944 |
| | Pruned by *ImF* | 2.460 | 2.460 | 0.931 | 1.116 | 1.834 | 1.294 | 1.193 | 1.209 | 1.569 | 1.404 | 1.547 |
| | After Merging | 2.699 | 2.699 | 1.611 | 1.116 | 1.834 | 1.294 | 1.193 | 1.632 | 0.887 | 1.404 | 1.637 |
| | SA–APSO | 2.470 | 2.470 | 1.497 | 1.146 | 1.446 | 1.129 | 1.141 | 1.551 | 0.877 | 1.199 | 1.493 |

**Table III: STLF PERFORMANCE ON HIDDEN NEURONS OF ANN AND SA–APSO ON JULY 01, 2007**

| | | Tst_01 | Tst_02 | Tst_03 | Tst_04 | Tst_05 | Tst_06 | Tst_07 | Tst_08 | Tst_09 | Tst_10 | **Avg**. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Initial ANN | No of Hidden Neurons | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| | Training Error | 0.007 | 0.010 | 0.007 | 0.008 | 0.007 | 0.015 | 0.012 | 0.011 | 0.013 | 0.013 | 0.011 |
| | Testing Error | 3.612 | 2.636 | 2.083 | 2.711 | 2.865 | 3.64 | 2.650 | 3.040 | 2.735 | 2.515 | 2.849 |
| Pruned by *ImF* | No of Hidden Neurons | 7 | 5 | 4 | 6 | 7 | 6 | 6 | 6 | 7 | 5 | 5.9 |
| | Training Error | 0.008 | 0.008 | 0.009 | 0.011 | 0.008 | 0.008 | 0.007 | 0.008 | 0.008 | 0.008 | 0.009 |
| | Testing Error | 3.226 | 1.692 | 1.436 | 1.573 | 2.709 | 1.667 | 1.817 | 2.384 | 2.152 | 2.290 | 2.095 |
| After Merging | No of Hidden Neurons | 5 | 5 | 2 | 6 | 6 | 6 | 6 | 4 | 5 | 5 | 5 |
| | Training Error | 0.007 | 0.008 | 0.007 | 0.011 | 0.008 | 0.008 | 0.007 | 0.008 | 0.007 | 0.008 | 0.008 |
| | Testing Error | 3.183 | 1.692 | 2.048 | 1.573 | 2.709 | 1.667 | 1.817 | 1.944 | 2.060 | 2.290 | 2.098 |
| SA–APSO | No of Hidden Neurons | 5 | 5 | 2 | 6 | 6 | 6 | 6 | 4 | 5 | 5 | **5** |
| | Training Error | 0.007 | 0.008 | 0.007 | 0.011 | 0.008 | 0.008 | 0.007 | 0.008 | 0.007 | 0.008 | 0.008 |
| | Testing Error | 3.084 | 1.559 | 1.937 | 1.858 | 2.422 | 1.605 | 1.791 | 1.907 | 2.051 | 1.989 | **2.021** |

**Table IV: MAPE COMPARISON OF ANN AND SA–APSO FOR JULY, 2007**

| Date | Day | ANN | SA-APSO |
|---|---|---|---|
| July 01,07 | Wed | 2.291 | 1.989 |
| July 02,07 | Thu | 3.077 | 2.932 |
| July 03,07 | Fri | 1.530 | 1.446 |
| .. | | | |
| July 29,07 | Wed | 8.434 | 8.467 |
| July 30,07 | Thu | 4.889 | 4.713 |
| July 31,07 | Fri | 2.708 | 2.544 |
| **Avg**. | | **3.041** | **2.951** |
| Best | | 0.999 | 0.935 |
| Worst | | 8.434 | 8.467 |

The following three diagrams Fig. 2-4, are based on the load forecasting results of July 1, 2007.

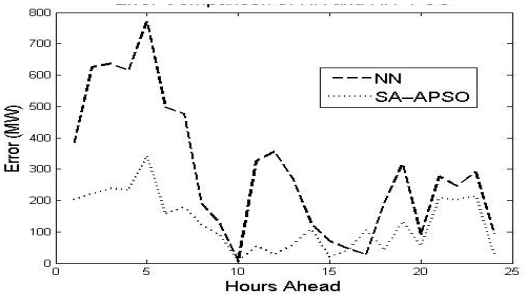


Fig. 2: Error comparison of ANN and SA-APSO

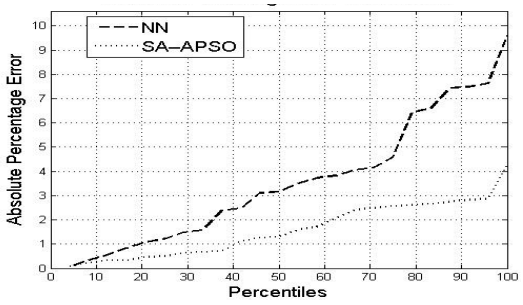


Fig. 3: Absolute Percentage Error Distributions

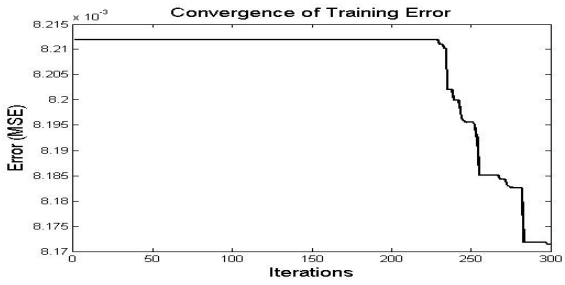


Fig. 4: Improvement of NN training error using PSO

In Fig. 2, the dashed and the doted curves represent the percentage of errors occurred by applying ANN and SA–APSO methods respectively. ANN produces an average error of 2.043 % with a maximum error of 6.295 %. Whereas the proposed SA–APSO's average error is 2.015 % and the maximum error is 6.248 %. In Fig. 3, the dashed and the doted curves represent the absolute percentage error distributions occurred by applying ANN and SA–APSO methods respectively. The SA–APSO curve shows that 60 % of the time the error is within 2% whereas the error is within 35 % in case of only ANN In Fig. 4, the curve shows how the error decreases in each iteration of SA–APSO exploration.

## 4. Conclusion

In this paper, a hybrid approach SA–APSO is used to design ANN architectures. SA–APSO can reduce the size of the ANN successfully as it tries to replace proper compensations at the time of each pruning or merging. Combination of both impact factor and correlation coefficient has ensured to reduce the size of ANN as much as possible and attain near optimal ANN. Besides the optimization of weight matrices by adaptive particle swarm optimization is a real addition in designing ANN. Through this optimization process the authors demonstrate that the optimization of weight matrices possible even on a trained ANN. Therefore this approach may be applied to get better performance of a trained ANN. In the adaptive PSO the authors consider two types of fitness for each candidate solution which is a real contribution in PSO for dealing ANN. This approach may be extended to apply on other classification benchmark dataset to show its effectiveness. In future the authors will try to show the performance of this approach for different classification benchmark datasets and other optimization problems. In its current implementation, SA-APSO has a few user-specified parameters although this is not unusual in the field. These parameters, however, are not very sensitive to moderate changes.ne of the future improvements to SA-APSO would be reduce the number of parameters or make them adaptive. In addition, the use of a different significance criterion in the merging operation of SA-APSO would also be an interesting future research topic.

## References

[1] M.M. R. Reed, "Pruning Algorithms- A survey," *IEEE Trans. on Neural Networks*, vol. 4, pp. 740 – 747, 1993.

[2] A.P. Engelbrecht, "A New Pruning Heuristic Based on Variance Analysis of Sensitivity Information," *IEEE Trans. on Neural Networks*, vol. 12, pp. 1386 – 1399, 2001.

[3] C. Xiang, Q. Ding and T. H. Lee, "Geometric Interpretation and Architecture Selection of MLP," *IEEE Trans. on Neural Networks*, vol. 16, pp. 84 – 96, 2005.
[4] D. Sabo and X. H. Yu, "A new pruning algorithm for neural network dimension analysis," *IEEE International Joint Conference on Neural Networks, IJCANN 2008*, pp. 3313 – 3318, Jun. 2008.
[5] H. Lee and C.H. Park, "A Pruning Algorithm of Neural Networks using Impact Factor Regularization," *proc. of the 9th International Conference on Neural Information Processing (ICONIP)*, vol. 5, pp. 2605 – 2609, 2002.
[6] D. S. Yeung and X. Q. Zeng, "Hidden neuron pruning for multilayer perceptrons using a sensitivity measure," *proc. of the First International Conference on Machine Learning and Cybernetics, Beijing*, pp. 1751 – 1757, 2002.
[7] Suman Ahmmed, Md. Saifur Rahman, Md. Monirul Islam *"ADCSP – A New Approach for Designing Artificial Neural Networks"*, Proceedings of 9th International Conference on Computer and Information Technology, ICCIT '06, pp. 408– 413, Dec., 2006.
[8] T. Ash, "Dynamic neuron creation in back-propagation networks," *Connection Science*, vol. 1, pp. 365-375, 1989.
[9] M.A. Costa, A. Braga and B.R de Menezes, "Constructive and Pruning Methods for Neural Network Design," *proc. of the VII Brazilian Symposium on Neural Networks, 2002*, pp.49 – 54, 2002.
[10] J. KeANNedy and R. C. Eberhart, "Particle swarm optimization," *Proceedings of IEEE Int. Conf. Neural Networks*, vol. 4, pp. 1942–1948, Nov., 1995.
[11] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of particle swarm optimization and back propagation as training algorithms for neural networks," *proc. of the 2003 IEEE Swarm Intelligence Symposium*, pp. 110 – 117, 2003.
[12] C. Zhang, H. Shao, and Y. Li, "Particle Swarm Optimization for Evolving Artificial Neural Network," *IEEE International Conference on Systems, Man and Cybernetics 2000*, vol. 4, pp. 2487 – 2490, 2000.
[13] B. Liu, L. Wang, Y. Jin, and D. Huang, "Designing neural networks using hybrid particle swarm optimization," *Advances in Neural Networks*, vol. 3496, pp 391-397, 2005.
[14] T. X. Lun, L. Y. Guo, and Z. Ling, "A hybrid particle swarm algorithm for the structure and parameter optimization of feedforward neural networks," *Advances in Neural Networks*, vol. 4493, pp 213-218, 2007.
[15] L. Prechelt, "Some notes on neural learning algorithm benchmarking," *Neurocomputing*, vol. 9, pp. 343 – 347, 1995.
[16] S. Ahmmed, D. M. F. Rahman, M. H. Khairul, A. Y. Saber, M. Z. Rahman, "Computational Intelligence Approach to Load Forecasting - a Practical Application for the Desert of Saudi Arabia," 12th *International Conference on Computers and Information Technology*, 2009, ICCIT'09, pp. 290 – 296, 21 – 23 December, 2009.

**Faisal Muhammad Shah** was born in Faridpur City, Bangladesh, on Novembor 18, 1980. He received the B.Sc. degree in Computer Science and Engineering from Ahsanullah University of Science and Technology, Dhaka, Bangladesh, in 2002 and the M.Sc. degree in Computer Science and Engineering from United International University, Dhaka, Bangladesh in 2010. He is currently working as a faculty of the Computer Science and Engineering Department, Ahsanullah University of Science and Technology. His research interests include computational intelligence, neural networks, optimization and their applications.

**Md. Khairul Hasan** was born in Rajshahi City, Bangladesh, on December 27, 1976. He received the B.Sc. degree in Computer Science and Engineering from Ahsanullah University of Science and Technology, Dhaka, Bangladesh, in 2000 and the M.Sc. degree in Computer Science and Engineering from United International University, Dhaka, Bangladesh in 2010. He is currently working as a faculty of the Computer Science and Engineering Department, Ahsanullah University of Science and Technology. His research interests include computational intelligence, neural networks, optimization and their applications.

**Suman Ahmmed** received the B.Sc. and M.Sc. degrees from the Computer Science and Engineering Department at Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 1997 and 2006, respectively. He is currently an Assistant Professor of Computer Science and Engineering and the Director of Student Affairs at United International University, Dhaka, Bangladesh. His research interests are load forecasting, optimization, neural networks, and optimization.