International Journal of Soft Computing 2 (3): 450-458, 2007

A Novel Algorithm for Designing Three Layered Artificial Neural Networks

[1]Suman Ahmmed, [2]Khondaker Abdullah-Al-Mamun and [3]Monirul Islam
[1]Department of Computer Science and Engineering, United International University, Dhaka-1209, Bangladesh
[2]Department of Computer Science and Engineering, Ahsanullah University of Science and Technology, Dhaka-1215, Bangladesh
[3]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka-1000, Bangladesh

Abstract: Architecture determination of Artificial Neural Networks (ANNs) is an important issue for the successful application of ANNs in many practical problems. It is well known that a three layered ANN can solve any kind of linear and nonlinear problems. This study proposes a new pruning algorithm, Architecture Designing by Correlation and Sensitivity Pruning (ADCSP), to determine the three layered near optimal ANN architectures automatically. The salient features of ADCSP are that it uses correlations, apply merging operation, uses computationally inexpensive formula, maintain its generalization ability and avoid overfitting. It has been tested extensively on a number of benchmark problems in machine learning and neural networks. The experimental results show that ADCSP can determine smaller architectures with good generalization ability compared to many other works.

Key words: ANN, ensemble of ANNs, correlations, generalization, overfitting, back-propagation

INTRODUCTION

In the domain of Artificial Neural Networks (ANNs) design, the most successful applications to the real-world problems reveal the need of designing optimal ANN structures rather than larger ones. 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 ANN architectures means to decide the number of layers in the ANN and the number of neurons in those layers. It is well known that a three layered ANN, consists of an input, a hidden and an output layer, can solve any kind of problem. Therefore, ANN architectures depend on the number of neurons in the hidden layers, since the numbers of input and output neurons are determined by the sizes of input and output vectors, respectively.

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 mostly depends on its architecture (Chauvin, 1990; Simon, 2003; Reed, 1993). 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 (Chauvin, 1990; Costa *et al*., 2002; Reed, 1993). 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 (Reed, 1993; Yeung and Zeng, 2002). To design an appropriate architecture for the solution of a given task is always an open challenge (Xiang *et al*., 2005; Yeung and Zeng, 2002).

There have been many attempts to design ANN architectures automatically, such as various constructive (Ash, 1989; Costa *et al*., 2002; Fahlman and Lebiere, 1990; Simon, 2003; Huang *et al*., 2005; Islam *et al*., 2003; Kwok and Yeung, 1999) pruning (Chung, 1998; Costa *et al*., 2002; Cun *et al*., 1990; Engelbrecht, 2001; Hagiwara, 1994; Hassibi and Stork, 1993; Lee and Park, 2002; Reed, 1993; Yeung and Zeng, 2002) and evolutionary (Fogel, 1995;

Corresponding Author: Suman Ahmmed, Department of Computer Science and Engineering, United International University, Dhaka, Bangladesh

Miller *et al*., 1989; Odri *et al*., 1993; Yao and Liu, 1997) 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. Unlike constructive and pruning algorithms, an evolutionary algorithm starts ANN design process with M networks where M is a user-specified parameter. It can add or delete neurons and/or connections during the evolution process until a near optimal architecture has been designed.

PROPOSED ALGORITHM

ADCSP mainly emphasis on correlation based pruning, but to make prior shortening, it reduces low information bearing (having lower standard deviation values) hidden neurons. Thereafter, it reduces the similar or dissimilar hidden neurons based on their correlations (both positive and negative) strength. As complete elimination of neurons could harm the network=s performance, merging strategy is used in ADCSP to handle this situation. To ensure the stability of the ANN after each reducing, ADCSP try to compensate proper substitutions so that the performance of the network does not fall under certain limit or the network could not move into wrong direction.

If a hidden neuron exhibits high correlation (either positive or negative) with other neurons in their responses, then it is considered redundant in ADCSP. It is therefore, possible to merge a pair of correlated neurons and replace them by a single neuron. The major steps of ADCSP are summarized in Fig. 1, which are explained further as follows.

Step 1: 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: Train the ANN by using Back Propagation (BP) algorithm until the error E reduces to a certain value. The training ends when the training error is still decreasing and the validation error starts to increase. In ADCSP, E is calculated according to following equation.

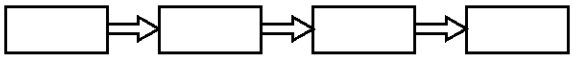

Fig. 1: Flow diagram of ADCSP

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

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: Compute the standard deviation of the output values for all hidden neurons in the ANN. If the standard deviations of the hidden neurons are under a threshold value, identify those as low information bearing neurons σ_{tr} and delete them all, otherwise go to next step. After pruning the error has been checked. If the error increases then send for BP retraining to reduce the errors. If error does not reduce under certain limit retrieve the pruned neurons. Go to next step.

Step 4: Compute correlations among hidden neurons pairs in an ANN. Both positive and negative correlations are calculated. The pairs having positive or negative correlations more than a threshold value ρ_{tr} are marked as M and they are selected for merging. If M marked pairs of neurons are found in the ANN then merge those pairs and replace the neurons. In spite of replacements, if the error of the ANN increases then send for BP retraining. If the error does not reduce under certain limit then restore the neurons and restart the algorithm further, otherwise fixes the final ANN architecture designed by ADCSP.

ADCSP uses replacement strategies to incur the losses due to pruning. The following subsections describe these components of ADCSP in detail.

Pruning by sensitivity: Standard deviation is used to measure how widely the output values of hidden neurons are dispersed from its average values. In ADCSP, a hidden neuron is identified as low information bearing if its standard deviation is small. Generally, a hidden neuron with small standard deviation delivers almost constant output value to the neurons in the succeeding layer. As a result one can easily replace the contributions of those neurons in ANN. To undermine the effect of pruning neurons, ADCSP implements a replacement strategy after each pruning. If a neuron (H_1) of hidden neuron is pruned then the connection weight of the bias neuron for the k^{th} output neuron is changed according to the following equation

$$W^{\circ}_{k,0}(t+1) = W^{\circ}_{k,0}(t) + W^{\circ}_{k,j}\bar{x}_j \quad (2)$$

Here, $W^{0}_{k,0}(t)$, the biases of the output neurons, where 0 represents biases, k represents output neurons i.e., {k = 1, 2, ...,..On},

$W^{0}_{k,j}\bar{x}j$, is the average weighted value feeds from the j^{th} pruned hidden neuron to the k^{th} output neuron.

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

Pruning by correlations: 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. If the error of the ANN increases after merging, ADCSP retrains the ANN to reduce the error. In the worst case scenario, if the retraining do not successful to reduce the error, ADCSP retrieves the last changes in the ANN architecture. The correlation between hidden neurons H_1 and H_2 is expressed by the following equation.

$$\rho_{H_1,H_2} = \frac{Cov(H_1,H_2)}{\sigma_{x_i}.\sigma_{y_i}} \quad (3)$$

Here, $\rho_{H1,H2}$ represents the correlation among H_1 and H_2 neurons. σ_{xi} and σ_{yi} are standard deviations of the output values of H_1 and H_2. $Cov(H_1, H_2)$ denotes the covariance. It can be calculated as:

$$Cov(H_1,H_2) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \quad (4)$$

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.

To compensate at merging, it is necessary to quantify the contribution of two correlated neurons in the ANN. If one can quantify the contribution of correlated neurons then he/she can try to make merging operation in such a way so that the contribution of the merged neuron will be nearly similar to that of the correlated pair of neurons.

Consider a three layered feed forward ANN and two correlated neurons H_1 and H_2 are selected to merge (Fig. 2). Let H_m be the merged neuron that will replace neurons H_1 and H_2 (Fig. 3). Thus H_m will be

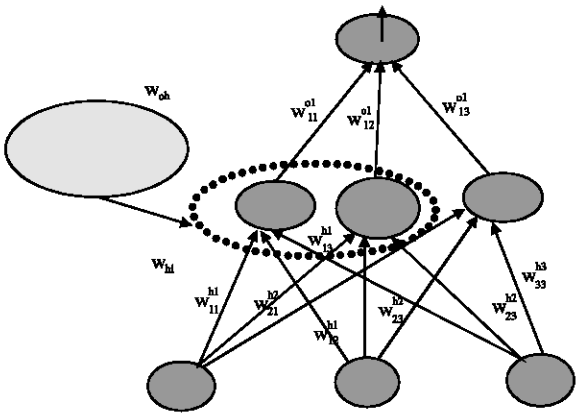

Fig. 2: Correlated neurons are selected

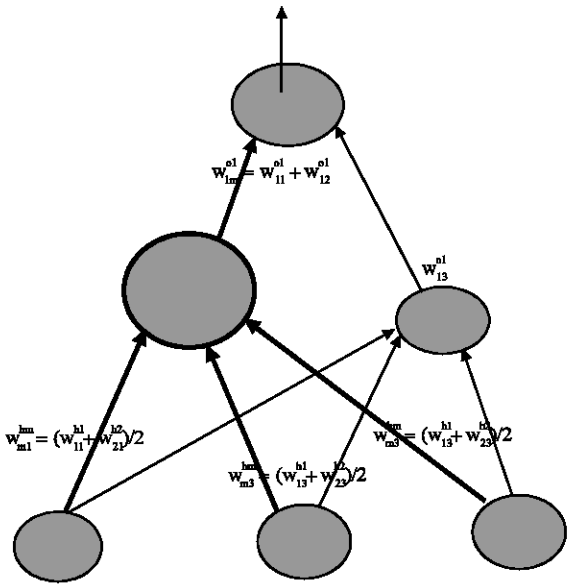

Fig. 3: Correlated neurons merged into one

$$W^{hm}_{m1} = \frac{W^{h1}_{11} + W^{h2}_{21}}{2},\quad W^{hm}_{m2} = \frac{W^{h1}_{12} + W^{h2}_{22}}{2},\quad W^{hm}_{m3} = \frac{W^{h1}_{13} + W^{h2}_{23}}{2} \quad (5)$$

This is true for all correlated hidden pairs. Besides Fig. 2 shows that, the total amount of input values supplied by the two correlated neurons (H_1 and H_2) to the output layer neuron O_1 is $(W^{o1}_{11} + W^{o1}_{12})$. ADCSP assigns summation weight to the merged neuron=s connection. Figure 3 shows, in case of merging H_1 and H_2 the new connection weight between O_1 and H_m is W^{o1}_{1m}, which is

$$W^{o1}_{1m} = (W^{o1}_{11} + W^{o1}_{12}) \quad (6)$$

Experimental results: ADCSP is applied on several well-known benchmark problems to evaluate its performance.

These are Australian Credit Card Assessment, Breast Cancer, Diabetes, Heart Disease, Iris, Soybean and Thyroid problems. The performance of ADCSP is measured in terms of the size of the ANNs, the number of epochs required for designing ANNs and classification accuracies of the designed ANNs. Each experiment was carried out 30 times in order to achieve fairness in the results.

Descriptions of the dataset: The detailed descriptions of the datasets are available at ics.uci.edu (128.195.11) in directory/pub/machine-learning-databases. Table 1 describes the characteristics of the different datasets.

Australian credit card assessment dataset: This is a credit card approval dataset problem. The problem is to assess the applications for Australian credit cards based on a number of attributes. This is a two-class problem. There are 690 examples in the dataset each of which consists of 51 real valued inputs. There are 307 examples of positive class and 383 examples of negative class.

Breast cancer dataset: This dataset was obtained from the University of Wisconsin Hospitals, Madison, Wisconsin, USA, by Dr. William H. Wolberg. The dataset was designed to diagnosis breast tumors as either benign or malignant. The dataset representing this problem contains a total number of 699 examples. Each example consists of 9 real value attributes as an input vector and represents two classes as output vector. Out of 699 examples 458 are benign examples and 241 are malignant examples.

Diabetes dataset: This dataset was originally donated by Vincent Sigillito from Johns Hopkins University by the National Institute of Diabetes and Digestive and Kidney Diseases. The problem posed here is to predict whether a patient would test positive or negative for diabetes according to criteria given by World Health Organization (WHO). This is a two-class problem with class value 1 and 2 interpreted as negative and positive results for diabetes. There are 500 examples of class 1 and 268 of class 2. There are 8 attributes for each example. The dataset is difficult to classify.

Heart disease dataset: This dataset comes from the Cleveland Clinic Foundation and was supplied by Rebert Detrano of the V. A. Medical center, Long Beach, CA. The purpose of the dataset is to predict the presence or absence of heart disease given the results of various medical tests carried out on a patient. There are two classes: presence or absence (of heart disease). The data here used has 920 examples each of which consists of 35 real valued inputs.

Table 1: Datasets used by ADCSP

	Number of		
Datasets	Total examples	Input attributes	Output classes
Australian credit card assessment	690	51	2
Breast cancer	699	9	2
Diabetes	768	8	2
Heart disease	920	35	2
Iris	150	4	3
Soybean	683	82	19
Thyroid	8124	21	3

Iris dataset: The classification of irises was chosen as an example of continuous valued input and binary output. Irises are classified into three categories: setosa, versicolor and virginia. Each category has 50 examples. Each example possesses four attributes: Sepal length, sepal width, petal length and petal width.

Soybean dataset: This dataset comes from the AANN version of the Asoybean dataset from the UCI machine learning repository. Donors of this dataset are Ming Tan and Jeff Schlimmer. These datasets were developed for developing an expert System for Soybean Disease Diagnosis. This is a problem of 19 output classes and has 82 input attributes. There are 683 examples are available in the dataset.

Thyroid dataset: This dataset comes from the Aann version of the Athyroid disease dataset from the UCI machine learning repository. Two files were provided Aann-train.data contains 3772 learning examples and Aann-test.data contains 3428 testing examples. There are 21 attributes for each example.

Experimental setup: In this experiment, the instructions described in benchmark methodologies (Prechelt, 1995, 1994, 1996) were followed. ADCSP partitioned datasets into three disjoint sets: a training set, a validation set and a test set. The training set is used to train ANNs by back-propagation learning algorithm (Simon, 2003). The validation set is used to evaluate the interim performance of ANNs and fixes the criteria for stopping the training of ANNs. Finally, the test set is used to measure the generalization ability of ADCSP. In these experiments, the output neurons are encoded 1 for representing a particular Class (C) and remaining neurons become 0. Actually, the most common winner-takes-all method is used in ADCSP to determine the class. Most widely used

sigmoid function is used as activation function in neurons of hidden and output layers. The number of input neurons was equal to the number of attributes of that dataset. The number of output neurons was equal to the number of classes of that dataset. Since ADCSP uses pruning approach, so the number of hidden neurons was taken larger than the sum of both input and output neurons divided by two. Initial connection weights of ANNs are assigned randomly between -0.20 and 0.20. Bias neurons in hidden and output layers are initially assigned 1.0 for all datasets. The learning rate varies from 0.50 to 0.70. The threshold values of standard deviation and correlation vary from 0.10 to 0.25 and 0.65 to 0.90, respectively. It becomes inevitable to take more neurons in the hidden layer when the input or output vector sizes seem larger. The initial hidden neurons for breast cancer, diabetes, heart and thyroid disease datasets are assigned from 7 to 10, 8 to 14, 8 to 18 and 8 to 16, respectively. For Australian credit card assessment, iris and soybean datasets the initial hidden neurons are chosen from 10 to 20, 6 to 8 and 10 to 30, respectively. For most of the data sets first 50, 25 and 25% number of examples are used as training set, validation set and test set, respectively.

Table 2 and 3 summarizes the results achieved by ADCSP for different problems. The number of epochs in Table 2 represents the total number of iterations required to attain the final ANN architectures. The classification error in Table 3 refers to the rate of wrong classification produced by designed ANNs on different datasets. It is clear from Table 2 and 3 that ANNs designed by ADCSP are very small in size, i.e., a small number of hidden neurons in the structure; convergence of ADCSP is very fast and classification errors of ADCSP are small. For example, an ANN having only two neurons can achieve an average error rate of 1.11% on the testing set for the cancer problem. In another case, ANNs designed by ADCSP for Australian credit card assessment dataset have only on average 1.2 hidden neurons and 13.64% classification error rate on testing set. Another important property of ADCSP is that the number of epochs required to finalize ANN architectures is also small in number. For example, for the diabetes problem, the average number of epochs required to finalize ANN was 44. The largest number of epochs required by ADCSP was for the thyroid problem and it was 138. This is not always true, however, that a problem having a large number of examples will need a large number of epochs in determining ANN architectures. Another difference between these two problems, as seen from Table 2 is the numbers of input attributes and output classes. The number of input attributes and output classes for the heart disease problem was 35 and 2, respectively, while they were 82 and 19 for the soybean problem. However, ADCSP required 8 and 51 number of epochs in determining near optimal ANN architectures for heart disease and soybean problems. This indicates that the number of epochs required in determining ANN architecture is not only dependent on the number of examples in problems. There are factors such as problem complexity, type and number of input attributes and output classes that might affect the number of required epochs to design a near optimal ANN architecture. The final number of hidden neurons for breast cancer is only 2. To achieve this ADCSP needs on average only 9 iterations. The minimum, maximum and average errors were 0.58, 1.15 and 1.11%, respectively.

Table 2: Architectures designed by ADCSP

Dataset		No. of epochs	No. of hidden neurons
Australian credit card assessment	Min	8	1
	Max	29	4
	Mean	12	1.2
Breast cancer	Min	2	2
	Max	16	2
	Mean	9	2
Diabetes	Min	21	2
	Max	117	3
	Mean	44	2.07
Heart disease	Min	8	2
	Max	35	4
	Mean	12	2.7
Iris	Min	146	2
	Max	267	2
	Mean	163	2
Soybean	Min	16	12
	Max	89	18
	Mean	51	15.17
Thyroid	Min	92	1
	Max	196	3
	Mean	138	1.7

Table 3: Classification accuracies of ANN designed by ADCSP

		Classification Error for		
Dataset		Training set (%)	Validation set (%)	Testing set (%)
Australian credit card assessment	Min	11.01	8.67	11.63
	Max	14.20	10.98	14.54
	Mean	12.01	9.50	13.64
Breast cancer	Min	3.43	2.29	0.58
	Max	4.00	2.86	1.15
	Mean	3.63	2.31	1.11
Diabetes	Min	18.49	22.40	22.40
	Max	22.66	23.96	27.60
	Mean	20.13	23.40	25.00
Heart disease	Min	6.74	16.96	16.96
	Max	15.44	18.70	21.74
	Mean	13.07	18.19	19.61
Iris dataset	Min	1.33	2.63	0.00
	Max	4.00	13.16	5.41
	Mean	2.58	6.75	1.44
Soybean dataset	Min	1.17	3.51	6.47
	Max	10.82	14.62	14.71
	Mean	2.63	4.99	7.57
Thyroid disease	Min	5.13	6.12	5.92
	Max	5.73	6.84	6.21
	Mean	5.38	6.33	6.08

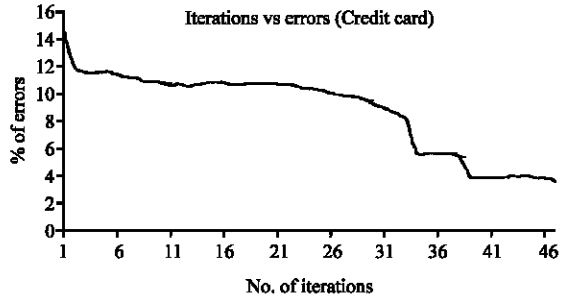

Fig. 4: Errors vs iterations

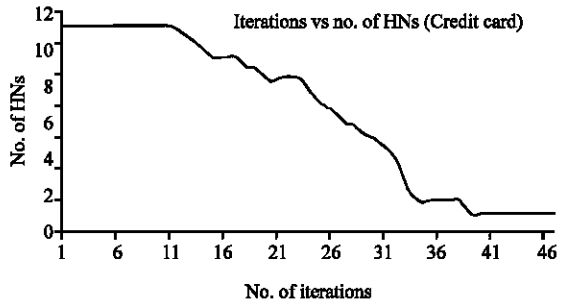

Fig. 5: No. of hidden neurons vs iterations

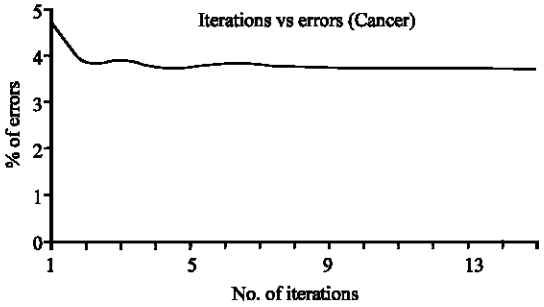

Fig. 6: Errors vs iterations

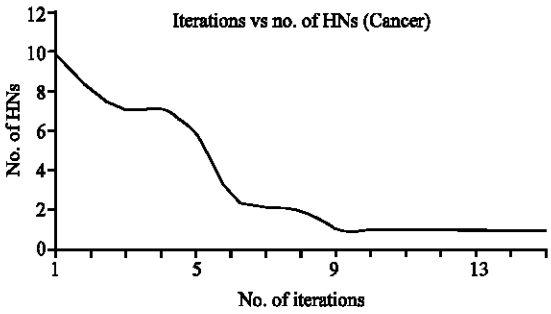

Fig. 7: Hidden neurons vs iterations

Figure 3 shows the architecture determination performance of ADCSP. In order to observe the architecture determination process for Australian Credit Card Assessment problem, Fig. 4 and 5 shows training processes of ADCSP in determining a near optimal ANN architecture for a particular problem. It is seen from Fig. 4 that the error of ANNs generally decreases as the training processes progress. However, it is seen that sometimes average errors increase temporarily and again started decrease after some iterations. This is due to effect of pruning hidden neurons from trained ANNs. The weight replacement strategy and retraining used in ADCSP help to reduce the errors quickly. For example, every hidden neuron of a trained ANN has some contributions to reduce its error. Whenever any hidden neuron is pruned from a trained ANN it does not get contribution from the pruned neuron. It is therefore obvious that the errors of ANNs will increase after pruning neurons. When ADCSP substitutes weights and retrains ANN after pruning, exiting neurons get chance for increasing their contributions so that the loss of pruned neurons can be compensated. This is the reason that after retraining the errors of ANN comes down in spite of temporary error increase due to pruning. Figure 5 shows the average number of hidden neurons with respect to the number of iterations for same dataset. At some points the average number of hidden neurons increases. This happens as this value is the average of 30 separate executions and in different executions the number of iterations varies. It is seen from Fig. 5 that the numbers of hidden neurons reduces and remains consistent. For example, for Australian credit card assessment dataset, the average numbers of hidden neurons remain same in 40 to 46 iterations which indicate that ADCSP could reduce the hidden neurons very quickly. Similarly, Fig. 6 and 7 show the converging process performance for cancer dataset. However for some dataset such as diabetes, ADCSP reduces hidden neurons very slowly. This is because diabetes problem is one of the hardest problems in machine learning and the data set is very noisy. Considering different datasets it can be concluded that ADCSP can design smaller sized ANNs for almost all datasets.

Robustness of ADCSP: The aim of this section is to observe that ADCSP can perform consistently in spite of the variations of its parameters, then it will prove the robustness of ADCSP. Basically, two control parameters are used in ADCSP: correlation threshold (σ) and standard deviation threshold (ρ). Another parameter is also considered for fairness i.e., the initial number of Hidden Neurons (HNs). These parameters were varied into a wide range to evaluate the robustness of ADCSP. Table 4 and 5 show the robustness performance of ADCSP for Australian Credit Card datasets. On the other hand Table 6 and 7 show the result set for Thyroid

Table 4: Number of epochs and HNS for Australian credit card assessment

Parameters threshold values		No. of epochs	No. of hidden neurons
σ = 0.15, ρ = ± 0.85, HN =14	Min	9	1
	Max	85	8
	Mean	22	2.17
σ = 0.10, ρ = ±0.85, HN = 14	Min	6	1
	Max	74	9
	Mean	19	2
σ = 0.15, ρ = ± 0.80, HN = 14	Min	9	1
	Max	85	8
	Mean	25	1.57
σ = 0.15, ρ = ± 0.85, HN = 14	Min	9	1
	Max	85	8
	Mean	22	2.17
σ = 0.15, ρ = ± 0.85, HN =16	Min	10	1
	Max	64	5
	Mean	21	2.1
σ = 0.15, ρ = ± 0.85, HN = 14	Min	9	1
	Max	85	8
	Mean	22	2.17

Table 5: Classification accuracy for Australian credit card assessment

Parameters threshold values		Classification error for Training set (%)	Validation set (%)	Testing set (%)
σ = 0.15, ρ = ± 0.85, HN = 14	Min	6.67	9.25	11.63
	Max	15.94	12.72	15.70
	Mean	11.88	10.56	13.93
σ = 0.10, ρ = ± 0.85, HN = 14	Min	5.22	9.25	11.05
	Max	15.36	13.30	17.44
	Mean	11.86	10.35	13.82
σ = 0.15, ρ = ± 0.80, HN = 14	Min	6.67	9.83	12.21
	Max	15.94	12.72	15.70
	Mean	11.83	10.67	13.95
σ = 0.15, ρ = ± 0.85, HN = 14	Min	6.67	9.25	11.63
	Max	15.94	12.72	15.70
	Mean	11.88	10.56	13.93
σ = 0.15, ρ = ± 0.85, HN = 16	Min	10.15	8.67	11.63
	Max	14.20	12.14	18.61
	Mean	11.87	10.64	13.93
σ = 0.15, ρ = ± 0.85, HN = 14	Min	6.67	9.25	11.63
	Max	15.94	12.72	15.70
	Mean	11.88	10.56	13.93

Table 6: Number of epochs and HNS for thyroid disease

Parameters threshold values		No. of epochs	No. of hidden neurons
σ = 0.25, ρ = ± 0.80, HN = 14	Min	92	1
	Max	196	3
	Mean	138	1.7
σ = 0.15, ρ = ± 0.80, HN = 14	Min	19	1
	Max	161	6
	Mean	52	3.23
σ = 0.20, ρ = ± 0.70, HN = 20	Min	73	1
	Max	208	4
	Mean	127	3.2
σ = 0.20, ρ = ± 0.70, HN = 12	Min	85	2
	Max	278	4
	Mean	128	2.73
σ = 0.20, ρ = ± 0.85, HN = 10	Min	27	0
	Max	99	4
	Mean	55	1.66

Table 7: Classification accuracy dataset

Parameters threshold values		Classification error for Training set (%)	Validation set (%)	Testing set (%)
σ = 0.25, ρ = 0.80, HN = 14	Min	5.13	6.12	5.92
	Max	5.73	6.84	6.21
	Mean	5.38	6.33	6.08
σ = 0.15, ρ = 0.80, HN = 14	Min	5.00	6.00	6.06
	Max	7.56	8.83	9.33
	Mean	6.53	7.61	7.92
σ = 0.20, ρ = 0.70, HN = 12	Min	4.65	5.96	5.95
	Max	5.69	7.87	6.94
	Mean	5.26	6.29	6.12
σ = 0.20, ρ = 0.70, HN = 20	Min	4.61	6.04	5.95
	Max	7.24	8.11	7.29
	Mean	5.21	6.41	6.26
σ = 0.20, ρ = 0.85, HN = 14	Min	4.69	6.04	5.98
	Max	5.97	8.11	7.96
	Mean	5.21	6.35	6.19

Table 8: Comparison between ADCSP and OBD (Eleuteri *et al*., 2005) for breast cancerdataset means not available

Algorithm	ADCSP	OBD
No. of epochs	24	-
No. of hidden neurons	1	10
No. of connections	14	-
Testing error rate	1.53%	3.50%

datasets. These tables also show that the performance of ADCSP is consistent in spite of varying different parameters.

Comparisons: Since ADCSP is an automated ANN architecture designing method, so the comparisons stand in terms of sizes of ANNs (number of hidden neurons and number of connections), convergences performance (number of epochs) and testing error rate (accuracies). Sometimes, direct comparisons among the ANN architecture designing methods become very difficult due to the lack of similar experimental assumptions and setups. In the followings, the outputs generated by ADCSP are compared with the results of some other prominent pruning algorithms.

Table 8 compares between ADCSP and OBD (Cun *et al*., 1990) for breast cancer dataset. The convergence performance of ADCSP is excellent. The average required number of epochs is only 24. Designed ANN has only one hidden neurons, on the other hand OBD requires ten hidden neurons; which is ten times larger. Besides the testing error rate of ADCSP is significantly better than OBD.

Similarly, Table 9 compares ADCSP with OBD for iris dataset.

Similarly Table 10-14 compare ADCSP with OBS (Hassibi and Stork, 1993) Impact Factor(ImF) (Lee and Park, 2002) Sensitivity analysis (Yeung and Zeng, 2002) variables selection (Eleuteri *et al*., 2005).

Table 9: Comparison between ADCSP and OBD (Eleuteri *et al.*, 2005) for iris dataset means not available

Algorithm	ADCSP	OBD
No. of epochs	163	-
No. of hidden neurons	2	-
No. of connections	19	28
Testing error rate	1.44%	13.00%

Table 10: Comparison between ADCSP and OBS (Eleuteri *et al.*, 2005) for thyroid dataset means not available

Algorithm	ADCSP	OBS
No. of epochs	195 (2 min pentium 4, 2.4GHz)	- (511min sparc Works. 200MHz)
No. of hidden neurons	2 (21-2-3)	7(8-7-3)
No. of connections	50	28
Testing error rate	6.66%	1.50%

Table 11: Comparison between ADCSP and ImF (Lee and Park, 2002) for breast cancer dataset

Algorithm	ADCSP	ImF
No. of epochs	24	51
No. of hidden neurons	1	1
No. of connections	14	14
Testing error rate	1.53%	2.22%

Table 12: Comparison between ADCSP and ImF (Lee and Park, 2002) for iris dataset

Algorithm	ADCSP	ImF
No. of epochs	163	455
No. of hidden neurons	2	1
No. of onnections	19	11
Testing error rate	1.44%	4.23%

Table 13: Comparison between ADCSP and sensitivity analysis Yeung and Zeng, 2002) for iris dataset means not available

Algorithm	ADCSP	Sensitivity measure
No. of epochs	66	22611
No. of hidden neurons	4	4
No. of connections	20	-
Testing error rate	5.73%	10.90%

Table 14: Comparison between ADCSP and variables selection (Eleuteri *et al.*, 2005) for iris dataset means not available

Algorithm	ADCSP	Variables selection
No. of epochs	24	-
No. of hidden neurons	1	5
No. of connections	14	-
Testing error rate	1.53%	2.20%

CONCLUSION

Correlation is such a property, thorough which the nature of different hidden neurons can be identified. Therefore, it becomes possible to design small ANNs for all the datasets. ADCSP uses effective replacement strategies to compensate the pruning effects. ADCSP always preserves its generalization property and avoid overfitting very strictly. This algorithm uses the formulae to find out standard deviations, correlations, errors and substituted weights etc., which were computationally inexpensive. As a result, the execution times of it for all datasets become fast. It tries to compensate the deleted neurons through proper replacements. Thus, the required numbers of epochs become smaller for almost all the datasets. It uses merge approach to design the ANN architectures automatically. ADCSP opens a new avenue that pruning or reduction is possible through merging. As per our best knowledge ADCSP is the first approach which uses merge operations to design ANNs automatically. Sometimes the accuracies of ADCSP are not the best. The main reason of it is that it never puts barriers to its generalization abilities.

FUTURE DIRECTIONS

ADCSP introduces correlations, there is a vast opportunity to apply this property into different ANN architecture designing methods, pruning or constructive methods etc. Besides, it is also expected that it will be useful in combining this technique with other techniques to enhance the overall performance of the ANN architecture. For discovering the full strength of this strategy, we will work on a wide variety of large datasets and try to analyze the algorithm further to improve its performance. The future aims on to establish correlation property, as a salient feature of ANNs, to show ADCSP as a performance enhancer for other methods, to develop this algorithm as computationally efficient to determine the architectures, to develop a module containing this method that will be applicable in different types of standard problems solvers, try to modify this method as less dependent on user-defined parameters or randomly assigned values.

REFERENCES

Ash, T., 1989. ADynamic neuron creation in back-propagation networks. Connection Sci., 1: 365-375.

Chauvin, Y., 1990. AGeneralization performance of over trained back propagation networks. Neural Networks, Proc. of the EUROSIP Workshop. Almedia, L.B. and C.J. Wellekens (Eds.), pp: 46-55.

Chung, S.B., 1998. A new pruning algorithm for structural Optimization of Neural Networks. Masters Thesis, Department of Electrical Engineering, KAIST, Taejon, Korea.

Costa, M.A., A. Braga and B.R. de Menezes, 2002. Constructive and Pruning Methods for Neural Network Design. Proc. of the VII Brazilian Symposium on Neural Networks (SBRN = 02).

Le Cun, Y., J.S. Denker and S.A. Solla, 1990. AOptimal Brain Damage., Proc. of the advances in Neural Information processing (2), Touretzky, D.S. (Ed.), pp: 598-605.

Eleuteri, A., R. Tagliaferri and L. Milano, 2005. A novel information geometric approach to variable selection in MLP networks. Neural Networks, 18: 1309-1318.

Engelbrecht, A.P., 2001. A New Pruning Heuristic Based on Variance Analysis of Sensitivity Information. IEEE Trans. Neural Networks, 12: 1386-399.

Fahlman, S.E. and C. Lebiere, 1990. The cascade-correlation learning architecture. Proc. of the Advances in Neural Information Processing System 2. Touretzky, D.S. (Ed.), pp: 524-532.

Fogel, D.B., 1995. Evolutionary Computation: Towards a New Philosophy of Machine Intelligence. New York, pp: 10017- 2394: IEEE. Press.

Hagiwara, M., 1994. Simple and effective method for removal of hidden units and weights. Neurocomputing, 6: 207-218.

Hassibi, B. and D.G. Stork, 1993. Second-order derivatives for network pruning: Optimal Brain Surgeon., proc. of the advances in Neural Information Processing Systems.

Lee Giles, C., S.J. Hanson and J.D. Cowan, Haykin, S., 2003. A Neural Networks: A Comprehensive Foundation (2nd Edn.), Pearson Education 5: 164-171.

Huang, G.B., P. Saratchandran and N. Sundararajan, 2005. Generalized Growing and Pruning RBF (CGAP-RBF) Neural Network for Function Approximation. IEEE Trans. Neural Networks, 16: 57-67.

Islam, M.M., X. Yao and K. Murase, 2003. Constructive Algorithm for Training Cooperative Neural Network Ensemble. IEEE Trans. Neural Networks, 14: 820-832.

Kwok, T.Y. and D.Y. Yeung, 1999. Constructive algorithms for structure learning in feedforward neural networks for regression problems. IEEE Trans. Neural Networks, 8: 630-645.

Lee, H. and C.H. Park, 2002. A Pruning Algorithm of Neural Networks using Impact Factor Regularization. Proc. of the 9th International Conference on Neural Information Processing (ICONIP)., 5: 2605-2609.

Miller, G.F., P.M. Todd and S.U. Hedge, 1989. Designing neural networks using genetic algorithms. Proc. of the 3rd International Conference on Genetic Algorithms and their Applications. Schaffer, J.D. (Ed.), Morgan Kaufmann, San Mateo, CA, pp: 379-384.

Mozer, M.C. and P. Smolensky, 1989. Skeletonization: A technique for trimming the fat from a network via relevance assessment. Proc. Adv. Neural Network Information Processing (1), Touretzky, D.S. (Ed.), pp: 107-115.

Odri, S.V., D.P. Petrovacki and G.A. Krstonosic, 1993. Evolutional development of a multilevel neural network. Neural Networks, 6: 583-595.

Prechelt, L., 1995. Some notes on neural learning algorithm benchmarking. Neurocomputing, 9: 343-347.

Prechelt, L., 1996. A quantitative study of experimental evaluation of neural network learning algorithms. Neural Network, 9: 457-462.

Prechelt, L., 1994. Proben1- A set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms. University Karlsruhe, Karlsruhe, Germany.

Reed, R., 1993. Pruning Algorithm-A survey. IEEE Trans. Neural Networks, 4: 740-747.

Shahjahan, M. and K. Murase, 2003. Dynamic Neuron Decaying Method for Pruning Artificial Neural Networks., E86-D: 736-751.

Xiang, C., Q. Ding and T. H. Lee, 2005. Geometric Interpretation and Architecture Selection of MLP. IEEE Trans. Neural Networks, 16: 84-96.

Yao, X. and Y. Liu, 1997. A new evolutionary system for evolving artificial neural networks. IEEE Trans. Neural Networks, 8: 694-713.

Yeung, D.S. and Xiao-Qin Zeng, 2002. Hidden neuron pruning for multilayer perceptrons using a sensitivity measure A. Proc. of the 1st International Conference on Machine Learning and Cybernetics, Beijing, pp: 1751-1757.