17th IEEE International Conference on Control Applications
Part of 2008 IEEE Multi-conference on Systems and Control
San Antonio, Texas, USA, September 3-5, 2008

Neural Network Dimension Selection for Dynamical System Identification

Devin Sabo, Xiao-Hua Yu

Abstract— **Choosing an appropriate size of a network is an important issue for any neural network applications. The common practice is to start with an "over-sized" network, then gradually reduces its size to find the optimal solution. In this paper, a new hybrid neural network pruning algorithm for multi-layer feedforward neural networks is investigated. Computer simulation results on system identification and pattern classification problems show this algorithm can significantly reduce the network dimension while still maintaining satisfactory identification and classification accuracy.**

I. Introduction

It is well known that before a neural network can be employed, its dimension (i.e., number of layers, number of neurons in each layer, and how they are connected) must be predetermined. In fact, a neural network is not fully utilized until it is properly trained with an appropriate size. However, finding the optimal dimension of a neural network is a very difficult task and often comes down to a guess work. A network that does not have enough parameters may be unable to learn the presented task; on the other hand, choosing a network that is larger than necessary may have some other limitations. A larger network yields more nodes, more weights, and more layers that result in additional arithmetic operations and high computation cost. For real time applications, the reduction of network size can save us precious hardware implementation time.

The ability to generalize, or to produce accurate values for the inputs not included in the training dataset, is one of the major benefits of using neural networks. An oversized network may over-fit the training data, and has poor generalization ability for the testing data. Of course, this is fine with a comprehensive training set since all possible input/output pairs are present and no generalization is needed. However, the amount of training data is usually limited; thus a trained network is expected to be able to perform well even on the previously unseen data. Therefore, the choice of an optimal network dimension is an important issue in neural network design and implementations. An ideal neural network should be able to perform well on both the training data and the unknown testing data while maintaining as compact a form as possible.

Aside from lucky guesses and extensive trial and error, there are two fundamental approaches to finding the appropriate size of a neural network. The first one is to start with a small network and slowly add more connections to it until an appropriate stopping criterion is satisfied [11]. The network is first trained at its minimum size; then more weights/nodes can be added and the new network will be retrained. This process can be repeated until certain performance index is met. The difficulties of this approach include when to start the growing process, and where to add the new connections/nodes in the network. In addition, the above procedures may be very tedious and time-consuming.

The second approach is to start with a network that is knowingly too large for the data, and then trim it down to the appropriate size. This is called "neural network pruning" and has been studied by many researchers in recent years ([1] [6] [9] [10] [12] [13]).

In this paper, a new pruning algorithm is investigated and compared with three existing ones, including the local sensitivity analysis method [13], the local variance sensitivity analysis [1], and the cross validation pruning algorithm [9]. This new algorithm combines the advantages of the above three methods, re-evaluates the network performance during pruning process, and iteratively prunes the neural network on a reduced set of connections if a pruning error occurs. Different data sets and various network configurations are studied in computer simulations. The results show that this new algorithm can significantly reduces the neural network size while still maintaining satisfactory generalization accuracy of the network, for both system identification and classification applications.

II. Review on Neural Network Pruning Algorithms

A typical neural network contains an input layer, an output layer, and one or more hidden layers. The number of outputs and inputs are usually fixed; while the number of hidden layers and number of hidden neurons in each hidden layer can be varied. In this research, we focus on the studies of pruning algorithms for multi-layer feedforward neural networks.

The simplest way to find the optimum network size is to use a brute force approach that produces all the combinations of networks within a desirable range, trains them, and chooses the best one. This process is usually not an efficient

Manuscript received February 4, 2008. This work was supported in part by the Department of the Navy, Office of Naval Research, under Award # N00014-06-1-1111.

Devin Sabo was with the Department of Electrical Engineering, California Polytechnic State University, San Luis Obispo, CA 93407, USA. He is now with Lockheed Martin Corporation, CA, USA.

Xiao-Hua Yu is with the Department of Electrical Engineering, California Polytechnic State University, San Luis Obispo, CA 93407, USA (e-mail: xhyu@calpoly.edu).

way to solve the problem. Another approach, the weight decay method (or the penalty method), is based on the assumption that smaller weights in a network have relatively small effect on the output of a node, especially when surrounded by significantly larger weights. This method adds a penalty term to the objective function to be minimized so that these smaller weights can eventually be forced to zero. However, this approach may eliminate weights that are actually crucial to the overall architecture of the network and thus produce a network with poor performance. Also, the added penalty term may create additional local minima on the error surface during training.

Huynh and Setiono [9] introduced the concept of cross-validation. The whole dataset is divided into two parts, i.e., the training set *T* and the cross validation set *C*. The pruning criterion is still based on the magnitude of each weight; however, a validation step is added to test the pruned network. At every pruning step, the performance of the network with reduced size is compared with the performance of the network before the current pruning phase. Let the performance criterion (objective function) on set *T* and set *C* be J_{TR} and J_{CV}, respectively (where $J(\bullet)$ can be the root-mean-square error (RMS), or the percentage of misclassified patterns). After pruning, a smaller neural network is obtained and the error on set *T* and set *C* be J'_{TR} and J'_{CV}, respectively. If

$$(J'_{TR} + J'_{CV}) < (J_{TR} + J_{CV}) \quad (1)$$

i.e., the pruned network outperforms the un-pruned one; then the pruned network is accepted and the pruning process can be continued. Otherwise, the network is restored to the size before the current pruning step. Obviously, the use of an additional cross validation set at each phase of the pruning takes into account that pruning is meant to not only reduce the size of a network, but also improve the network generalization capacity.

Rather than focusing on the magnitude of the weights in the network, the sensitivity based approach attempts to find the contribution of each weight in the network and then prunes the weights that have the least effect on the objective function. Mozer and Smolensky [12] suggested that the sensitivity of each weight can be found by measuring the difference on the performance of the network with/without that weight, i.e.,:

$$\begin{aligned} s_{j,i} &= J(w_{j,i} = 0) - J(w_{j,i} = w^f_{j,i}) \\ &= J(\text{without } w_{j,i}) - J(\text{with } w_{j,i}) \end{aligned} \quad (2)$$

where $s_{j,i}$ is the sensitivity (with respect to the removal of connection $w_{j,i}$); $w_{j,i}$ is the weight of the neural network from node *i* to node *j*; $w^f_{j,i}$ is the final value of weight $w_{j,i}$ when training is finished; and $J(\bullet)$ is the objective function.

Calculating Eq. (2) directly may be very time-consuming. Karnin [10] found an effective way to approximate it for the back-propagation algorithm:

$$s_{j,i} = \sum_{t=0}^{T-1} [\Delta w_{j,i}(t)]^2 \ \frac{w^f_{j,i}}{\eta (w^f_{j,i} - w^i_{j,i})} \quad (3)$$

where *T* is the total number of iterations (training epochs) needed to minimize the objective function *J*; η is the learning rate and $\Delta w_{j,i}$ is the change on weight $w_{j,i}$ in one training iteration. The absolute value of the estimated sensitivity for each weight, $|s_{j,i}|$, is then compared with a pre-determined threshold to determine whether the weight should be pruned or not. Note that this algorithm relies heavily on the selection of a threshold which must be determined beforehand which may differ between data sets and applications. If the threshold is too high, too many weights will be pruned and the pruned network may not function as desired; but if the threshold is too small, no weights will be pruned at all. Also, in this method, all the sensitivities in the network are compared with the same threshold, i.e., they are treated equally for pruning.

Ponnapelli et al. [13] suggested that the sensitivities of weights should only be compared with those related with the same node in the same layer. Thus, the concept of local relative sensitivity index (LRSI) is defined as the ratio of the sensitivity of a particular weight and the sum of all the sensitivities of the weights that are connected to the same node from the previous layer:

$$LRSI_{j,i} = \frac{|s_{j,i}|}{\sum_{m=1}^{M} |s_{j,m}|} \quad (4)$$

where *M* is the total number of connections to node *j* from the previous layer. For each node, any weight that has a local sensitivity less than a threshold will be pruned:

$$LRSI_{j,i} \le \beta \quad (5)$$

Even though the choice of the threshold (i.e., β) still depends on the rule of thumb, it is now a percentage which is relatively easier to be chosen. Note that this algorithm only considers weight removal; node pruning is not included. Theoretically, if all the weights that are connected to a single node are pruned, then this node can also be eliminated. However, this may take several rounds of pruning and training so it may not be a feasible solution in practice.

Engelbrecht [6] proposed a modified approach to sensitivity analysis. Instead of using the value of the sensitivity directly, Engelbrecht found the average sensitivity of a network parameter (e.g., weight or node) over all the patterns, and then developed a new measure called variance nullity. That is, if the variance of sensitivity of a network parameter over all the patterns (denoted by $\sigma^2_{\theta_k}$ for parameter θ_k) is close to zero and the average sensitivity (also over all the patterns) is small, then we conclude that this parameter has little or no effect on the output of the neural network over all patterns and therefore can be eliminated. The variance of sensitivity is defined as:

$$\sigma_{\theta_k}^2 = \frac{\sum_{p=1}^{P} (s_{\theta_k}^{(p)} - \tilde{s}_{\theta_k})^2}{P-1} \quad (6)$$

where P is the total number of patterns under consideration and $\tilde{s}_{\theta_i}$ is the average sensitivity over all the patterns:

$$\tilde{s}_{\theta_k} = \frac{\sum_{p=1}^{P} s_{\theta_k}^{(p)}}{P} \quad (7)$$

The parameter variance nullity (PVN) for each parameter is then defined as:

$$\gamma_{\theta_k} = \frac{(P-1)\,\sigma_{\theta_k}^2}{\sigma_0^2} \quad (8)$$

where σ_0^2 is a small constant value related with hypothesis test $H: \sigma_{\theta_k}^2 < \sigma_0^2$ [6].

This algorithm allows for pruning of both nodes and weights, with each parameter having a separate formula for the sensitivity calculation. The extension of a sensitivity measurement to nodes (not just weights) allows for the possibility of finding a smaller network, and also decreases the number of times to retrain the network before obtaining its final size.

However, as we discussed earlier, relying on one single value of σ_0^2 for the entire network can lead to problems. Fnaiech et. al. [1] suggested that parameters within the same layer should be considered "locally" rather than "globally", and defined a new pruning index called the local parameter variance nullity (LPVN). The PVN for all parameters in the same layer are summed up; then the LPVN for each parameter (which represents the relative importance of PVN of a parameter in the layer) can be obtained and used for pruning:

$$L\gamma_{\theta_k^{[l]}} = \frac{\gamma_{\theta_k^{[l]}}}{\sum_{k=1}^{K} \gamma_{\theta_k^{[l]}}} \quad (9)$$

where $L\gamma_{\theta_k^{[l]}}$ is the LPVN for layer l, and K is the total number of parameters in layer l. Note that in this algorithm, the pruning decision is still based on the hypothesis test H; thus choosing the appropriate threshold for LPVN is crucial to the success of this algorithm.

As a summary, all the above algorithms have their own advantages and limitations. For example, in the cross validation pruning algorithm (CVP) [9], the concept of cross-validation is introduced to provide a better criterion to evaluate the neural network performance (before and after pruning) at every step; however, this criterion still depends on the magnitudes of neural network weights. To avoid this problem, in the local sensitivity analysis method (KLSA) [13], local relative sensitivity is suggested; however, only weight pruning is considered in this algorithm (node removal is not included). The local variance sensitivity analysis (LVSA) [1] overcomes this limitation, but it still relies on the value of a threshold related with the hypothesis test.

III. The Novel Hybrid Pruning Algorithm

In this section, a novel pruning algorithm called hybrid sensitivity analysis with re-pruning (HSAR) is investigated. Both weight pruning and node pruning are considered. Pruning is based on sensitivity calculation and local variance nullity, and the performance of the neural network is re-evaluated using cross-validation at every pruning step.

One of the disadvantages of the existing algorithms is the tendency to get carried away with too many parameters pruned from the network in one step. Testing revealed that when pruning too many parameters in any single step leads to poor network performance, a pruning restoration is required. That is, all the nodes, weights, and biases in the network need to be restored from the configuration in the previous step. Therefore, in a multi-step algorithm (such as LVSA and CVP), more than one pruning restorations may be required before the pruning process could finish. Weights and nodes originally selected for elimination would remain in the network due to these pruning restorations. To overcome this limitation, the proposed algorithm iteratively prunes the neural network on a reduced set of connections if a pruning error occurs.

In this algorithm, the performances of the newly pruned and trained network are evaluated using the following:

$$\rho(\zeta J'_{TR} + J'_{CV}) < (\zeta J_{TR} + J_{CV}) \quad (10)$$

where ρ is a constant that gives priority to pruned networks, ζ ($\zeta < 1$) is another constant that encourages generalization capacity by favoring the cross validation error over the training error. If the new network fails to show an improvement over the old network, then restore the network to its last working configuration and start the re-pruning process; otherwise continue to the next pruning mode (either pruning weight or node).

In the re-pruning process, pruning is performed on a reduced parameter list:

$$n_{rp}(t+1) = n_{rp}(t)(1-\lambda) \quad (11)$$

where n_{rp} is the number of weights to re-prune and λ is the percentage of weights to deduct at each re-pruning step. For node, we have:

$$n_{rp}(t+1) = n_{rp}(t) - 1 \quad (12)$$

i.e., each successive re-pruning step only subtracts one node from the previous attempt. If there are no possible parameters to re-prune, go to the next layer; otherwise continue to prune the network using the reduced pruning list in Eq. (11) and (12). The flow chart of the algorithm is shown in Fig. 1.

IV. Computer Simulation Results

In this section, the proposed algorithm is applied to solve different pattern classification problems and compared with three existing algorithms described above (i.e, the local

sensitivity analysis method (KLSA) [13], the local variance sensitivity analysis (LVSA) [1], and the cross validation pruning algorithm (CVP) [9]). The datasets used in simulation can be found in [4], which is available from the machine learning repository, University of California, Irvine. The "Computer hardware data set" (CPU test) describes the relationship between the CPU performance and computer parameters such as machine cycle time, main memory and cache memory size, number of channels, etc. The "Iris data set" (Iris test), one of the best known data set in pattern classification applications, classifies the types of iris based on the width and length of its petals and sepals.

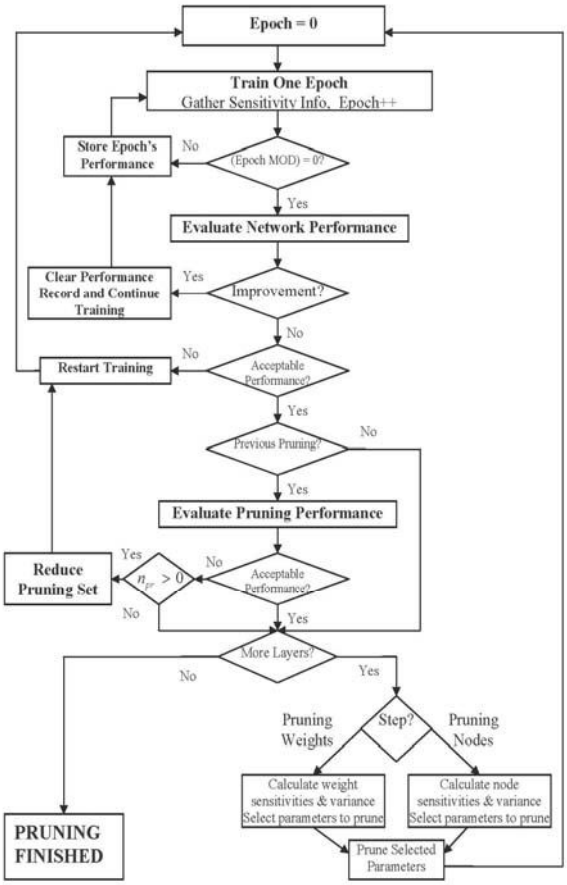

Fig. 1. Flow chart of the algorithm

In the following simulation examples, different initial network configurations are considered to fully test the abilities of each pruning algorithm on a variety of hidden node and hidden layer setups. Configuration 1 has one hidden layer with fifteen hidden nodes; configuration 2 has two hidden layers with ten nodes in each hidden layer; and configuration 3 has three hidden layers with five nodes in each hidden layer. Furthermore, each of the two data sets is divided into ten equal sub-sets, where eight of them are used for training, one is used for validation, and the remaining one is for testing. Each of the sub-dataset is used for training, validation, and testing on a rotation basis, resulting in a total of ten different data configurations. For example, the CPU test dataset contains totally 209 instances (with 9 attributes in each instance); so for each sub-dataset, there are about 21 instances. Similarly, the Iris data set contains totally 150 instances (with 4 attributes in each instance), results in 15 instances per sub-dataset.

The weights of all the neural networks are initialized at random before training. The same initial conditions are applied to all the pruning algorithms in each test. To minimize the influences of initial conditions to the test results, ten different sets of initial conditions are chosen for each neural network configuration and each data configuration. Therefore, for each neural network configuration, a total of 10*10 = 100 simulation runs are performed. This process is repeated for each of the four pruning algorithms for both CPU and Iris tests.

In the applications presented in this paper, the system outputs are all positive; so the following sigmoid (or logistic) function is chosen to be the activation function for each neuron:

$$g(u)=\frac{1}{1+e^{-u}} \tag{13}$$

For system identification problem (CPU test dataset), the objective function is to minimize the following performance index:

$$E=\frac{1}{MK}\left[\frac{1}{2}\sum_{m=1}^{M}\sum_{k=1}^{K}(y_{km}-d_{km})^2\right] \tag{14}$$

where y_{km} is the output at node k for pattern m, d_{km} is the target or desired value at node k for pattern m, M is total number of patterns or samples, and K is the total number of outputs.

The Iris dataset is a multi-class classification problem. Similar to [9], we define the cross-entropy objective function as:

$$E=-\sum_{m=1}^{M}\sum_{k=1}^{K}d_{km}\ln(y_{km}) \tag{15}$$

The backpropagation with momentum algorithm is employed for neural network training:

$$w(t)=w(t-1)+\Delta w(t) \tag{16}$$

$$\Delta w(t)=\eta\frac{\partial E}{\partial w}+\beta\Delta w(t-1) \tag{17}$$

As we discussed in section 2, one of the drawbacks of the existing algorithms is that they intend to prune too many parameters in one step. Multiple pruning restorations may be needed in a multi-step algorithm (such as LVSA and CVP) before the pruning process could finish. This is verified in Fig. 2 – 4 for LVSA algorithm, where Fig. 2 illustrates the average number of restorations needed in the CPU test for configuration 1 (i.e., 3-layer network), Fig. 3 shows the results of configuration 2 (i.e., 4-layer network) and Fig. 4 shows the results of configuration 3 (i.e., 5-layer network)

Fig. 2 indicates that for a typical 3-layer feedforward neural network and LVSA algorithm, the ideal case, i.e., zero pruning restoration only has 6% rate of occurrence; while the percentages for 1-, 2-, 3-, and 4-restoration are 25%, 29%, 25%, and 15%, respectively. For configuration 2 and 3, the

percentage of zero (or non-) restoration is 0. In Fig. 3, the percentages for 1-, 2-, 3-, 4-, 5-, and even 6-restoration are 7%, 22%, 39%, 21%, 8%, and 3%, respectively. Similarly, in Fig. 4, the percentages for 1-, 2-, 3-, 4-, 5-, and even 6-restoration are 4%, 10%, 20%, 31%, 18%, and 17%, respectively. In other words, the chances of having pruning restorations increase as the sizes of the original networks increase.

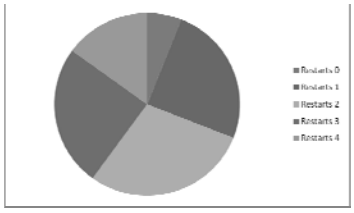

Fig. 2. Average pruning restores (CPU test, LVSA, configuration 1)

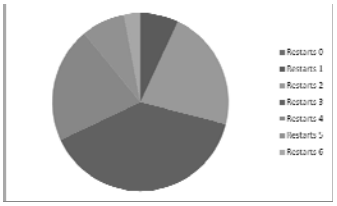

Fig. 3. Average pruning restores (CPU test, LVSA, configuration 2)

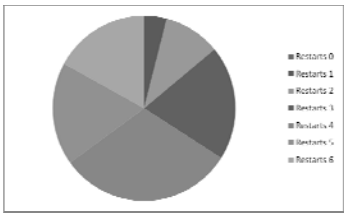

Fig. 4. Average pruning restores (CPU test, LVSA, configuration 3)

Similar trend can also be found in CVP algorithm. In fact, CVP is a two stage process (pruning the weights and then the nodes); so two is the largest possible number of restorations it may have. Simulation results show that all the neural networks have to experience at least one restoration during pruning; i.e., for all three configurations and data sets, the percentage of non-restoration is 0. For the first configuration, 39% for 1-restoration and 61% for 2-restorations; for the second configuration, the percentage for 1-restoration is reduced to 19% while the percentage for 2-restorations is increased to 81%. In configuration 3, the percentages for 1-restoration and 2-restoration are 22% and 78%, respectively.

Table 1 below shows the overall pruning capability of each of the four tested algorithms by displaying the mean and standard deviation (presented as (mean) ± (std)) of the pruning percentage (with respect to the original network). For example, in the CPU test, for the first neural network configuration, the KLSA algorithm can prune about 14.97% of the total neural network weights (an average for 100 runs with different initial conditions and data rotations), with the standard deviation of 14.55%. Similarly, under the same condition, the proposed HSAR algorithm can prune about 55.21% of the total neural network weights, with a standard deviation of 21.90%. Obviously, the new algorithm outperforms the KLSA algorithm.

Table 2 outlines the performance of each algorithm in terms of identification error (for the CPU test) and classification accuracy (for the Iris test) on the test dataset. The identification error gives a measure of the mean-square-error of the desired output and NN output; while the classification accuracy gives the percentage of the correct classification over the total patterns. It is shown that the overall accuracy of the new algorithm is similar or even better than other algorithms.

In table 3 and 4, the detailed information of where pruning occurs for each configuration and each algorithm is shown, where the first column shows the configuration, the second column shows the number on layers (e.g., layer 1 is the input layer, layer 2 is the first hidden layer, and layer 3 (if applicable) is the second hidden layer, etc.). Note that when a specific input doesn't have much effect on the network output performance, the input node or weight can also be removed. In column 3, "N" represents pruning on nodes and "W" represents pruning on weights. The rest of the columns show the numbers of nodes or weights pruned for different algorithms (average over 100 simulation runs as described before). For example, for the first NN configuration in CPU test, the average number of weights that can be pruned by CVP algorithm is about 28.38 while the average number of weights that can be pruned by the proposed algorithm is about 47.27. Note that the KLSA algorithm doesn't remove any NN node.

TABLE I
A Comparison of Each Algorithm's Pruning Percentages

Data	NN	KLSA	LVSA	CVP	HSAR
CPU Test	1	14.97 ± 14.55	35.15 ± 26.03	27.70 ± 17.50	55.21 ± 21.90
	2	13.41 ± 12.64	31.38 ± 22.16	16.92 ± 31.94	46.18 ± 21.72
	3	9.26 ± 6.69	28.21 ± 15.70	19.75 ± 28.40	40.20 ± 19.07
Iris Test	1	4.71 ± 5.76	18.53 ± 16.90	12.37 ± 14.46	55.26 ± 12.14
	2	5.98 ± 6.79	16.94 ± 18.20	24.42 ± 28.90	65.62 ± 13.26
	3	2.22 ± 3.96	15.44 ± 13.48	15.86 ± 18.59	62.60 ± 11.48

TABLE II
A Comparison of Each Algorithm's Accuracy

	KLSA	LVSA	CVP	HSAR
CPU Test				
NN 1	0.0037	0.0027	0.0035	0.0025
NN 2	0.0034	0.0031	0.0034	0.0030
NN 3	0.0055	0.0034	0.0052	0.0032
Iris Test				
NN 1	93.67%	93.53%	93.57%	92.73%
NN 2	94.17%	94.60%	94.52%	95.40%
NN 3	93.93%	93.50%	93.84%	93.60%

TABLE III
The Detailed Results for the CPU Test

Config	Layer	Node/ Weight	KLSA	LVSA	CVP	HSAR
1	1	N	-	1.20	0.05	0.97
	1	W	11.83	27.76	28.38	47.27
	2	N	-	0.07	0.00	0.06
	2	W	2.18	4.65	0.19	5.63
2	1	N	-	0.74	0.00	0.77
	1	W	5.16	18.6	14.15	23.96
	2	N	-	0.36	0.06	0.34
	2	W	12.83	33.9	8.61	46.92
	3	N	-	0.03	0.00	0.01
	3	W	0.93	2.03	0.07	3.60
3	1	N	-	0.56	0.00	1.00
	1	W	2.96	9.62	8.60	12.49
	2	N	-	0.32	0.04	0.39
	2	W	1.79	7.58	7.31	9.26
	3	N	-	0.21	0.02	0.17
	3	W	2.76	7.65	3.69	12.92
	4	N	-	0.04	0.00	0.06
	4	W	0.44	0.69	0.08	1.25

TABLE IV
The Detailed Results for the Iris Test

Config	Layer	Node/ Weight	KLSA	LVSA	CVP	HSAR
1	1	N	-	0.31	0.01	0.57
	1	W	0.61	9.44	7.31	30.51
	2	N	-	0.08	0.01	0.03
	2	W	2.78	12.41	0.61	32.64
2	1	N	-	0.31	0.05	1.03
	1	W	0.36	4.7	7.27	23.94
	2	N	-	0.2	0.01	0.6
	2	W	3.65	17.49	11.1	71.65
	3	N	-	0	0	0.07
	3	W	1.35	3.26	0.5	21.39
3	1	N	-	0.14	0	0.52
	1	W	0.5	2.03	4.12	9.35
	2	N	-	0.07	0.02	0.27
	2	W	0.63	3.81	4.46	14.6
	3	N	-	0.05	0.03	0.2
	3	W	1.01	5.02	2.75	14.54
	4	N	-	0	0.01	0.05
	4	W	0.45	2.78	0.5	8.38

In summary, the simulation results consistently indicate the HSAR algorithm can reduce the neural network size significantly without sacrificing the network performance.

V. CONCLUSION

In this research, a novel hybrid iterative pruning algorithm that can prune multi-layer feedforward artificial neural networks very effectively is presented and tested. Based on sensitivity analysis, cross validation, and iterative pruning, this algorithm outperforms three other existing pruning algorithms. Satisfactory simulation results are demonstrated in this paper; and more tests will be conducted to further investigate the performance of this new algorithm.

REFERENCES

[1] Fnaiech, N., Abid, S., Fnaiech, F., and Cheriet, M., "A modified version of a formal pruning algorithm based on local relative variance analysis", First International Symposium on Control, Communications and Signal Processing (2004), pp.849-852

[2] Abid, S., Fnaiech, F., and Najim, M., "A fast Feed-Forward Training Algorithm Using a Modified Form of the Standard Back-Propagation Algorithm", IEEE Transactions on Neural Network (2001), Volume 12, Issue 2, pp.424-430.

[3] Andrews, R., Diederich, J., Golea, M., and Tickle, A.B., "The truth will come to light: Directions and challenges in extracting knowledge embedded within trained artificial neural networks", IEEE Transactions on Neural Networks (1998), Volume 9, Issue 6, pp.1057-1068.

[4] Blake, E. K. C. and Merz, C., "UCI repository of machine learning databases", University of California, Irvine, Department of Information and Computer Science, 1998.

[5] Efe, M.O., Iplikci, S., Kayank, O., and Wilamowski, B., "An Algorithm for Fast Convergence in Training Neural Networks", International Joint Conference on Neural Networks, pp. 1778-1782, Washington DC, July 15-19, 2001.

[6] Engelbrecht, A.P., "A new pruning heuristic based on variance analysis of sensitivity information", IEEE Transactions on Neural Networks (2001), Volume 12, Issue 6, pp.1389-1399.

[7] Giles, C.L. and Lawrence, S., "Overfitting and Neural Networks: Conjugate Gradient and Backpropagation", Proceedings of the IEEE International Conference on Neural Networks (2000), pp.114-119.

[8] Haykin, S., "Neural networks: a comprehensive foundation," Upper Saddle River, N.J., Prentice Hall, 1999.

[9] Huynh, T.Q. and Setiono, R., "Effective neural network pruning using cross-validation", IEEE International Joint Conference on Neural Networks (2005), Volume 2, pp.972-977.

[10] Karnin, E.D., "A simple procedure for pruning back-propagation trained neural networks", IEEE Transactions on Neural Networks (1990), Volume 1, Issue 2, pp.239-242.

[11] Marsland, S., Nehmzow, S.U., and Shapiro, J., "A self-organizing network that grows when required", in Neural Networks, Volume 15 (2002), Isuue 8-9, pp.1041-1058.

[12] Mozer, M. C. and Smolensky, P., "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in Advances in Neural Information Processing (1989), D.S. Touretzky, Ed., pp.107-115.

[13] Ponnapalli, P.V.S., Ho, K.C., and Thomson, M., "A formal selection and pruning algorithm for feedforward artificial neural network optimization", IEEE Transactions on Neural Networks (1999), Volume 10, Issue 4, pp.964-968.

[14] Fnaiech, N., Fnaiech, F., and Cheriet, M., "A new feedforward neural network pruning algorithm: SSM-iterative pruning (SSMIP)", IEEE International Conference on Systems, Man and Cybernetics (2002)

[15] Fnaiech, F., Fnaiech, N., and Najim, M., "A new feedforward neural network hidden layer neuron pruning algorithm", IEEE International Conference on Acoustics, Speech, and Signal (2001), pp.1277-1280.