A COMPARISON OF CONSTRUCTIVE AND PRUNING ALGORITHMS TO DESIGN NEURAL NETWORKS

KAZI MD. ROKIBUL ALAM

Department of Computer Science and Engineering
Khulna University of Engineering and Technology
Khulna-9203, Bangladesh
Email: rokbcse@yahoo.com

BIKASH CHANDRA KARMOKAR

Department of Computer Science and Engineering
Khulna University of Engineering and Technology
Khulna-9203, Bangladesh
Email: bikash_kuet@yahoo.com

MD. KIBRIA SIDDIQUEE

Department of Computer Science and Engineering
Khulna University of Engineering and Technology
Khulna-9203, Bangladesh
Email: shakil_2k7@live.com

Abstract

This paper presents a comparison between constructive and pruning algorithms to design Neural Network (NN). Both algorithms have advantages as well as drawbacks while designing the architecture of NN. Constructive algorithm is computationally economic because it simply specifies straightforward initial NN architecture. Whereas the large initial NN size of pruning algorithm allows reasonably quick learning with reduced complexity. Two popular ideas from two categories: "cascade-correlation [1]" from constructive algorithms and "skeletonization [2]" from pruning algorithms are chosen here. They have been tested on several benchmark problems in machine learning and NNs. These are the cancer, the credit card, the heart disease, the thyroid and the soybean problems. The simulation results show the number of iterations during the training period and the generalization ability of NNs designed by using these algorithms for these problems.

Keywords**:** *Constructive algorithm; Pruning Algorithm; Generalization ability.*

1. Introduction

While designing NN, the architecture and the training parameters are the crucial issues of its research because they significantly affect the accuracy of the resultant model as well as the training time. Selection of NN architecture by manual design may be appropriate when there are experienced human experts with sufficient prior knowledge of the problem to be solved. However, it is certainly not the case for those real-world problems where much prior knowledge is unavailable. In back-propagation algorithm [3], NN deploys a fixed architecture, whereas constructive and pruning, both algorithms adopt dynamic NN architectures. Both of them are supervised learning algorithms [4] and their dynamic architectures are widely applied to solve real-world problems. However their mechanisms of attaining dynamic NN architectures are completely different. Therefore to deal real-world problems, the selection of suitable NN architecture is still a question.

In constructive algorithm, to specify an initial NN architecture is simple. Therefore it is usually faster and cheaper to build a NN. It can easily be generalized to add a group of hidden nodes, instead of just one, to the NN simultaneously, by requiring optimizing the same objective functions. It starts a NN with a small architecture *i.e*. small number of hidden layers, nodes and connections. It always searches for small NN solution first, then adds hidden nodes and weights incrementally until a satisfactory solution is found [5]. It is thus computationally more economic than pruning algorithm. However to train successively smaller NNs until the smallest one is found; can be time consuming. Besides, it may be sensitive to initial conditions and learning parameters and be more likely to become trapped in local minima [6].

A pruning algorithm performs the opposite as the constructive algorithm does, i.e. trains a NN larger than necessary and deletes unnecessary layers, nodes and connections during training [6]. The large initial size allows the NN to learn reasonably quickly with less sensitivity to initial conditions and local minima while the reduced complexity of the trimmed system favors improved generalization. However for pruning algorithm, one does not know in practice how big the initial NN should be. Also because of layers, nodes, connections etc. deletion, there is a possibility of losing information, and the majority of the training time is spent on NNs larger than necessary [6]. In this paper a comparison between constructive and pruning algorithms has been presented.

The rest of this paper is organized as follows: Section 2 describes "cascade–correlation" (CC) [1], a major category of constructive algorithm. Section 3 illustrates "skeletonization" [2], a foremost example of pruning algorithm. Section 4 presents results of the experimental studies. Finally, section 5 concludes with a summary of the paper.

2. Cascade−Correlation

Cascade-correlation is the combination of two key ides: the first is the cascade architecture, in which hidden node is added to the NN one at a time and do not change after they have been added. The second is the learning algorithm, which creates and installs the new hidden node. For each new hidden node, it is attempted to maximize the magnitude of the correlation between the node's output and the residual error signal is tried to eliminate [1].

The cascade architecture begins with some input and one or more output nodes, but no hidden node. The number of inputs and outputs is dictated by the problem and by the I/O representation. Every input is connected to every output node by a connection with an adjustable weight. There is also a bias input, permanently set to +1. The output node may just produce a linear sum of their weighted inputs, or they may employ some non-linear activation function. Hidden node is added to the NN one by one. Each new hidden node receives a connection from each of the NN's original inputs and also from every pre-existing hidden node. The hidden node's input weights are frozen at the time the node is added to the NN. Only the output connections are trained repeatedly. Each new node therefore adds a new one-node "layer" to the NN, unless some of its incoming weights happen to be zero. There are a number of possible strategies for minimizing the NN depth and fan-in as new nodes are added.

At some point, this training will approach an asymptote. When no significant error reduction has occurred after a certain number of training cycles, a new node is added in hidden layer. If it is satisfied with the NN's performance, training is stopped [1].

2.1. *Major steps of cascade−correlation*

Step 1: If further training yields no appreciable reduction of error, a hidden node is 'recruited'.
Step 2: A pool of hidden nodes (generally 8) is created and trained until their reduction halts. The hidden node with the greatest correspondence to overall error (the one that will affect it the most) is then installed in the NN and the other is the discarder.
Step 3: The new hidden node 'rattles' the NN and significant error reduction is accomplished after each hidden node is added. This ingenious design prevents the moving target problem by training feature detectors one by one and only accepting the best.
Step 4: The weights of hidden node are static; once they are initially trained, they are not touched again. The features they identify are permanently cast into the memory of the NN [1]. Correspondence to overall error (the one that will affect it the most) is then installed in the NN and the other is the discarder.

3. Skeletonization

Skeletonization estimates the least important units of the NN and deletes them during training [6]. A measure of the relevance ρ of a unit is the error when the unit is removed minus the error when it is left in place. Instead of calculating this directly for each and every unit, ρ is approximated by introducing a gating term α for each unit such that $o_j = f(\sum_i w_{ji} \alpha_i o_i)$ (1)

where o_j is the activity of unit j, w_{ji} is the weight from unit i to unit j, and f is the sigmoid function. If α = 0, the unit has no influence on the network; if α = 1, the unit behaves normally. The importance of a unit is then approximated by the derivative $\hat{\rho}_i = -\frac{\partial E^l}{\partial \alpha_i}\Big|_{\alpha_i=1}$ (2)

which can be computed by back-propogation [3]. Since this is evaluated at α = 1, α is merely a notational convenience rather than a parameter that must be implement in the NN. When $\hat{p}_i$ falls below a certain threshold, the unit can be deleted.

The usual sum of squared errors is used for training. The error used to measure relevance is

$$E^l = \sum \left| t_{pj} - o_{pj} \right| \quad (3)$$

rather than the sum of squared errors, because this provides a better estimate of relevance when the error is small. An exponentially decaying average is used to suppress fluctuations

$$\hat{\rho}_i(t+1) = .8\hat{\rho}_i(t) + .2\frac{\partial E(t)}{\partial \alpha_i} \quad (4)$$

The mechanism proposed in [7] studied the effect of this pruning method on the fault tolerance of the system. Interestingly, they found that the pruned system is not significantly more sensitive to damage even though it has fewer parameters. When the increase in error is plotted as a function of the magnitude of a weighted deleted by a fault, the plots for the pruned and unpruned NNs are essentially the same. They also found that the variance of the weights into a node is a good predictor of the node's relevance and that the relevance of a node is a good predictor of the increase in root mean square (rms) error expected when the node's largest weight is deleted [6].

4. Experimental Studies

Cascade-correlation [1] and Skeletonization [2] algorithms have been applied for well-known benchmark problems such as the cancer, the credit card, the heart disease, the thyroid and the soybean datasets. All the datasets were obtained from the UCI machine learning benchmark repository. To calculate the generalization ability of CC [1] and Skeletonization [2], winner−takes−all combination method was investigated.

The learning rate parameter was set between [0, 1]. The initial random weights were set between [–0.5, 0.5]. The sigmoid activation function that have been used is $[1/(1+e^{(-ax)})]$. To get the result, the NN have been trained several times and the best results for every problem have been chosen.

4.1. *Description of datasets*

4.1.1. The Breast Cancer Dataset

This problem has been subject of several studies on NN design. The dataset representing this problem contains 9 attributes and 699 examples. This is a two−class problem. The purpose of the dataset was to classify a tumor as either benign or malignant [8].

4.1.2. The Credit Card Dataset

Each example represents a real credit card application and the output describes whether the bank (or similar institution) granted the credit card or not. Predict the approval or non-approval of a credit card to a customer. This dataset contains 51 inputs, 2 outputs and 690 examples. This dataset has a good mix of attributes: continuous, nominal with small number of values, and nominal with large number of values. There are also a few missing values in 5% of the examples, 44% of the examples are positive [8].

4.1.3. The Heart Disease Dataset

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. This is a reduction of the number of classes in the original dataset in which there were four different degrees of heart disease. The dataset contains 35 inputs, 2 outputs, 920 examples [8].

4.1.4. The Soybean Dataset

Most attributes have a significant number of missing values. Many results for this learning problem have been reported in the literature, but these were based on a large number of different versions of data. The dataset contains 82 inputs, 19 outputs, and 683 examples [8].

4.1.5. The Thyroid Dataset

Based on patient query data and patient examination data, the task is to decide whether the patient's thyroid has over function, normal function or under function. This dataset has 21 inputs 3 outputs and 7200 examples. For various attributes there are missing values, which are always encoded using a separate input. Since some results for this dataset using the same encoding are reported in the literature thyroid is not a permutation of the original data but retains the original order instead example [8].

Table 4.1. Variation of NN architecture for some datasets by using cascade-correlation.

Dataset	Initial Architecture	Final Architecture
Cancer	9–1–2	9–7–2
Credit card	51–1–2	51–8–2
Heart disease	35–1–2	35–7–2
Soybean	82–1–19	82–20–19
Thyroid	21–1–3	21–5–3

Table 4.2. Variation of NN architecture for some datasets by using skeletonization.

Dataset	Initial Architecture	Final Architectur
Cancer	9–9–2	9–4–2
Credit card	51–11–2	51–5–2
Heart disease	35–10–2	35–4–2
Soybean	82–25–19	82–16–19
Thyroid	21–8–3	21–5–3

Table 4.3. Number of iterations and error for cascade-correlation.

Dataset	Iterations	Error (%)	
		Training	Classification
Cancer	101	0.990	1.892
Credit card	240	0.73	13.375
Heart disease	210	4.40	16.965
Soybean	200	0.0009	5.987
Thyroid	240	0.69	5.144

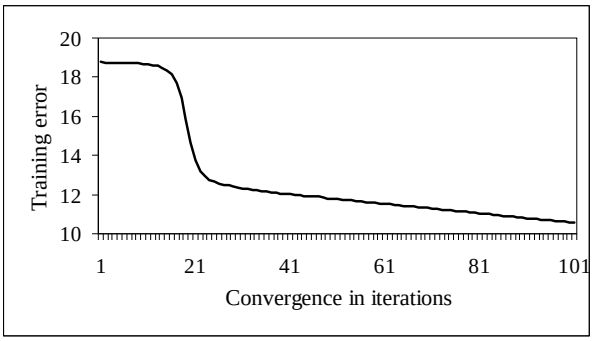
Table 4.4. Number of iterations and error for skeletonization.

Dataset	Iterations	Error (%)	
		Training	Classification
Cancer	94	1.196	1.716
Credit card	233	0.69	13.114
Heart disease	217	4.53	17.135
Soybean	193	0.0039	6.278
Thyroid	253	0.73	5.233

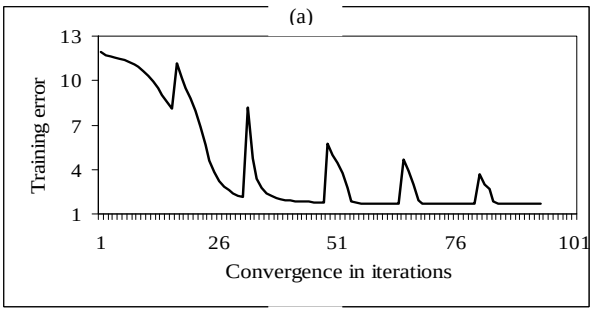
4.2. *Experimental results*

Variation of architecture of NN for CC [1] algorithm has been presented in Table 4.1 and for skeletonization [2] algorithm has been presented in Table 4.2. Also the number of iterations, the training and the classification error for CC [1] algorithm has been presented in Table 4.3 and for skeletonization [2] algorithm has been presented in Table 4.4. Error curve of individual NN in Fig. 4.1 (a) and Fig. 4.1 (b) have been shown for CC [1] and skeletonization [2] respectively for the cancer dataset. Also in Fig. 4.2 (a) and 4.2 (b), curve of hidden nodes addition for CC [1] and hidden nodes deletion for skeletonization [2] algorithms are shown respectively for the cancer dataset.

In case of the cancer dataset, for example, the NN started with architectures (9−1−2) (Fig.3.2 (a)). CC [1] algorithm was applied to determine the appropriate number of hidden nodes of individual NN. Here nodes were added because they possessed insufficient architecture. After hidden nodes addition, the final architecture of the NN was (9−7−2). Similarly for the cancer dataset, skeletonization [2] algorithm was applied to determine the appropriate architecture of individual NN. Here the NN started with architecture (9–9–2) (Fig.3.2 (b)), then nodes were deleted because it had the architecture larger than necessary. After hidden nodes deletion, the final architecture of the NN was (9−4−2).

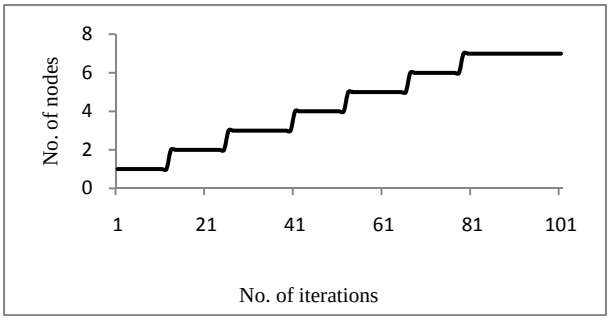


(a)

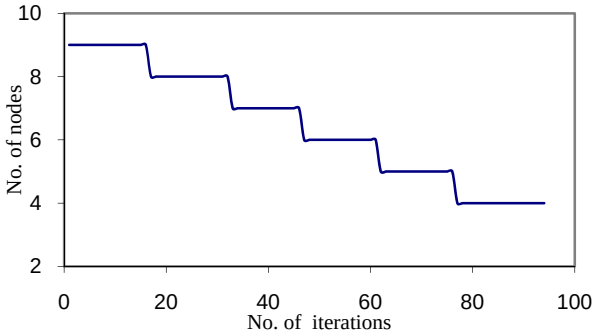


(b)

Fig 4.1 The error of the NN for the cancer dataset for (a) CC algorithm and (b) skeletonization algorithm.



(a)



(b)

Fig 4.2 For NN training for the cancer dataset, hidden nodes: (a) addition by CC algorithm and (b) deletion by skeletonization algorithm.

4.3. *Comparison*

In this section, the results of the CC [1] and skeletonization [2] algorithms were compared with single NN architecture (Stan) by Optiz and Maclin [9]. For results they have used UCI machine learning dataset repository.

Table 4.5. Comparison of classification errors among CC [1], skeletonization [2] and Stan [9].

Dataset	CC (%)	Skeletonization (%)	Stan (%)
Cancer	1.892	1.716	3.4
Credit Card	13.375	13.114	14.8
Heart disease	16.965	17.135	18.6
Soybean	5.987	6.278	9.2
Thyroid	5.144	5.233	–

4.4. *Discussion*

For CC [1] and skeletonization [2] algorithms, better results have been found than Stan [9]. Back propagation uses fixed architecture whereas skeletonization [2] algorithm starts with architecture larger than necessary *i.e.* with huge hidden nodes. In skeletonization [2] algorithm, nodes are deleted one after another as required. When the architecture is sufficient to cope pace with the problem space, node deletion is stopped.

In CC [1] algorithm, new information is added to an already trained NN. Training on a new set of examples may alter a NN's output weights. At any given time, only one layer of weights in the NN can be trained. The rest of the NN is not changing, so results can be cached. There is never any need to propagate error signals backwards through the NN connections. A single residual error signal can be broadcast to all candidates. The weighted connections transmit signals in only one direction [1].

The comparison of errors for the datasets represented in Table 4.5 shows that for some problems the best result is generated by CC [1] algorithm and for some other problems the best result is generated by skeletonization [2] algorithm. Table 4.5 also shows that both CC [1] and skeletonization [2] algorithms perform better than Stan [9].

5. Conclusions

Comparisons have been presented between CC [1] and skeletonization [2], the most common constructive and pruning algorithms to design NN. The algorithms have been applied to benchmark problems, which are popular both in case of machine learning and NN community. The experimental results for the cancer, the credit card, the heart disease, the thyroid and the soybean problems show the generalization ability of NN by using CC [1] and skeletonization [2] algorithms. For every problems CC [1] and skeletonization [2] algorithms outperforms over back-propagation [3] algorithm.

References

[1] Fahlman S. E.; Lebiere C., (1990): The Cascade–Correlation learning architecture. in D. S. Touretzky, *Advances in Neural Information Processing Systems* 2, pp. 524–532, San Maeto, CA, Margan Kaufman.

[2] Mozer M. C.; Smolensky P., (1989): Skeletonization: A technique for trimming the fat from a network via relevance assessment. in D. S. Touretzky, *Advances in Neural Information Processing Systems* 1, pp. 107–115, Ed. Denver.

[3] Rumelhart D. E.; Hinton G. E., Williams R. J., (1986): Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition.* vol. I, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, pp. 318–362.

[4] Kwok T. Y.; Yeung D. Y., (1997b): Objective functions for training new hidden units in constructive neural networks. *IEEE Transactions on Neural Networks*, 8, pp. 1131–1148.

[5] Ash T., (1989): Dynamic node creation in back propagation networks. *Connection Science*, 1, pp. 365−375.

[6] Reed R., (1993): Pruning algorithms−a survey. *IEEE Transactions on Neural Networks*, vol 4, no. 5, pp 740−747.

[7] Segee B. E.; Carter M. J., (1991): Fault tolerance of pruned multilayer networks. in *International Joint Conference on Neural Networks*, vol. II (Seattle), pp. 447–452.

[8] Prechelt L., (1994): PROBEN1−a set of benchmarks and benchmarking rules for neural network training algorithms. *Technical Report* 21/94, Faculty of Informatics, University of Karlsruhe, Germany.

[9] Opitz D.; Maclin R., (1999): Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, vol 11 pp. 169–198.