

A COMPARISON OF DIFFERENT CONSTRUCTIVE ALGORITHMS TO DESIGN NEURAL NETWORKS

Kazi Md. Rokibul Alam, Md. Shamsul Huda, Md. Mirza Hafiz Al-asad and Md. Mostafa Zaman

Department of Computer Science and Engineering

Khulna University of Engineering and Technology, Khulna-9203, Bangladesh.

Email: rokib30@yahoo.com, shuda9203@yahoo.com, mirza_755@yahoo.com, rajib307@yahoo.com

ABSTRACT

This paper presents a comparison among the constructive algorithms to design Neural Network (NN). Constructive algorithms provide a large number of benefits for NN architecture design. It is straightforward to specify the initial NN architecture. NNs with constructive algorithms are computationally economic. Also they are universal approximator. Two most popular constructive algorithms: “dynamic node creation” and “cascade-correlation” 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.

1. INTRODUCTION

Constructive algorithms are supervised learning algorithms for NN [1]. They start individual NN with a small architecture (i.e. small number of hidden layers, nodes and connections), then add hidden nodes and weights incrementally until a satisfactory solution is found [2]. The main categories of constructive algorithms are “dynamic node creation (DNC) [2]”, “cascade-correlation (CC) [3]” etc. In this paper we have presented a comparison between two constructive algorithms, the ‘DNC’ and the ‘CC’ to design NN.

Constructive algorithms can learn fast and have dynamic architecture, where in case of back-propagation algorithm [7], NN has fixed architecture. They can build deep NN (high-order feature detectors) without the dramatic slowdown [7]. Constructive algorithms are straightforward to specify an initial NN and moreover, 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.

A pruning algorithm performs the opposite as the constructive algorithm does, i.e. deletes unnecessary layers, nodes and connections during training [4]. For this, there is a possibility of losing information. In constructive algorithms it is straightforward to specify an initial NN whereas for pruning algorithms, one does not know in practice how big the initial NN should be. Constructive algorithms always search for small NN solution first. They are thus more computationally economic than pruning algorithms, in which the majority of the training time is spent on NNs larger than necessary [4].

The rest of this paper is organized as follows: Section 2 describes DNC [2] and CC [3] the main two categories of constructive algorithms. Section 3 presents results of our experimental study. Finally, section 4 concludes with a summary of the paper.

2.1 Description of dynamic node creation

DNC adjusts the weights in a NN by training topology. It begins with minimal NN, then trains and add new hidden node one by one that establishing a multiplayer structure. It begins a single node in a hidden layer, and then starts training. After some iteration if the error is not minimized then it adds new hidden node and trains again. This procedure is continued until the error is minimized [2].

Major steps of dynamic node creation

Step 1: Create an initial NN with only input and output nodes. Randomly initialize the connection weights of the NN within a certain range.

Step 2: Train the NN by using a training algorithm (e.g., back-propagation [7]) for a certain number of training epochs.

Step 3: Calculate the error of the NN by using the training set or the validation set. Stop the training

process if the minimum error criterion has been met. Otherwise, continue.

Step 4: Add one hidden node to the NN if the error of the NN does not significantly reduce after some training epochs. The assumption is that the NN has an inappropriate architecture. Go to the step 2 [2].

2.2 Description of cascade-correlation

Cascade-correlation is the combination of two key ideas: 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, we attempt to maximize the magnitude of the correlation between the node's output and the residual error signal we are trying to eliminate [3].

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 [3].

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 [3]. 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 EXPERIMENTAL STUDIES

We have applied DNC [2] and CC [3] algorithms 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 DNC [2] and CC [3], 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]. We have used sigmoid activation function $[1/(1+e^{-ax})]$. To get the result, we have trained our NN several times and have chosen the best result for every problem.

3.1 Description of datasets

A. 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 [5].

B 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 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 [5].

C. 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 [5].

D. 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 [5].

E. 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 23 inputs 2 outputs 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 [5].

Table 3.1: Variation of NN architecture for some datasets by using dynamic node creation.

Dataset	Initial Architecture	Final Architecture
Cancer	9-1-2	9-6-2
Credit card	51-1-2	51-9-2
Heart disease	35-1-2	35-8-2
Soybean	82-1-19	82-22-19
Thyroid	21-1-3	21-6-3

Table 3.2: Variation of NN architecture for some datasets by using cascade-correlation.

Dataset	Initial Architecture	Final Architecture
Cancer	9-1-2	9-5-2
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 3.3: Number of iterations and error for dynamic node creation.

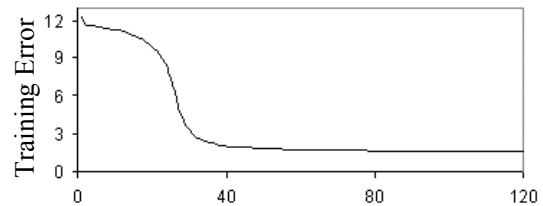
Dataset	No. of Iterations	Error (%)	
		Training	Classification
Cancer	116	1	2.156
Credit Card	319	0.77	14.531
Heart disease	280	5.00	17.425
Soybean	300	0.015	6.029
Thyroid	350	0.71	5.891

Table 3.4: Number of iterations and error for cascade-correlation.

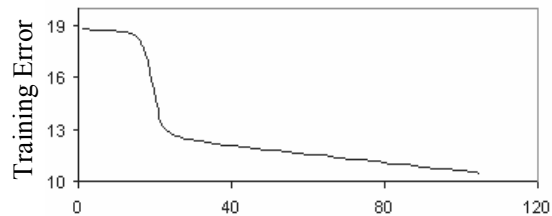
Dataset	No. of Iterations	Error (%)	
		Training	Classification
Cancer	101	0.990	1.892
Credit card	240	0.73	13.375
Heart disease	210	4.40	16.965
Soybean	200	0.0009	5.987
Thyroid	240	0.69	5.144

3.2 Experimental results

Variation of architecture of NN for DNC [2] algorithm has been presented in Table 3.1 and for CC [3] algorithm has been presented in Table 3.2. Also the number of iterations, the training and the classification error for DNC [2] algorithm has been presented in Table 3.3 and for CC [3] algorithm has been presented in Table 3.4. We have shown error curve of individual NN for which DNC [2] was applied for the cancer and the heart disease datasets. Also curve of hidden nodes addition is shown for cancer dataset for CC [3] algorithm.



(a)



(b)

Fig 3.1 The error of the individual NN by dynamic node creation algorithm for (a) the Cancer dataset (b) the Heart disease dataset.

In case of the cancer dataset, for example, the NN started with architectures (9–1–2) (Fig.3.2). CC [3] algorithm was applied to determine the appropriate numbers of hidden nodes of individual NNs. Here nodes were added because they possessed insufficient architecture. After hidden nodes addition, the final architecture of the NN was (9–5–2). Similarly for heart disease dataset, we

started with NN architecture (35-1-2) and finally got (35-8-2) for DNC [2] algorithm.

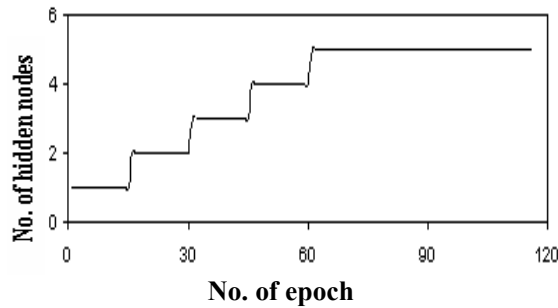


Fig. 3.2 Hidden nodes addition for NN training with cascade-correlation for the Cancer dataset.

3.3 Comparison

In this section, we have compared the results of the DNC [2], CC [3] with single NN architecture (Stan) by Optiz and Maclin [6]. For results they have used UCI machine learning dataset repository.

Table 3.5: Comparison of errors among DNC [2], CC [3] and Stan [6].

Dataset	DNC (%)	CC (%)	Stan (%)
Cancer	2.156	1.892	3.4
Credit Card	14.531	13.375	14.8
Thyroid	5.891	5.144	–
Soybean	6.029	5.987	9.2
Heart disease	17.425	16.965	18.6

3.4 Discussion

For DNC [2] here we have got better results than Stan [6]. Back propagation uses fixed architecture whereas DNC [2] starts with minimal hidden nodes. In DNC [2] nodes are added one after another as required. If the architecture is sufficient, then node addition is stopped.

In CC [3] 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 [3].

The comparison of errors for the datasets represented in Table 3.5 shows that the best result is generated by CC [3] algorithm for every problem.

Table 3.5 also shows that DNC [2] algorithm also performs better than Stan [6].

4.0 CONCLUSION

We have compared between DNC [2] and CC [3], the most common constructive 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 cancer, credit card, the heart disease, the thyroid and the soybean problem show the generalization ability of NN by using DNC [2] and CC [3] algorithms. For almost every problems CC [3] outperforms over DNC [2].

REFERENCES

- [1] Kwok, T. Y., and Yeung, D. Y., "Objective functions for training new hidden units in constructive neural networks," *IEEE Transactions on Neural Networks*, 8, 1131–1148, (1997b).
- [2] Ash, T. "Dynamic node creation in back propagation networks," *Connection Science*, 1, 365–375, 1989.
- [3] Fahlman, S. E. and Lebiere, C. "The Cascade–Correlation learning architecture," In D. S. Touretzky, *Advances in neural information processing systems 2* (pp. 524–532). San Mateo, CA: Morgan Kaufman, 1990.
- [4] R. Reed, "Pruning algorithms—a survey," *IEEE Transactions on Neural Networks*, vol 4, no. 5, pp 740–747, September 1993.
- [5] Lutz Prechelt, "PROBEN1—a set of benchmarks and benchmarking rules for neural network training algorithms," Technical Report 21/94, Faculty of Informatics, University of Karlsruhe, Germany, (1994).
- [6] David Opitz and Richard Maclin, "Popular Ensemble Methods: An Empirical Study," *Journal of Artificial Intelligence Research* 11 pp. 169–198, 1999.
- [7] Rumelhart, D. E., Hinton, G. E., & Williams, R. J., "Learning internal representations by error propagation," In D. E. Rumelhart and J. L. McClelland (Eds), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. I pp. 318–362, MIT Press, Cambridge, MA, 1986.