Prove of Convergence of Extended Divide & Conquer Networks

Steve G. Romaniuk

Department of Information Systems and Computer Science

National University of Singapore

10 Kent Ridge Crescent

Singapore 0511

e-mail: stever@iscs.nus.sg

Abstract

The task of determining an effective architecture for multi-layer feed forward back-propagation like neural networks can be a time consuming effort. Over the past couple years several algorithms were proposed for dynamically constructing network architectures. Some of these algorithms have been shown to converge for binary data. Unfortunately, the results do not carry over for the non-binary input case. For classificatory problems no guarantees are provided, which would suggest otherwise. In this paper, we present an extension to the basic network growing algorithms, which allows constructing networks in bounded time. The algorithm guarantees convergence for any classificatory domain, as long as no contradictory training examples are present. The extension is described for the Divide & Conquer Networks (DCN) algorithm. The derived mathematical model can be readily incorporated into other network growing approaches to ensure convergence. The model is dependent on the usage of simple threshold cells, which can be applied to a variety of learning rules.

1 Introduction

In recent years network growing learning algorithms have received considerable attention, due to their ability to overcome some of the major shortcomings of more traditional neural network approaches. It is a well known fact, that in order to apply feed forward neural networks to learning requires making several choices, such as determining the number of hidden units, choosing a transfer function and fine tuning learning parameters. Several approaches have been proposed to dynamically create network structures, such as Divide & Conquer Networks [6], Cascade Correlation [3], Upstart Algorithm [4], and Extentron [2]. For several of these algorithms it was observed that after phases of rapid improvement (minimizing overall error) stages of no improvement would be entered, or even performance would decrease. These stages of stagnation would persist even though new hidden units would be introduced. Eventually, improvements would be obtained and solutions found. In some cases this resulted in exceedingly large network structures. For all cases only a limited number of patterns could not be recognized suggesting a general problem with picking at least a single example from the remaining set (not yet correctly recognized by the network) and learning it in conjunction with what is already known. In order to guarantee convergence and avoid unnecessary introduction of new hidden units (features) without improving the networks performance, it was deemed important to extent the DCN algorithm [6]. This extension allows picking at least one additional training pattern apart from those already correctly learned.

2 Divide & Conquer Networks

DCN dynamically creates network architectures based upon the domain examples presented, which allows more effective learning and does not require any preconceived guess on part of the human user. Training is done on the weights of the input links to one cell at any given time. Hence, backpropagation of error through hidden cells is not required. The architecture created may have hidden layers with multiple cells in them, as well as multiple hidden layers. Each cell introduced into the network has connections from all cells in the network that were previously introduced. One output at a time is trained in cell sharing mode, otherwise outputs are trained in parallel. The first cell introduced on a hidden layer is trained on all of the training examples. If it classifies the training set correctly, the cell will serve as an output (Conquer Phase). Succeeding cells on a layer are trained on subsets of the original train set with some (or all) examples correctly recognized omitted from the training set (Divide Phase). DCN attempts to economize on hidden layers to provide for more parallelism. The algorithm can effectively make use of various local learning rules, including quickprop, delta, and perceptron rule. Each cell in the network (other than the original input cells) serves as a form of feature detector. In introducing connections to new cells in the network a form of constructive induction (of features) takes place.

DCN differs from other approaches such as [1] in that it will add cells in multiple hidden layers. However, it can add multiple nodes on each layer unlike Cascade Correlation [3]. Furthermore, DCN does not require the use of any correlation measure. There is no dependence upon a tree structure as in [4]. Divide and Conquer Networks only require training examples that have not already been correctly classified during the divide phase.

3 Derivation of Extension Model

In this section we prove that given a feature F_K , which can correctly classify a subset of the original train set, a new feature F_{K+1} can be constructed that allows the recognition of at least one additional training pattern. Repeated application of this procedure - in case of failure of the incremental learning process - yields guaranteed convergence for any classificatory problem, unless contradictory training patterns are present in the train set. We state the above in the following theorem.

Theorem 1 *Let the current feature cell be* $F_K = O(S_K)$*, where* $S_K = \sum_{i=1}^{K} F_{i-1} * W_{K,i}$ *and the transfer function is the threshold function,*

$$O(a) = \begin{cases} 0, & a \leq 0 \\ 1, & a > 0 \end{cases}$$

Furthermore, let $T(I_l) \in \{0,1\}$ *be the output for training example* I_l*. Let* E *be the set of all training examples and* E_{F_K} *be a true subset of* E*,* $E_{F_K} \subset E$*. Assume,* $\forall I_l \in E_{F_K}\ F_K = O(I_l * W_K) = T(I_l)$*. Here,* W_K *represents the weight vector for feature* F_K*. Then,* $\exists I_s \in E, I_s \ni E_{F_K}\ O(I_s * W_K) \neq T(I_s)$*. Given this condition the goal is to find a feature* F_{K+1} *such that:* $O(I_s * W_{K+1}) = T(I_s) \wedge \forall I_l \in E_{F_K}\ O(I_l * W_{K+1}) = T(I_l)$*.*

Proof Theorem 1 *The problem reduces to finding a weight vector* W_{K+1} *for feature* F_{K+1}*. The first case we investigate is for* $T(I_s) = 0$*. Given the initial constraints we can derive the following four conditions that have to be met in order to satisfy the earlier stated goal.*

$$\begin{array}{ll} O(I_l * W_K) = T(I_l), & \forall I_l \in E_{F_K}\ (a) \\ O(I_s * W_K) = 1, & I_s \ni E_{F_K}\ (b) \\ O(I_l^{'} * W_{K+1}) = T(I_l), & \forall I_l \in E_{F_K}\ (c) \\ O(I_s^{'} * W_{K+1}) = 0, & I_s \ni E_{F_K}\ (d) \end{array} \tag{1}$$

We define $I_l^{'} = (I_l, F_K)$ *and* $I_s^{'} = (I_s, F_K)$ *as the composite of the original vectors* I_l *and* I_s *augmented by feature* F_K*. From Eq. (2d) it follows,*

$$O(I_s^{'} * W_{K+1}) = O(I_s * W_K^{'} + F_K * W_{K+1,K+1}) = 0 \tag{2}$$

From the above condition we can derive the next set of conditions:

$$I_s * W_K^{'} + F_K * W_{K+1,K+1} \leq 0 \Rightarrow F_K * W_{K+1,K+1} \leq -I_s * W_K^{'} \tag{3}$$

Since $F_K = O(I_s * W_K) = 1$ *(Eq. (1b)) we conclude from Eq. (3),*

$$W_{K+1,K+1} \leq -I_s * W_K^{'} \tag{4}$$

Using Eq. (2a) and (2c) we state,

$$O(I_l * W_K) \ = O(I_l^{'} * W_{K+1}) = O(I_l * W_K^{'} + F_K * W_{K+1,K+1}) \tag{5}$$

We consider two separate cases:

1. Case $T(I_l) = 1$*:*

$$O(I_l * W_K^{'} + F_K * W_{K+1,K+1}) = 1 \Rightarrow I_l * W_K^{'} + F_K * W_{K+1,K+1} > 0 \tag{6}$$

2. Case $T(I_l) = 0$*:*

$$\begin{array}{l} O(I_l * W_K) = 0 \Rightarrow I_l * W_K \leq 0 \wedge \\ O(I_l^{'} * W_{K+1}) = O(I_l * W_K^{'} + F_K * W_{K+1,K+1}) = 0 \Rightarrow \\ I_l * W_K^{'} + F_K * W_{K+1,K+1} \leq 0 \end{array} \tag{7}$$

The above result leads to the following two conditions which need to be met for the case $T(I_s) = 0$.

$$\begin{array}{ll} W_{K+1,K+1} \leq -I_s * W_K^{'} & (a) \\ F_K * W_{K+1,K+1} > -I_l * W_K^{'} \vee F_K * W_{K+1,K+1} \leq -I_l * W_K^{'} & (b) \end{array} \tag{8}$$

Since F_K *is either,*

$$F_K = O(I_l * W_K) = 1 \vee F_K = O(I_l * W_K) = 0 \tag{9}$$

we obtain the following set of conditions,

$$\begin{array}{ll} W_{K+1,K+1} \leq -I_s * W_K^{'} & (a) \\ W_{K+1,K+1} > -I_l * W_K^{'} \vee 0 \leq -I_l * W_K^{'} & (b) \end{array} \tag{10}$$

From Eq. (11a) and (11b) we derive:

$$- I_l * W_K^{'} < W_{K+1,K+1} \leq -I_s * W_K^{'} \Rightarrow I_l * W_K^{'} > I_s * W_K^{'} \tag{11}$$

Finally, we can state for $T(I_s) = 0$,

$$\begin{array}{l} (I_l - I_s) * W_K^{'} > 0 \wedge W_{K+1,K+1} \leq -I_s * W_K^{'},\ T(I_l) = 1 \\ W_{K+1,K+1} \leq -I_s * W_K^{'} \wedge I_l * W_K^{'} \leq 0,\ T(I_l) = 0 \end{array} \tag{12}$$

For the second case we consider $T(I_s) = 1$. From Eq. (1) and (2) we obtain,

$$\begin{array}{l} O(I_s^{'} * W_{K+1}) = O(I_s * W_{K+1}^{'} + F_K * W_{K+1}K + 1) = 1 \Rightarrow \\ I_s * W_K^{'} + F_K * W_{K+1,K+1} > 0 \end{array} \tag{13}$$

Note, that the outputs of Eq. (1a) and (1c) have been reversed. Now, since $F_K = O(I_l * W_K) = 0$, it follows, $I_s * WK \leq 0$. By setting Eq. (1a) and (1c) equal we get,

$$O(I_l * W_K) = O(I_l^{'} * W_{K+1}) = T(I_l) \tag{14}$$

We again investigate two cases separately.

1. Case $T(I_l) = 0$:

$$O(I_l^{'} * W_{K+1}) = 0 \ \Rightarrow I_l^{'} * W_{K+1} \leq 0 \Rightarrow I_l * W_K{}^{'} + F_K * W_{K+1,K+1} \leq 0 \tag{15}$$

2. Case $T(I_l) = 1$:

$$\begin{array}{ll} O(I_l^{'} * W_{K+1}) = 1 & \Rightarrow \\ I_l^{'} * W_{K+1} > 0 & \Rightarrow \\ I_l * W_K{}^{'} + F_K * W_{K+1,K+1} > 0 & \Rightarrow \\ I_l * W_K^{'} > 0 & \end{array} \tag{16}$$

This leads to the final set of conditions for $T(I_s) = 1$:

$$\begin{array}{l} I_s * W_K^{'} > 0 \wedge I_l * W_K^{'} \leq 0,\ T(I_l) = 0 \\ I_s * W_K^{'} > 0 \wedge I_l * W_K^{'} > 0,\ T(I_l) = 1 \end{array} \tag{17}$$

From Eq. (17) it follows that weight $W_{K+1,K+1}$ is not required. This of course further implies that the original feature F_K is also not required.

Q.E.D.

4 Application of Extension Model to DCN

A summary of the necessary conditions for implementing the decision strategy of the extension algorithm is displayed in Table 1. Solutions to this problem may be obtained by using results from operations research. The problem requires determining the vector $W_{K+1}^{'}$ with respect to the conditions imposed on the product between said vector and training patterns $I_l \in E_{F_K}$ and $I_s \ni E_{F_K}$. This can be accomplished using algorithms for solving linear programming problems [5]. Note, that the conditions for $T(I_s) = 1$ do not require the feature F_K, which implies the cell that implements this feature is redundant and can therefore be pruned from the network.

Table 1. Decision Strategy for Extension Model

$T(I_l)/T(I_s)$	0	1
0	$W_{K+1,K+1} \leq -I_s * W_K^{'} \wedge I_l * W_K^{'} \leq 0$	$I_s * W_K^{'} > 0 \wedge I_l * W_K^{'} \leq 0$
1	$W_{K+1,K+1} \leq -I_s * W_K^{'} \wedge (I_l - I_s) * W_K^{'} > 0$	$I_s * {W_K}^{'} > 0 \wedge I_l * W_K^{'} > 0$

We state the Extended Divide & Conquer algorithm in Figure 1. Note, that by the very nature of the algorithm convergence for classificatory domains (continuous inputs and binary outputs) is guaranteed. If the DCN algorithm can not improve on the ErrorBits parameter (Number

(1) Let $ErrorBits = card(E_{F_K}) \wedge K = 0$
(2) While $ErrorBits > 0$ do
  (2.1) Perform DCN
  (2.2) Determine $NewErrorBits$
  (2.3) If $NewErrorBits \geq ErrorBits$ then
    (2.3.1) Select $I_s \in E \wedge I_s \ni E_{F_K}$
      (2.3.1.1)For all $I_l \in E_{F_K}$ do
        (2.3.1.1.1) Given $T(I_s) \wedge T(I_l)$ satisfy conditions in Table 1
  (2.4) $ErrorBits = NewErrorBits$
  (2.5) Create New Feature F_K
(3) Halt

Figure 1: High Level Description of Extended DCN Algorithm

of Training Patterns incorrectly classified by current feature F_K), then the extension part of the algorithm is executed. It guarantees that any training patterns $I_s \in E \wedge I_s \ni E_{F_K}$ will be recognized by the new feature F_{K+1}. Under worst case conditions K such features are generated and the condition in step (2.3) is true.

5 Summary

In this paper an extension for dynamic network growing algorithms is presented, which not only proves convergence of algorithms utilizing this method (for all classificatory domains), but also supports deriving simpler network architectures in cases, were no improvement in reduction of number of incorrectly classified training patterns is possible during installation of a new hidden unit. Derivation of the extension model is motivated by observing that during several experiments using Divide & Conquer Networks (as well as other methods), extended stages of no improvement (no reduction of error bits) were encountered. Extension of other network growing approaches is paramount, due to a similar inability to appropriately cope with this problem. Picking up at least a single (not yet correctly recognized) example pattern during training by a new feature cell is not guaranteed, - at least within some pre-determined number of epochs. Applying the extension algorithm in such a case, can guarantee improvement by at least one additional training pattern and eventual convergence by the network growing algorithm.

References

[1] Ash, T. (1989). *Dynamic Node Creation in Backpropagation Networks.* (Tech. Report ICS Report 8901), Inst. for Cognitive Science, University of California, San Diego, La Jolla, Ca.

[2] Baffes, P.T. and Zelle, J.M (1992). Growing Layers of Perceptrons: Introducing the Extentron Algorithm, *Proceedings of the 1992 International Joint Conference on Neural Networks* (pp. II-392- II-397), Baltimore, MD., June.

[3] Fahlman, S.E. and Lebiere, C. (1990). The Cascade-Correlation Learning Architecture, In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2* (pp. 524-532). San Mateo, CA.: Morgan Kaufmann.

[4] Frean, M. (1991). The Upstart Algorithm: A Method for Constructing and Training FeedForward Neural Networks, *Neural Computation*, 2, 198-209.

[5] Phillips, D.T., Ravindran, A., Solberg, J. (1976) Operations Research: Principles and Practice, John Wiley & Sons, Inc.

[6] Romaniuk, S.G., Hall, L.O. (1993) Divide and Conquer Networks. To appear Neural Networks.