Pruning Strategies for the MTiling Constructive Learning Algorithm

Rajesh Parekh, Jihoon Yang, and Vasant Honavar *
Artificial Intelligence Research Group
Department of Computer Science
Iowa State University
Ames IA 50011. U.S.A.
{parekh|yang|honavar}@cs.iastate.edu

Abstract

We present a framework for incorporating pruning *strategies in the* MTiling *constructive neural network learning algorithm. Pruning involves elimination of redundant elements (connection weights or neurons) from a network and is of considerable practical interest. We describe three elementary sensitivity based strategies for pruning neurons. Experimental results demonstrate a moderate to significant reduction in the network size without compromising the network's generalization performance.*

1. Introduction

Constructive neural network learning algorithms offer an interesting paradigm for incremental construction of near minimal architectures for pattern classification problems [4, 5, 6, 7]. Traditional algorithms for multi-layer feed forward neural network training (such as the *backpropagation* [14]) search for an appropriate weight setting using a gradient descent based error minimization in an a-priori fixed network architecture. Constructive algorithms grow the network incrementally by training one or more *threshold logic units* (TLUs or *neurons*) and integrating these trained neurons into the existing network architecture. These algorithms offer several benefits over the traditional gradient descent based training mechanisms: They obviate the need for an a-priori (often ad-hoc) choice of network topology; do not require the expensive error backpropagation; and are guaranteed to converge to zero classification errors (under certain assumptions). Several constructive learning algorithms have been proposed in the literature — *Tower*, *Pyramid* [4], *Tiling* [9], *Upstart* [2], *Perceptron Cascade* [1], and *Sequential Learning* [8]. These algorithms differ from each other in the design choices viz. representation of input patterns (binary/bipolar valued or real-valued); when and where to add a new TLU (or a group of TLUs); connectivity of the newly added neuron(s); training the TLUs; and training the sub-network affected by the modification of the network topology. These differences in design choices result in constructive learning algorithms with different representational and inductive biases. Provably correct and practical extensions of these algorithms to handle real-valued pattern attributes and multiple output categories are studied in [10, 11, 15].

The success of the constructive learning algorithms depends partly on the algorithm used to train the individual TLUs because the convergence to zero classification errors is based on the fact that the TLU training algorithm can find a suitable weight setting such that the total number of misclassifications is reduced by at least one each time a new neuron (or a group of neurons) is added and trained and the network outputs are recomputed. Algorithms such as the *Pocket algorithm* with *ratchet modification* [4], the *Thermal perceptron algorithm* [3], and the *Barycentric correction procedure* [12] are commonly used for training individual TLUs (or groups of TLUs) in constructive algorithms. We denote such a suitable TLU training algorithm by $\mathcal{A}$.

An exhaustive search through the space of neural network architectures is computationally infeasible. Constructive learning algorithms adopt a greedy strategy by incrementally adding TLUs during the network training phase. The training of individual TLUs is based on local information in the sense that during training the weights of the remainder of the network are frozen and the training set for these neurons is constructed with the objective of reducing the residual classification error. Owing to the *design biases* and the *locality of training* it is possible that the incrementally grown networks are larger than necessary for the given classification task. Other things being equal, smaller (more compact) networks are desirable because of lower classification cost; potentially superior generalization per-

*This research was partially supported by the National Science Foundation grants IRI-9409580 and IRI-9643299 to Vasant Honavar.

formance; and transparency of the acquired knowledge in applications which involve extraction of rules from trained networks. These reasons motivate the study of pruning techniques in constructive learning algorithms.

Network pruning involves elimination of connection elements (i.e., weights or neurons) that are deemed unnecessary in that their elimination does not degrade the network's performance. Pruning can be performed either after the entire network is trained or can be integrated into the training process itself. In this paper we study the application of pruning techniques to MTiling , an extension of the *Tiling* algorithm to handle real-valued pattern attributes and multiple output classes [10, 11, 15]. The remainder of this paper is organized as follows: Section 2 outlines the MTiling constructive neural network learning algorithm. Section 3 describes pruning strategies for eliminating unwanted neurons from a MTiling network. Section 4 presents the results of experiments. Finally, section 5 concludes with an analysis of the experiments with pruning and suggestions for future research.

2. The MTiling Algorithm

The 2-category *Tiling* algorithm [9] constructs a strictly layered network of TLUs. The bottom-most layer receives inputs from each of the N input neurons. The neurons in each subsequent layer receive inputs from the neurons in the layer immediately below itself. All TLUs within the network are trained using the chosen TLU training algorithm ($\mathcal{A}$). Each layer maintains a *master neuron* that acts as the output neuron for that layer. If the master neuron cannot correctly classify all training patterns then ancillary neurons may be added to the layer and trained to ensure a *faithful representation* of the training patterns. The *faithfulness* criterion states that no two training examples belonging to different target classes should produce identical outputs at any given layer. Faithfulness is a necessary condition for convergence in strictly layered networks [9]. The *Tiling* algorithm is guaranteed to converge to zero classification errors (under certain assumptions) for finite, non-contradictory datasets where the pattern attributes are bipolar valued (i.e., 1 or -1 only).

The proposed extension to multiple output classes involves constructing layers with M master neurons (one for each of the output classes). Groups of one or more ancillary neurons are trained at a time in an attempt to make the current layer faithful. Fig. 1 shows the construction of a MTiling network.

2.1. Algorithm

1. Train a layer of M master neurons each of which is connected to the N inputs.

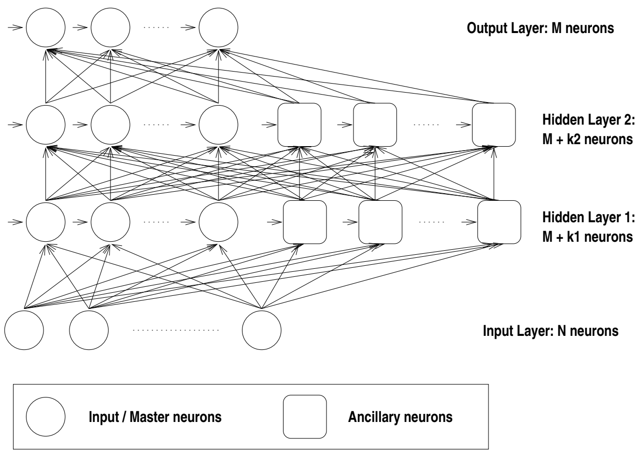

Figure 1. Tiling Network

2. If the master neurons can achieve the desired classification accuracy then stop.

3. Otherwise, if the current layer is not faithful, add ancillary neurons to the current layer to make it faithful as follows, else go to step 4.

   (a) Among all the unfaithful outputs at the current layer, identify the one that the largest number of input patterns map to. (Note that each output is a vector of $M + k^{'}$ elements where $k^{'}$ is the total number of ancillary neurons added to the layer so far).

   (b) The set of patterns that generate the output identified in step 3(a) will form the training set for ancillary neurons.

   (c) Add a group of k ($1 \leq k \leq M$) ancillary neurons where k is the number of target classes represented in the set of patterns identified in step 3(b) and train them.

   (d) Repeat these last three steps (of adding and training ancillary neurons) till the representation of the patterns in the current layer is faithful.

4. Train a new layer of M master neurons that are connected to each neuron in the previous layer and go to step 2.

2.2. Convergence Proof

The convergence of the MTiling algorithm to zero classification errors is proved in two parts. First, it is proved that a faithful representation can be attained for any finite, non-contradictory dataset (possibly having real-valued attributes) using only a finite number of ancillary neurons. It is then demonstrated that with each additional layer the number of classification errors is reduced by at least one. Since the number of training patterns is finite, the total number of classification errors made by the MTiling network will eventually become zero. The reader is referred to [11] for the detailed convergence proof.

3. Pruning Strategies

An excellent survey of neural network pruning strategies appears in [13]. Reed outlines two types of pruning techniques — *sensitivity calculations* and *penalty terms* for feed-forward neural networks trained using the backpropagation algorithm. The former investigates the sensitivity of the error function to the removal of a network element. Elements with the least sensitivity are pruned. The second group of techniques involves incorporating a penalty term in the error function which causes the irrelevant connection weights to be driven toward zero during the minimization of the error function. The group of constructive algorithms mentioned in section 1 does not explicitly define a cost (error) function for minimization during training. Thus, it is not clear whether penalty term based pruning techniques can be directly applied to these constructive learning algorithms.

The sensitivity of a neuron is defined as the *error* introduced in the network upon the removal of the neuron. The error can be defined in a manner that is most suited to the context. Thus, for a neuron i in layer L (that has been made faithful by addition of ancillary neurons), the sensitivity $S(i)$ is defined as:

$$\begin{aligned} S(i) &= 1 \text{ if eliminating } i \text{ renders } L \text{ unfaithful} \\ S(i) &= 0 \text{ otherwise} \end{aligned}$$

We assign a sensitivity of 1 to all master neurons since the master neurons cannot be pruned. In the case of the MTiling network, pruning is combined with the training process and is invoked after each layer is trained and made faithful. Three simple techniques for identifying neurons with sensitivity $S(i) = 0$ are described below. These neurons are pruned and training is continued with the addition of a new layer of M master neurons.

Dead Neurons:
Ancillary neurons with exactly the same output (i.e., 1 or -1) for all patterns in the training set are called *dead neurons*. As seen in Fig. 2, pruning dead neurons does not affect the faithfulness of the current layer. Thus, dead neurons are assigned a sensitivity of 0.

Correlated Neurons:
Pairs of ancillary neurons that give either exactly the same or exactly opposite output for each pattern in the training set (i.e., the product of the outputs of the two units is either 1 for all patterns or -1 for all patterns) are said to be correlated. As seen in Fig. 3, one of these neurons can be safely pruned without affecting the faithfulness of the current layer. Ancillary neurons are taken two at a time and their outputs (for each pattern) are compared to determine if the neurons are correlated. One neuron from the correlated pair is dropped as soon as it is identified and is not considered any further in the search for correlated neuron pairs.

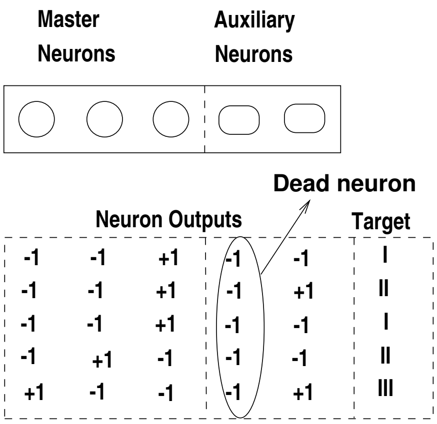

Figure 2. Dead Neurons

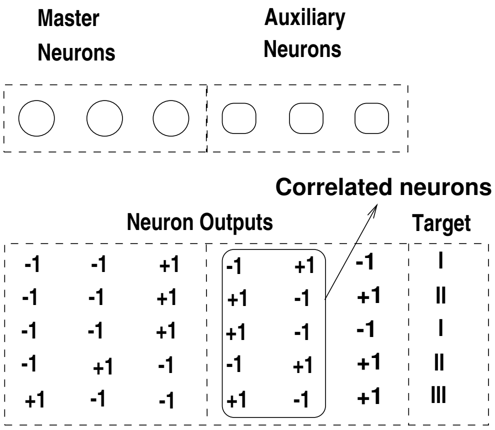

Figure 3. Correlated Neurons

Redundant Neurons:
Determination of redundant neurons involves dropping ancillary neurons one at a time and comparing the remaining outputs for faithfulness. If the outputs are not faithful, the dropped neuron is restored. Otherwise the dropped neuron is redundant and is assigned a sensitivity $S(i) = 0$ (see Fig. 4). The redundant neuron is immediately pruned and the search for redundant neurons is continued starting with the first ancillary neuron. This identification and pruning of redundant neurons is continued until no further redun-

dant neurons can be identified. In the worst case this search for redundant neurons makes $\mathcal{O}(k^2)$ calls to the routine that checks for faithfulness of the current layer (assuming that the current layer has k ancillary neurons to start with).

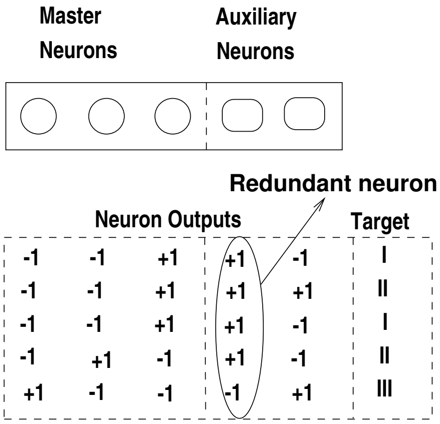

Figure 4. Redundant Neurons

The search for neurons with $S(i) = 0$ incurs an additional cost. Identification of the neurons that are eventually pruned involves a comparison of the current layer's outputs for each pattern. It should be noted that these outputs are computed while checking for faithfulness during training and hence are readily available to the pruning step. Thus, the only cost incurred by the pruning step is that of determining the neurons with $S(i) = 0$.

4. Experimental Results

We have conducted several experiments with pruning using a variety of *artificial* and *real-world* datasets. Table 1 summarizes the properties of the datasets where **Tr**, **Te**, **N**, **M**, and **A** denote the size of the training set, the size of the test set, the number of inputs, the number of output neurons, and the attribute type (*r* - real and *i* - integer), respectively. The 3 concentric circles dataset **3c** considers points belonging to three concentric circles centered at the origin and having radii 2, 4, and 6 respectively. Points were generated at random and were assigned to one of three output classes depending on the distance ($d(x, O)$) of the point (x) from the origin (O): $\forall x,\ 0 \leq d(x, O) \leq 2 \Longrightarrow x \in$ class 1 ; $2 < d(x, O) \leq 4 \Longrightarrow x \in$ class 2 ; and $4 < d(x, O) \leq 6 \Longrightarrow x \in$ class 3 . The other datasets viz. 2 spirals (**2sp**), BUPA liver disorder (**liver**), image segmentation (**seg**), wisconsin diagnostic breast cancer (**wdbc**), and wine recognition (**wine**) are available at the *UCI Machine Learning Repository* [1].

10 runs of the MTiling algorithm were conducted for each dataset with and without pruning. Individual TLUs were trained using the *Thermal perceptron algorithm* for 500 epochs with random initial weights in the range $[-1..1]$. The *winner-take-all* (WTA) strategy was used to compute the outputs for datasets involving more than two pattern classes. For runs with network pruning the number of neurons pruned by each of the three pruning strategies, the total time for pruning (in seconds), the network size (number of hidden and output units), the total training time (in seconds), and the generalization performance (classification accuracy on the test set) over the 10 runs were recorded. For runs without network pruning the network size, the training time, and the generalization performance over the 10 runs were recorded. A few runs for the **wdbc** and the **wine** failed to converge to zero classification errors even after the addition of 100 ancillary neurons. These runs were considered as failures and not included in results reported below.

Dataset	**Tr**	**Te**	**N**	**M**	**A**
3c	900	900	2	3	r
2sp	194	–	2	1	r
liver	234	117	34	1	i,r
seg	210	2100	19	7	i,r
wdbc	380	189	30	1	r
wine	120	58	13	3	i,r

Table 1. Datasets

Figure 5 compares the average network size of the MTiling algorithm with and without pruning. This demonstrates a modest to significant reduction in the size of the network with pruning. The generalization performance of the network is not affected by pruning as can be seen from Figure 6.

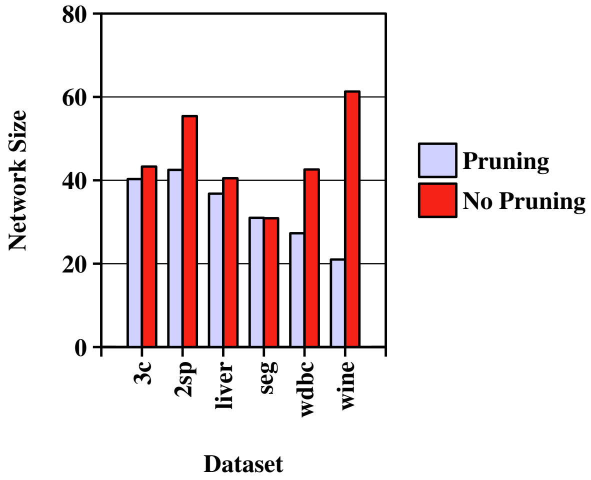

Figure 5. Comparing the Network Size with and without Pruning

Table 2 reports the average number of neurons pruned by

[1] [http://www.ics.uci.edu/AI/ML/MLDBRepository.html]

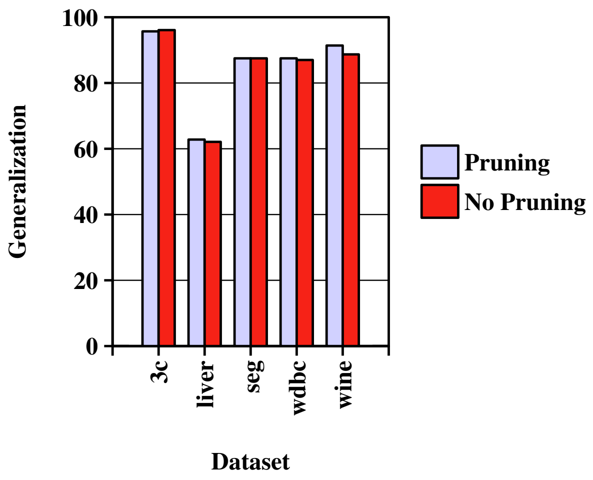

Figure 6. Comparing the Generalization with and without Pruning

each of the three pruning strategies (**D** – dead neurons, **C** – correlated neurons, and **R** – redundant neurons), the average time for pruning (**PTime**) in seconds and the average total training time (**Time**) in seconds for the MTiling algorithm. A majority of the neurons pruned belong to the class of redundant neurons. Further, the total time spent in searching for neurons with $S(i) = 0$ is a small fraction of the total training time in each case.

Dataset	D	C	R	PTime	Time
3c	0.1	0.0	5.2	133.8	856.3
2sp	2.2	0.6	16.0	13.32	134.8
liver	0.0	0.1	5.5	7.22	156.87
seg	0.6	0.0	2.3	4.01	175.17
wdbc	5.9	0.0	7.4	23.21	27.25
wine	15.4	4.8	14.8	8.45	142.31

Table 2. Summary of simulation results

We repeated the above experiments using the *Barycentric correction procedure* instead of the *Thermal perceptron algorithm* for training the individual TLUs. The results for **wdbc** and **wine** were significantly different from those reported above (see Table 3). We see that training using the *Barycentric correction procedure* resulted in MTiling networks with practically no redundancy as opposed to the case of the *Thermal perceptron algorithm* where a significant reduction in network size was attained as a result of pruning. Table 3 describes the total number of neurons pruned by the three pruning strategies combined (**TP**), the network size (**Size**), the time for pruning (**PTime**), the total training time (**Time**), and the generalization accuracy (**%**) for the two datasets. The results are averaged over 10 runs of the MTiling algorithm using the *Barycentric correction procedure* .

Note that the results of pruning on the other datasets were comparable to those obtained when the *Thermal perceptron algorithm* was used for training TLUs in the MTiling network.

Dataset	TP	Size	PTime	Time	%
wdbc	0.7	20.6	9.4	234.6	89.4
wine	0.0	7.9	0.1	30.3	94.0

Table 3. MTiling **networks trained using the** *Barycentric correction procedure*

5. Discussion

Owing to the inherent *biases* and the *locality of training*, network growing algorithms do not necessarily construct a minimal network to learn a given pattern classification task. Network pruning can be employed to eliminate unnecessary network elements. We have described three simple methods for pruning neurons in the MTiling algorithm. The experimental results demonstrate a moderate to significant reduction in the network size. Smaller networks simplify the task of knowledge extraction from neural networks since the extracted rules are easier to interpret if the networks are smaller. It must be noted that these improvements come at the additional cost of identifying the neurons that can be pruned. Our experiments demonstrate that this additional time spent in pruning is a small fraction of the total training time of the MTiling network. This approach presents a natural tradeoff between training time and network size.

The generalization performance of the networks remained nearly the same with or without pruning. This might be attributed to the fact that the pruning methods we studied simply eliminate the redundancy in the network. Other pruning strategies might however significantly affect the network's generalization performance.

The redundancy introduced in a MTiling network while learning to classify a particular dataset might actually depend on the choice of the TLU training algorithm (as seen in the results described in section 4). It is not clear whether there exists a single TLU training algorithm which when used for training TLUs in the MTiling network results in the construction of superior networks (in terms of network size and generalization ability) on all datasets.

Sensitivity based pruning of individual weights requires computing the network error after removing each weight independently. This is computationally infeasible even for moderately large networks. A strategy that computes or approximates the sensitivity of the weight during the training itself merits investigation. The characteristics of the MTiling algorithm can be used to identify *dominant* connection weights as follows. Since the MTiling network is strictly layered and uses bipolar TLUs (with outputs either 1 or -1) in the hidden and output layers, the inputs for the TLUs in all the layers starting at the second hidden layer are guaranteed to be bipolar valued (i.e., the inputs values can only be either 1 or -1). Consider a TLU with the weight vector $\mathbf{W} = \{W_0, W_1, \ldots, W_n\}$. If this TLUs inputs are guaranteed to be bipolar valued, we say that the connection weight W_j is the dominating connection weight if $\mid W_j \mid > \sum_{i=0, i \neq j}^{n} \mid W_i \mid$. Note that if a TLU has a dominant connection weight then the output of the TLU is determined solely by the input to this dominant connection and does not depend on the inputs to the other connections. Thus, in the case of MTiling networks, if any TLU in the second hidden layer (or above) contains a dominant connection weight, then all connection weights except the dominant one for this TLU can be pruned.

We are currently exploring additional, possibly more informed pruning strategies for the MTiling algorithm and studying appropriate pruning strategies for other constructive neural network learning algorithms.

References

[1] N. Burgess. A constructive algorithm that converges for real-valued input patterns. *International Journal of Neural Systems*, 5(1):59–66, 1994.

[2] M. Frean. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209, 1990.

[3] M. Frean. A thermal perceptron learning rule. *Neural Computation*, 4:946–957, 1992.

[4] S. Gallant. Perceptron based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, June 1990.

[5] S. Gallant. *Neural Network Learning and Expert Systems*. MIT Press, Cambridge, MA, 1993.

[6] V. Honavar. *Generative Learning Structures and Processes for Generalized Connectionist Networks*. PhD thesis, University of Wisconsin, Madison, 1990.

[7] V. Honavar and L. Uhr. Generative learning structures and processes for connectionist networks. *Information Sciences*, 70:75–108, 1993.

[8] M. Marchand, M. Golea, and P. Rujan. A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11(6):487–492, 1990.

[9] M. Mézard and J. Nadal. Learning feed-forward networks: The tiling algorithm. *J. Phys. A: Math. Gen.*, 22:2191–2203, 1989.

[10] R. G. Parekh, J. Yang, and V. G. Honavar. Constructive neural network learning algorithms for multi-category classification. Technical Report ISU-CS-TR95-15a, Department of Computer Science, Iowa State University, 1995.

[11] R. G. Parekh, J. Yang, and V. G. Honavar. Constructive neural network learning algorithms for multi-category real-valued pattern classification. Technical Report ISU-CS-TR97-06, Department of Computer Science, Iowa State University, 1997.

[12] H. Poulard. Barycentric correction procedure: A fast method of learning threshold units. In *Proceedings of WCNN'95, July 17-21, Washington D.C.*, volume 1, pages 710–713, 1995.

[13] R. Reed. Pruning algorithms — a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations into the Microstructure of Cognition*, volume 1 (Foundations). MIT Press, Cambridge, Massachusetts, 1986.

[15] J. Yang, R. G. Parekh, and V. G. Honavar. Mtiling - a constructive neural network learning algorithm for multi-category pattern classification. In *Proceedings of the World Congress on Neural Networks'96*, pages 182–187, San Diego, 1996.