Iterative pruning in second-order recurrent neural networks

Giovanna Castellano, Anna Maria Fanelli, Marcello Pelillo

Dipartimento di Informatica, Universita degli Studi di Bari
Via Orabona 4, I-70126 Bari, Italy

Abstract. An iterative pruning method for second-order recurrent neural networks is presented. Each step consists in eliminating a unit and adjusting the remaining weights so that the network performance does not worsen over the training set. The pruning process involves solving a linear system of equations in the least-squares sense. The algorithm also provides a criterion for choosing the units to be removed, which works well in practice. Initial experimental results demonstrate the effectiveness of the proposed approach over high-order architectures.

1. Introduction

Recurrent neural networks are particularly attractive because of their potential in dealing with such problems as pattern completion and temporal sequence processing [1], [2].

As for feed-forward networks, the choice of the proper number of hidden units is still an open question that usually turns out to be a trade-off between generalization and learning abilities [2]. Various methods have been proposed for feed-forward topologies [3], which solve this problem by pruning excessive weights and/or units in a larger than necessary network after or during training. By contrast, such methods for recurrent neural networks are lacking. To our knowledge, the only attempt in this direction was made by Giles and Omlin [4], who proposed a simple heuristic for pruning trained recurrent neural networks.

In this paper we propose a formal method of pruning second-order recurrent neural networks, which is a generalization of an algorithm initially developed for feed-forward architectures [5] and successively extended to first-order recurrent neural networks [6]. The method is based on the idea of removing hidden units and adjusting the remaining weights in such a way that the overall input-output network's behavior is kept approximately unchanged over the entire training set. The pruning process is formulated in terms of a linear system, that is solved in the least-squares sense by means of a very efficient preconditioned conjugate gradient procedure.

Initial experimental results over a simple problem of grammatical inference demonstrate the effectiveness of the pruning algorithm, in terms of both speed and generalization capability of the reduced networks.

2. The network

Let the second-order network be represented by a directed graph $\mathcal{N} = (\mathcal{V}, \mathcal{E}, w)$. $\mathcal{V} = \{0,...,n\}$ is the set of units, which is divided into a subset $\mathcal{V}_I$ of input units and a subset $\mathcal{V}_R$ of recurrent units, that includes the set $\mathcal{V}_H$ of hidden units and the set $\mathcal{V}_O$ of output units. $\mathcal{E} \subseteq \mathcal{V}_R \times \mathcal{V}_R \times \mathcal{V}_I$ is the set of connections. Each connection $(i, j, k) \in \mathcal{E}$ is associated with a weight $w_{ijk} \in R$. For each unit $i \in \mathcal{V}_R$, let us define its "projective" field $P_i = \{j \in \mathcal{V}_R \mid (i, j, k) \in \mathcal{E}; \forall k \in \mathcal{V}_I\}$ and its "receptive" field $R_i = \{j \in \mathcal{V}_R \mid (j, i; k) \in \mathcal{E}; \forall k \in \mathcal{V}_I\}$.

At each time step $t = 0, 1, \ldots$ the output of unit $i \in \mathcal{V}_R$ is $x_i^{t+1} = f_i(u_i^t)$ where f_i is a nonlinear differentiable activation function (here assumed to be the logistic function for all units), and

$$u_i^t = \sum_{k \in \mathcal{V}_I} \sum_{j \in R_i} w_{ijk} x_j^t I_k^t \qquad (1)$$

is the net input. I_k^t denotes the output of unit $k \in \mathcal{V}_I$ at time t. At each time step t, the network is presented a single symbol $\sigma_p(t)$ of an input pattern p, which is mapped onto the input set $\mathcal{V}_I$.

Unary input mapping is usually adopted [8], that associates one input neuron to each distinct symbol. This means that when symbol $\sigma(t)$ is presented, only the correspondent input neuron is on. The net input becomes:

$$u_i^t = \sum_{j \in R_i} w_{ij\sigma}(t) x_j^t \qquad (2)$$

From (1) and (2) it is clear that with this mapping, each input neuron $k \in \mathcal{V}_I$ acts as a selector of the subset of connections $\mathcal{E}_k = \{(i, j, k) \in \mathcal{E}\}$. For each input pattern, the response of the network is given by the output values of neurons in $\mathcal{V}_O$ when the last symbol is met.

3. The pruning algorithm

Suppose that unit $h \in \mathcal{V}_H$ has been identified to be removed. Our approach to network pruning involves eliminating all its incoming and outgoing connections, and then adjusting the weights incoming into h's projective field in such a way that the net input of every unit $i \in P_h$ remains approximately unchanged. This amounts to requiring that the following relation holds:

$$\sum_{k \in \mathcal{V}_I} \sum_{j \in R_i} w_{ijk} x_j^t I_k^t = \sum_{k \in \mathcal{V}_I} \sum_{j \in R_i - \{h\}} \left(w_{ijk} + \delta_{ijk} \right) x_j^t I_k^t \qquad (3)$$

for each unit $i \in P_h - \{h\}$, for each training pattern p and for each symbol $\sigma_p(t)$. The δ_{ijk}'s arefactors to be computed in order to adjust the weights w_{ijk}'s. Simple algebraic manipulations yield:

$$\sum_{k \in \mathcal{V}_I} \sum_{j \in R_i - \{h\}} \delta_{ijk} x_j^t I_k^t = \sum_{k \in \mathcal{V}_I} w_{hjk} x_h^t I_k^t \qquad (4)$$

that, according to the unary input encoding becomes:

$$\sum_{j \in R_i - \{h\}} \delta_{ij\sigma(t)} x_j^t = w_{hj\sigma(t)} x_h^t \qquad (5)$$

Note that in first order architectures, the symbols of a pattern are given in one shot to the network, thus condition (3) is applied after the network has reached a stable state. In second-order recurrent networks, the same condition must be applied after the presentation of each symbol in a pattern; this is done in order to keep trace of the subset of connection weights involved in computing the output at each time step.

System (5), which can be conveniently represented as $A\bar{y} = \bar{b}$, is then solved in the least-square sense by means of a very efficient preconditioned conjugate-gradient method proposed by Bjiörck and Elfving [7]. It begins with an initial solution $\bar{y}_0$ and iteratively produces a sequence of points $\{\bar{y}_k\}$ so as to decrease the residuals $\rho_k = \| A\bar{y}_k - \bar{b}_k \|$. This naturally suggests a criterion for choosing the units to be removed: pick the unit for which the initial residual ρ_0 is minimum. Since the initial point $\bar{y}_0$ is usually chosen to be the null vector, this amounts to selecting the unit for which the norm of $\bar{b}$ is minimum. To prevent the algorithm producing "useless" units, the selected unit $h \in \mathcal{V}_H$ should satisfy the following conditions: $P_i - \{h\} \neq \emptyset$; for each $i \in R_h$ and $R_i - \{h\} \neq \emptyset$ for each $i \in P_h$. Summarizing, the proposed algorithm is:

1. s := 0
2. repeat
   (a) identify the "minimum-norm" unit $h \in \mathcal{V}_H$ in network $\mathcal{N}^{(s)} = (\mathcal{V}^{(s)}, \mathcal{E}^{(s)}, w^{(s)})$,
   (b) compute δ_{ijk} by solving system (5) in the least-squares sense
   (c) construct the new network $\mathcal{N}^{(s+1)} = (\mathcal{V}^{(s+1)}, \mathcal{E}^{(s+1)}, w^{(s+1)})$ as follows:

$$\begin{aligned} V_H^{(s+1)} &:= V_H^{(s)} - \{h\} \\ \mathcal{E}^{(s+1)} &:= \mathcal{E}^{(s)} - \left(\{h\} \times P_h^{(s)} \cup R_h^{(s)} \times \{h\} \right) \\ w_{ijk}^{(s+1)} &:= w_{ijk}^{(s)} + \delta_{ijk} \quad \text{for } i \in P_h - \{h\} \end{aligned} \qquad (6)$$

3. s := s+1

until a stopping condition is satisfied.

Note that the stopping condition has been left purposely undefined in the algorithm. In fact the proposed pruning procedure can be stopped with different criteria. If the application at hand requires keeping the original network behavior over the training data, one can stop the pruning algorithm by evaluating the performance of the pruned network over the training set. As well, if a good generalization ability is requested to the reduced network, a stopping condition that takes into account the performance over the test set can be used, regardless of the behavior over the training data.

4. Experimental results

To assess the performance of the pruning algorithm on second-order network architectures, we chose the well known problem of grammatical inference, in which the network has to infer a grammar which best describes a set of positive and negative example strings of a regular language. As pointed out in [8], second order recurrent networks are more reliable than first order ones for this kind of problem, as they converge more quickly to a good solution. A benchmark for the grammatical inference problem is the set of seven small regular grammars introduced by Tomita [10], that generate languages with strings of arbitrary length over the alphabet {0, 1}. In our simulations an extra end symbol e was added to the

alphabet, since, as suggested by Giles et al. [9], this helps the network to find a good solution.

We carried out some experiments on Tomita's first grammar, which can be described as $T_1 = 1^*$,where (*string*)* represents a string repeated zero or more times. Simulations were carried out over the simple training set proposed by Tomita [10], consisting of 16 strings, uniformly distributed in the two classes of positive and negative examples, with increasing lenght up to 9 (including the final symbol). The network training was carried out by means of the incremental algorithm proposed in [8], with both learning rate and momentum term set to 0.5. Initial weight values were drawn randomly from a uniform distribution in [-1.0, 1.0]. The network was considered to classify a string correctly when $|x_0 - T| < \varepsilon$, where ε is a threshold fixed at 0.2 and T is the target value for the input string, that is assumed to be 0.8 for positive examples and 0.2 for negative ones.

We trained networks with 3 non-recurrent input neurons (including the one corresponding to the final symbol *e*), one output unit and 15, 12, 9, 7, 5 and 3 hidden neurons respectively. In order to avoid dependence on starting conditions each trial was repeated seven times with different starting weights. The averaged results are reported in table 1. For a large number of neurons, the network successfully recognizes the whole training set within few epochs. On the contrary, as soon as the number of hidden units becomes less than 7, the training procedure did not converge at all within 500 epochs.

Hidden nodes	Epochs	MSE
15	97	0.009
12	139	0.010
9	254	0.015
7	325	0.011
5	-	-
3	-	-

Table 1. *Results of the training phase over networks with different sizes.*

Next, the pruning algorithm was applied to the successfully trained networks with no stopping condition, in order to see how well the time required to train and prune a large solution network compares with that of directly training a small network. As an illustrative example, figures 1a and 1b show respectively the averaged misclassification error (tolerance $\varepsilon = 0.5$) and MSE of the12-node network during the pruning process. We observed that the reduced network with 7-hidden units has the same MSE as the network trained with 7 hidden units (see

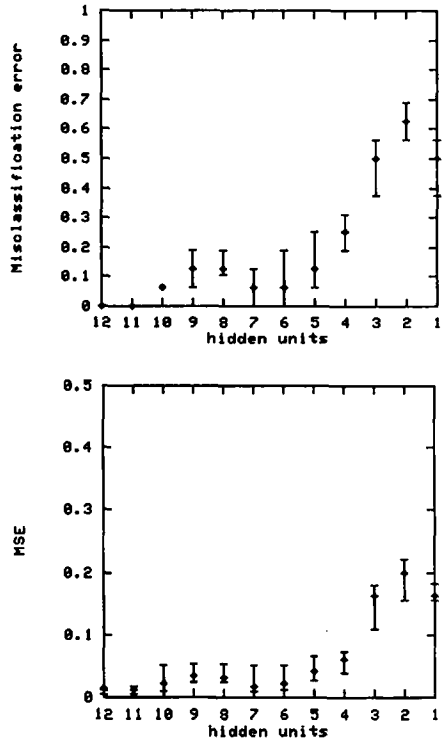

Fig. 1. *Averaged performance during pruning of 12-node networks. Up: (a). Bottom: (b).*

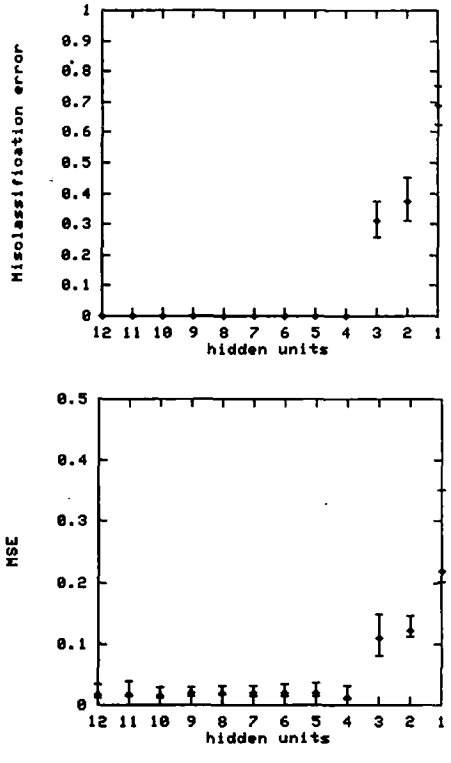

Fig. 2. *Averaged generalization of the reduced networks. Up: (a). Bottom: (b).*

table 1), but the former was obtained in fewer iterations. This suggests that training an overdimensioned network and then reducing it to a smaller size with the proposed method requires less time than simply training a network of that size.

Moreover, we tested the generalization of the reduced networks obtained at each step of the pruning procedure. The test set was composed of all the positive and negative strings with length less than 13 that were not in the training set. The results of this testing phase (figures 2a and 2b) show that the networks with 11 to 4 hidden units correctly generalize all the test strings. Therefore, if the pruning is stopped when the misclassification error computed on the test set becomes worse than the original one of 0.01 or more, we obtain a 4-hidden units network with the same generalization ability as the original 12-hidden network.

5. Conclusions

In this letter, a method of reducing the size of second-order recurrent neural networks has been developed.

The iterative nature of the proposed pruning procedure allows the network's designer to monitor the performance of the reduced networks, in order to define a stopping condition according to his own requirements. Therefore, the algorithm turns out to be very flexible.

Moreover, for the proposed procedure to work, no parameter needs to be set: this feature is in contrast with most pruning procedures existing in literature, that often require a preliminary phase of parameter tuning.

Initial experimental results demonstrated the effectiveness of the proposed method, that quickly reduces the network size while maintaining a good generalization ability.

Further work is in progress to test the feasibility of using the pruning method in more complex problems.

References

[1] L.B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment, *Proc. Int. Conference on Neural Networks*, San Diego, CA, vol. 2, pp. 609-618, 1987.

[2] J. Hertz, A. Krogh, R.G. Palmer. *Introduction to the theory of neural computation.* Addison-Wesley, Redwood City, CA, 1991.

[3] R. Reed. Pruning algorithms - a survey, *IEEE Trans. on Neural Networks*, vol.4, no. 5, pp. 740-747, 1993.

[4] C.L. Giles, C.W. Omlin. Pruning recurrent neural networks for improved generalization performance, *IEEE Trans. on Neural Networks*, vol. 5, no. 5, pp. 848-851, 1994.

[5] M. Pelillo, A. M. Fanelli. A method of pruning layered feed-forward neural networks, *New Trends in Neural Computation*, J. Mira, J. Cabestany, A. Prieto, eds., pp. 278-283, Springer-Verlag, Berlin, 1993.

[6] G. Castellano, A.M. Fanelli, M. Pelillo. Pruning in recurrent neural networks, *Proc. Int. Conf. on Artificial Neural Networks (Sorrento, Italy)*, pp. 451-454, 1994.

[7] A. Björck, T. Elfving. Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations, *BIT* vol. 19, 145-163, 1979.

[8] C.B. Miller, C.L. Giles. Experimental comparison of the effect of order in recurrent neural networks, *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 4, pp. 849-872, 1993.

[9] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun , Y.C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks, *Neural Computation*, vol.4, pp. 393-405, 1992.

[10] M. Tomita. Dynamic construction of finite-state automata from examples using hill-climbing, *Proc. Fourth Annual Cognitive Science Conf.*, Ann Arbor, MI, pp. 105-108, 1982.