Pruning Recurrent Neural Networks for Improved Generalization Performance *

C. Lee Giles [a,b], Christian W. Omlin [a,c]

[a] NEC Research Institute, 4 Independence Way, Princeton, NJ 08540

[b] UMIACS, U. of Maryland, College Park, MD 20742

[c] Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180

Abstract

Determining the architecture of a neural network is an important issue for any learning task. For recurrent neural networks no general methods exist that permit the estimation of the number of layers of hidden neurons, the size of layers or the number of weights. We present a simple pruning heuristic which significantly improves the generalization performance of trained recurrent networks. We illustrate this heuristic by training a fully recurrent neural network on positive and negative strings of a regular grammar. We also show that if rules are extracted from networks trained to recognize these strings, that rules extracted after pruning are more consistent with the rules to be learned. This performance improvement is obtained by pruning and retraining the networks. Simulations are shown for training and pruning a recurrent neural net on strings generated by two regular grammars, a randomly-generated 10-state grammar and an 8-state triple parity grammar. Further simulations indicate that this pruning method can gives generalization performance superior to that obtained by training with weight decay.

1 MOTIVATION

Choosing an appropriate architecture for a learning task is an important issue in training neural networks. Because general methods for determining a 'good' (recurrent) network architecture prior to training are lacking, algorithms that adapt the network architecture during training have been developed. These algorithms can be classified according to their different objectives as *constructive* or *destructive* training methods. This paper introduces a simple technique for pruning trained *recurrent* neural networks to significantly improve their generalization performance. To our knowledge, no such technique for recurrent neural networks has been previously published. Good generalization results have also be reported using weight decay ([8, 10]). We will compare our pruning method with weight decay for different decay rates.

*Published in *IEEE Trans. on Neural Networks* vol. 5, no. 5, p. 848, 1994.

2 PRUNING A RECURRENT NETWORK

To test our pruning heuristic, we incrementally trained discrete-time, fully recurrent temporally-driven neural networks with second-order weights W_{ijk} to learn regular languages [1, 4, 13, 15]. The network accepts a time-ordered sequence of inputs and evolves with dynamics defined by the following equations:

$$S_i^{(t+1)} = g(\Xi_i + b_i), \qquad \Xi_i \equiv \sum_{j,k} W_{ijk} S_j^{(t)} I_k^{(t)},$$

where g is a sigmoid discriminant function and b_i is the bias associated with hidden recurrent state neurons S_i. The output is the activation of one of the state neurons S_0 inspected at the end of a temporal sequence. The weights W_{ijk} were updated according to a second-order form of the real-time recurrent learning (RTRL) algorithm for recurrent neural networks ([16]). For more details see [4].

The heuristic used for extracting rules from recurrent networks in the form of deterministic finite-state automata (DFA's) is described in detail in [4]. Different approaches are discussed in [1, 15]. The quality of the extracted rules has been discussed in [5]. The algorithm used to extract a finite-state automaton from a network is based on the observation that the outputs of the recurrent state neurons of a trained network tend to cluster. Our hypothesis is that collections of these clusters correspond to the states of the finite-state automaton the network has learned. For the DFA extraction, we consider network states where the output of the output neuron S_0 is larger than 0.5 to correspond to accepting DFA states. Otherwise, the current network state corresponds to a rejecting DFA state. The problem of DFA extraction is thus reduced to identifying clusters in the output space $[0,1]^N$ of all state neurons. We use a dynamical state space exploration which identifies the DFA states and at the same time avoids exploration of the entire space which is computationally not feasible.

The extraction algorithm divides the output of each of the N state neurons into q intervals of equal size, yielding q^N partitions in the space of outputs of the state neurons. We also refer to q as the *quantization level.* Clearly, the extracted DFA depends on the quantization level q chosen, i.e., in general, different DFA will be extracted for different values of q. A subsequent standard minimization algorithm yields a unique, minimal representation of the extracted DFA's ([9]). Many different DFA extracted thus collapse into *equivalence classes.* Several minimized DFA's may be consistent with the training set. Thus, it becomes necessary to select among the candidate DFA's the one model which best approximates the unknown source grammar. Based on simulation results in [11], we choose the consistent minimized DFA which was extracted with the smallest quantization level q as the best model. As it turns out, the best model is also always the DFA with shortest description length (number of states). The specific issues of the extraction algorithm and quality of the extracted rules are discussed in [11, 12].

Our goal is to train networks of small size with improved generalization performance and also to improve the quality of the extracted rules. We start by training a large network for a known regular grammar and apply our network pruning and retraining strategy to the trained network. Whenever the training is successful, the state neuron with the smallest weight vector is removed and the network is retrained using the same training set. This process is repeated until either a network with satisfactory generalization performance is obtained or until the retraining fails to converge within a certain number of epochs. When the current network fails to converge, we choose the network trained in the previous prune/retrain cycle as our solution network.

3 SIMULATION RESULTS

3.1 Experiments

We trained recurrent networks on two different sequentially-presented training sets consisting of positive and negative strings. The first set of strings was obtained from the randomly generated 10-state DFA shown in figure 1a and the second set from the 8-state DFA shown in figure 1b (this DFA accepts only strings which have an even number of 0's, 1's and 2's, triple parity). Note that all DFA accept strings of *arbitrary length.* A finite disjoint subset of these strings is chosen for the training and test sets. It consists of the first 500 positive and 500 negative example strings in alphabetical order with alternating positive and negative strings. Since this a second-order modification of RTRL, training occurs at the end of each presented string. An incremental training method discussed in [4] was used for the string presentation. The initial training set consisted of 30 strings. This training set was successfully learned and a small number of incorrectly classified strings were then added to the original set. The process was repeated until all strings in the training set were correctly classified. The learning rate and the momentum were set to 0.5. We started training with a network of 15 state neurons and the weights were initialized to random values in the interval [-1.0, 1.0].

All networks were trained on the same training set. However, because of the incremental training heuristic, none of the networks needed to be trained on all strings of the training set. The learning of the internal representation of the DFA states from short strings permitted the network to correctly classify longer strings without actually training on these strings.

3.2 Generalization Performance

For each (re)training/pruning cycle, we show in table 1 the number of state neurons, the (re)training time, the size of the training set necessary for successful training, the generalization performance of the trained network, the quantization level q used for DFA extraction, the size of a good minimized DFA extracted from the network and its generalization performance.

The results for the randomly generated 10-state DFA are shown in table 1. A network with 15 state neurons learned the training set relatively easily (197 epochs) and only a fraction of the entire training set was necessary (142 strings). The generalization performance of the trained network on all strings of length up to 20 (2,097,150 strings) is fairly good with only 6.75% of all strings misclassified. For DFA extraction, we only considered DFA's that were consistent with the training set, i.e. the DFA's correctly classified all strings

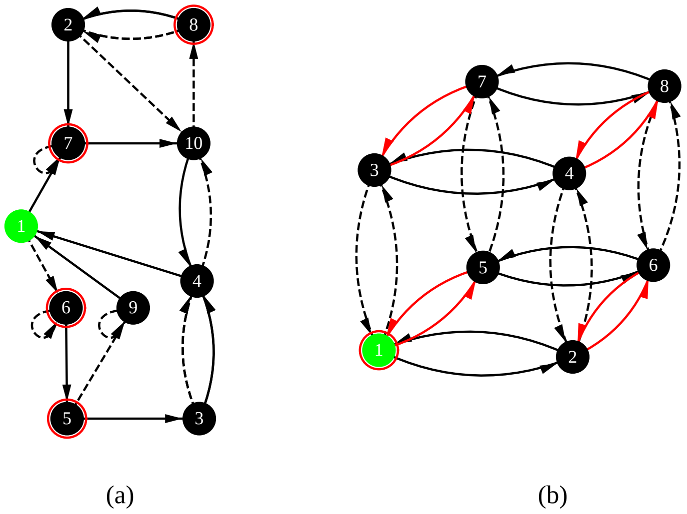

Figure 1: **Inferred DFA's.** The shaded state is the start state; circled states are accepting states. (a) Randomly generated 10-state DFA with two input symbols. (b) DFA that accepts only strings with an even number of 0's, 1's and 2's (triple parity).

of the training set. As a good model of the regular language, we chose the consistent DFA extracted with the smallest quantization level q ([5]). For the trained 15-neuron network, we were able to extract a consistent DFA for q=2; however, the minimized DFA had 382 states as compared to 10 states for the DFA that generated the training set. The extracted DFA's generalization performance is impressive with only 0.41% of all test strings misclassified, thus outperforming the trained network (this is often the case, see for example [5]).

After pruning the first state neuron of the network, the retraining time was negligible (7 epochs), indicating that the pruned state neuron did not contribute significantly to the internal representation of the learned DFA. The generalization performance of the pruned and retrained network and the extracted DFA were comparable to the performance of the larger network.

Retraining became more difficult with fewer state neurons but the network generalization performance improved by an order of magnitude (0.14% for the 7-neuron network). This improvement would come as no surprise if the maximum size of the training set were also increasing; with more training strings used, one would certainly expect the network to perform better. However, the generalization improvement was achieved in most cases *without* additional new training strings. At each stage of the pruning/retraining process, the size of the training set was smaller than the size of the training set used to train the 15-neuron network. This clearly shows that the performance improvement is due to the reduced size of the network.

Neurons	Time	Size	NN Performance	q-level	DFA States	DFA Performance
15	197	142	6.75%	2	382	0.41%
14	7	46	6.89%	2	484	1.57%
13	98	99	2.61%	2	314	0.35%
12	11	62	1.51%	2	10	0.00%
11	14	67	0.97%	2	10	0.00%
10	22	83	1.26%	2	135	0.05%
9	111	157	2.95%	2	151	0.62%
8	102	140	2.44%	4	505	1.21%
7	104	118	0.14%	2	10	0.00%

Table 1: **Random DFA:** Network performance after each pruning cycle; epochs; maximal size of the maximal training set; NN classification errors on test set; quantization level; size of extracted DFA; DFA classification errors.

After retraining the 7-neuron network, we attempted to further reduce the size of the network. However, the 6-neuron network failed to converge within 50,000 epochs. Thus, the 7-neuron network was our final network. Trained recurrent networks make generalization errors because the internal representation of DFA states is unstable, i.e. with increasing string length, well-separated neuron activation clusters formed during training begin to merge together ([17]). The extracted DFA's do not share this problem and thus show consistently better generalization performance. The DFA extracted from the smallest network (7 neurons) was identical with the original DFA. The quality of the extracted rules also tends to improve with decreasing network size.

The results shown for triple parity (table 2) confirm our findings that our pruning/retraining algorithm is an effective tool for improving the generalization performance of both the trained network as well as the extracted DFA. Note that in this case the resultant 3-state neural network is the least size neural network for "representing" the 8-state DFA *if* the internal state representations of the network are confined to the rails of the sigmoid ([17]).

Neurons	Time	Size	NN Performance	q-level	DFA States	DFA Performance
15	183	209	13.64%	3	3105	8.42%
14	3	42	13.85%	3	2560	4.17%
13	17	84	10.08%	2	128	0.00%
12	22	99	9.62%	2	81	0.00%
11	12	65	5.45%	2	8	0.00%
10	23	92	3.87%	2	46	0.00%
9	20	83	3.27%	3	124	0.00%
8	23	91	4.07%	2	8	0.00%
7	36	93	3.16%	2	8	0.00%
6	29	96	3.98%	3	8	0.00%
5	39	87	0.58%	2	8	0.00%
4	29	85	2.08%	2	8	0.00%
3	179	92	0.75%	3	8	0.00%

Table 2: **DFA for Triple-Parity:** Network performance after each pruning cycle; epochs; maximal size of the maximal training set; NN classification errors on test set; quantization level; size of extracted DFA; DFA classification errors.

3.3 Comparison with Weight Decay

It has been observed in simulations that weight decay can improve the generalization performance of feed-forward networks ([8, 10]). Weight decay suppresses irrelevant components of weight vectors by choosing a small vector that solves the learning problem.

For networks trained using weight decay, the error function is expanded to include an error term which penalizes large weights: The weight update then becomes

$$\Delta w_{ijk} = -\alpha \, \frac{\partial E_0}{\partial w_{ijk}} - \lambda w_{ijk}$$

The results in table 3 show a comparison of the performances of pruned networks with the generalization of networks trained with weight decay for varying decay rates λ. In all but one case, the pruned networks outperformed the networks with weight decay. The training time for pruned networks includes the initial time necessary to train a 15-neuron network and the retraining time for each pruning step. The pruning always resulted in networks with 7 state neurons. The training times for pruned networks and networks with weight decay were comparable, although pruning causes fewer weight updates after each pruning/retraining cycle. The methods refer to plain training (none), training with pruning (pruning), and training with weight decay ($\lambda = 0.0001$, $\lambda = 0.0005$, $\lambda = 0.001$). The pruning heuristic always improved both the network generalization performance and the extracted DFA, especially when the ideal DFA with 10 states was not already extracted in the original 15-neuron network. The convergence time for training with weight decay increases with increasing decay rate. None of the runs converged for values of λ larger than the ones shown. In cases where the original network was not well trained (table 3a), weight decay improved network generalization and the extracted rules. However, in all other cases (tables 3b-d), networks trained with weight decay can show worse generalization performance and DFA's were extracted that were consistent with the training data, but not identical with the ideal 10-state DFA. We can conclude that our pruning heuristic generally results in better trained networks and smaller extracted DFA's that explain the training data than weight decay methods. In addition we did not have the weight decay disadvantage of possible failure to converge to a good solution or the need to set the decay rate λ prior to training.

Method	Time	NN Performance	DFA States
none	197	6.75%	382
pruning	666	0.14%	10
$\lambda = 0.0001$	199	4.18%	10
$\lambda = 0.0005$	257	3.18%	30
$\lambda = 0.001$	401	2.20%	10

(a)

Method	Time	NN Performance	DFA States
none	175	2.76%	81
pruning	437	0.21%	10
$\lambda = 0.0001$	141	1.32%	13
$\lambda = 0.0005$	186	1.03%	10
$\lambda = 0.001$	362	2.92%	10

(b)

Method	Time	NN Performance	DFA States
none	151	0.90%	10
pruning	375	0.00%	10
$\lambda = 0.0001$	154	1.93%	87
$\lambda = 0.0005$	169	0.97%	10
$\lambda = 0.001$	305	1.74%	10

(c)

Method	Time	NN Performance	DFA States
none	161	2.14%	10
pruning	282	1.12%	10
$\lambda = 0.0001$	172	0.61%	10
$\lambda = 0.0005$	200	3.28%	72
$\lambda = 0.001$	351	2.49%	13

(d)

Table 3: **Comparison Pruning vs. Weight Decay:** The methods refer to plain training, training with pruning, and training with weight decay rates ($\lambda = 0.0001$, $\lambda = 0.0005$, $\lambda = 0.001$).

4 CONCLUSIONS

We have presented a destructive method for improving the generalization performance of recurrent neural networks, trained to recognize strings of regular languages. Our simulation results demonstrate that pruning combined with retraining can significantly improve the performance of the networks themselves and also of the extracted symbolic rules. The pruning procedure is a repetitive cycle of reducing the size of the architecture and retraining the network. Our method is based on a simple heuristic which assesses the relevance of recurrent state neurons according to the magnitude of the incoming weights. State neurons with small weights tend to contribute less to the overall computation and thus are promising pruning candidates. The pruned network needs to be retrained to achieve its performance prior to the pruning step. As to be expected, the retraining becomes harder as the size of the network decreases; the performance improves while generally using fewer strings than were necessary to train the original network. Our preliminary results where the generalization performance improves by an order of magnitude using a simple pruning heuristic are encouraging. The performance improvements of pruned networks are generally superior to networks trained with weight decay; training is faster due to the shrinking network size and there is no need for determining a decay rate prior to training. It would be interesting to compare our method with other weight pruning methods ([2, 7]).

An open question is whether this pruning method produces the smallest network necessary to learn (or represent) the deterministic finite automata (DFA) to be learned. Certainly for the triple parity DFA, a 3-state neural network is the smallest possible if the neuron state activations are confined to the rails of the sigmoid. But the 10-state random DFA should have had a 4 state recurrent network. However, that did not occur; training failed to converge. It would be interesting to see if knowledge inserted into the network before or during training [3, 6, 14] aids or impedes the pruning process.

References

[1] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural Computation*, vol. 1, no. 3, pp. 372–381, 1989.

[2] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2* (D. Touretzky, ed.), (San Mateo, CA), pp. 598–605, Morgan Kaufmann Publishers, 1990.

[3] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "A unified approach for integrating explicit knowledge and learning by example in recurrent networks," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 1, p. 811, IEEE 91CH3049-4, 1991.

[4] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee, "Learning and extracting finite state automata with second-order recurrent neural networks," *Neural Computation*, vol. 4, no. 3, p. 380, 1992.

[5] C. L. Giles and C. W. Omlin, "Extraction, insertion and refinement of symbolic rules in dynamically-driven recurrent neural networks," *Connection Science*, vol. 5, no. 3,4, pp. 307–337. Special Issue on Architectures for Integrating Symbolic and Neural Processes.

[6] C. L. Giles and C. W. Omlin, "Inserting rules into recurrent neural networks," in *Neural Networks for Signal Processing II, Proceedings of The 1992 IEEE Workshop* (S. Kung, F. Fallside, J. A. Sorenson, and C. Kamm, eds.), pp. 13–22, IEEE Press, 1992.

[7] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5* (S. Hanson, J. Cowan, and C. Giles, eds.), (San Mateo, CA), Morgan Kaufmann Publishers, 1993 - to be published.

[8] G. E. Hinton, "Learning translation invariant recognition in a massively parallel network," in *PARLE: Parallel Architectures and Languages Europe*, pp. 1–13, Berlin: Springer Verlag, 1987. Lecture Notes in Computer Science.

[9] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley Publishing Company, Inc., 1979.

[10] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems 4* (J. Moody, S. Hanson, and R. Lippmann, eds.), (San Mateo, CA), pp. 950–957, Morgan Kaufmann Publishers, 1992.

[11] C. W. Omlin and C. L. Giles, "Extraction of rules from discrete-time recurrent neural networks," Tech. Rep. 92-23, Computer Science Department - Rensselaer Polytechnic Institute, Troy, NY 12180, 1992.

[12] C. W. Omlin, C. L. Giles, and C. B. Miller, "Heuristics for the extraction of rules from discrete-time recurrent neural networks," in *Proceedings International Joint Conference on Neural Networks 1992*, vol. I, pp. 33–38, June 1992.

[13] J. B. Pollack, "The induction of dynamical recognizers," *Machine Learning*, vol. 7, pp. 227–252, 1991.

[14] J. W. Shavlik, "A framework of combining symbolic and neural learning," Tech. Rep. TR 1123, Computer Sciences Dept., Computer Sciences Dept, U of Wisconson - Madison, 1992.

[15] R. L. Watrous and G. M. Kuhn, "Induction of finite-state languages using second-order recurrent networks," *Neural Computation*, vol. 4, no. 3, p. 406, 1992.

[16] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

[17] Z. Zeng, R. M. Goodman, and P. Smyth, "Learning finite state machines with self-clustering recurrent networks," *Neural Computation*, vol. 5, no. 6, pp. 976–990, 1993.