Reducing Network Depth in the Cascade-Correlation Learning Architecture

Shumeet Baluja & Scott E. Fahlman

October 17, 1994

CMU-CS-94-209

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

The Cascade-Correlation learning algorithm constructs a multi-layer artificial neural network as it learns to perform a given task. The resulting network's size and topology are chosen specifically for this task. In the resulting "cascade" networks, each new hidden unit receives incoming connections from all input and pre-existing hidden units. In effect, each new unit adds a new layer to the network. This allows Cascade-Correlation to create complex feature detectors, but it typically results in a network that is deeper, in terms of the longest path from input to output, than is necessary to solve the problem efficiently. In this paper we investigate a simple variation of Cascade-Correlation that will build deep nets if necessary, but that is biased toward minimizing network depth. We demonstrate empirically, across a range of problems, that this simple technique can reduce network depth, often dramatically. However, we show that this technique does not, in general, reduce the total number of weights or improve the generalization ability of the resulting networks.

Shumeet Baluja is supported by a National Science Foundation Fellowship. This research was supported by the Department of the Navy, Office of Naval Research under Grant No. N00014-93-1-0806 and by the National Science Foundation under grant IRI9214873. The conclusions presented in this paper are those of the authors and not necessarily those of the U.S. government or the sponsoring agencies.

Keywords

Artificial Neural Networks, Cascade-Correlation, Network Depth, Constructive Neural Network Learning Algorithm

1 INTRODUCTION

In 1990, the Cascade-Correlation learning algorithm (sometimes shortened to “Cascor”) was introduced by Scott E. Fahlman and Christian Lebiere [Fahlman and Lebiere, 1990]. Cascor offers two major advantages over standard back-propagation networks: First, the user does not have to guess in advance the proper size and topology for a network to fit the problem at hand. Cascor develops a good, though not necessarily optimal, network topology in the course of learning each new problem. Second, Cascor's greedy learning algorithm can be several orders of magnitude faster than standard backpropagation, which must engage in a complex, dynamic, competitive process in order to sort out which hidden unit will do which job within the network [Fahlman, 1988].

Cascor begins with a minimal network, then automatically trains new hidden units one by one and adds them to the active net. Each new hidden unit receives a connection from each of the network's original inputs and also from each of the previously installed hidden units. Thus, the addition of each hidden unit effectively creates a new layer in the network, lengthening the longest signal path from input to output by one device and one layer of weights. One of the advantages of this method is that each new unit creates a high-order feature detector, able to compute a more complex function than the units in earlier layers. However, the resulting networks may be much deeper than necessary for solving the problem efficiently.

It is well known that a network with a single layer of hidden units with sigmoid activation functions is sufficient to approximate any continuous function [Hornik, Stinchcombe, White, 1989]. However, the number of units required in this single layer may be very large. For some problems, such as the “two spirals” problem addressed later in this paper, the number of hidden units and the time required for training can be greatly reduced by using more hidden layers. Cascor handles such problems well, but since each hidden unit occupies a layer of its own, the networks are almost always deeper than is either necessary or useful.

Is this excess depth a problem? Many people regard it as one. There have been a number of papers which dismiss Cascor on the grounds that the networks it produces are too deep. See, for example, [Phatak & Koren, 1994], in which the authors limit the network topology to a strictly layered form, with all the layers limited to the same number of hidden units, pre-determined by the user. Other alternative schemes propose constructive learning algorithms that use only a single layer of hidden units, or that impose some other tight restriction on network depth. Such schemes work well when the problem fits into the limited architecture that they support, but they either do poorly or fail, when the problem can benefit from a deep net. Another often noted criticism of the Cascor algorithm is the extra time required to propagate a signal through many excess layers in very deep networks. The growth of the hidden units' fan-in as the network grows deeper has also been regarded as a drawback of the Cascor algorithm. Additionally, the increased fan-in and depth of the network make it hard for humans to interpret the functions being performed, or the features being detected, by units deep in the network. Although only some of these problems may be pertinent to a given application of the algorithm, all have been cited as potential problems which need to be addressed.

In this paper we will show that if it is desired to reduce excess network depth, a simple modification of the Cascor algorithm will suffice. The resulting algorithm still retains the ability to build deep nets when the problem calls for them. In this paper, we will also investigate whether the shallower nets produced by this method are better at generalizing to new cases that are similar but not identical to those in the training set.

1.1 Introduction to Cascade-Correlation

The Cascade-Correlation network is initialized with no hidden units. There is only a direct connection from each input to each output unit. These initial connections are trained as well as possible. Any single-layer network learning algorithm could be used to train the output weights. In the simulations reported here, the Quickprop algorithm is used [Fahlman, 1988].

After training ceases to yield significant improvement, and if the error has not been reduced to a satisfactory level, a single new hidden unit is trained (see below) and added to the active network. After the new unit is added, its input weights are frozen, and all of the weights entering the output units are again trained using Quickprop. This cycle is repeated until the error is satisfactorily reduced.

To train a new hidden unit, we begin with a "candidate unit" that receives an incoming connection from each of the network's original inputs and from each pre-existing hidden unit. During training, the candidate unit's output is unconnected, so it has no effect on the active network's outputs. The incoming connections are initialized with small random weights. Holding the active network fixed, we cycle through the set of training examples, adjusting the incoming weights so as to iteratively increase the magnitude of **S**, a measure of the correlation between the candidate unit's output value and the residual output error observed at the output units. The term we want to maximize can be expressed by equation 1, in which **V** is the candidate unit's value, and $\mathbf{E_o}$ is the residual output error observed at output **o**. In a manner similar to the derivation of backpropagation, the derivative $\partial\mathbf{S}/\partial\mathbf{w}$ with respect to each of the unit's incoming weights is determined, and the weights are then modified (using Quickprop) so as to increase **S**.

$$S = \sum_{outputs\,(o)} \left| \sum_{patterns\,(p)} (V_p - \overline{V})\,(E_{p,o} - \overline{E_o}) \right| \qquad (1)$$

Instead of training only a single hidden unit at a time, a pool of candidate units, each with a different set of random initial weights, can be trained simultaneously. When all of the candidate units have ceased to improve their **S** scores, the candidate unit with the best score is chosen for installation into the active network; the others are discarded. This use of a candidate pool reduces the chance of installing a new, permanent unit that has become stuck in a local minimum of the learning space, and that would contribute little to the network's performance. It also makes it possible for a mixture of unit-types or units with different input connectivity to compete for a place in the active network. On a parallel machine, the candidate units can be trained in parallel, since there is a no interaction between these units except to select a winner [Fahlman & Lebiere, 1990].

2 LEARNING WITH SIBLING AND DESCENDANT UNITS

A simple modification of Cascor is effective in reducing the depth of the resulting networks. We split the pool of candidate units into two groups. One group, the *descendant units*, operates just as described earlier: each candidate unit receives inputs from all pre-existing hidden units, and deepens the active net by one layer when it is installed. The other group, the *sibling units*, receive inputs from earlier layers of the net, but not from those units that are currently in the deepest layer of the net. When a sibling unit is added to the active net, it becomes part of the current deepest layer — a sibling of the deepest units created so far —but it does not begin a new, deeper layer in the network. We call this revised algorithm "Sibling/Descendant Cascade-Correlation" or *SDCC*.

This use of separate sibling and descendant pools in Cascor was first investigated by Christian Lebiere and Scott Fahlman. Lebiere did observe a reduction in network depth, but the work was abandoned after a few experiments and was never published. More recently, this architecture and several other Cascor variants were studied by Waugh and Adams [1994], but using only the two-spirals problem and a very similar "double helix" problem. Their results are consistent with ours, but it is hard to draw any general conclusions from so limited a set of benchmark problems.

During candidate training, the sibling and descendant units compete with one another. The one with the best **S** score is chosen for installation in the active network. Thus, the network will only be deepened in cases where a descendant unit is able to do a better job of matching the residual error than any of the sibling units. It may seem surprising that a sibling unit ever wins the contest, since they share all the inputs of the sibling units, plus one or more additional inputs. In principle, these additional inputs should only help the descendant units; if they do not, the learning algorithm should set these weights to zero. However, in practice, the sibling units, with fewer inputs, are able to converge more quickly, and they often maintain a slight edge in the **S** score unless the extra inputs give the descendant units some real advantage.

If we want to create a stronger bias in favor of shallow networks, we can penalize the descendant units by multiplying their **S** scores by a factor $\lambda \leq 1.0$. For example, if $\lambda = 0.5$, a descendant unit will only be selected if its **S** score is twice that of the best sibling unit. In the experiments reported below, we test six values of λ for each problem.

This modification in the algorithm leads to large differences in the network architecture. For example, consider the cross recognition problem (described in detail in the next section). In Figure 1, we illustrate the network architecture developed using standard Cascor and two typical runs of SDCC with $\lambda = 0.9$. The depth of the network is reduced from 7 hidden layers to one or two.

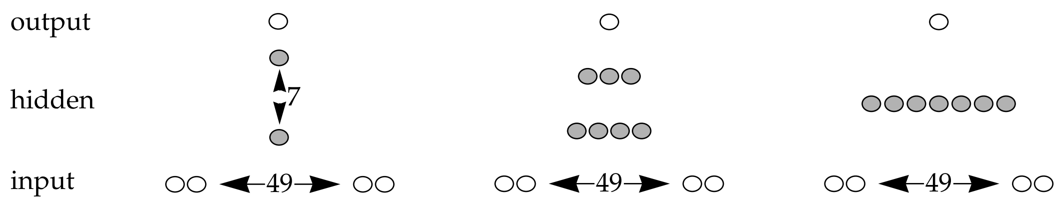


Figure 1: Network Architectures created by standard Cascade-Correlation (left) and by Sibling/Descendant Cascade-Correlation (middle and right), with $\lambda = 0.9$. The architecture on the left has 7 hidden layers, the one in the middle 2, the one on the right, 1. Hidden units are shown shaded. Each hidden unit has an input connection from every unit in the layers below its own.

3 EMPIRICAL ANALYSIS

To analyze the benefit of using siblings rather than descendants alone, a large number of test were performed on benchmark problems. These tests can be broadly divided into two groups. The first set of tests are to determine the new algorithm's ability to generalize, while the second are to determine the new architecture's ability to memorize. Each group of tests is done with six settings for the λ parameter for the descendant units. Each of the results reported here is the average of at least 25 training sessions per λ setting.

All of the parameters used in this experiment are set to the default values of the Cascade-Correlation simulator available through Carnegie Mellon University's Neural Network Archives. Unless otherwise noted, there were 8 candidate units in the candidate pool. In the SDCC algorithms four of the candidate units were trained as descendents, and four were trained as siblings. In the original version of Cascade-Correlation, all of the eights candidate units were descendent units. All of the candidate units were sigmoid units, with output values in the range of ±0.5. No weight decay was used. As suggested in [Fahlman, 1988], no weight was allowed to grow more than by a factor of 2 from the previous time step (Mu parameter). Also as suggested in [Fahlman, 1988], an offset of 0.1 is added to the sigmoid prime values of the output units to eliminate the flat spot where the derivative of the sigmoid function approaches zero (SigPrimeOffset Parameter). Training, in all of the experiments, is continued until all of the training examples are correctly classified.

3.1 Generalization Tasks

Five classes of problems were empirically examined. The suite of test problems included the parity problems, classification of the Vowels database, classification of the Sonar database, a checkerboard problem, and a cross recognition problem.

3.1.1. The Parity Benchmark

Three versions of the parity problem were attempted to ascertain generalization ability. The problems attempted here included 10 bit parity problems with 512 training points and 512 testing points, the 10 bit parity with 256 training points and 768 testing points, and the 12 bit parity with 1024 training points and 3096 testing points. The results are shown below, in Table 1. In all of the tables, percent correct refers to the performance on the test set.

Table 1: The Parity Benchmark

TASK	Table Interpretation	Standard Cascor	Sibling/Descendant Cascade-Correlation					
			$\lambda = 1.0$	$\lambda = 0.95$	$\lambda = 0.9$	$\lambda = 0.8$	$\lambda = 0.5$	$\lambda = 0.2$
PARITY 10	Percent Correct	97.6	97.7	97.7	97.4	97.0	95.0	94.2
512 Train	Hidden Units	5.8	5.8	6.1	6.4	7.0	8.6	8.9
512 Test	Hidden Layers	5.8	3.8	3.2	3.1	2.7	1.16	1.0
	Connections	94.5	92.4	97.7	100.3	107.3	115.9	117.6
PARITY 10	Percent Correct	86.4	85.9	85.4	85.1	81.2	75.4	72.5
256 Train	Hidden Units	6.5	6.7	6.6	6.8	7.4	8.7	9.5
768 Test	Hidden Layers	6.5	4.2	3.5	3.0	2.4	1.1	1.0
	Connections	108.4	107.8	105.1	106.1	111.3	116.2	124.5
PARITY 12	Percent Correct	96.5	97.5	97.4	98.0	97.7	94.2	92.3
1024 Train	Hidden Units	8.5	8.2	8.4	7.6	10.1	13.5	15.6
3096 Test	Hidden Layers	8.5	5.1	4.8	3.8	3.2	1.5	1.0
	Connections	170.5	154.0	158.1	160.1	174.2	211.5	234.0

3.1.2. The Vowel and Sonar Classification Problems

The vowel classification task data is taken from [Robinson, 1989]. The task is to classify eleven steady state vowels of British English using a training set of lpc derived log area ratios. The input size is 10 units, the output is represented as 11 binary units of which only one is turned on in a given example. The sonar signal classification task is to discriminate between sonar signals which bounce off of a metal cylinder and those that bounce from roughly cylindrical rock [Gorman and Sejnowski, 1988]. The number of inputs was 60, a single binary output was used. The data for both of these experiments were obtained from the Carnegie Mellon University's Neural Network Bench Archives. The Sonar classification task was also attempted with a large weight-decay (0.03) applied during candidate-unit training, which improves generalization performance in this case. The results for all of these experiments are shown in Table 2.

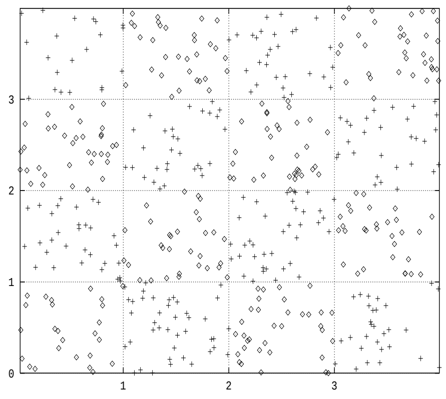
The percentage of correct values for the vowel set are computed in a non-standard manner. The percent correct refers to the sum of the number of correct outputs for each sample in the testing set, divided by the number of total outputs in the testing set.

Table 2: The Vowel and Sonar Classification Benchmark

TASK	Table Interpretation	Standard Cascor	Sibling/Descendant Cascade-Correlation					
			λ = 1.0	λ = 0.95	λ = 0.9	λ = 0.8	λ = 0.5	λ =0.2
VOWELS	Percent Correct Hidden Units Hidden Layers Connections	89.6 15.7 15.7 581.8	89.7 15.4 8.8 561.9	89.4 15.7 6.8 566.1	89.7 16.4 5.2 579.9	89.5 17.2 2.7 580.0	89.7 18.5 1.0 527.4	89.9 18.4 1.0 525.6
SONAR	Percent Correct Hidden Units Hidden Layers Connections	75.5 1.4 1.4 146.3	76.0 1.0 1.0 125.5	76.8 1.0 1.0 123.0	75.8 1.0 1.0 125.5	75.6 1.0 1.0 125.5	76.2 1.0 1.0 123.0	76.3 1.1 1.0 128.0
SONAR decay= 0.03	Percent Correct Hidden Units Hidden Layers Connections	85.6 6.6 6.6 488.9	84.6 6.5 5.8 479.9	85.0 6.8 6.0 501.6	85.3 6.8 5.3 503.1	85.5 6.7 4.8 494.3	85.2 7.5 3.2 540.0	85.5 11.2 1.0 752.9

3.1.3. The Checkerboard Problem

This problem is a variant of the traditional exclusive-or (XOR) problem. In this problem, there are two continuous valued inputs, representing the (x,y) coordinates in a 2D plane, and one binary output. The task is to determine whether the point designated by the inputs would fall on a black or red square of a standard checkerboard. Three versions of this problem were studied: the first had 500 training examples and 2500 testing points, the second had 250 training examples and 2750 testing points, and the third had 125 training examples and 2875 testing points. The training and test sets for the first trial are shown below, in Figure 2. The results of the experiments are shown in Table 3.



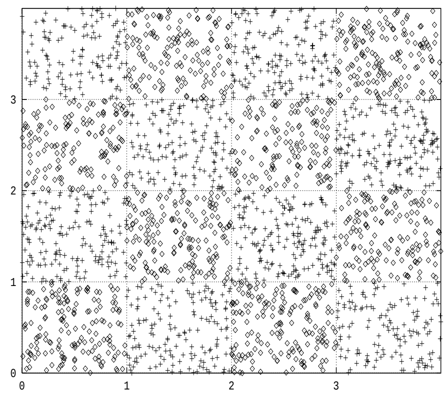


Figure 2: Training (left) and Testing (right) data for the checkerboard problem, version 1. Crosses represent positive examples, diamonds represent negative examples.

Table 3: The Checkerboard Problem

TASK	Table Interpretation	Standard Cascor	Sibling/Descendant Cascade-Correlation					
			λ = 1.0	λ = 0.95	λ = 0.9	λ = 0.8	λ = 0.5	λ =0.2
CHECKER- BOARD Train 500 Test 2500	Percent Correct Hidden Units Hidden Layers Connections	88.4 24.1 24.1 382.0	88.4 24.7 16.8 387.0	88.6 24.7 14.1 382.8	88.5 24.5 12.2 370.6	89.0 25.0 8.9 367.7	90.0 29.2 4.0 413.9	91.6 34.3 2.4 421.8
CHECKER- BOARD Train 250 Test 2750	Percent Correct Hidden Units Hidden Layers Connections	81.3 17.3 17.3 215.1	80.7 17.5 12.3 212.6	81.7 17.8 10.5 215.7	81.0 18.3 8.6 221.1	82.0 18.2 6.6 208.4	82.6 20.5 2.9 210.2	84.0 24.1 2.0 178.2
CHECKER- BOARD Train 125 Test 2875	Percent Correct Hidden Units Hidden Layers Connections	75.4 12.8 12.8 131.4	74.6 13.2 8.8 132.0	74.6 13.6 7.9 135.7	74.8 13.5 6.5 130.4	74.2 14.0 5.3 134.4	75.3 15.6 2.4 125.6	76.2 19.2 2.0 121.2

3.1.4. Cross Recognition

Two cross recognition problems were attempted, the first had 36 inputs, the second had 49 inputs. Each had 1 output. All of the inputs and output are binary. For visualization, the inputs are arranged as a two dimensional 6x6 and 7x7 grid for the two problems, respectively. The task is to determine if exactly one cross shaped pattern of activation is present within the input grid. Each input presentation has the same number of inputs turned on. The training set was designed as follows: there can exists 0, 1, or 2 crosses in the input grid. The center of each of the crosses, if any are present in the example, occur somewhere on the upper-left to bottom-right diagonal, as shown in Figure 3. A positive example is one in which there is only 1 cross. Positive noise was added to each input example. If this noise caused the existence of a new cross, the example was deleted, and a new example created. A sample of the classifications is shown below, in Figure 3. The performance for both of the problems is given in Table 4. For both of the problems, 1000 examples were used for training, and 1500 for testing.

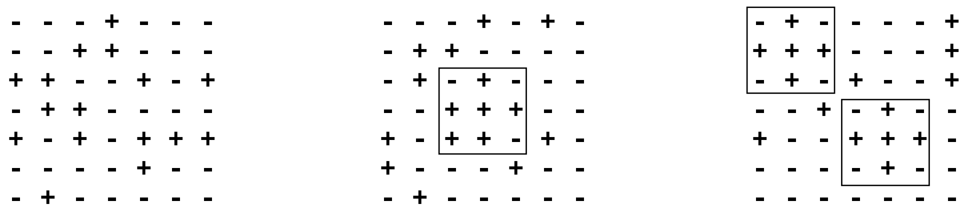


Figure 3: Three sample inputs for the cross recognition problem. Only the second example should be classified as positive, as it contains exactly one cross. Samples are taken from the 7x7 training set.

Table 4: The Cross Recognition Problem

TASK	Table Interpretation	Standard Cascor	Sibling/Descendant Cascade-Correlation					
			λ = 1.0	λ = 0.95	λ = 0.9	λ = 0.8	λ = 0.5	λ =0.2
CROSS	Percent Correct	90.9	90.7	91.3	91.1	90.6	90.2	90.4
(6x6)	Hidden Units	5.4	5.4	5.3	5.7	5.7	5.8	5.9
1000 Train	Hidden Layers	5.4	3.6	2.2	1.8	1.0	1.0	1.0
1500 Test	Connections	252.5	251.5	245.1	257.2	254.4	258.9	260.4
CROSS	Percent Correct	83.4	82.3	82.3	82.2	82.0	81.6	82.2
(7x7)	Hidden Units	6.2	6.2	6.5	6.5	6.7	6.7	6.8
1000 Train	Hidden Layers	6.2	3.7	2.4	1.6	1.2	1.0	1.0
1500 Test	Connections	384.7	376.6	392.2	384.6	392.2	392.7	396.8

3.2 Memorization Tasks

To test the ability of the revised Cascade-Correlation to memorize training sets, three problems were studied. The first is the two spirals problem. The task is as follows: given two concentric spirals, determine to which spiral a given point belongs. A version of this problem was also attempted with four spirals with a single output node. The desired output was one of four values which were evenly spaced in the region of -0.5 to +0.5. The training points for the two and four spirals problems are shown in Figure 4. The last problem in this section is the 10 parity problem, with all 1024 points used for training. The results are shown in Table 5. In the 0.2 column, the row marked with N/A indicates that with this setting for λ, the revised algorithm was unable to find a solution with the addition of 100 hidden units in all but one run.

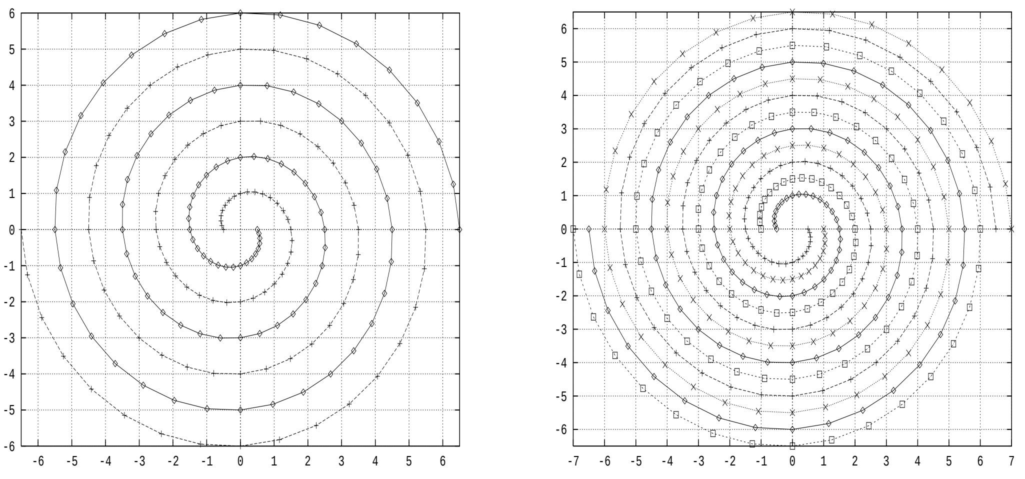


Figure 4: Training points for the Two Spiral (left) and Four Spiral (right) problems.

Table 5: Memorization Tasks

TASK	Table Interpretation	Standard Cascor	Sibling/Descendant Cascade-Correlation					
			λ = 1.0	λ = 0.95	λ = 0.9	λ = 0.8	λ = 0.5	λ =0.2
Two Spiral	Hidden Units Hidden Layers Connections	13.4 13.4 128	14.0 11.0 133	14.2 10.0 133	13.9 8.8 128	14.6 7.3 134	17.6 4.2 170	38.5 2.0 409
Four Spiral	Hidden Units Hidden Layers Connections	39.5 39.5 891.6	39.2 28.2 864.8	43.3 23.8 904.2	39.9 21.2 875.4	40.9 14.2 882.1	52.5 5.6 1211.2	N/A
PARITY 10 (1024 examples)	Hidden Units Hidden Layers Connections	6.2 6.2 102.6	6.7 4.8 108.3	7.0 3.2 109.3	6.5 2.6 98.9	7.5 2.2 113.0	10.2 1.2 134.2	10.6 1.0 138.7

4 CONCLUSIONS AND FUTURE RESEARCH

These experiments show that the Sibling/Descendant variation of Cascade-Correlation is able to very substantially reduce the number of layers in the network, as compared to the standard Cascade-Correlation algorithm. Applying a relatively small penalty to units which create a new layer ($\lambda = 0.8$) reduces the network depth by a factor of 2–5 without a large impact on the network's ability to generalize. Using a larger penalty factor ($\lambda = 0.2$) reduces the depth still more, though sometimes this causes a modest reduction in the generalization score. However, in some of the more difficult generalization problems, the use of $\lambda = 0.2$ may cause the number of hidden units to explode or may prevent the network from converging altogether.

Despite the reduction in network depth and fan-in, we observed no consistent reduction in the total number of free parameters (trainable weights) in the network, and often an increase for very small values of λ. It appears that any reduction in connections due to the use of sibling units is balanced by a corresponding increase in the total number of hidden units. Similarly, we observed no overall increase in the quality of generalization due to the use of the SDCC variant.

As noted in the introduction, minimizing network depth may be important in applications in which forward propagation time of the trained network is crucial, or in which interpretability of the function of the hidden units is desired. SDCC can be a simple and effective method for automatically building an appropriate network topology of modest depth.

Possible extensions of this work include the automatic choice of the λ parameter and the investigation of more complicated techniques that would add units to earlier layers as well as the deepest current layer.

5 ACKNOWLEDGEMENTS

Thanks are due to Geoffrey Gordon and Dean Pomerleau for their discussions about the contents of the paper and the Cascade-Correlation learning algorithm.

6 REFERENCES

Fahlman, S. E. (1988) "Faster-Learning Variations on Back-Propagation: An Empirical Study" in *Proceedings, 1988 Connectionist Models Summer School*, D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski (eds.), Morgan Kaufmann Publishers, Los Altos CA, pp. 38-51.

Fahlman, S. E. and Lebiere, C. (1990) "The Cascade-Correlation Learning Architecture", in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky (ed.), Morgan Kaufmann Publishers, Los Altos CA, pp. 524-532.

Gorman, R. P. and Sejnowski, T. J. (1988) "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets" in *Neural Networks*, Vol. 1. p 75-89.

Hertz, J., Krogh, A, & Palmer, G. (1993) *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Company.

Hornik, K., Stinchcombe, M., and White, H. (1989) "Multilayer Feedforward Networks Are Universal Approximators", in *Neural Networks 2, 359-336.*

Phatak, D. S. and Koren, I. (1994) "Connectivity and Performance Tradeoffs in the Cascade Correlation Learning Architecture" in *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, November 1994, pages 930 - 935.

Robinson, A. J. (1989) *Dynamic Error Propagation Networks*, Ph.D. Thesis, Cambridge University Engineering Department.

Waugh, S. and Adams, A. (1994) "Connection Strategies in Cascade-Correlation" in Proceedings: The Fifth Australian Conference on Neural Networks, Brisbane, pp. 1-4.