Neural Networks, Vol. 4, pp. 67–79, 1991
Printed in the USA. All rights reserved.

0893-6080/91 $3.00 + .00

ORIGINAL CONTRIBUTION

Creating Artificial Neural Networks That Generalize

JOCELYN SIETSMA AND ROBERT J. F. DOW

Materials Research Laboratory (MRL)—DSTO

(*Received 6 September 1989; revised and accepted 25 June 1990*)

Abstract—*We develop a technique to test the hypothesis that multilayered, feed-forward networks with few units on the first hidden layer generalize better than networks with many units in the first layer. Large networks are trained to perform a classification task and the redundant units are removed ("pruning") to produce the smallest network capable of performing the task. A technique for inserting layers where pruning has introduced linear inseparability is also described. Two tests of ability to generalize are used—the ability to classify training inputs corrupted by noise and the ability to classify new patterns from each class. The hypothesis is found to be false for networks trained with noisy inputs. Pruning to the minimum number of units in the first layer produces networks which correctly classify the training set but generalize poorly compared with larger networks.*

Keywords—Neural Networks, Back-propagation, Pattern recognition, Generalization, Hidden units, Pruning.

INTRODUCTION

One of the major strengths of artificial neural networks is their ability to recognize or correctly classify patterns which have never been presented to the network before. Neural networks appear unique in their ability to extract the essential features from a training set and use them to identify new inputs. It is not known how the size or structure of a network affects this quality.

This work concerns layered, feed-forward networks learning a classification task by back-propagation. Our desire was to investigate the relationship between network structure and the ability of the network to generalize from the training set. We examined the effect of noise in the training set on network structure and generalization, and we examined the effects of network size. This second could not be done simply by training networks of different sizes, since trained networks are not necessarily making effective use of all their hidden units.

To address the question of the effective size of a network, we have been developing a technique of training a network which is known or suspected to be larger than required and then trimming off excess units to obtain the smallest network (Sietsma & Dow, 1988). This is slower than training the "right" size network from the start, but it proves to have a number of advantages.

When we started we had some assumptions about the relationship between size and ability to generalize, which we wished to test. If a network can be trained successfully with a small number of units on the first processing layer, these units must be extracting features of the classes which can be compactly expressed by the units and interpreted by the higher layers. This compact coding might be expected to imply good generalization performance. To have many units in a layer can allow a network to become overspecific, approximating a look-up table, particularly in the extreme where the number of units in the first processing layer is equal to the number of examplars in the training set.

It has been suggested that networks with more layers, and fewer units in the early layers, may generalize better than "shallow" networks with many units in each layer (Rumelhart, 1988). However, narrow networks with many layers are far harder to train than broad networks of one or two layers. Sometimes we found that after rigorous removal of inessential units, more layers were required to perform the task. This suggested a way of producing long, narrow networks. A broad network could be trained, trimmed to the fewest possible units, and then extra layers inserted to enable the network to relearn the solution. This avoids both the training difficulties and the problem that the smallest number of units needed for a task is not generally known. We could then test

Acknowledgement: The authors wish to thank Mr. David A. Penington for his valuable contributions, both in ideas and in excellent computer programs.

Requests for reprints should be sent to J. Sietsma, USD, Material Research Laboratory, P.O. Box 50, Ascot Vale, Victoria 3032, Australia.

the hypothesis that narrow networks generalize better.

A simple task was chosen which could provide straightforward tests of the ability of the trained networks to classify new inputs. Several networks were trained using training sets containing different levels of noise, which provided us with a variety of structures of multilayered networks, whose performance on the generalization tasks could be compared.

The structure of these networks was analyzed by pruning, using two different degrees of severity. Pruning out nonvarying and parallel units showed the effective structure of the network and improved its performance. A second pruning reduced a layer to the minimum set of units able to distinguish the classes.

The results found were that adding noise to the training set greatly improves the performance of the network, and causes more of the hidden units to be used. Reducing networks to the smallest size capable of classifying the training set degraded their ability to generalize, indicating that in some circumstances networks with many hidden units generalize better than networks with few hidden units. The long, narrow networks produced by the technique described above performed poorly.

The next four sections describe our experimental set-up: they detail the classification task, the networks, the training regimes and the pruning techniques that we used, including a very brief discussion of other approaches to pruning. This is followed by the sections containing the results of the experiments. These describe the effects of pruning clean-trained networks, the effects of adding noise to the training set, and the effect of lengthening the network. We discuss the structure of some of the solution networks, give results of training networks of the same size as the pruned networks, and finally discuss some of the more general issues raised by this work.

THE TASK

The aim of this work was to discover how best to train neural networks that would generalize well from their training set. This investigation examined the questions of the effect of the size of the network, the effects of different training regimes and the effects of pruning and extending the trained network.

To explore these questions we set up a classification task which we hoped would present a complex task to an artificial neural network, be comparatively simple to analyze, and provide an obvious test of the ability of the solution network to generalize. We were not interested in (and make no assertions about) comparisons between network generalization and traditional signal processing techniques.

The training inputs were sine waves of different frequencies. Two different tests of ability to generalize were used. The first test was to correctly identify the frequency of sine waves with phase shifts different from the phases of the waves in the training set. The second test of generalization was to correctly identify input patterns from the training set which had been corrupted by random noise.

The training set consisted of 41 exemplars or input vectors in four classes. Each class contained waves of a particular frequency, covering two amplitudes (0.8 and 1.0) and five phases. The invariant in each class, which the networks were to learn and use in generalizing, was the underlying number of cycles, or frequency, of the input.

This problem is trivial using traditional signal processing techniques. It would be possible to hand-craft a solution network, using our knowledge of sine waves and Fourier transforms, so that no training was required at all. However as we were interested in learning about the properties of different structures of back-propagation networks, it would have been inappropriate to build into the network any prior knowledge about this particular problem.

The first three classes each contained 10 exemplars. Each exemplar consisted of 64 samples from a sine wave. In the first class the 64 points covered two complete periods, input patterns in the second class contained four periods and in the third class, six periods.

Class "frequency = 2" had target outputs (0, 0, 1) and inputs **x** given by

$$x_i = A\sin\left(2i\frac{2\pi}{64} + p\right),$$

where $i = 1, 2, \ldots, 64,$

$A = 0.8$ or $1.0,$

$p = 0, \frac{2\pi}{5}, \frac{4\pi}{5}, \frac{6\pi}{5}, \frac{8\pi}{5}.$

Class "frequency = 4" had target outputs (0, 1, 0) and inputs **x** given by

$$x_i = A\sin\left(4i\frac{2\pi}{64} + p\right),$$

where $i = 1, 2, \ldots, 64.$

Class "frequency = 6" had target outputs (1, 0, 0) and inputs **x** given by

$$x_i = A\sin\left(6i\frac{2\pi}{64} + p\right),$$

where $i = 1, 2, \ldots, 64.$

The fourth class was added to teach the network to make (0, 0, 0) the default output. It was intended to contain everything which was not two, four, or six cycles of a sine wave. We had noticed previously, when working with a similar but simpler problem, that small networks can adopt a strategy of a "default" class when solving multiclass problems. (This was another frequency-discrimination task, using frequencies 3, 6, and 9, with only four phases and one amplitude in each class.) Trained to separate patterns into three classes, the network will develop first layer units sensitive to the characteristics of two of the classes, and anything to which these units do not respond is put into the third class. This led to an odd result with the three-class frequency discrimination problem mentioned above (where the training exemplars did not adequately cover the phase space)—that, while most inputs with a frequency of 3 were correctly identified, they were classed as frequency 9 if they were phase shifted $3\pi/4$. The addition of this class succeeded in making the networks actively recognize all three frequencies.

The fourth class had target outputs (0, 0, 0) and contained 11 exemplars:

five straight lines—$x_i = 0$,

$x_i = 0.2$,

$x_i = -0.2$,

$x_i = i/320$

$x_i = 0.2 - i/64$

two low-frequency waves—

$$x_i = 0.2 \sin(i \frac{2\pi}{64} + p),$$

$$p = 0, \pi,$$

four high-frequency waves—

$$x_i = 0.2 \sin(fi \frac{2\pi}{64} + p),$$

$$p = 0, \pi,$$

$$f = 8, 16,$$

where $i = 1, 2, \ldots, 64$ in each case.

The input pattern consisting of a zero at each input was presented three times in each presentation of the entire training set. This meant that a presentation consisted of 43 input–target pairs, of which three were the same.

This training set allowed us to test performance under two types of variation. Novel input patterns in the classes could be generated using phases between those of the training exemplars, and random noise distortion could be added to the training set to produce a noisy test set.

INITIAL NETWORKS

The size of the starting network was chosen to be three layers (two hidden layers) with 64 inputs, 20 units (nodes, processing elements) in the first layer, eight units in the second layer and three output units. These are shown as (64)-20-8-3 networks. A similar though simpler task (frequency discrimination with four phases, one amplitude and no zero class) had a solution with only two layers (one hidden layer) but it was difficult to find the solution. As the task described in this paper was more complex it was commenced with three layers.

From previous work we expected the solution network to require three units in the first layer for each frequency class, that is, nine units in the first layer. Thus from experience it was possible that the smallest net which could perform this task would have one hidden layer of nine units. There were a number of reasons for starting with a larger network. First and most importantly, our intention was to investigate the effects of excess units and of pruning networks. Secondly, the minimum network theoretically capable of a task can be very difficult or impossible to train. Using extra units, and so adding dimensions to the error space, can remove local minima (Rumelhart, Hinton, & Williams, 1986). Finally, the (64)-9-3 size was an informed guess, and couldn't be relied upon. Three units can define a single frequency, but other solutions may be found by the training process.

TRAINING REGIMES

Three different initial random networks were generated. Initial bias weights were random values uniformly distributed between -0.5 and 0.5. The other weights were between $(-0.5/\text{fan-in})$ and $(0.5/\text{fan-in})$, where the fan-in of a unit is the number of units which are inuts to it. In these fully connected networks, the fan-in at each layer is equal to the number of units in the layer below.

The training algorithm used was back-propagation with the generalized delta rule (Rumelhart, Hinton, & Williams, 1985), with a momentum term, a decay term, learning factor inversely proportional to fan-in (Plaut, Nowlan, & Hinton, 1986) and the errors summed over the set of training exemplars before the weights are updated.

Consider a network with n_m units on layer m. The inputs to the network are treated as activations of a layer 0. The net input to unit j on layer m is

$$\text{net}_{jm} = \sum_{i=1}^{n_{m-1}} w_{jim} a_{i,m-1} + \text{bias}_{jm},$$

where w_{jim} is the weight which unit j on layer m gives to input i. The output or activation of the unit is

$$a_{jm} = f(\text{net}_{jm}) = \frac{1}{1 + e^{-\text{net}_{jm}}}.$$

The error function δ, at a unit for the input pattern–target output pair p is given by

$$\delta_{jm}^{(p)} = f'(\text{net}_{jm})(t_j^{(p)} - a_{jm}), \text{ when } m \text{ is the output layer,}$$

where $t_j^{(p)}$ is the target output for unit j for pattern p,

$$\delta_{jm}^{(p)} = f'(\mathrm{net}_{jm}) \sum_{k=1}^{n_{m+1}} w_{k,j,m+1}\delta_{k,m+1}^{(p)},$$

when m is a hidden layer.

The change in each weight is given by

$$\Delta w_{jim}(t) = \frac{\eta}{n_{m-1}} \sum_{\substack{p\in \\ \text{pattern} \\ \text{set}}} \delta_{jm}^{(p)} a_{i,m-1}^{(p)} + \alpha \Delta w_{jim}(t-1) - h w_{jim},$$

where η is the learning factor, α is the momentum coefficient, and h is the decay factor.

Three starting networks were trained using $\eta = 5$, $\alpha = 0.5$, $h = 10^{-5}$. The networks were initially trained for 2,000 presentations of the training set. Two of them (A and C) then correctly classified all the training patterns. Network B had not reached a solution and was given another 3,000 training runs, after which all of its outputs were correct.

Several researchers have reported that adding noise to the training inputs improves the performance of the network (e.g., Elman & Zipser, 1988) and we have found previously that it causes more of the available units to be used in the solution (Sietsma & Dow, 1988). As well as training with the original, or clean, training set we started from the same random networks and trained with three levels of noise distortion. Noise consisted of random values added to the points in the input vectors. Random values were generated from populations with two different standard deviations and different values were added to each pattern at each presentation.

"PRUNING" A NETWORK

Pruning is the name we have given to the process of examining a solution network, determining which units are not necessary to the solution and removing those units (Sietsma & Dow, 1988).

In this process the network is analyzed by examining the outputs of the hidden units across all the training set inputs. We identify two stages or types of pruning. Stage One is removing units that can be considered as not contributing to the solution. Stage Two is removing units that give information not required at the next layer.

(i) Noncontributing Units (Stage One)

These are units which either have approximately constant output across the training set, or have outputs across the training set which mimic the outputs of another unit. These units can be removed and the weights given to their outputs redistributed in such a way as to make almost no change to the network's performance over the training set.

If the output of a unit is approximately constant, then it is acting like an additional bias to all units to which it feeds. If the average output of unit i is a, and all its outputs (across the training set) fall within some range ($a \pm d$), where d is small, then for each unit, j, on the next layer, substitute for the bias weight

$$\mathrm{bias}_j' = \mathrm{bias}_j + w_{ji}a.$$

Unit i can then be removed.

If one unit p, gives approximately the same output as a second unit q, for all the training set, only one of these units is needed. Each unit in the next layer receives input from both p and q which contains the same information. To remove p without changing the solution, at each unit j, on the next layer,

$$w_{jq}' = w_{jq} + w_{jp}.$$

Two units can convey the same information, not by giving the same output, but by giving opposite outputs. If the output of unit p is high for any input for which the output of q is low and vice versa, then one unit is redundant. That is, $a_p \approx 1 - a_q$. (The activation or output function of the units is bounded by zero and one.) For any unit j which depends on p and q:

$$\begin{aligned}
\mathrm{net}_j &= \sum_i w_{ji}a_i + \mathrm{bias}_j \\
&= \cdots + w_{jp}a_p + \cdots + w_{jq}a_q + \cdots + \mathrm{bias}_j \\
&\approx \cdots + w_{jp}(1 - a_q) + \cdots + w_{jq}a_q + \cdots + \mathrm{bias}_j \\
&\approx \cdots + (w_{jq} - w_{jp})a_q + \cdots + (\mathrm{bias}_j + w_{jp})
\end{aligned}$$

So to remove unit p, replace w_{jq} by $w_{jq} - w_{jp}$ and $bias_j$ by $bias_j + w_{jp}$.

(ii) Unnecessary-Information Units (Stage Two)

Stage Two pruning is removing units which are independent of the other units in the layer but give information which is not required at the next layer. For example, a layer with three units, over a training set containing four patterns, may give approximate outputs:

	Unit		
Pattern	1	2	3
A	0	1	1
B	1	0	1
C	0	0	1
D	1	1	0

Assume that the next layer gives one output for patterns A and B and a different output for patterns C and D. The outputs of units 1, 2, and 3 are linearly independent, but 1 and 2 alone are sufficient to give a unique identification for each class to the next layer. With a minimum information content criterion unit 3 is unnecessary and could be removed. This example also illustrates a potential problem with this pruning. If unit 3 is removed, requiring the next layer to separate A and B from C and D is equivalent to requiring it to perform an Exclusive Or, which cannot be done with a single layer. Removing units which contribute unnecessary information to the next layer may lead to the outputs of the pruned layer being linearly inseparable with respect to the classes or outputs of the layer above.

(iii) Classes (an aside)

To clarify the concept of classes imposed by the next layer, consider the example of the usual (2)-2-1 solution to the XOR problems:

Inputs	Layer 1 Outputs	Outputs
0, 0	1, 1	0
1, 0	0, 1	1
0, 1	0, 1	1
1, 1	0, 0	0

The target outputs put the input vectors into two classes:

$$\text{Class } 0 = \{(0, 0), (1, 1)\},$$
$$\text{Class } 1 = \{(1, 0), (0, 1)\}.$$

These classes are not linearly separable. That is, there does not exist a line in the plane such that all members of class 0 lie on one side of the line and all members of class 1 lie on the other side. The first layer outputs put the input vectors into three classes:

$$A = \{(0, 0)\},$$
$$B = \{(1, 0), (0, 1)\},$$
$$C = \{(1, 1)\}.$$

These are linearly separable—one line separates A from B and C, and one line separates C from A and B. Alternatively, each unit could be considered separately. Unit 1 imposes:

$$\text{Class } 1 = \{(0, 0)\},$$
$$\text{Class } 0 = \{(0, 1), (1, 0), (1, 1)\}.$$

Unit 2 imposes:

$$\text{Class } 1 = \{(0, 0), (0, 1), (1, 0)\},$$
$$\text{Class } 0 = \{(1, 1)\}.$$

Each of these divisions independently is linearly separable. This is equivalent to considering the three classes imposed by the layer as a whole. So for this network the inputs are not separable with respect to the network targets but are separable with respect to the classes imposed by the first layer.

(iv) Lengthening a Network

Linear inseparability introduced by Stage Two pruning is the basis of a technique for developing longer networks. When a layer has been pruned, the remaining outputs are tested for linear separability with respect to the classes imposed by the next layer outputs. If the output vectors are not linearly separable, a small network is trained to take the outputs of the pruned layer as inputs and reproduce the outputs of the layer above. This small network is then inserted into the original network. This technique has enabled us to build narrow, many-layered networks from broad, shallow ones. We could then compare the performance of these networks, without trying to train many-layered networks from scratch.

(v) Pruning Tools

A number of tools were developed to assist with the two types of pruning. A computer program called FindRedund searched a file of unit outputs and, using preset limits for what should be approximated to zero or one and what should be left as ambiguous, determined which units did not vary, which units were the same and which units were inverses of each other. The program also gave advice on which unit in each set it would be best to remove. FindRedund indicated which units should be removed for Stage One pruning.

For Stage Two pruning a program called MinRed was developed. This also read the file of outputs and targets and determined the maximum number of units which could be removed while still keeping the classes distinct. This could be done with respect to either the classes imposed by the next layer's outputs or the target outputs of the network.

Typically both programs were told to treat outputs of less than 0.35 as zero, outputs greater than 0.65 as one, and the rest as ambiguous. These were our initial choices for these parameters. We had intended to adjust them as necessary over the course of our investigation, but as they worked well (no networks were seriously degraded by pruning using these parameters) the initial values were retained.

A further program was developed to edit the weights of a neural network: to remove units, updating the weights according to the formulae above, to insert layers, and to merge networks.

(vi) Other Techniques for Simplifying Networks

The pruning techniques we describe in this paper are two of a variety of methods of producing simpler networks. Another possible approach to identifying inessential units would be a form of sensitivity analysis. In this approach the activation of a unit is set to zero for all training set inputs and the effect on network output assessed. This would identify units which either had activation always near zero, or whose activation was given little weight by units in the next layer.

The units with activation always near zero would also be identified by Stage One pruning. In the case of units being given small weight by the next layer, the likeliest cause would be that the unit's activation was not varying significantly, in which case it, too, would be identified by Stage One pruning. If a unit has high activation for some training inputs and low for others, but is not being used by the network, then it is not making essential distinctions and the unit will be removed by Stage Two pruning.

Sensitivity analysis would not remove a unit which had activations parallel to another unit. In general, the units which would be removed by a sensitivity analysis of the trained network are likely to be the same as, or a subset of, those that are removed by Stage One pruning, and will definitely be a subset of those removed by Stage Two pruning.

Our Stage One pruning technique examines the matrix of activations for parallel and anti-parallel units (including units parallel to the bias input). A similar but more general approach would be to test for linear dependence between all units. This is mentioned in Kung and Hwang (1988). We experimented a little with this and found that the number of units removed by the two techniques was similar, but that tests for complete linear independence using row echelon reduction were very sensitive to the strictness of the test for identity.

Another form of pruning which has been discussed, though we were not able to find any analyses in the literature, is setting small weights to zero. This is equivalent to removing connections between units, rather than removing entire units. It is not known what effect this would have on the performance of a network, but it could be a very fruitful approach to analyzing the logic of a solution network.

It has also been suggested that some form of pruning could be implemented as part of the training process. This could be done as a dynamic form of sensitivity analysis, with a strictness parameter that increased as the network approached solution, or by adding a cost, related to the size or number of connections, to the error cost function. These are completely different approaches to the ones we have described in this paper, which apply solely to trained solution networks.

TABLE 1
Midphase Test Results

Network	A	B	C
Number of Patterns Incorrectly Classified	2	3	3
SSE (midphase test)	1.5	2.4	2.7

RESULTS FOR CLEAN-TRAINED NETWORKS

The three networks trained with the clean set of input patterns were tested with fifteen input patterns, all sine waves of amplitude 0.9, at the same three frequencies (2, 4, and 6) and with phase shifts of $\pi/5$, $3\pi/5$, π, $7\pi/5$, and $9\pi/5$; that is, exactly midway between the training inputs. This is referred to as the midphase test.

Each network correctly classified some of these but failed with others. The performance of the network is given as the sum squared error (SSE), the sum of the squared differences between the outputs and the targets summed over all outputs and patterns (Table 1). As the targets are unattainable, the sum squared error is never zero, but for all solution networks it was less than 0.1 for the training set. (The maximum possible value for SSE for the midphase test approaches 45, when all 3 outputs are maximally wrong for each of the 15 input patterns.)

We tested the ability of the networks to generalize to noise-distorted inputs by presenting them with noise-distorted versions of the training set. Two levels of noise distortion were used. In noise test A a random number from a pseudo-Gaussian distribution with mean 0 and standard deviation of 0.5 was added to each point of each input pattern. This can be translated into a signal-to-noise ratio for the frequency class inputs, although S/N is not a very meaningful measure for the zero class. For input patterns with amplitude 0.8, noise with standard deviation 0.5 corresponds to S/N = 1 dB and for amplitude 1.0, S/N = 3 dB. Let us state once again that we are not looking at this as a signal processing problem. Simple preprocessing could remove almost all of this noise, but then there would be no point in adding it. As we were using this as a general example we did not wish to use any problem-specific coding or filtering. Noise test B was similar, with noise drawn from a distribution with a standard deviation of 1.0 (S/N = −3 and −5 dB for the frequency classes).

A presentation was counted as a "fail" if any of the three network outputs differed from its target value of 0.0 or 1.0 by 0.5 or more. Between 500 and 5,000 presentations of the pattern set were used in each test, depending on the performance of the network, in order to give a reasonable confidence interval on the failure rate. Only noise test A, standard deviation equals 0.5, was applied to these unpruned networks (Table 2).

TABLE 2
Noise Test Results

	Network (Clean trained, full-size)		
	A	B	C
Failure Rate, Noise Test A	17%	24%	17%

These three networks were pruned by removing nonvarying and duplicate units and retraining (Stage One pruning). There was a lot of redundancy in these networks. Network A was pruned from (64)-20-8-3 to (64)-10-4-3, network B was pruned to (64)-10-3-3 and network C to (64)-7-4-3. The pruned networks gave essentially the same performance on the mid-phase tests as the unpruned originals. The noise tests, however, showed a significant improvement (Table 3).

These networks could not be pruned to the second stage using our programs. The hidden unit outputs were not concentrated near zero and one but were scattered over the interval. The network was using small differences in outputs to discriminate between patterns, and the MinRed program, which looks for the maximum number of units which can be removed while keeping the classes distinct, was not able to organize them into classes. This probably contributes to the poor noise performance. If the response of a layer of a network is different when one of its inputs is 0.5 from when it is 0.6, then the network becomes vulnerable to distortion.

EFFECTS OF ADDING NOISE TO THE TRAINING SET

(i) Unpruned Networks

Originally the three networks were trained by several thousand presentations of the forty-one clean exemplars. To examine the effect of training with noisy inputs, we distorted the input patterns differently in each presentation by adding random values from a pseudo-Gaussian distribution to each point in the pattern. (That is, a new random number was generated for each point in each pattern at each presentation of the training set.) This avoids any danger of the network learning the characteristics of a particular set of distortions.

TABLE 3
Performance On Noise and Midphase Tests

	Network (Clean Trained, Pruned to Stage One)		
	A	B	C
SSE (midphase test)	2.0	2.0	2.7
Failure Rate, Noise Test A	12.9%	12.2%	12.0%
Failure Rate, Noise Test B	37%	42%	43%

Three different regimes of training with noise were used. These were:

(1) adding noise with a standard deviation of 0.5 throughout the training program (2,000 to 3,500 presentations),
(2) training for 5,000 presentations with a standard deviation of 1.0 and then continuing training for another 1,000[1] with a standard deviation of 0.5,
(3) lengthy training (10,000 presentations) with a standard deviation of 1.0.

All trained networks correctly identified the clean training inputs as before. However, this training greatly improved the networks' ability to classify noise-distorted patterns as is illustrated in Table 4.

Training with noise also improved the ability of the network to generalize to input patterns that were phase-shifted with respect to the training set. Only one of the nine networks trained with noise failed to identify successfully the phase-shifted patterns in that it misclassified two inputs (SSE = 1.0). The three networks trained with clean input patterns all failed to identify several of the patterns in the test set.

The other effect of learning with noise, which must contribute to the improved generalization, is that more of the units are used and contribute independently to the solution. That is, fewer units can be removed in Stage One pruning.

(ii) Pruned Networks

The nine noisy-trained networks were pruned to Stage One. Results for the standard tests are given in Table 5.

As the amount of noise used in training increased, the number of units which could be removed decreased. This process stopped short of using all the available units. In each case generalization to other phases was not harmed by pruning and noise performance was generally improved. For the networks trained with the greatest amount of noise, training regime (3), only a very small number of units could be removed and the networks declined slightly in performance after pruning. Thus units which appeared redundant were contributing to the solution. This suggests that although the units were similar they were not identical and this difference was being used constructively. For the other networks, Stage One pruning generally improved network perform-

[1]Actually set to do 5,000 presentations but 1,000 were sufficient.

TABLE 4
Tests of Noise-Trained, Full-Sized Networks

Training Regime:	1			2			3		
Starting Net:	A	B	C	A	B	C	A	B	C
SSE (Midphase Test)	.09	.002	.006	.002	.002	1.0	.004	.06	.17
Failure Rate Noise Test A	.27%	.10%	.18%	.03%	.05%	.05%	.05%	.16%	.20%
Failure Rate Noise Test B	15%	14%	17%	8.6%	11.0%	9.1%	6.0%	5.5%	5.7%

ance. The greatest improvement occurred for the networks trained with the least noise. These are the networks trained most easily and quickly.

Stage Two pruning, the removal of units which are not needed to distinguish between the classifications, was then attempted on all of these networks.

For the majority of networks the first layer could be pruned to nine units. Before we did this it was necessary to check whether the resulting (41) output vectors in the nine-dimensional activation space would be linearly separable, with respect to either the target output classes or the classes of the second layer. There is a lack of easily-implemented tests for linear separability but there is a convergence theorem for the one layer perception which states that if the two classes of vectors are linearly separable, the perceptron will converge to the separating plane (e.g., Nilsson, 1965). We did not do a strict linear separability test but attempted to train a single-layer network with back propagation to separate the output vectors. This had the advantage that if training was successful, the single-layer network could be substituted directly into the original network.

A single-layer network (three units) with nine inputs was trained to take the outputs of the pruned first layer and produce the target outputs of the network. This learned easily, so the second layer was redundant and the task could be performed by a (64)-9-3 network. This was the result of Stage Two pruning of five of the nine networks. The performance of these networks on the generalization tests is given in Table 6.

TABLE 5
Tests of Noise Trained Networks Pruned to Stage One

Networks Trained With Noise Regime (1)	A	B	C
Size of Network	(64)-14-3-3	(64)-12-3-3	(64)-12-3-3
SSE (Midphase Test)	0.04	0.001	0.02
Failure Rate, Noise Test A	0.07%	0.33%	0.06%
Failure Rate, Noise Test B	12.3%	14.2%	12.6%

Networks Trained With Noise Regime (2)	A	B	C
Size of Network	(64)-15-4-3	(64)-16-4-3	(64)-19-4-3
SSE (Midphase Test)	0.001	0.001	1.3
Failure Rate, Noise Test B	8.4%	9.9%	8.6%

Networks Trained With Noise Regime (3)	A	B	C
Size of Network	(64)-17-5-3	(64)-19-4-3	(64)-19-5-3
SSE (Midphase Test)	0.01	0.004	0.17
Failure Rate, Noise Test B	6.8%	6.9%	5.9%

HOW DOES THE (64)-9-3 NETWORK WORK?

The combination of training with noise and pruning produces a network where most hidden unit outputs (over the training set) are close to either zero or one. This means the network is comparatively simple to analyze. Units can be considered as being turned on or off by their inputs.

Each of the nine units in the first layer has a default state of either 0.9 to 1 or 0 to 0.1, depending on the sign of the bias weight. Each unit changes state for two phases of a particular frequency. There are three units for each frequency class, covering the five phases by responding to two phases each. The second layer then has only to perform a fairly simple "OR" operation: unit 1 give a high output if 1 or 2 or 3 is "on" (i.e., not in its default state), unit 2 is high if 4 or 5 or 6 is "on," etc. The analysis of how the first layer units perform these tasks is being reported elsewhere (Sietsma, 1990).

One network was pruned to (64)-8-4-3 and one (network B, training (3)) found a two-layer solution needing only eight units ((64)-8-3). This was done by having two units each respond to one phase of two frequencies (instead of two phases of one fre-

TABLE 6
Performance of (64)-9-3 Networks (Stage Two Pruned)

	Regime (1) A	Regime (1) B	Regime (1) C	Regime (2) A	Regime (3) A
SSE (Midphase Test)	.005	.004	.007	0.04	0.04
Failure Rate, Noise Test A	0.09%	0.03%	0.05%	0.14%	0.03%
Failure Rate, Noise Test B	12.7%	11.2%	12.2%	14.1%	12.1%

quency). Unit 2 turned on for one phase of two cycles and one phase of four cycles. Unit 5 turned on for the same phase of four cycles and a phase of six cycles. This meant if unit 2 was on (high output) alone, the input was six cycles; if unit 5 alone was on, the input was two cycles; but if both units were on, the input was four cycles. This was an interesting case of a neural network finding a quite unusual solution. All these networks successfully classified the midphase test set.

LONGER NETWORKS

On two occasions the network was lengthened by inserting extra layers after Stage Two pruning had introduced linear inseparability. The two networks were C trained with noise regime (2) and C trained with noise regime (3). Network C trained with noise regime (2) was initially reduced to (64)-19-4-3 by Stage One pruning. This network was not good at generalizing since two of the midphase inputs gave the wrong response (both two cycles, outputs (0, 0, 0.35) and (0, 0, 0.1). The MinRed program gave the information that we could remove eleven units from the first layer and still preserve enough information to make all of the distinctions used at the second layer. The resulting 8-vectors were not linearly separable with respect to the output classes nor with respect to the classes imposed by the next layer.

Eventually a three-layer network, with eight inputs and four outputs, was trained with the forty-one first-layer outputs to reproduce the second-layer output patterns. This network was pruned to (8)-4-4-4 and inserted into the network to replace the old second layer. This gave a solution network consisting of five layers of units: (64)-8-4-4-4-3.

This five-layer network generalized very poorly, worse than its "parent" network and also gave a high error rate when tested with noisy inputs. This is not because eight units in the first layer are incapable of covering all phases, as the two other networks with eight units correctly classified all phases.

The narrowest network we produced was by Stage Two pruning of the C net trained with SD = 1 (Regime 3). The first layer of this reduced to only six units and the network lengthened to five layers. The resulting network had structure (64)-6-4-4-3-3.

Stage One pruning in general had little effect on a network's ability to generalize, and slightly improved its classification of noisy patterns. We reduced the size of the network by up to one-half without impairing performance. Where Stage Two pruning reduced or did not change the number of layers in the network, recognition of other phases did not change. The effect on noise performance was ambiguous, usually giving a small decrease but occasionally giving a small improvement. These networks have been pruned smaller than was possible with the clean-trained networks, yet still give better performance. Where Stage Two pruning was followed by a lengthening of the network, the number of other phases correctly identified went down and noise performance declined even more (Table 7). Thus building long, narrow networks did not improve generalization.

ANALYSIS OF A FIVE-LAYER NETWORK

The final size of network C trained with noise regime (3) makes one ask how a network with only six units in the first layer can perform this classification task.

Any classification task can be performed by a three-layer network (Lippmann, 1987; Longstaff & Cross, 1987) or indeed by a two-layer network (Hornik, Stinchcombe & White, 1989; Funahashi, 1989), though the number of hidden units required may be very large, and there are no guarantees that a particular training algorithm can find the solution. This five-layer network was produced by training a three-layer network and then examining its hidden units to determine which were essential to keep the classes at each layer distinct. Another network was then trained to take the outputs of this subset of the units in the first layer and reproduce the (pruned) outputs of the second layer. This required another three-layer network. This seems to suggest that the task set by the pruned first layer was harder than the original task, which normally requires only two layers.

The first layer of the (64)-6-4-4-3-3 network does very little analysis or grouping. The 41 input patterns

TABLE 7
Tests of Noise-Trained Networks After Stage Two Pruning

Networks Trained With Noise Regime (1)	A	B	C
Size of Network	(64)-9-3	(64)-9-3	(64)-9-3
SSE (Midphase Test)	0.005	0.004	0.007
Failure Rate, Noise Test A	0.09%	0.03%	0.05%
Failure Rate, Noise Test B	12.7%	11.2%	12.2%
Networks Trained With Noise Regime (2)	**A**	**B**	**C**
Size of Network	(64)-9-3	(64)-8-4-3	(64)-8-4-4-4-3
SSE (Midphase Test)	0.04	0.002	3.13
Failure Rate, Noise Test A	0.14%	0.7%	12.3%
Failure Rate, Noise Test B	14.1%	19.4%	34%
Networks Trained With Noise Regime (3)	**A**	**B**	**C**
Size of Network	(64)-9-3	(64)-8-3	(64)-6-4-4-3-3
SSE (Midphase Test)	0.02	0.007	2.96
Failure Rate, Noise Test A	0.03%	0.24%	15.3%
Failure Rate, Noise Test B	12.1%	15.6%	36%

are reduced to 17 patterns of outputs, compared to only 10 in a nine-unit solution. All input patterns in the fourth or "default" class give the same output pattern and input patterns which differ only in amplitude in all but one case generate the same responses. Each phase, however, has a distinct output pattern at the first layer.

The first layer units are not specialists. Most units turn on for some phases of all three frequencies. For example unit 1 mainly has output one (positive bias), but is zero for phases 0 and $2\pi/5$ of two cycles, phases 0 and $8\pi/5$ of four cycles, and phases $2\pi/5$ and $4\pi/5$ of six cycles.

The more usual solution has nine units in the first layer, each sensitive to some phases of one frequency only, which gives a simple task to the second layer. This six-unit first layer has done no such simple preprocessing and the outputs are very difficult to sort out.

This network fails to classify correctly three of the test patterns; two cycles at phase $3\pi/5$, and six cycles at π and $9\pi/5$. Each of these gives output (0, 0, 0), so this has become the default output.

BEGINNING WITH SMALL NETWORKS

The original starting size of (64)-20-8-3 for the networks was a somewhat arbitrary choice, which was expected to be larger than required for solution. To test the effect of beginning with a smaller network, the random initial (64)-20-8-3 networks were pruned in the same ways as their trained descendants.

The second layer was removed from the initial networks to give three random (64)-20-3 networks. These were trained with noise regime (1). All three networks trained to solution. (Three layers were not needed.)

Units that had been found to become redundant after training were removed from the first layer of the untrained (64)-20-3 networks to produce three (64)-14-3 networks with untrained weights. A size of fourteen units was chosen as typical after Stage One pruning. Two of the three networks trained to solution. The third network, after 7,000 presentations of the training set, was stable at a nonsolution. One might assume that this was because the initial random weights of some units were too similar for the training process to separate them. However the units with these initial weights had become independent of each other when trained as part of the (64)-20-8-3 network.

The untrained (64)-20-8-3 networks were also pruned to (64)-9-3 networks by removing units which Stage Two pruning had removed from the trained networks. None of these found a solution. This implies that at least some of the "excess" units were needed to allow the networks to learn their solutions.

Ten new random (64)-9-3 networks were then trained. Here 5,000 presentations of the training set were used, with learning factor = 3/fan-in, momentum = 0.3, decay = 10^{-5} and standard deviation of noise = 0.5. Only two of the networks found a solution, both after 1,000 passes. Therefore it is not impossible for this size of network to learn or find a solution but only two out of 13 random starts did so. Although networks trained with 20 first-layer units had between 10 and 19 independent first-layer units (1 to 10 could be removed by Stage One criteria), training (64)-14-3 networks did not lead to 14 independent units. The two networks successfully trained from 14 units had one and two units which could be removed by Stage One pruning. The two nine-unit networks which learned used all the units.

In conclusion, the size of the initial network was indeed larger than required. In particular, two layers of units would have been sufficient to learn to perform this task. Interestingly, if a network has more units in a hidden layer, more units appear to be used, provided the network is trained with noise. For example, a network with 20 units in the first hidden layer developed 14 distinct units when trained with noise regime (1), whereas a network with 14 units in the first hidden layer, trained under the same regime, developed only 12 distinct units. This is not universally true, but there appears to be difficulty in using all of the available units.

ZERO/ONE NETWORKS

One of the effects of training with noisy data, followed by pruning, is that the outputs of the hidden units tend more to zero and one than in the clean-trained, full-sized networks. This is because of the training process and because pruning preferentially removes units with activation close to one-half. The result is units which generally can be characterized as either on or off—near one or near zero.

This makes the solution network potentially simpler to implement in hardware, as units could be implemented with threshold functions rather than modeling the sigmoid output. However, this has some drawbacks as it is not possible to have intermediate outputs at the first layer, which later layers may successfully interpret, or may pass on as intermediate outputs, indicating the ambiguous nature of the input signal.

The greatest appeal of 0/1 or extreme units in simulation is that they are easy to analyze. If, over the training set, each unit can be characterized as on or off, examining the unit outputs can indicate how the solution works. For example, the activations of the first layer of the (64)-9-3 solution networks form overlapping clusters of a certain phase width for each frequency, regardless of amplitude, and all the zero class patterns are put in a single cluster. A further stage of analysis is to examine the weights used to form these clusters and the way in which higher layers interpret the information. Another advantage of 0/1 units is that the solutions should be more stable. A network where the difference between classes (over the training set) corresponds to differences in hidden outputs close to the maximum difference possible is less likely to be confused by noise, intermediate inputs or degradation of the weights than a network which is using hidden outputs at intermediate values to discriminate between the classes.

DISCUSSION OF NOISE TRAINING, PRUNING AND LONG NETWORKS

Adding noise to the training set creates more stable, robust networks which use more of the available units. Networks rarely use all units independently. Training with noise has the surprising effect of improving the ability of the network to correctly classify phases which were not in the training set, as well as improving the classification of noise corrupted inputs. Thus adding one type of perturbation to the training set aids with generalizing over other variations. Adding noise to the training set would be useful even when the application of the network is not going to involve noisy signals.

Stage One pruning, the removal of constant output and duplicate units, considerably improves the noise performance of clean-trained networks. The effect is less clear on noise-trained networks, giving a small improvement after training regimes (1) and (2) and no change or a small degradation after training regime (3). This suggests that in clean-trained networks, constant output (i.e., unused) units and duplicate units propagate noise further up the network, leading to a decrease in signal-to-noise ratio at higher levels of the network and poorer performance. Networks which have been trained with large amounts of noise are able to minimize this effect.

Stage Two pruning, the removal of unique units that are not essential for class discrimination, led in general to poorer performance. However, the performance of Stage Two pruned, noise-trained networks was still better than that of the clean-trained networks, even though the networks were smaller. Thus Stage Two pruning could be useful in circumstances where the smallest possible network was desired and robust performance was not important. Generalization to the other phases was not usually affected by either stage of pruning. The exception is the two cases where Stage Two pruning led to linear inseparability and a longer network. These five-layer networks also had the worst noise performance, but even when Stage Two pruning reduced the number of layers, performance usually declined. This effect increased with the amount of noise used in training.

Do Large Networks Follow Insignificant Detail?

If networks used extra units (above the minimum needed to perform the task) to become overspecific, this would imply that networks with many units on the first hidden layer should yield poorer generalization to new phases than small networks. Overspecificity means that units would become accurately attuned to the training set exemplars and the network would only respond correctly to inputs closely matching them in frequency, phase, and amplitude. In the extreme there could be five units for each frequency class (plus some for the zero class), each sensitive to exactly one training phase. Each of these hypothetical units has a unique activation pattern, so none would be removed by Stage One pruning. In this case adding noise to the training set would blur the phase of the inputs, which may somehow enable the network to "realize" that phase was not a relevant feature. This is the process assumed by Elman and Zipser (1988), among others. Therefore, noisy training would lead to fewer units being used, because if phase is not important only three units are required to identify a particular frequency. If noisy training prevented overspecificity and enabled a network to ignore irrelevant features, noisy-trained networks would have more units removable by Stage One pruning than clean-trained networks. Stage Two pruning would lead to better generalization still, as idiosyncratic units which are not essential in distinguishing the classes must be responding to irrelevant features of the inputs.

In fact, precisely the opposite occurs. Training with noise led to networks which used more units, each with their own idiosyncrasies, and which classified new inputs more reliably than clean-trained networks. Therefore adding noise did not prevent units from becoming overspecific, but caused them to become usefully specific. Stage Two pruning, which removes unique units which are not essential to the classification, impaired the ability of these networks to generalize.

The idea of excess units becoming overspecific is not convincing when one considers movement in the error space. Consider a two-dimensional feature space (Figure 1) with a connected, closed region as class A and the rest of the plane as class B. The training points or examplars of class A show a shape with considerable fine structure. If there is a blank area between the training inputs from A and B (Figure 1(a)), gradient descent in error space is not going to form decision boundaries which draw a dot-to-dot around the exact perimeter of A. Once the classes are separated, any fine-tuning is done at the output layer. If the training points from B fit close to A (Figure 1(b)) like a jigsaw puzzle, the details become significant.

CONCLUSION

The aim of this work was to investigate techniques for creating artificial neural networks that would generalize well from their training sets. In particular, we wished to test the idea that using fewer first-layer units would encourage networks to generalize better.

From this work we conclude that to obtain the neural networks which generalize best, networks larger than the minimum required to perform the task should be trained with noise distorted inputs. If a small network is desired, removing useless and repetitive units (Stage One pruning) has no ill effects and can have beneficial effects on the ability of the network to generalize. If for some reason the network cannot be trained with noise, Stage One pruning of clean-trained networks considerably improves noise immunity as well as giving a smaller network.

Training with random noise dramatically improves a network's ability to recognize noisy inputs. The failure rates for the full-sized, clean-trained networks of 17 to 24% were reduced by noisy training to less than 0.5%. Training with noise also improves the ability of a network to recognize representatives of the classes which were not in the training set. Of the nine noisy-trained networks, eight successfully classified all phases, where the three clean-trained networks each misclassified two or three inputs of other phases.

Networks were pruned from their starting size of (64)-20-8-3 to a minimum of (64)-8-3, which is a re-

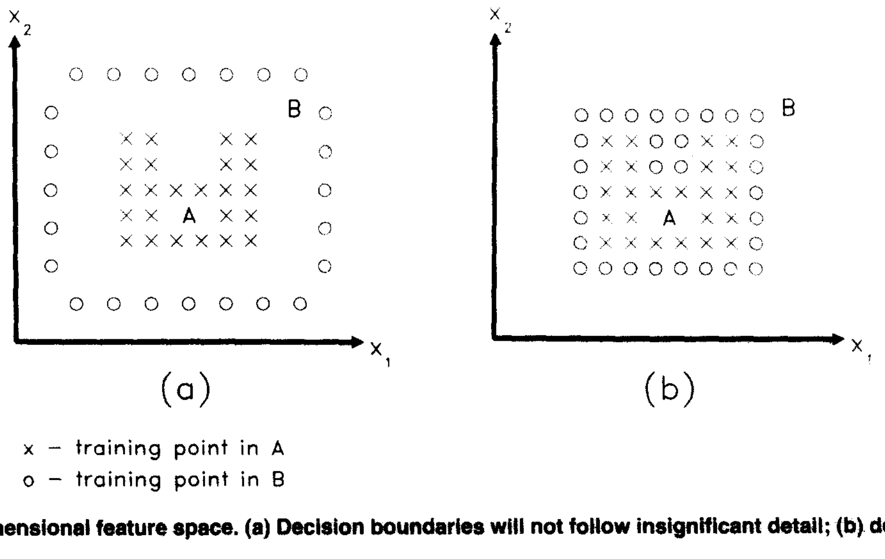

FIGURE 1. Two-dimensional feature space. (a) Decision boundaries will not follow insignificant detail; (b) detail is significant.

duction in size by approximately two thirds. This corresponds to a processing speed increase of roughly three times. Noise immunity was impaired but, for all pruned networks which did not require lengthening, recognition of other phases was not impaired. This minimum information content pruning (Stage Two pruning) would be appropriate where a small, fast network is required and recognition of noisy patterns is not so important. Performance on clean signals is unimpaired. Stage Two pruning of networks trained with random noise produced networks which were comparable in size to Stage One pruned, clean-trained networks but which generalized considerably better. The networks that were pruned to the narrowest base, and which required the insertion of additional layers, performed far worse than other noise-trained networks, although not badly compared with the clean-trained networks.

Fewer units led to better generalization for our networks trained with clean input patterns. For the networks trained with noise-distorted, varying patterns, more units were used and removing apparently unused units had less effect. The two narrow, five-layer networks generalized far worse than their "parent" networks, indicating that fewer first-layer units, followed by more processing stages, does not lead to better generalization.

Analysis of the solution networks was made possible by the fact that, for the training set, hidden units usually had activations close to either zero or one. This is a consequence of training with noisy data followed by pruning redundant units. Examining the pattern of units turned on or off by the inputs shows how the network has clustered the input set. This method of dissecting a solution network has been reported elsewhere (Sietsma, 1990) and is a subject of our continuing research.

REFERENCES

Elman, J. L., & Zipser, D. (1988). Learning the hidden structure of speech. *Journal of the Acoustical Society of America*, **83**(4), 1615–1626.

Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, **2**(3), 183–192.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**(5), 359–366.

Kung, S. Y., & Hwang, J. N. (1988). An algebraic projection technique for optimal hidden units size and learning rates in back-propagation networks. *Proceedings of the IEEE Conference on Neural Networks, San Diego 1988* (Vol. 1, pp. 363–370).

Lippmann, R. (1987). An introduction to computing with neural nets. *IEEE Acoustics, Speech and Signal Processing Magazine*, **4**, 4–22.

Longstaff, I. D., & Cross, J. F. (1987). A pattern recognition approach to understanding the multi-layer perceptron. *Pattern Recognition Letters*, **5**, 315–319.

Nilsson, Nils (1965). *Learning machines: foundations of trainable pattern-classifying systems*. New York: McGraw-Hill.

Plaut, D. C., Nowlan, S. J., & Hinton, G. E. (1986). *Experiments on learning by back-propagation* (CMU-CS-86-126). Pittsburgh, PA: Carnegie-Mellon University.

Rumelhart, D. E. (1988, July). *Parallel distributed processing*. Plenary Session, IEEE International Conference on Neural Networks 1988, San Diego, CA.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland and the PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition: Vol. 1, Foundations* (pp. 318–362). Cambridge, MA: Bradford Books/MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, **323**, 533–536.

Sietsma, Jocelyn (1990). The Effect of Pruning a Back-Propagation Network. *First Australian Conference on Neural Networks (ACNN '90)*, Sydney, Australia, 1990.

Sietsma, Jocelyn, & Dow, R. J. F. (1988). Neural net pruning—Why and how. *Proceedings of the IEEE International Conference on Neural Networks 1988, Vol. 1* (pp. 325–333).