Connection Science
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713411269

Using Relevance to Reduce Network Size Automatically

Michael C. Mozer [a]; Paul Smolensky [a]
[a] Department of Computer Science &, Institute of Cognitive Science, University of Colorado, Boulder, CO, USA.

Online Publication Date: 01 January 1989
To cite this Article: Mozer, Michael C. and Smolensky, Paul (1989) 'Using Relevance to Reduce Network Size Automatically', Connection Science, 1:1, 3 - 16
To link to this article: DOI: 10.1080/09540098908915626
URL: http://dx.doi.org/10.1080/09540098908915626

Using Relevance to Reduce Network Size Automatically

MICHAEL C. MOZER & PAUL SMOLENSKY

This paper proposes a means of using the knowledge in a network to determine the functionality or relevance *of individual units, both for the purpose of understanding the network's behavior and improving its performance. The basic idea is to iteratively train the network to a certain performance criterion, compute a measure of relevance that identifies which input or hidden units are most critical to performance, and automatically remove the least relevant units. This* skeletonization *technique can be used to simplify networks by eliminating units that convey redundant information; to improve learning performance by first learning with spare hidden units and then removing the unnecessary ones, thereby constraining generalization; and to understand the behavior of networks in terms of minimal 'rules'.*

One thing that connectionist networks have in common with brains is that if you open them up and peer inside, all you can see is a big pile of goo. Internal organization is obscured by the sheer number of units and connections. Although techniques such as hierarchical cluster analysis (Sejnowski & Rosenberg, 1987) have been suggested as a step in understanding network behavior, one would like a better handle on the role that individual units play. This paper proposes one means of using the knowledge in a network to determine the functionality or *relevance* of individual units. Given a measure of relevance for each unit, the least relevant units can be automatically removed from the network to construct a *skeleton* version of the network.

Skeleton networks have several potential applications:

- *Constraining generalization.* By eliminating input and hidden units that serve no purpose, the number of parameters in the network is reduced and generalization will be constrained (and hopefully improved).
- *Speeding up learning.* Learning is fast with many hidden units, but a large number of hidden units allows for many possible generalizations. Learning is slower with few hidden units, but generalization tends to be better. One idea for speeding up learning is to train a network with many hidden units and then

Correspondence to Michael C. Mozer, Department of Computer Science & Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0430, USA. Our thanks to Colleen Seifert for conversations that led to this work; to Dave Goldberg, Geoff Hinton, and Yann le Cun for their feedback; and to Eric Jorgensen for saving us from computer hell. This work was supported by grant 87-2-36 from the Sloan Foundation to Geoffrey Hinton, a grant from the James S. McDonnell Foundation to Michael Mozer, and a Sloan Foundation grant and NSF grants IRI-8609599, ECE-8617947, and CDR-8622236 to Paul Smolensky. An abridged version of this paper appeared as Mozer & Smolensky (1989).

eliminate the irrelevant ones. This may lead to a rapid learning of the training set and then, gradually, an improvement in generalization performance.[1]

- *Understanding of the behavior of a network in terms of 'rules'.* One often wishes to get a handle on the behavior of a network by analyzing the network in terms of a small number of rules instead of an enormous number of parameters. In such situations, one may prefer a simple network that performed correctly on 95% of the cases over a complex network that performed correctly on 100%. The skeletonization process can discover such a simplified network.

Several researchers (Chauvin, 1989; Hanson & Pratt, 1989; David Rumelhart, personal communication, 1988) have studied techniques for the closely related problem of reducing the number of free parameters in back propagation networks (Rumelhart *et al.*, 1986). Their approach involves adding extra cost terms to the usual error function that cause nonessential weights and units to decay away. We have opted for a different approach—the all-or-none removal of units—which is not a gradient descent procedure. The motivation for our approach was twofold. First, our initial interest was in designing a procedure that could serve to focus 'attention' on the most important units, hence an explicit relevance metric was needed. Second, our impression is that it is a tricky matter to balance a primary and secondary error term against one another. One must determine the relative weighting of these terms, weightings that may have to be adjusted over the course of learning. In our experience, it is often impossible to avoid local minima—compromise solutions that partially satisfy each of the error terms. This conclusion is supported by the experiments of Hanson & Pratt (1989).

Another class of techniques related to skeletonization is based on principal component analysis of hidden unit activities or weights (Elman, 1989). Sanger (1989) applies principal component analysis to vectors what he calls *contributions*—the activity of a given hidden unit times its weight to a given output unit for a given training input—and shows that this can be effectively used to understand what work is being done by various patterns of hidden units. Principal component analysis, like cluster analysis, identifies important *patterns* in the network; while it may be possible to develop methods for reducing the size of a network based on this pattern information, these methods would undoubtedly be less straightforward than the method we study here: simply removing individual units that are determined to be less relevant for good performance than others.

Determining the Relevance of a Unit

Consider a multi-layer feedforward network. How might we determine whether a given unit serves an important function in the network? One obvious source of information is its outgoing connections. If a unit layer l has many large-weighted connections, then one might expect its activity to have a big impact on higher layers. However, this need not be. The effects of these connections may cancel each other out; even a large input to units in layer $l+1$ will have little influence if these units are near saturation; outgoing connections from the innervated units in $l+1$ may be small; and the unit in l may have a more-or-less constant activity, in which case it could be replaced by a bias on units in $l+1$. Thus, a more accurate measure of the relevance of a unit is needed.

What one really wants to know is, what will happen to the performance of the network when a unit is removed? That is, how well does the network do *with* the unit

versus *without* it? For unit i, then, a straightforward measure of the relevance, ρ_i, is

$$\rho_i = E_{\text{without unit } i} - E_{\text{with unit } i},$$

where E is the error of the network on the training set. The problem with this measure is that to compute the error with a given unit removed, a complete pass must be made through the training set. Thus, the cost of computing ρ is $O(np)$ stimulus presentations, where n is the number of units in the network and p is the number of patterns in the training set. Further, if the training set is not fixed or is not known to the experimenter, additional difficulties arise in computing ρ.

We therefore set out to find a computationally cheaper alternative to ρ. Before presenting this alternative, it is first necessary to introduce an additional bit of notation. Suppose that associated with each unit i is a coefficient α_i which represents the *attentional strength* of the unit (see Figure 1). This coefficient gates the flow of activity from the unit:

$$a_j = f\left(\sum_i w_{ji} o_i\right)$$

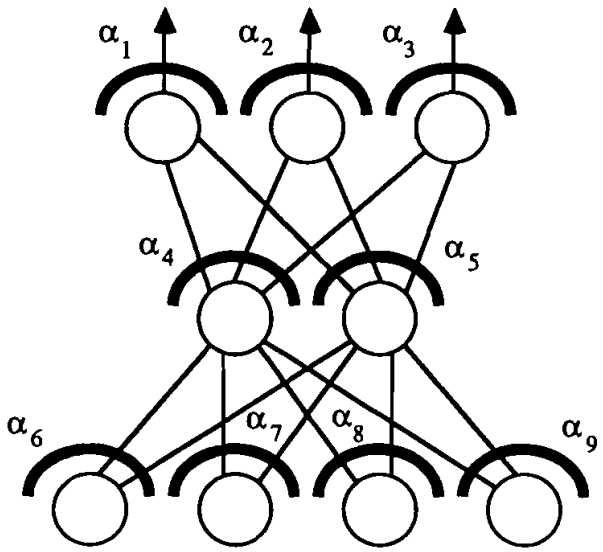

Figure 1. A 4–2–3 network with attentional coefficients on all units.

and

$$o_j = \alpha_j a_j,$$

where a_j is the activity of unit j, o_j is its output as observed by the rest of the network (for input and hidden units) or by the teacher (for output units), w_{ji} is the connection strength to j from i, and f is the sigmoid squashing function. If $\alpha_i = 0$, unit i produces the constant output 0 and thus has no influence on other units; if $\alpha_i = 1$, unit i behaves like a conventional unit. In terms of α_i, the relevance of unit i can then be rewritten as

$$\rho_i = E_{\alpha_i=0} - E_{\alpha_i=1}.$$

The simpler alternative we have adopted for ρ_i, which we term $\hat{\rho}_i$, is just the negative of the derivative of the error function with respect to α_i, evaluated at the point where all $\alpha_j = 1$:

$$\hat{\rho}_i = -\left.\frac{\partial E}{\partial \alpha_i}\right|_{(\alpha_1, \ldots, \alpha_n)=(1, \ldots, 1)}$$

This definition is derived from the simple fact that

$$\rho_i = E_{\alpha_i=0} - E_{\alpha_i=1} = -\int_0^1 \frac{\partial E}{\partial \alpha_i} d\alpha_i.$$

Thus, ρ_i is the area under the $-\partial E/\partial \alpha_i$ curve between $\alpha_i=0$ and $\alpha_i=1$; $\hat{\rho}_i$ is the derivative evaluated at $\alpha_i=1$—the rightmost value of the curve.

To motivate the use of $\hat{\rho}_i$ instead of ρ_i, consider the simple task of estimating the relevance of two output units based on a single training pattern. Assume that both units have a target activity level of 1 for this pattern, and the activity level of unit 1 is 0.2 and unit 2 is 0.5. Using the usual quadratic error function, unit 1 should be judged as less relevant than unit 2 because the removal of unit 1 will affect the error less than the removal of unit 2.

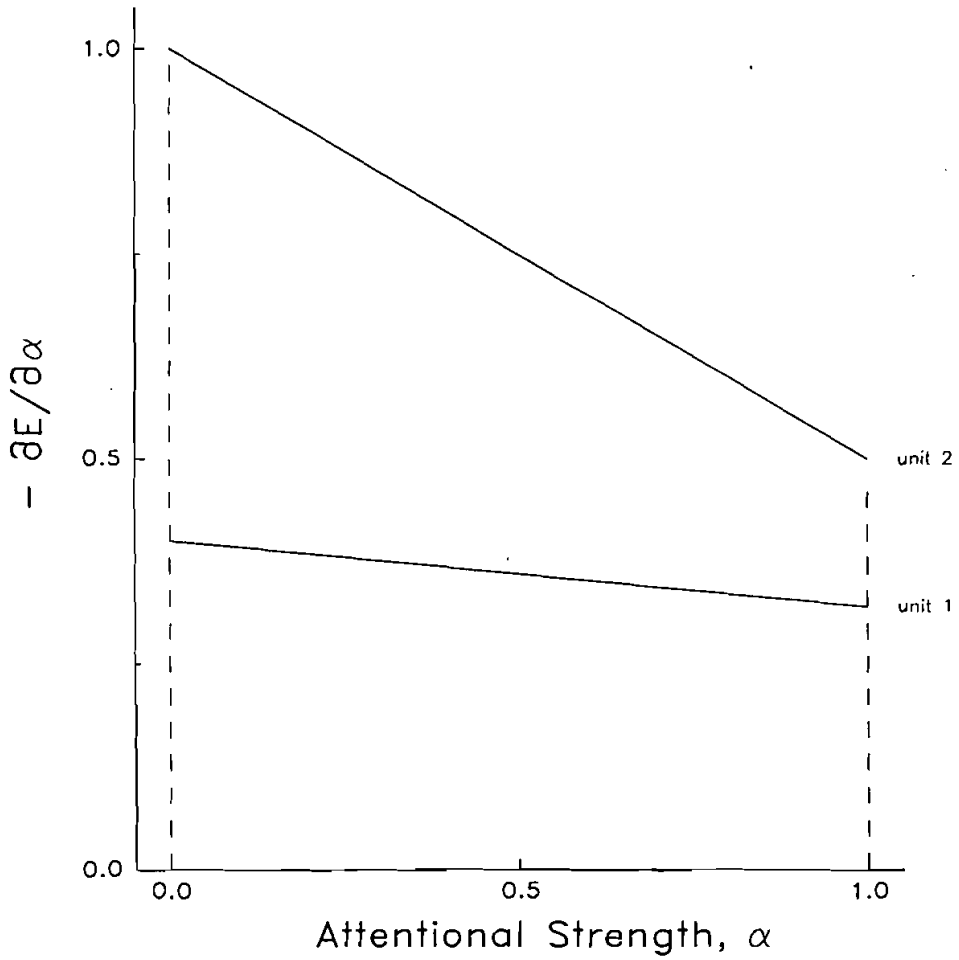

Figure 2. The derivative of the error function with respect to α_i for two output units plotted against α_i. The target activity level of both units is assumed to be 1, and the actual activity level of unit 1 is 0.2 and unit 2 is 0.5. ρ_i is the area under the $-\partial E/\partial \alpha_i$ curve, and $\hat{\rho}_i$ is the value of the curve at $\alpha_i=1$—the rightmost point.

Figure 2 shows the derivative of the error function with respect to α_i, $-\partial E/\partial\alpha_i$, for each output unit. Clearly, ρ_i—the area under each curve—is different than $\hat{\rho}_i$—the rightmost value of each curve. To what extent, then, can $\hat{\rho}_i$ serve in place of ρ_i? Because our goal is to decide which unit is least relevant so that we can eliminate it from the network, what matters is whether comparing $\hat{\rho}_1$ and $\hat{\rho}_2$ gives the same result as comparing ρ_1 and ρ_2. It need not in principle, but in practice it serves us well. We can appreciate why from Figure 2. If the height of the $-\partial E/\partial\alpha$ curve at $\alpha=1$ for unit 2, $\hat{\rho}_2$, is significantly greater than that for unit 1, $\hat{\rho}_1$, it will follow that the area under the curve for unit 2, ρ_2, is greater than that for unit 1, ρ_1, unless the curve for unit 1 not only crosses over that for unit 2 between $\alpha=1$ and $\alpha=0$, but does so quite dramatically. As Figure 2 shows, the relation $\rho_2>\rho_1$ is correctly predicted by the relation $\hat{\rho}_2>\hat{\rho}_1$. While the conditions under which the $\hat{\rho}_i$ have the same ordering as the ρ_i can be worked out analytically for this simple case, the general case of multiple patterns and hidden units is difficult to analyze. Fundamentally, the justification for our method is its empirical success, as we report below.

The derivative defining $\hat{\rho}_i$ can be computed using an error propagation procedure very similar to that used in adjusting the weights with the back propagation algorithm. Additionally, note that because the definition of $\hat{\rho}_i$ assumes that all α_j are 1, the α_j never need be changed. Thus, the α_j are not actual parameters of the system, just a bit of notational convenience used in estimating relevance.

In practice, we have found that $\partial E/\partial\alpha_i$ fluctuates strongly in time and a more stable measure that yields better results is an exponentially decaying time average of the derivative. In the simulations reported below, we use:

$$\hat{\rho}_i(t+1)=0.8\hat{\rho}_i(t)-0.2\frac{\partial E(t)}{\partial\alpha_i}.$$

Relevance Computation is Based on a Linear Error Function

One problem with the proposed relevance measure is that the quadratic error function ordinarily used in back propagation networks is not suitable for estimating relevance. This error function,

$$E^q=\sum_p\sum_j(t_{pj}-o_{pj})^2$$

(where p is an index over patterns, j over output units; t_{pj} is the target output, o_{pj} the actual output), has the serious drawback that its derivative goes to zero as the total error decreases, and consequently, if $\hat{\rho}_i=-\partial E^q/\partial\alpha_i$, the relevance of all units will tend to zero as the error decreases. This certainly violates intuitions as to how the relevance measure should behave.

To see the problem more clearly, consider again the situation of estimating the relevance of an output whose target activity level is 1. Using the quadratic error function,

$$\hat{\rho}^q=-\left.\frac{\partial E^q}{\partial\alpha}\right|_{\alpha=1}=2o(1-o).$$

As the unit's output, o, approaches the target value of 1, the relevance goes to zero. Thus, $\hat{\rho}^q$ grossly underestimates the relevance of outputs that are close to the target. Figure 3 depicts the situation graphically. The solid curve is the 'true' relevance, ρ, as a function of the activity of the output unit. Note that if the activity is close to zero, so is the relevance because suppressing the unit by setting α to zero will not alter its

output much; if the activity is close to 1, setting α to zero will have maximal effect. The short-dashed curve is $\hat{\rho}^q$, rescaled by a multiplicative constant to best fit the ρ curve. This measure is reasonably good only when the activity of the unit is less than 0.5. As the activity of the output unit approaches 1, $\hat{\rho}^q$ decreases whereas ρ increases.[2]

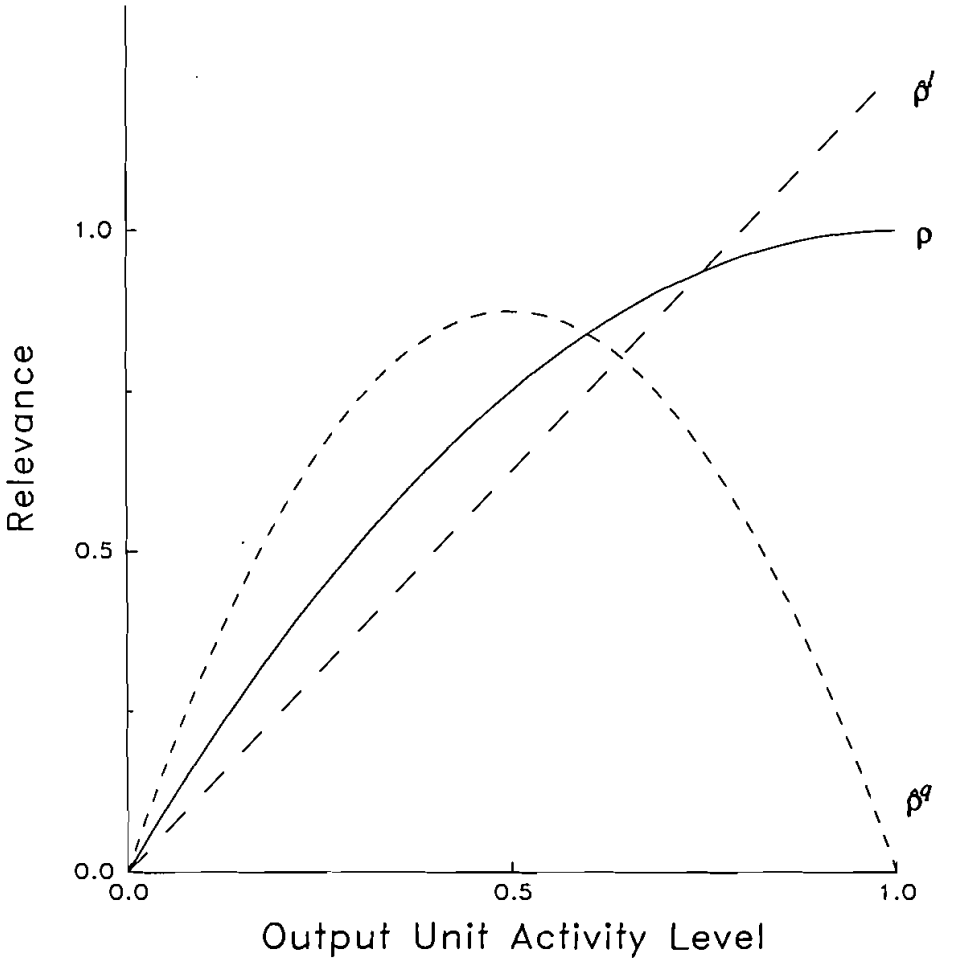

Figure 3. Three measures of the relevance of an output unit whose target activity is 1, as a function of the unit's activity level. The solid curve labeled ρ is the true relevance, $E^q_{\alpha=0} - E^q_{\alpha=1}$; the short-dashed curve is the measure using the derivative of E^q; and the long-dashed line is the measure using the derivative of E^l. The two derivative measures have been rescaled by a multiplicative constant to best fit the ρ curve.

One possibility is to make use of $\hat{\rho}^q$ only for ranges of output unit activities where the measure is good. In practice, this turns out to be quite difficult. Instead, we have opted for a different solution: to use a *linear* error function,

$$E^l = \sum_p \sum_j |t_{pj} - o_{pj}|$$

in computing the relevance. The derivative of E^l does not depend on how close the output unit activity is to the target, and thus does not go to zero as the error decreases. Figure 3 shows an estimate of relevance, $\hat{\rho}^l$, computed from the derivative of E^l.

Unlike $\hat{\rho}^q$, $\hat{\rho}^l$ is monotonically increasing, which is particularly important because the skeletonization procedure we will present uses $\hat{\rho}$ to rank order the units by relevance.

To summarize, in the results reported below, while E^q is used as the error metric in training the weights via conventional back propagation, $\hat{\rho}$ is measured using E^l. This involves separate back propagation phases for computing the weight updates and the relevance measures.

In the next two sections, we discuss sample applications of relevance assessment. We then discuss how to use relevance assessment for the construction of skeleton networks.

A Simple Example: The Cue Salience Problem

Consider a network with four inputs labeled A–D, one hidden unit, and one output. We generated ten training patterns such that the correlations between each input unit and the output are as shown in the first row of Table I. (In this particular task, a hidden layer is not necessary. The inclusion of the hidden unit does not affect any of the reported results, and in fact increases the difficulty of relevance assessment. It simply allowed us to use a standard three-layer architecture for all tasks.)

Table I

	Input unit			
	A	B	C	D
Correlation with output unit	1.0	0.6	0.2	0.0
Input-hidden connection strengths	3.15	1.23	0.83	−0.01
ρ_i	5.36	0.07	0.06	0.00
$\hat{\rho}_i$	0.46	−0.03	0.01	−0.02

In this and subsequent simulations, unit activities range from −1 to 1, input and target output patterns are binary (−1 or 1) vectors. Training continues until all output activities are within some acceptable range of the target value. Additional details of the training procedure and network parameters are described in the Appendix.

To perform perfectly, the network need only attend to input A. This is not what the input-hidden connections do, however; their weights have the same qualitative profile as the correlations (second row of Table I).[3] In contrast, the relevance values for the input units show A to be highly relevant while B–D have negligible relevance. Additionally, the qualitative picture presented by the profile of $\hat{\rho}_i$ is identical to that of the ρ_is. Thus, while the weights merely reflect the statistics of the training set, $\hat{\rho}_i$ indicates the *functionality* of the units.

The Rule-Plus-Exception Problem

Consider a network with four binary inputs labeled A–D and one binary output. The task is to learn the function $AB+\overline{ABCD}$: the output unit should be on whenever both A and B are on, or in the special case that all inputs are off. With two hidden units, back propagation arrives at a solution in which one unit responds to AB—the rule—and the other to $\overline{ABCD}$—the exception (Figure 4). Clearly, the AB unit is more relevant to the solution; it accounts for 15 cases whereas the $\overline{ABCD}$ unit accounts for only one. This

fact is reflected in the $\hat{\rho}_i$: in 100 replications of the simulation, the mean value of $\hat{\rho}_{AB}$ was 1.49 whereas $\hat{\rho}_{\overline{ABCD}}$ was only 0.17. These values are extremely reliable; the standard errors are 0.003 and 0.005, respectively.

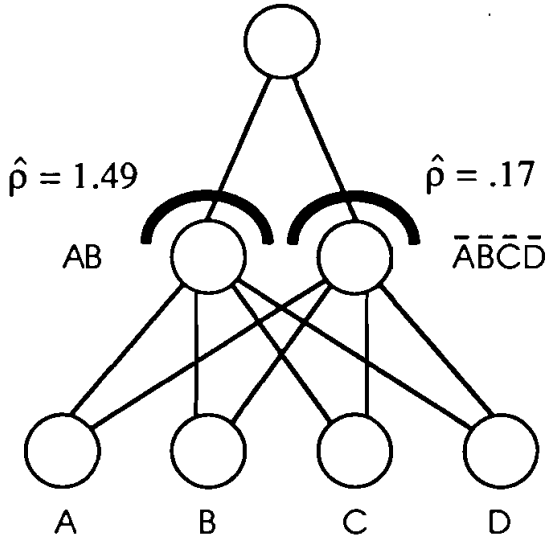

Figure 4. A 4–2–1 network used to learn the rule-plus-exception problem. One hidden unit learns to respond to AB—the rule—and the other to $\overline{ABCD}$—the exception.

Relevance was also measured using the quadratic error function. With this metric, the AB unit is incorrectly judged as being *less* relevant than the $\overline{ABCD}$ unit: $\hat{\rho}^q_{AB}$ is 0.029 and $\hat{\rho}^q_{\overline{ABCD}}$ is 0.033. As explained above, the basis of the failure of the quadratic error function is that $\hat{\rho}^q$ grossly underestimates the true relevance as the output error goes to zero. Because the one exception pattern is invariably the last to be learned, the output error for the 15 non-exception patterns is significanly lower, and consequently, the relevance values derived from the non-exception patterns are much smaller than that derived from the one exception pattern. This results in the relevance assessment based on the exception pattern dominating the overall relevance measure, and in the incorrect relevance assignments described above. However, this problem can be avoided by assessing relevance using the linear error function.

If we attempted to 'trim' the rule-plus-exception network by eliminating hidden units, the logical first candidate would be the less relevant $\overline{ABCD}$ unit. This trimming process would leave us with a simpler network—a skeleton network—whose behavior is easily characterized in terms of a simple rule, but which could only account for 15 of the 16 input cases.

Constructing Skeleton Networks

In the remaining examples we construct skeleton networks using the relevance metric $\hat{\rho}$. The procedure is as follows: (1) train the network until all output unit activities are within some specified range of the target value (see Appendix); (2) compute $\hat{\rho}$ for each unit; (3) remove the unit with the smallest $\hat{\rho}$; and (4) repeat steps 1–3 a specified number of times. In the examples below, we have chosen to trim away either the input

units or the hidden units, not both simultaneously, but there is no reason why this could not be done.

We have not yet addressed the crucial question of how much to trim away from the network. At present, we specify in advance when to stop trimming. However, the procedure described above makes use only of the ordinal values of the $\hat{\rho}$. One untapped source of information that may be quite informative is the magnitudes of the $\hat{\rho}$. A large increase in the minimum $\hat{\rho}$ value as trimming progresses may indicate that further trimmings will seriously disrupt performance in the network.

The Train Problem

Consider the task of determining a rule that discriminates the 'east' trains from the 'west' trains in Figure 5. There are two simple rules—simple in the sense that the rules require a minimal number of input features: east trains have *a long car and triangle load in car* or *an open car or white wheels on car*. Thus, of the seven features that describe each train, only two are essential for making the east/west discrimination.

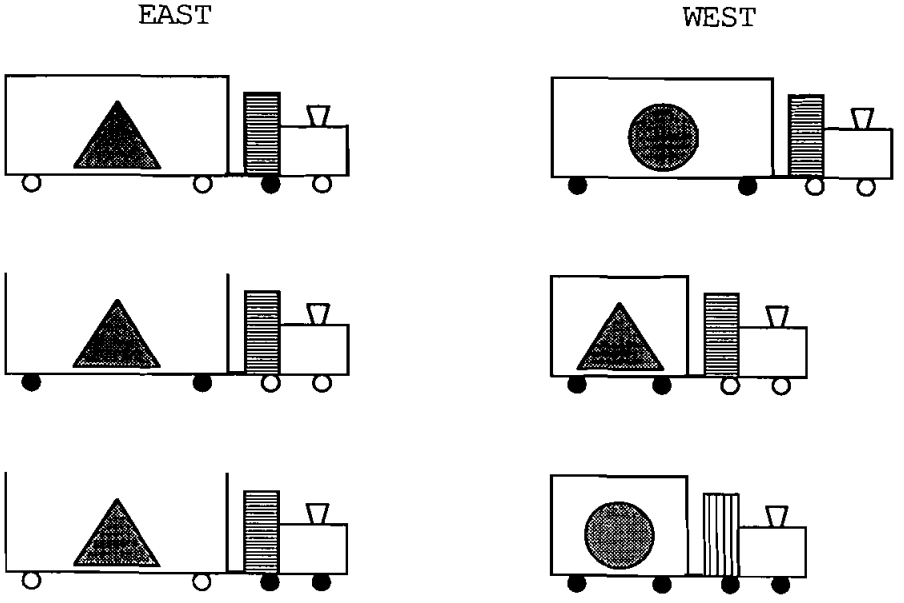

Figure 5. The train problem. Adapted from Medin *et al.* (1987).

A 7–1–1 network trained on this task using back propagation learns quickly, but the final solution takes into account nearly all the inputs because 6 of the 7 features are partially correlated with the east/west discrimination. When the skeletonization procedure is applied to trim the network from 7 inputs to 2, however, the minimal set of input features is discovered—either *long car* and *triangle load*, or *open car* and *white wheels on car*—on each of 100 replications of the simulation we ran.

The trimming task is far from trivial. The expected success rate with random removal of the inputs is only 9.5%. Other skeletonization procedures we experimented with resulted in success rates of 50–90%.

The Four-bit Multiplexor Problem

Consider a network that learns to behave as a four-bit multiplexor. The task is, given 6 binary inputs labeled A–D, M_1, and M_2, and one binary output, to map one of the inputs A–D to the output contingent on the values of M_1 and M_2. The logical function being computed is $\overline{M_1}\overline{M_2}A+\overline{M_1}M_2B+M_1\overline{M_2}C+M_1M_2D$.

A standard 4-hidden-unit back propagation network was tested against a skeletonized network that began with 8 hidden units initially and was trimmed to 4 (an *8—→4 network*). If the network did not reach the performance criterion within 1000 training epochs, we assumed that the network was stuck in a local minimum and counted the run as a failure.

Table II

Architecture	Failure (%)	Median epochs to criterion (with 8 hidden)	Median epochs to criterion (with 4 hidden)
Standard 4-hidden network	17	—	52
8—→4 network	0	25	45

Performance statistics for the two networks are shown in Table II, averaged over 100 replications. The standard network fails to reach criterion on 17% of the runs, whereas the 8—→4 network always obtains a solution with 8 hidden units and the solution is not lost as the hidden layer is trimmed to 4 units.[4] The 8—→4 network with 8 hidden units reaches criterion in about half the number of training epochs required by the standard network. From this point, hidden units are removed one at a time from the 8—→4 network, and after each cut the network is retrained to criterion. Nonetheless, the total number of epochs required to train the initial 8 hidden unit network and then trim it down to 4 is still less than that required for the standard network with 4 units. Furthermore, as hidden units are removed, the performance of the 8—→4 network remains close to criterion, so the improvement in learning is substantial (Figure 6).

The Random Mapping Problem

The problem here is to map a set of random 20-element input vectors to random 2-element output vectors. Twenty random input–output pairs were used as the training set. Ten such training sets were generated and tested. A standard 2-hidden unit network was tested against a 6—→2 network. For each training set and architecture, 100 replications of the simulation were run. If criterion was not reached within 1000 training epochs, we assumed that the network was stuck in a local minimum and counted the run as a failure.[5]

As Table III shows, the standard network failed to reach criterion with two hidden units on 17% of all runs, whereas the 6—→2 network failed with the hidden layer trimmed to two units on only 8.3% of runs. In 9 of the 10 training sets, the failure rate of the 6—→2 network was lower than that of the standard network. Both networks required comparable amounts of training to reach criterion with two hidden units, but the 6—→2 network reaches criterion much sooner with six hidden units, and its

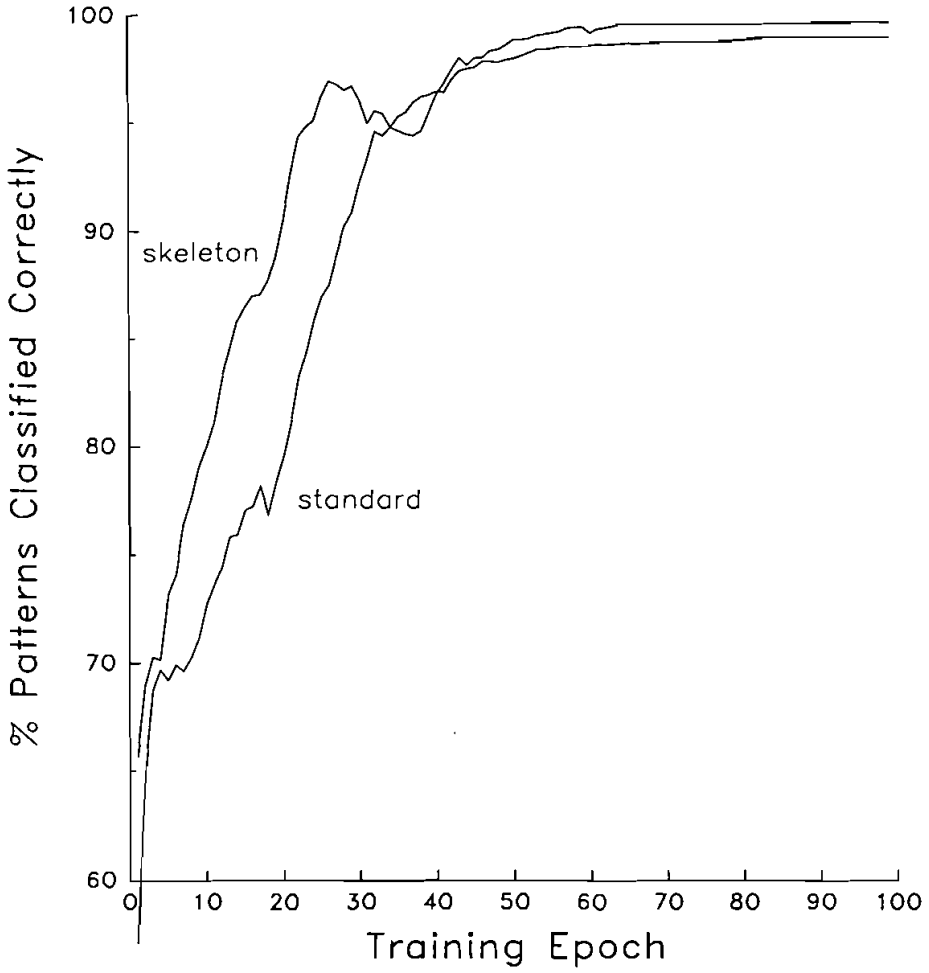

Figure 6. Average performance on the four-bit multiplexor task.

Table III

	Standard network		6—2 network		
Training set	Failures (%)	Median epochs to criterion (2 hidden)	Failures (%)	Median epochs to criterion (6 hidden)	Median epochs to criterion (2 hidden)
1	14	20	7	11	22
2	16	69	13	12	47
3	25	34	0	7	14
4	33	38	0	10	21
5	38	96	55	35	⟨max⟩
6	9	17	1	9	17
7	9	28	5	14	43
8	6	13	0	8	16
9	8	12	0	8	17
10	12	12	2	8	17

performance does not significantly decline as the network is trimmed. These results parallel those of the four-bit multiplexor.

Summary and Conclusions

We proposed a method of using the knowledge in a network to determine the relevance of individual units. The relevance metric can identify which input or hidden units are most critical to the performance of the network. The least relevant units can then be removed to constuct a skeleton version of the network.

Skeleton networks have application in two different scenarios, as our simulations demonstrated:

- Understanding the behaviour of a network in terms of 'rules'
  - —*The cue salience problem.* The relevance metric singled out the one input that was sufficient to solve the problem. The other inputs conveyed redundant information.
  - —*The rule-plus-exception problem.* The relevance metric was able to distinguish the hidden unit that was responsible for correctly handling most cases (the general rule) from the hidden unit that dealt with an exceptional case.
  - —*The train problem.* The relevance metric correctly discovered the minimal set of input features required to describe a category.
- Improving learning performance
  - —*The four-bit multiplexor.* Whereas a standard network was often unable to discover a solution, the skeleton network never failed. Further, the skeleton network learned the training set more quickly.
  - —*The random mapping problem.* As in the multiplexor problem, the skeleton network succeeded considerably more often with comparable overall learning speed, and less training was required to reach criterion initially.

Basically, the skeletonization technique allows a network to use spare input and hidden units to learn a set of training examples rapidly, and gradually, as units are trimmed away, to discover a more concise characterization of the underlying regularities of the task. In the process, local minima seem to be avoided without increasing the overall learning time. It is difficult to say how this result will scale up with network size, but experiments with a larger version of the random mapping problem suggest that the approach is promising.

One somewhat surprising result is the ease with which a network is able to recover when a unit is removed. Conventional wisdom has it that if a network is given excess hidden units, it will memorize the training set, thereby making use of all the hidden units available to it. However, in our simulations, the network does not seem to be distributing the solution across all hidden units because even with no further training, removal of a hidden unit often does not drop performance below the criterion. In any case, there generally appears to be an easy path from the solution with many units to the solution with fewer.

Although we have presented skeletonization as a technique for removing units from a network, there is no reason why a similar procedure could not operate on individual connections instead. Basically, an α coefficient would be required for each connection, allowing for the computation of $\partial E/\partial \alpha$. Yann le Cun (personal communication, 1989) has independently developed a procedure quite similar to our skeletonization technique which operates on individual connections and uses the second derivative of the quadratic error function instead of the first derivative of the linear error function.

In simple problems where it is possible to find a minimal network that completely solves the problem, skeletonization provides an effective means of automatically trimming a larger network down to the minimal size. In more complex problems, where it is likely that even the large initial network will not be capable of learning the training set perfectly, as network size shrinks, the achievable performance on the training set will simply degrade monotonically. The network size selects a point in the trade-off between the poorer performance on the training set that comes from smaller networks and the greater consumption of resources, greater difficulty of analysis, and likely poorer generalization that come from larger networks. Analyses such as that of Baum & Haussler (1989) provide theoretical insight into the trade-off, while skeletonization provides a computational process by which networks can systematically explore this trade-off for themselves.

Notes

1. Overall learning speed improves only if the total time spent learning with a decreasing number of units is less than the time that would have been spent learning with just the minimal number of units. It is not obvious *a priori* that reducing a large network will be faster than starting with a small network: the process of converting the end product of learning with more units to the initial weights for learning with fewer units—here, by discarding units according to a relevance measure—must be well designed. Learning can be slower overall if the end product of learning with more units does not provide a good starting point for learning with fewer units; for example, if the starting point is no better than random, all the time spent on learning with more units is wasted.
2. Note that the example illustrated in Figure 2 behaves as desired with the quadratic error function because the units—with activity values 0.2 and 0.5 and target values 1—are not in the range of small error.
3. The values reported in Table I are an average over 100 replications of the simulation with different initial random weights. Before averaging, however, the signs of the weights were flipped if the hidden-output connection was negative.
4. Here and below we report median epochs to criterion rather than mean epochs to avoid aberrations caused by the large number of epochs consumed in failure runs.
5. To eliminate floor and ceiling effects, we rejected training sets that were either extremely simple to learn—the standard network could learn in under 20 epochs and were impossible to learn—the standard network was never able to find a solution.

References

Baum. E.B. & Haussler, D. (1989) What size net gives valid generalization? *Neural Computation*, **1**, 151–160.

Chauvin, Y. (1989) A back-propagation algorithm with optimal use of hidden units. In D. Touretzky (Ed.) *Advances in Neural Network Information Processing Systems 1.* San Mateo, CA: Morgan Kaufmann.

Elman, J.L. (1989) *Structured representations and connectionist models* (CRL Technical Report 8901). La Jolla, CA: University of California, San Diego, Center for Research in Language.

Hanson, S.J. & Pratt, L.Y. (1989) Some comparisons of constraints for minimal network construction with back propagation. In D. Touretzky (Ed.) *Advances in Neural Network Information Processing Systems 1.* San Mateo, CA: Morgan Kaufmann.

Medin, D.L., Wattenmaker, W.D. & Michalski, R.S. (1987) Constraints and preferences in inductive learning: an experimental study of human and machine performance. *Cognitive Science*, **11**, 299–339.

Mozer, M.C. & Smolensky, P. (1989) Skeletonization: a technique for trimming the fat from a network via relevance assessment. In D. Touretzky (Ed.) *Advances in Neural Network Information Processing Systems 1.* San Mateo, CA: Morgan Kaufmann.

Rumelhart, D.E., Hinton, G.E. & Williams, R.J. (1986) Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1. Foundations*, pp. 318–362. Cambridge, MA: MIT Press/Bradford Books.

Sanger, D. (1989) *Contribution analyis: A technique for assigning responsibilities to hidden units in connectionist networks* (Technical Report CU-CS-435-89). Boulder, CO: University of Colorado, Department of Computer Science.

Sejnowski, T.J. & Rosenberg, C.R. (1987) Parallel networks that learn to pronounce English text. *Complex Systems*, **1**, 145–168.

Appendix: Training Procedure

All simulations made use of a three-layer feedforward network with input-hidden and hidden-output connections, as described in Rumelhart *et al.*, (1986). For several tasks, the hidden layer was not strictly necessary, but was included to keep the architecture uniform across problems. The inclusion of the hidden layer in no way affected the qualitative results. Input and target output patterns were binary (-1 or 1) vectors. Hidden and output units used the conventional sigmoidal activity function with an activity range of -1 to 1: $f(x)=\tanh(x/2)$.

Initially, connection strengths are selected at random from a mean-zero Gaussian distribution and are rescaled such that the L_1 norm of the connection strengths feeding into a unit is 2.0. Consequently, if a unit has many inputs, weights tend to be smaller. With the initial weights chosen in this manner, each unit has some variation in its activity but tends to operate in the linear range of the sigmoid curve. Weights are updated only after a complete pass through the entire training set, and training continues until all output unit activities are within some acceptable range around the target value (the *margin*, see Table IV).

Table IV

Simulation	Margin	Learning rate	Layer trimmed
Cue salience	0.1	4.0	input
Rule plus exception	0.1	3.0	hidden
Train	0.1	3.0	input
4-bit multiplexor	0.9	2.0	hidden
Random mapping	0.9	3.0	hidden

A fixed learning rate was determined for each simulation to optimize performance for the standard network. This learning rate was divided by the number of inputs to a unit, so that it would scale up as units were trimmed from the network. Dividing the learning rate by the fan-in is probably an overly conservative procedure and, if anything, penalizes the simulations involving skeletonization of the hidden layer due to the larger number of hidden units initially. It also allows for different learning rates in the input-hidden and hidden-output connections.

Momentum was used, with a generic momentum parameter of 0.5—probably not large enough to help a lot, but not large enough to hurt either.