PERGAMON

Neural Networks 12 (1999) 479–498

Neural Networks

Contributed article

Template-based procedures for neural network interpretation

J.A. Alexander, M.C. Mozer*

Department of Computer Science, University of Colorado, Boulder, CO 80309-0430, USA

Received for publication 8 January 1999

Abstract

Although neural networks often achieve impressive learning and generalization performance, their internal workings are typically all but impossible to decipher. This characteristic of the networks, their *opacity*, is one of the disadvantages of connectionism compared to more traditional, rule-oriented approaches to artificial intelligence. Without a thorough understanding of the network behavior, confidence in a system's results is lowered, and the transfer of learned knowledge to other processing systems – including humans – is precluded. Methods that address the opacity problem by casting network weights in symbolic terms are commonly referred to as *rule extraction* techniques. This work describes a principled approach to symbolic rule extraction from standard multilayer feedforward networks based on the notion of *weight templates*, parameterized regions of weight space corresponding to specific symbolic expressions. With an appropriate choice of representation, we show how template parameters may be efficiently identified and instantiated to yield the optimal match to the actual weights of a unit. Depending on the requirements of the application domain, the approach can accommodate *n*-ary disjunctions and conjunctions with O(k) complexity, simple *n*-of-*m* expressions with O(k^2) complexity, or more general classes of recursive *n*-of-*m* expressions with O(k^{L+2}) complexity, where k is the number of inputs to an unit and L the recursion level of the expression class. Compared to other approaches in the literature, our method of rule extraction offers benefits in simplicity, computational performance, and overall flexibility. Simulation results on a variety of problems demonstrate the application of our procedures as well as the strengths and the weaknesses of our general approach.

Keywords: Neural networks; Rule extraction; Connectionist networks; Sigmoidal units; Boolean

1. Rule extraction

Although neural networks often achieve impressive learning and generalization performance, their internal workings are typically all but impossible to decipher. Unlike their rule-based counterparts, they cannot explain their decisions. Networks are often described as *opaque*; one cannot easily look inside them to ascertain how they produce their results (Minsky, 1991). Although this characteristic may contribute to some of the mystique associated with connectionist systems, it is a serious problem in many situations. This article is concerned with a specific approach for attacking the opacity problem in neural networks, an approach that provides visibility into a network's operation by casting its weights in the form of symbolic rules. Our approach gives rise to several procedures for performing what is commonly termed *rule extraction.*

Why is connectionist rule extraction important? Consider a network whose units use the conventional inner product activation and sigmoidal output functions with a $[-1, +1]$ range, as is common in feedforward networks trained by back propagation (Fig. 1). Although networks of these units have been successfully used to learn many interesting tasks, in most cases it is difficult to explain the behavior of the trained system by inspecting the weights. As a simple example of this situation, consider the following vector of 16 weights plus a bias. (We will adopt the convention that the bias is the last element of the vector.)

$$\mathbf{w} = -0.38\ 0.73\ 1.88\ -4.36\ -0.63\ 0.43\ -0.89\ -0.77\ 1.18\ -1.22\ 1.99\ -0.84\ 0.01\ 0.38\ 0.64\ -0.32\ 1.84.$$

These weights represent the knowledge embedded in a trained 16-0-1 sigmoidal network. (The notation '16-0-1' indicates that there are 16 input units, no hidden units, and a single output unit.) This particular network was trained to predict the political party affiliation (Democrat or Republican) for members of the US House of Representatives based on their voting history on 16 key issues. The network performs its task very well – over 95% accuracy on both training and test data – but exactly *how* does it reach its answers? One way to produce an explanation of the network behavior is to fall back on the activation equation shown in Fig. 1. However, this approach leads to a mathematical description that does not much distill the raw information

0893-6080/99/$ - see front matter
PII: S0893-6080(99)00009-X

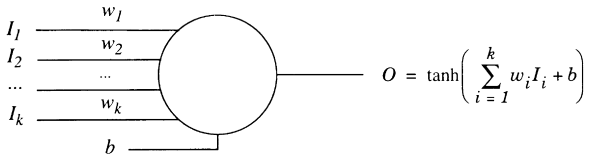

Fig. 1. Sigmoidal unit.

of the weights. An alternative approach is to describe the behavior of the network at a higher level, a level where most of the mathematical information is abstracted away. For example, we might envision a purely symbolic explanation that related the party affiliation by some sort of *if-then* rule such as the following:

> *If (voted for health care bill or voted against capital gains tax reduction bill) then (affiliation is Democratic).*

Symbolic extractions of this sort are valuable for several reasons. Compared to numbers, symbols are easier for people to understand. Human understanding is critical if one wishes to build confidence in a network's performance among people. This is particularly so in mission-critical applications, where the consequences of failure are so great that opaque, black-box systems are not perceived as trustworthy. Human understanding is also important if we wish to transfer the learning of a network to a person. This might be the case if a network was used as an expert assistant to provide input to a human decision maker. Symbolic rules are also required should one wish to transfer the learning of a network to an expert system that uses rules as the fundamental unit of knowledge. This situation might arise if the network was being used as part of the knowledge engineering phase of an expert system project.

Symbolic rules hide the detail and thus provide an efficient base for subsequent levels of reasoning. The hiding of detail often results in a certain loss of precision, but this is usually an acceptable trade off. One might even conjecture that the loss of precision brought about by rule extraction from neural networks might actually improve performance in some domains, because the rules help suppress noise and provide constraints that restrict generalization.

Table 1
Unit output for all input patterns

Inputs				Net input	Output	
I_1	I_2	I_3	I_4		Actual	Thresholded at 0
−1	−1	−1	−1	−3	−0.906	−1
−1	−1	−1	+1	3	0.906	+1
−1	−1	+1	−1	−9	−1.000	−1
−1	−1	+1	+1	−3	−0.906	−1
−1	+1	−1	−1	−3	−0.906	−1
−1	+1	−1	+1	3	0.906	+1
−1	+1	+1	−1	−9	−1.000	−1
−1	+1	+1	+1	−3	−0.906	−1
+1	−1	−1	−1	3	0.906	+1
+1	−1	−1	+1	9	1.000	+1
+1	−1	+1	−1	−3	−0.906	−1
+1	−1	+1	+1	3	0.906	+1
+1	+1	−1	−1	3	0.906	+1
+1	+1	−1	+1	9	1.000	+1
+1	+1	+1	−1	−3	−0.906	−1
+1	+1	+1	+1	3	0.906	+1

Connectionist rule extraction is also valuable because it provides a link between neural networks and traditional, symbolic approaches to artificial intelligence (AI). This is important because of a growing realization among researchers that AI is an eclectic endeavor, one that must draw upon both connectionist and symbolic approaches to make significant progress. By showing how neural networks may be interpreted in symbolic terms, connectionist rule extraction makes a contribution toward bridging the gap between the two paradigms. Effective use of rule extraction and other hybrid models may well show the space between connectionist and symbolic AI to be a continuum rather than a true gap.

Connectionist rule extraction is thus a topic worthy of investigation. Nevertheless we are not the first to reach this conclusion; methods for analyzing connectionist networks using various sorts of rules have already been proposed by many researchers. We discuss some of these methods later. For now, we simply state that none of the existing methods are so universally applicable as to eliminate the need for additional work in this area.

2. A template-based approach to rule extraction

Any method that attempts a symbolic reformulation of network behavior must first choose a language of description. The language of description used in this work is based on *n*-of-*m* expressions. An *n*-of-*m* expression consists of a list of *m* terms and a value *n* such that $1 \leq n \leq m$. The overall expression is true when at least *n* of the *m* terms are true. An example of an *n*-of-*m* expression stated using logical variables is the majority voter function

$$X = 2 \text{ of } (A, B, C). \tag{1}$$

As a means of symbolic description, *n*-of-*m* expressions are interesting because they are capable of describing behaviors intermediate to standard Boolean OR ($n = 1$) and AND ($n = m$) functions. These intermediate behaviors are quite powerful, as they reflect a limited form of two-level Boolean logic. To see why this is true, note that Eq. (1) is equivalent to the logic expression

$$OR[AND(A, B),\ AND(B, C),\ AND(A, C)].$$

Later, we illustrate even more general behaviors that result from taking a recursive view of these expressions. In addition to the descriptive power they afford, *n*-of-*m* expressions are amenable to human comprehension, and *n*-of-*m* expressions are a practical choice for use with sigmoidal units due to the monotonic output function of such units.

To see how a set of connectionist weights may be reformulated as a symbolic *n*-of-*m* expression, consider a four-input sigmoidal unit. Suppose that after training the unit's weights (and bias) are represented by the vector:

$$\mathbf{w} = 6\ \ 0\ \ -6\ \ 6\ \ 0.$$

Assuming that four $-1/+1$ inputs I_1-I_4 are applied to these weights, what symbolic function does this unit compute? Treating -1 inputs and outputs as the Boolean state *false*, and $+1$ inputs and outputs as the Boolean state *true*, Table 1 suggests that the unit computes a nearly exact version of the *n*-of-*m* expression, 2 of $(I_1, \bar{I}_3, I_4)$. (The overbar notation is used to indicate negation of a term.)

In this example we can be fairly certain that no other *n*-of-*m* expression better describes the function being computed by the unit. In general, however, the correspondence between weights and symbolic expressions is less clear, and a principled way of exploring many possible symbolic interpretations is required. Essentially we must mount a search over the *n*-of-*m* expression space, comparing the actual weights to each possible expression using some similarity to metric. To manage the search space, we introduce the notion of a *weight template* to represent each *n*-of-*m* expression under consideration. Following McMillan (1992), we define a weight template as a parameterized region of the weight space that corresponds to a given symbolic expression. Each weight template is parameterized to reflect invariances in the symbolic input–output behavior of sigmoidal units. The template that describes the expression given earlier is:

$$\mathbf{t} = p\ \ 0\ \ -p\ \ p\ \ 0.$$

As long as the parameter p is positive, the symbolic interpretation of the unit's behavior (obtained by treating the sigmoidal units as a binary threshold unit, as in Table 1) is independent of the actual value. The parameter p thus allows many sets of actual weights to map onto the same weight template. In this example, setting p to six causes the actual weights to match the template exactly. As exact matches are rare for real problems, in general we need to compute the degree of fit between the actual weights and the candidate templates. Once the template that best fits the actual weights has been identified, we have extracted a rule, because each template is associated with a unique *n*-of-*m* expression.

The procedure for rule extraction that we have sketched – and will elaborate in Section 3 – requires only the weights in a trained network and not the training data. Instead of evaluating the candidate rules on the basis of weight differences, one might consider evaluating them on the basis of output differences over the training set. However, by emphasizing the training data, such a procedure might fail to model the regularities that the network has learned, thus compromising generalization performance.

3. The algorithm in detail

Having presented an overview of our approach to the connectionist rule extraction, we now describe our extraction methods in detail. We begin by characterizing the general form of templates for *n*-of-*m* expressions, and then elaborate a three-step extraction process. We also discuss several interesting variations on the basic extraction algorithm that demonstrates the generality of our approach.

3.1. Weight templates for n-of-m expressions

To illustrate how weight templates can be used to represent *n*-of-*m* expressions, consider a generalized weight vector for a four-input sigmoidal unit:

$$\mathbf{w} = w_1\ \ w_2\ \ w_3\ \ w_4\ \ b.$$

Now consider the following two template vectors:

$$\mathbf{t}_1 = 0\ \ p\ \ 0\ \ p\ \ p,$$

$$\mathbf{t}_2 = p\ \ -p\ \ p\ \ 0\ \ -2p.$$

These templates are parameterized by the variable p, stipulated to be positive. Given a large positive value of p (say 5.0) and an input vector $\mathbf{I}$ whose components are approximately -1 and $+1$, $\mathbf{t}_1$ describes the symbolic expression 1 of (I_2, I_4), and$\mathbf{t}_2$ describes the symbolic expression 3 of $(I_1, \bar{I}_2, I_3)$. We can use truth tables (e.g. Table 1) to confirm that $\mathbf{t}_1$ and $\mathbf{t}_2$ really do represent the aforementioned expressions, but how can we *generate* the weight template for a given symbolic expression? For sigmoidal units with $-1/+1$ inputs, a general prescription for forming an *n*-of-*m* template is as follows:

1. set each of the weight values associated with the m inputs of interest to $\pm p$, choosing $+p$ for each *positive* (normal) term, $-p$ for each *negated* term;
2. set all other weight values to zero;
3. set the bias value to $(1+m-2n)p$.

This prescription covers the expressions where $1 \leq m \leq k$ and $1 \leq n \leq m$. For completeness, it is useful to describe templates for two additional expressions: one for a constant *true* expression and one for a constant *false* expression. To be consistent with the general form of *n*-of-*m* templates, these constant expression templates have zero values in all weight positions except the bias position, where a true template has the value $+p$ and a false template has the value $-p$.

Now that we have specified how weight templates may be generated for symbolic expressions within our language of description, we are ready to explore the process of using weight templates to find the *optimal* or the best-fitting expression for a given set of weights. Optimal is defined as the expression corresponding to a valid template, $\mathbf{t}^*$, having the parameter p^*, such that the Euclidean distance

$$d = \|\mathbf{t}^* - \mathbf{w}\|^2 \tag{2}$$

is minimized. Finding an optimal expression involves three steps. First, a set of candidate templates is identified. For efficiency, an informed selection algorithm allows the set of candidate templates to be much smaller than the set of all possible *n*-of-*m* expressions. Second, each candidate template's parameters are then instantiated with optimal values. Third, the instantiated template best fitting the actual weights is chosen as the basis for symbolic interpretation. We discuss these three steps in turn.

3.2. Establishing candidate templates for a given unit

Given an actual weight vector for a unit with k inputs ($k+1$ elements when the bias is counted), the total number of *n*-of-*m* expressions that can be associated with that weight vector is:

$$T = \sum_{m=1}^{k} \sum_{n=1}^{m} 2^m \binom{k}{m} = \sum_{m=1}^{k} m 2^m \frac{k!}{(k-m)!m!}. \quad (3)$$

For $k = 10$, T is 393660, and for $k = 20$, T is more than 46 billion. One reason why T is so large is because it accounts for (using the exponential term) all possible combinations of positive and negated terms. The other reason is because it accounts for (using the factorial terms) all possible subsets of m terms among the k inputs. As we explain next, these high-complexity terms can be eliminated, yielding a value for T that is only quadratic in k.

3.2.1. Pruning the search space using the signs of the actual weights

To show how the signs of the actual weights may be used to reduce the required number of templates, we must refer to our optimality measure, Eq. (2). Any template whose jth element, t_j, has sign opposite to the jth element of the weight vector, w_j, will yield a larger distance measure than any other template whose t_j has the same sign as w_j. Thus, we can use the sign of the actual weight to determine a single logical sense (positive or negated) for each term. We can similarly use the sign of the actual bias to reduce the range of the values of n for which *n*-of-*m* templates must be generated. The results are quite simple: for each j, a positive term is considered for the corresponding input only if the actual weight is positive and a negated term is considered for the corresponding input only if the actual weight is negative. With regard to the bias, only *n*-of-*m* templates for which the sign of the template bias value agrees with the sign of the actual bias value are considered. (For this method a template bias value of 0 is considered both positive and negative.) Theorem 1 in Appendix A shows that this procedure is valid, i.e. it will not rule out the optimal template.

As an example of the implications of this sort of pruning, consider the following weight vector:

$$\mathbf{w} = -4\ 0\ 1\ 5\ 3.$$

If we are interested in generating templates whose terms involve I_1 and I_3, then there are eight possible templates of this form, as shown in the following equations:

$$\mathbf{t}_1 = -p\ 0\ -p\ 0\ -p,$$

$$\mathbf{t}_2 = -p\ 0\ -p\ 0\ p,$$

$$\mathbf{t}_3 = -p\ 0\ p\ 0\ -p,$$

$$\mathbf{t}_4 = -p\ 0\ p\ 0\ p,$$

$$\mathbf{t}_5 = p\ 0\ -p\ 0\ -p,$$

$$\mathbf{t}_6 = p\ 0\ -p\ 0\ p,$$

$$\mathbf{t}_7 = p\ 0\ p\ 0\ -p,$$

$$\mathbf{t}_8 = p\ 0\ p\ 0\ p.$$

Theorem 1 proves that of these eight templates, we need to consider only $\mathbf{t}_4$, which represents the expression 1 *of* $(\bar{I}_1, I_3)$ When instantiated with its optimal value of p, this template is guaranteed to have a smaller distance to w than any of the other aforementioned templates.

3.2.2. Pruning the search space using the magnitudes of the actual weights

The factorial terms in Eq. (3) reflect the fact that there are in general many possible groups of m terms among a set of k inputs, even when the logical sense of each term is fixed in accordance with Theorem 1. Fortunately we need not generate weight templates for each possible grouping of m terms. Theorem 2 in Appendix A shows that for a given choice of n and m, we need only consider *one* template, the one whose m terms correspond to the m highest absolute value actual weights.

For an example of the implications of Theorem 2, we return to the weight vector:

$$\mathbf{w} = -4\ 0\ 1\ 5\ 3.$$

If we are considering templates for 1-of-2 expressions, then taking into account the results of Theorem 1, there are six possible templates of this form:

$$\mathbf{t}_1 = -p\ p\ 0\ 0\ p,$$

$$\mathbf{t}_2 = -p\ 0\ p\ 0\ p,$$

$$\mathbf{t}_3 = -p\ 0\ 0\ p\ p,$$

$$\mathbf{t}_4 = 0\ p\ p\ 0\ p,$$

$$\mathbf{t}_5 = 0\ p\ 0\ p\ p,$$

$$\mathbf{t}_6 = 0\ 0\ p\ p\ p.$$

Theorem 2 proves that of these six templates, we need to generate only one, $\mathbf{t}_3$, which represents the expression 1 of $(\bar{I}_1, I_4)$, and when instantiated with its optimal value of p, it

Table 2
Enumeration of candidate templates for example weight vector

Expression type	Candidate template					p^*	Instantiated template					Distance from w	Extraction error
Constant	0	0	0	0	p	3.00	0.00	0.00	0.00	0.00	3.00	42.0	80.8%
1−of−1	0	0	0	p	0	5.00	0.00	0.00	0.00	5.00	0.00	26.0	51.0%
1−of−2	$-p$	0	0	p	p	4.00	−4.00	0.00	0.00	4.00	4.00	3.0	5.9%
1−of−3	$-p$	0	p	p	$2p$	2.29	−2.29	0.00	2.29	2.29	4.58	14.4	28.2%
2−of−3	$-p$	0	p	p	0	3.33	−3.33	0.00	3.33	3.33	0.00	17.7	34.7%
1−of−4	$-p$	p	p	p	$3p$	1.46	−1.46	1.46	1.46	1.46	4.38	23.2	45.5%
2−of−4	$-p$	p	p	p	p	2.60	−2.60	2.60	2.60	2.60	2.60	17.2	33.7%

is guaranteed to have a smaller distance to **w** than any of the other aforementioned templates.

The consequence of Theorems 1 and 2 is that the number of *n*-of-*m* templates required for a unit with *k* inputs is reduced to a polynomial function of *k*, i.e.

$$T = \sum_{m=1}^{k} \sum_{n=1}^{\left\lfloor \frac{(m+1)}{2} \right\rfloor} 1 = \left\lfloor \frac{1}{4}k^2 + \frac{1}{2}k + \frac{1}{4} \right\rfloor. \tag{4}$$

This number is a dramatic reduction over the original value of T [Eq. (3)]. Values for T for $k = 10$ and 20 are now 30 and 110, respectively. Thus, an efficient and exhaustive search can be performed through the space of the possible *n*-of-*m* expressions.

3.3. Instantiating template parameters

Having described the process of establishing candidate templates, we proceed to the second step of the rule interpretation algorithm: *instantiating* a weight template-selecting the value of the parameter p, call it p^* that minimizes d in Eq. (2) for a given template **t**. As d is quadratic in p, a unique minimum exists. The general solution for any *n*-of-*m* template is given by the following equation:

$$p^* = \frac{\sum_{i=1}^{k+1} w_i u_i}{\sum_{i=1}^{k+1} u_i^2}, \tag{5}$$

where $u_i \in \{-1, 0, 1\}$ depending on whether t_i is $-p$, 0, or $+p$, respectively, and index $k+1$ corresponds to the bias term and $u_{k+1} = m-2n+1$. When **t** is consistent with the result of Theorem 1 (i.e. when each non-zero u_i agrees in sign with the corresponding w_i), p^* will always be positive.

Applying Eq. (5) to the example vectors for **w** and $\mathbf{t}_3$ in Section 3.2.2 yields $p^* = 4.0$, which can be seen by inspection to be the best choice. The minimal distance d for this instantiated template is then 3.0.

3.4. Finding the best-fitting template

The third step in interpreting the algorithm is identifying the best-fitting template. To do this, the distance between each instantiated template and the actual weight vector is computed [Eq. (2)], and the minimum-distance template, **t***, is selected. We can use **t*** as part of a rudimentary check on the validity of the extraction process, via the *extraction error*

$$E_{\text{extraction}} = 100\% \times \frac{\|\mathbf{t}^* - \mathbf{w}\|^2}{\|\mathbf{w}\|^2}. \tag{6}$$

This quantity measures how close the original weights are to the nearest symbolic rule. When the nearest symbolic rule is still relatively far from the actual weights, it suggests that the unit in question is not computing a symbolic function, or is computing a symbolic function beyond the ability of the present algorithm to model. We can also examine the value of p^* used in **t***. Small values of p^* translate into activation changes below a unit's full dynamic range, compromising the assumption of Boolean outputs propagating to subsequent inputs. Thus, $E_{\text{extraction}}$ and p^* serve as diagnostics on the likely quality of the extracted rule.

3.5. Algorithm summary

The development of the preceding sections is summarized in the following algorithm. Here and elsewhere in this article, *sign*(x) is taken as $+1$ for $x \geq 0$, -1 otherwise.

```
Given an actual weight vector
w = [w1 w2 ... wk wk+1],
For m = 1 to k

  For n = 1 to m
    If (sign(1+m−2n) == sign(wk+1)) OR
    (1+m−2n == 0) {

      1. Initialize template coefficient
         vector u to all 0's.
      2. Set the template weight coefficients
         corresponding to the m highest
         absolute value weights in w (not
         including wk+1) to the sign of the
         corresponding weight in w.
      3. Set the template bias coefficient
         uk+1 to 1+m−2n.
      4. Solve for p* using Eq. (5).
```

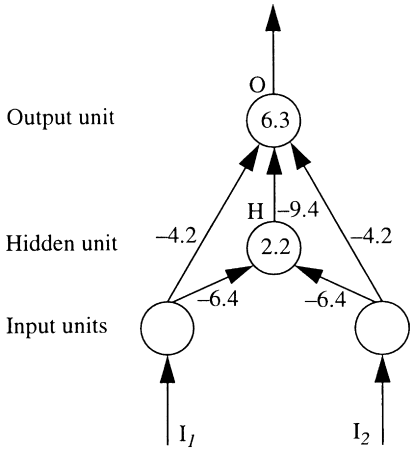

Fig. 2. 2-1-1 network for the xor problem.

5. *Compute* $\mathbf{t}$ *by multiplying each element of* $\mathbf{u}$ *by* p^*.
6. *Compute the distance* $d = \|\mathbf{t}-\mathbf{w}\|^2$.
7. *Save* $\mathbf{t}$, p^*, *and* d *if* d *is the minimum found so far.*

Return the n-of-m condition corresponding to the template with minimal d.

3.6. A simple example

We present an example of the complete interpretation process for the weight vector

$$\mathbf{w} = -4\ 0\ 1\ 5\ 3.$$

Table 2 shows the seven templates generated by the algorithm. The distances in the table indicate that the best template is

$$\mathbf{t}_{1\text{-of-}2} = -4\ 0\ 0\ 4\ 4.$$

For the purpose of interpreting, this unit may be said to be computing a function fairly close to the symbolic expression 1 *of* $(\bar{I}_1, I_4)$. The extraction error, $E_{\text{extraction}}$, is low at 5.9%, and $p^* = 4$ indicates essentially the Boolean behavior. Construction of a truth table offers further confirmation that the aforementioned expression is a valid interpretation of this unit's behavior under binary inputs.

3.7. Variations on the basic algorithm

In this section we explore several variations on the basic rule interpretation algorithm. The variations treated here are by no means an exhaustive set, but they do serve to illustrate the basic flexibility inherent in our approach to the connectionist rule extraction.

3.7.1. Grouped-input representations

The derivation of our basic algorithm presumed that the inputs to a network independent of each other. In problems where the inputs possess some structure, we can utilize this structure to constrain rule extraction. A common example of structure is the grouped-input encoding, where instead of k independent inputs, we have g groups of h inputs each, such that each group codes a local (1 of h) representation of a given input feature.[1] To see how the interpretation algorithm can be informed by this type of input structure, consider the following weight vector, which has $g = 2$ groups of $h = 3$ alternatives.

$$\mathbf{w} = w_{11} w_{12} w_{13} w_{21} w_{22} w_{23} b.$$

Each element is indexed by its group and its position within the group. It can be seen that for any of the values c_j, $j \in \{1,\ldots,g\}$, the following weight vector is an activation invariant of w for any $-1/+1$ inputs adhering to the encoding scheme described earlier.

$$\mathbf{w}' = w_{11}+c_1\ \ w_{12}+c_1\ \ w_{13}+c_1\ \ w_{21}+c_2\ \ w_{22}+c_2\ \ w_{23}+c_2\ \ b+(h-2)(c_1+c_2).$$

A convenient way of accounting for this invariance is to add unknown values for the c_j's to the candidate weight templates. Thus, a 1-of-2 template might appear as follows:

$$\mathbf{t} = c_1\ \ c_1+p\ \ c_1\ \ c_2+p\ \ c_2\ \ c_2\ \ p+c_1+c_2.$$

Instead of a single parameter p, the template now has $g+1 = 3$ parameters requiring instantiation. Fortunately, even with the added template parameters, the expression for the Euclidean distance between a template and an actual weight vector [Eq. (2)] describes a simple hyperparabolic surface, and thus the parameters yielding the minimum distance can be found by differentiating and solving a set of $g+1$ linear equations in $g+1$ variables.

3.7.2. Augmented n-of-m expressions

Although the n-of-m expressions treated so far are quite powerful, we have observed an interesting class of symbolic behaviors exhibited by single units that cannot be described using n-of-m expressions. The simplest example of this type of behavior may be seen in the single hidden-unit *xor* network described by Rumelhart et al. (1986), reproduced in Fig. 2. In this figure, the numbers inside the circles represent unit bias values. Given that this network was trained using 0/1 inputs, a simple analysis shows that the hidden unit H has learned the expression $AND\,(\bar{I}_1, \bar{I}_2)$, and the output unit O has learned the expression

$$AND\,[OR\,(\bar{I}_1, \bar{I}_2), \bar{H}],$$

which may also be written as

$$2 \text{ of } [1 \text{ of } (\bar{I}_1, \bar{I}_2), \bar{H}].$$

This expression can be viewed as a nested or recursive form of n-of-m expression, where at least one of the m terms is

[1] The analysis here is easily extended to cases where each group has its own value of h.

itself an *n*-of-*m* expression. Arbitrary recursive expressions such as 1 of [2 of (I_1, I_2, I_3), 2 of (I_4, I_5, I_6)] cannot be computed by a single sigmoidal unit, because such expressions are not generally linearly separable. However, at least one significant subclass of recursive *n*-of-*m* expressions is linearly separable. Here, we develop a description of this subclass and show how templates for such expressions may included in our rule extraction methodology.

The subclass of linearly separable recursive *n*-of-*m* expressions of interest consists of the following two forms:

$$OR\ (C_{n\text{-of-}m}, A_1, A_2, \ldots, A_q),$$

$$AND\ (C_{n\text{-of-}m}, A_1, A_2, \ldots, A_q),$$

where $C_{n\text{-of-}m}$ is an arbitrary *n*-of-*m* expression, $1 \leq n \leq m$, and the $A_i, i \in \{1, \ldots, q\}$, are simple terms. We call these forms *augmented* *n*-of-*m* expressions because they extend simple *n*-of-*m* expressions with additional disjuncts or conjuncts.

In considering how weight templates reflecting these forms may be generated, recall that templates for simple *n*-of-*m* expressions have a fundamental scaling parameter *p* for the *m* non-zero weights and a bias multiplier on *p* that is a function of *n* and *m*. The templates are set up such that when the given expression is minimally satisfied, the net input of the unit is *p*, and when the expression is minimally violated, the net input is $-p$. To achieve a similar behavior with an augmented *n*-of-*m* template, an additional multiplier on *p* is required for the *q* non-zero template weights associated with the additional terms A_j. This is because each of those weights must affect the net input as much as the entire inner $C_{n\text{-of-}m}$ expression. (A single active additional disjunct makes the entire augmented *OR* forms true, while a single inactive additional conjunct makes the entire augmented *AND* form false.) Templates for augmented *n*-of-*m* expressions should thus have the following structure:

1. *m* of the template weights is set to $\pm p$, where *p* stands for a positive quantity.
2. *q* of the template weights is set to $\pm K_q p$, where K_q is a positive integer constant.
3. The rest of the template weights are set to 0.
4. The template bias is set to $K_b p$, where K_b is an integer constant.

Appropriate values for the multipliers K_q and K_b may be derived by considering certain boundary conditions for each type of the augmented expression. For the *OR* type, the unit must be active when *n* of the *m* terms in $C_{n\text{-of-}m}$ are true and none of the *q* terms A_i are true. The unit must also be active when none of the *m* terms in $C_{n\text{-of-}m}$ are true and one of the *q* terms A_i is true. The unit must be inactive when *n*-1 of the *m* terms in $C_{n\text{-of-}m}$ are true and none of the *q* terms A_i are true. When $-1/+1$ inputs are used, these conditions may be expressed as:

$$(2n-m)p - qK_q p + K_b p > 0,$$

$$-mp + (2-q)K_q p + K_b p > 0,$$

$$[2(n-1)-m]p - qK_q p + K_b p < 0.$$

Equating the left-hand sides of these conditions to *p*, *p*, and $-p$, respectively, gives the solutions for K_q and K_b as follows:

$$K_q = n, \tag{7}$$

$$K_b = m + 1 + n(q-2). \tag{8}$$

These solutions are appealing because they are consistent with those for simple *n*-of-*m* expressions, as can be seen by setting $q = 0$, where the value for K_b then reverts to $m+1-2n$, as we would expect. A different sort of degeneracy occurs when $n = 1$. Now K_q is 1 and K_b is $(m+q)-1$, so the overall expression looks like a simple disjunction of $m+q$ terms.

Similar boundary conditions may be established for the *AND* type. In this case the relevant conditions become:

$$(2n-m)p + qK_q p + K_b p > 0,$$

$$[2(n-1)-m]p + qK_q p + K_b p < 0,$$

$$mp + [2(q-1)-q]K_q p + K_b p < 0.$$

When the left-hand sides of these conditions are equated to $p, -p$ and $-p$, respectively, the solutions are:

$$K_q = m + 1 - n, \tag{9}$$

$$K_b = (m+1)(1-q) + n(q-2). \tag{10}$$

These solutions also give $K_b = m+1-2n$ when $q = 0$. The other degeneracy for this type of expression occurs when $n = m$. In this situation K_q is 1, K_b is $1-(m+q)$, and the overall expression looks like a simple conjunction of $m+q$ terms. The existence of these special cases for both the augmented types reflects the fact that the augmented *n*-of-*m* expressions generalize simple *n*-of-*m* expressions in the same way the simple *n*-of-*m* expressions generalize pure disjunctions and conjunctions. Although it is possible that other template forms exist to model the augmented *n*-of-*m* expressions, the simulations detailed in Section 5 show that single units readily evolve weights consistent with the templates described here.

3.7.2.1. Combinatorics As $1 \leq n \leq m$, the condition $K_q > 1$ holds for the non-degenerate cases of both the *OR* and *AND* augmented types. This means that the *q* template weights associated with the additional disjuncts or conjuncts will always be greater in magnitude than the *m* weights associated with $C_{n\text{-of-}m}$. Note also that for the *OR* type the relationship $K_b > 0$ holds, while for the *AND* type the relationship $K_b < 0$ holds. Together these observations allow us to eliminate many possible candidate templates based on the sign of the bias and the absolute values of

the actual weights we are matching against. The incremental number of templates that must be generated to account for augmented *n*-of-*m* expressions is then given by:

$$\sum_{m=2}^{k-1}\sum_{n=1}^{m-1}\sum_{q=1}^{k-m}1=\frac{1}{6}k^3-\frac{1}{2}k^2+\frac{1}{3}k.$$

When added to T [Eq. (4)], the total number of templates required is:

$$T_{\text{aug}}=\left\lfloor\frac{1}{6}k^3-\frac{1}{4}k^2+\frac{5}{6}k+\frac{1}{4}\right\rfloor.$$

Although this figure is O(k) worse than for simple *n*-of-*m* expressions, it is still polynomial in k and is quite manageable for many problems.

3.7.2.2. Algorithm summary A procedure for including templates for augmented *n*-of-*m* expressions in the extraction process is as follows. This procedure is for illustrative purposes, and is not meant to suggest the most efficient implementation.

Given an actual weight vector $w=[w_1 w_2 \ldots w_k w_{k+1}]$,
Find the best simple n-of-m template using the procedure described earlier; save this template's value of d for comparison as follows.

For $m=2$ *to* $k-1$

If w_{k+1} *is positive*

Let $n_1=2$
Let $n_2=m$
Let type $=$ *OR*

Else if w_{k+1} *is negative*

Let $n_1=1$
Let $n_2=m-1$
Let type $=$ *AND*

For $n=n_1$ *to* n_2,

For $q=1$ *to* $k-m$,

1. *Initialize template coefficient vector* $\mathbf{u}$ *to all 0's.*
2. *Set the template weight coefficients corresponding to the q highest absolute value weights in* $\mathbf{w}$ *(not including* w_{k+1}*) to* $K_q\,\text{sign}(w_i)$*, using* K_q *from Eq. (7) (for type* $=$ *OR) or Eq. (9) (for type* $=$ *AND).*
3. *Set the template weight coefficients corresponding to the next m highest absolute value weights to* $\text{sign}(w_i)$.
4. *Set the template bias coefficient* u_{k+1} *to* K_b *from Eq. (8) for (type* $=$ *OR) or Eq. (10) (for type* $=$ *AND).*
5. *Solve for p*.*
6. *Compute* $\mathbf{t}$ *by multiplying each element of* $\mathbf{u}$ *by p*.*
7. *Compute the distance* $d=\|t-w\|^2$.
8. *Save* $\mathbf{t}$*, p*, and d if d is the minimum found so far.*

Return the n-of-m condition corresponding to the template with minimum d.

3.7.2.3. Other augmented forms The subclass of linearly separable recursive *n*-of-*m* expressions discussed previously can be extended by noting that the augmented *n*-of-*m* expressions can themselves play the role of an inner $C_{n\text{-of-}m}$ expression. That is we can nest the *OR* and *AND* augmented forms once again, by arriving at *level*-2 augmented expressions of the following forms:

$$AND\,[OR\,(C_{n-of-m},A_1,A_2,\ldots,A_q),B_1,B_2,\ldots,B_r],$$

$$OR\,[AND\,(C_{n-of-m},A_1,A_2,\ldots,A_q),B_1,B_2,\ldots,B_r],$$

where the B_i, $i\in\{1\ldots r\}$, are additional simple terms. Nesting of this sort can be repeated indefinitely, resulting in level-L augmented expressions. The complexity of rule extraction for a level-L expression is O(k^{L+2}). Each level of nesting requires the introduction of another multiplier (for another group of non-zero weights) in the weight templates that model expressions of that nesting level. Thus for higher levels of nesting, units must evolve weight groups whose values span very large ranges. We omit a further discussion on template generation for these nested forms as they are not used in any of our simulations. (Given the weight ranges required for a single unit to implement these complex expressions, in practice it seems more likely that a network will distribute responsibility for them across units in different layers.)

3.7.3. Extraction algorithm for 0/1 inputs

Our development of the extraction algorithm has presumed units with $-1/+1$ inputs. Extraction can also be performed with 0/1 inputs. The overall process differs a little from that of $-1/+1$ inputs, but there is a significant difference in the number of candidate templates required for the 0/1 case.

For the $-1/+1$ representation, Theorems 1 and 2 proved that the signs and magnitudes of the actual weights and bias could be applied to select the single best template for each possible combination of n and m values. This result is possible because the template definitions allow the sign of each template weight (including the bias) to be chosen independently. The difficulty with templates for 0/1 inputs is that the choice of bias is coupled to the choice of which m template weights to make non-zero. As a result, the total

number of *n*-of-*m* templates that must be considered for a unit with *k* inputs is given by the following equation:

$$T = \sum_{m=1}^{k} \sum_{m_{\text{neg}}=0}^{m} \sum_{n=1}^{m} 1 = \frac{1}{3}k^3 + k^2 + \frac{2}{3}k,$$

which is O(*k*) worse than Eq. (4), the expression for the number of potential templates with $-1/+1$ inputs.

4. Related work

Having laid down the basic principles underlying our template-based symbolic interpretation of non-linear connectionist units, we describe alternative rule extraction techniques in the literature that make the most contact with the present work, and contrast our approach with these alternatives.

4.1. McMillan

The RuleNet system of McMillan (1992) and McMillan et al. (1991) was developed to demonstrate the merits of connectionist rule learning using integrated symbolic and subsymbolic processing. The RuleNet architecture is closely related to the mixture-of-experts configuration described by Jacobs et al. (1991). Although developed independently, RuleNet can usefully be viewed as a specialization of the Jacobs configuration. This configuration features a system of competing *expert networks* working in conjunction with a *gating network*. The outputs of the gating network are normalized (Bridle, 1989) to form probabilities that determine which expert network's output is selected as the overall output of the system. The error function is such that the gating network is encouraged to divide the training patterns into several subsets. The output mapping for each subset is then produced by a different expert network. McMillan used an approach like this to develop an architecture that learned condition–action rules (Holland et al., 1986). His *condition subnet* plays the role of the gating network, and the *action subnet* plays the role of the expert networks. RuleNet is significant for several reasons, but we are most interested in the rule extraction component of the system.

McMillan performed template-based rule extraction on both the condition and action subnets. The templates used for the action subnet were highly specialized to the RuleNet's learning task and are not directly comparable to those discussed in the present work. Templates for the condition subnet, however, are directly comparable, and McMillan's extraction of conditions is the focus of this commentary. Abstracting the exact nature of the conditions learned by RuleNet, the templates treated by McMillan fall into three general types: simple, binary disjunctive, and binary conjunctive. In the language of *n*-of-*m* expressions, these types are described as 1-of-1, 1-of-2, and 2-of-2, respectively. Our method would generate three candidate templates for these types, whereas McMillan's method required a total of g^2 (where g is the number of input groups as described in Section 3.7.1), a substantial difference in search efficiency.

As RuleNet was designed to explore more than just connectionist rule extraction, it is not surprising that the generality and efficiency of the extraction techniques was not the focus. Nevertheless, McMillan's work with RuleNet must be credited as laying down the basic ideas behind template-based rule extraction.

4.2. Towell

Towell (1991) and Towell and Shavlik (1991) have attempted to combine the merits of explanation-based learning and empirical (inductive) learning. To carry out a learning task using Towell's hybrid approach requires both an initial domain theory—a set of rules—for the task and a set of labeled training examples. These items are used in a three-step learning process centered around Knowledge-Based Artificial Neural Networks, or KBANNs. The first step transforms the a priori domain knowledge into a structured initial connectionist network. The second step trains the network using the labeled training examples. The final step is rule extraction, whereby the weights of the trained network are converted into a set of rules. As with McMillan's work, we are most concerned here with the rule extraction step.

Towell's network training method works to ensure that the assumption of Boolean activation behavior is satisfied. Rule extraction is then performed on each unit to determine the conditions causing that unit to become strongly active. These conditions are expressed using a variant of the *n*-of-*m* formalism used in this work. Towell's first step is to iteratively cluster the unit's weights, excluding the bias, into groups. The procedure then checks each training pattern against each weight group and eliminates weight groups that do not affect the classification of any pattern. The next step is to train the network biases only, other weights being frozen; this step helps maintain Boolean unit behavior in the modified network. Finally, rules are extracted by searching for input combinations that activate the unit. In the best case, these rules can be stated strictly in symbolic terms, i.e. without using numeric quantities. The following example, adapted from Towell and Shavlik (1991), illustrates the results for a single unit in such a case:

$$\mathbf{w} = 6.2\ \ 1.2\ \ 6.0\ \ 1.0\ \ 1.0\ \ 6.0\ \ 1.2\ \ -10.9,$$

where the rule extracted is $O = 2$ of (I_1, I_3, I_6).

In this example, conversion of the rule from numeric to symbolic form was possible because of the fact that after group elimination the unit had only one group of weights. When more than one group remains, Towell typically leaves the rule in numeric form, rather than enumerating all the conditions that cause the unit to become active. Even by using *n*-of-*m* expressions, such an enumeration is

problematic because of the combinations involved in the activation equation. An example of a rule extracted from the part of a network trained on the promoter recognition task from molecular biology is:

$$\begin{aligned} \mathit{Minus}35 = -10 < {} & +5.0\mathit{nt}(@-37\text{`--}T\text{-}G\text{--}A\text{'}) \\ & + 3.1\mathit{nt}(@-37\text{`---}GT\text{---'}) \\ & + 1.9\mathit{nt}(@-37\text{`----}C\text{-}CT\text{'}) \\ & + 1.5\mathit{nt}(@-37\text{`---}C\text{--}A\text{-'}) \\ & - 1.5\mathit{nt}(@-37\text{`------}GC\text{'}) \\ & - 1.9\mathit{nt}(@-37\text{`--}CAW\text{---'}), \end{aligned}$$

where *nt*() returns the number of true terms, @-37 locates the terms on the DNA strand, and ''-'' indicates a do not-care term.

This rule is considerably more complicated than a basic *n*-of-*m* expression and is not a complete description of cases under which the Minus35 condition is true. In Towell's Prolog-like formulation, disjunctive conditions are indicated by separate rule enumerations. The aforementioned rule describes only one of the four disjuncts relevant to the Minus35 condition. Nevertheless, rules of this sort are much simpler than the complete networks from which they are extracted and were found to be useful in the domains studied by Towell and his collaborators.

Towell's use of an iterative clustering step stands in contrast to our use of template-based procedures specialized for dividing weights into a fixed number of groups: one group of value 0 and one group of value $\pm p$ for simple *n*-of-*m* templates, additional non-zero groups for augmented *n*-of-*m* templates. Our simpler method appears to be more efficient, but Towell's extraction algorithm is more general in that it can accommodate situations by clustering results in numerous groups of non-zero weights.

However, there are disadvantages associated with Towell's method. The primary disadvantage is that the rules produced by his method are in general not completely symbolic. Although numeric expressions were convenient for the domains Towell studied, in applications where one is only interested in symbolic interpretations such expressions might be seen as providing too much detail, and be difficult for people to interpret and reason about. Sometimes one wants to determine the nearest symbolic interpretation of unit behavior rather than to extract the most precise mathematical description. Our method offers a simpler paradigm for doing this.

In summary, Towell's approach is capable of more expressive descriptions of unit behavior and is therefore more general. The trade off is that the descriptions are more complex, and less comprehensible, than the purely symbolic ones provided by our method. The two approaches represent two different points on the complexity–expressiveness curve, and each approach will have applications where it is most appropriate. Thus we conclude that both methods have their place in rule extraction tool kits.

4.3. Other research

Other methods for the connectionist rule extraction have been suggested by Fu (1989, 1991) and Hayashi (1990). Towell (1991) also describes several alternatives to the *n*-of-*m* method described earlier. The work of Gallant (1988) was an important early example of integrating connectionist and rule-based approaches. Other examples of integrated approaches include Dolan and Smolensky (1989), Touretzky and Hinton (1998) and Sun (1991). Hybrid AI systems have become quite popular in the recent years, and several such systems are discussed in Dinsmore (1992).

Although explicit rule extraction is the focus of this work, there are other ways to combat the problem of network opacity. These include visualization techniques such as those described by Hinton (1989), and statistical approaches such as hierarchical cluster analysis (Sejnowski and Rosenberg, 1987). Other approaches that can be used to facilitate network description include those described in Mozer and Smolensky (1988) and Nowlan and Hinton (1991).

5. Simulations

We performed six sets of simulation experiments that were designed to test the effectiveness of our method of rule extraction. The simulations range from small-scale problems for which we know the solution in advance and can verify whether the interpretation procedure produces the correct outcome, to larger-scale problems from the University of California at Irvine machine learning repository.

5.1. Random functions

Extraction performance was first tested using random *n*-of-*m* functions known to be learnable by a single sigmoidal unit. We tested 100 simple *n*-of-*m* functions of 10 Boolean variables in groups of 10 using a 10-0-10 feedforward network. An example of one of these groups of functions is shown as follows:

$f_1 = 2 \text{ of } (\bar{I}_2, I_{10})$

$f_2 = 1 \text{ of } (\bar{I}_2)$

$f_3 = 6 \text{ of } (I_1, I_2, I_3, \bar{I}_4, I_5, I_6, I_{10})$

$f_4 = 2 \text{ of } (I_1, I_2, I_3, \bar{I}_4, I_5, I_6, I_7, I_8, \bar{I}_9, \bar{I}_{10})$

$f_5 = 5 \text{ of } (I_4, \bar{I}_5, I_7, \bar{I}_8, \bar{I}_9, I_{10})$

$f_6 = 7 \text{ of } (\bar{I}_1, I_3, \bar{I}_5, I_7, I_8, \bar{I}_9, \bar{I}_{10})$

$f_7 = 1 \text{ of } (\bar{I}_3, I_6)$

Table 3
Summary of extraction performance on random functions

Problem	Network topology	p^*		Extraction error	
		Mean	Standard deviation	Mean (%)	Standard deviation (%)
Random *n*-of-*m* functions	10-0-10	5.20	0.19	0.01	0.01
Random augmented *n*-of-*m* functions	10-0-1	4.54	0.32	0.01	0.01

$f_8 = 7$ of $(\bar{I}_1, \bar{I}_2, I_3, \bar{I}_4, \bar{I}_5, I_6, \bar{I}_7, I_8, I_{10})$

$f_9 = 4$ of $(\bar{I}_1, \bar{I}_2, I_3, I_4, I_5, \bar{I}_6, \bar{I}_8, I_9, I_{10})$

$f_{10} = 1$ of $(I_2, I_4, \bar{I}_8, \bar{I}_9)$

The network was trained by back propagation on the standard squared-differences error function (Rumelhart et al., 1986) using the complete set of $2^{10} = 1024$ patterns for each function. Training was stopped when outputs for all patterns were within 0.05 of their target $-1/+1$ values, and testing was performed by reinjecting the extracted rules and thresholding outputs at 0. In all simulations the network learned all the functions and *the extracted rules matched the target functions exactly*. The extracted rules therefore correctly mapped 100% of the patterns. Table 3 summarizes the extraction performance in terms of the two measures p^* and $E_{\text{extraction}}$. The high mean p^* and low mean extraction error indicate that the weight configurations taken on by the network closely match those described by the weight templates, further confirming the accuracy of the extraction process. The statistics reported in Table 3 are the ensemble values obtained by replicating the experiment 10 times with different initial weights but with the same 100 target functions. The low standard deviations for p^* and $E_{\text{extraction}}$ indicate that the extraction process is consistent across both the functions and the experimental replications.

Table 3 also gives the results of testing 20 random augmented *n*-of-*m* functions of 10 inputs. The functions were *strictly* augmented in the sense that values of *n*, *m*, *q*, and the type of expression (*OR* or *AND*) were chosen such that none of the functions had an equivalent simple *n*-of-*m* form. In this case the functions were tested one at a time, over 10 replications. In all cases *the extracted rules matched the target functions exceptionally* and thus correctly mapped all 1024 patterns. Statistical results for these functions are similar to those for the simple *n*-of-*m* functions. The figures for p^* and $E_{\text{extraction}}$ are encouraging because they indicate that the more complicated descriptions of templates for augmented *n*-of-*m* expressions are valid models of weight configurations taken on by actual networks.

The experiments in this section are admittedly simple, but they provide evidence supporting the basic ideas behind a template-based approach to the connectionist rule extraction. In particular, the experiments show that the information on the weights is sufficient to extract a function responsible for generating the training data. Extracting a function from the training data itself is relatively easy; extracting a function from the weights is yet another matter.

5.2. Simple logic problems

We also tested the extraction algorithms on a simple set of logic problems that require multilayer nets for their solution. The problems are as follows:

- the *rule-plus-exception* problem, defined as $O = OR\,[AND\,(A, B), AND\,(\bar{A}, \bar{B}, \bar{C}, \bar{D})]$;
- *xor*-1 is the 2-1-1 version of *xor* shown in Fig. 2;
- *xor*-2 is a strictly layered (2-2-1) version of *xor* (Rumelhart et al., 1986);
- *negation*, also described in Rumelhart et al. (1986), is a problem in which one of the four inputs controls if the other inputs appear unchanged or negated at the outputs.

The *xor*-1 and *negation* problems make use of direct input–output connections. The negation problem required the use of a secondary error term to encourage binary hidden unit activities and therefore allow accurate rule extraction as

Table 4
Summary of extraction performance on simple logic functions

Problem	Network topology	Hidden unit penalty term	Average p^*		Extraction error		Patterns correctly classified by rules (%)
			Hidden unit(s)	Output unit(s)	Hidden unit(s) (%)	Output unit(s) (%)	
Rule-plus-exception	4-2-1	–	2.72	6.15	0.8	1.3	100
xor-1	2-1-1	–	5.68	4.40	0.1	0.1	100
xor-2	2-2-1	–	4.34	5.68	0.4	1.0	100
Negation	4-3-4	Activation	5.40	5.17	0.2	2.2	100

follows:

$$E_{\text{act}} = \eta_{\text{act}} \sum_{\text{h}} (1 - o_{\text{h}}^2).$$

This error penalizes non-Boolean hidden activations for symmetric $(-1/+1)$ units; h is an index over the set of hidden units, and o_{h} denotes the activation of hidden unit h. For experiments utilizing E_{act}, the regularization parameter η_{act} was initialized at 0 and gradually increased throughout the course of the training using

$$\eta_{\text{act}} = 0.2\left(1 - \text{e}^{-\text{epoch}/1000}\right),$$

where 'epoch' denotes the current training epoch number.

Networks for these experiments were trained using the cross-entropy function (Hinton, 1989). Other simulation parameters were the same as those given in the previous section. Networks were trained on the complete pattern space of each function; rules were then extracted and tested against all the patterns. Table 4 summarizes the results over 10 replications of each problem with different initial weights. In addition to the perfect classification performance of the rules, the high values of p^* and low values of extraction error provide evidence that the extraction process is accurate.

Symbolic solutions for these problems often come in forms that differ from the canonical form of the function. For example, the extracted rules for the *rule-plus-exception* problem show a level of negation within the network as follows:

$$H_1 = OR\,(A, B, C, D)$$

$$H_2 = AND\,(A, B)$$

$$O = OR\,(\bar{H}_1, H_2)$$

In general the networks we tested make use of negation to form equivalent solutions. The extraction process easily captures these solutions, as long as the units conform the assumption of approximately Boolean activation behavior. The following three sets of rules are among some of the equivalent forms extracted for the *xor*-2 problem:

$$H_1 = AND\,(\bar{I}_1, \bar{I}_2) \quad H_1 = OR\,(\bar{I}_1, I_2) \quad H_1 = AND\,(I_1, I_2)$$

$$H_2 = OR\,(\bar{I}_1, \bar{I}_2) \quad H_2 = OR\,(I_1, \bar{I}_2) \quad H_2 = AND\,(\bar{I}_1, \bar{I}_2)$$

$$O = AND\,(\bar{H}_1, H_2) \quad O = OR\,(\bar{H}_1, \bar{H}_2) \quad O = AND\,(\bar{H}_1, \bar{H}_2)$$

Shown later are the three equivalent rule sets extracted for *xor*-1. Note that each solution demonstrates the use of an augmented *n*-of-*m* expression.

$$H = AND\,(\bar{I}_1, I_2)$$

$$O = OR\,[AND\,(I_1, \bar{I}_2), H]$$

$$H = OR\,(\bar{I}_1, I_2)$$

$$O = OR\,[AND\,(\bar{I}_1, I_2), \bar{H}]$$

$$H = OR\,(\bar{I}_1, \bar{I}_2)$$

$$O = AND\,[OR\,(I_1, I_2), H]$$

For the *negation* problem, we found that without the help of E_{act}, the network for this problem would sometimes solve the problem using hidden activations that were not completely Boolean, which causes the extraction process to fail. Shown here is an example of a set of rules extracted in such a case.

$H_1 = AND\,(I_1, \bar{I}_2, I_3)$ Extraction error = 8.4%

$H_2 = AND\,[OR\,(\bar{I}_2, \bar{I}_3), \bar{I}_1]$ Extraction error = 8.4%

$H_3 = AND\,(I_1, \bar{I}_4)$ Extraction error = 0.0%

$O_1 = OR\,[AND\,(\bar{I}_2, \bar{I}_3, I_4, H_1, \bar{H}_2, H_3), I_1]$ Extraction error = 6.7%

$O_2 = 2$ of $(\bar{I}_1, H_1, \bar{H}_2)$ Extraction error = 12.7%

$O_3 = AND\,[OR\,(I_1, \bar{I}_2, I_3), \bar{H}_1, \bar{H}_2]$ Extraction error = 6.7%

$O_4 = OR\,[AND\,(\bar{I}_1, I_4), H_3]$ Extraction error = 0.8%

The values for extraction error suggest that the rules do not match the original weights exactly. Nevertheless, the rules correctly account for 56 of the 64 output activities, much better than one would expect by chance. However, the rules correctly map only eight of the 16 total patterns. For comparison, following is a set of correct rules extracted from the negation network with the help of E_{act}:

$H_1 = OR\,(I_1, \bar{I}_4)$ Extraction error = 0.0%

$H_2 = OR\,(\bar{I}_1, \bar{I}_2)$ Extraction error = 0.4%

$H_3 = AND\,(I_1, \bar{I}_3)$ Extraction error = 1.4%

$O_1 = (I_1)$ Extraction error = 2.2%

$O_2 = AND\,[OR\,(I_1, I_2), H_2]$ Extraction error = 2.2%

$O_3 = OR\,[AND(\bar{I}_1, I_3), H_3]$ Extraction error = 0.1%

$O_4 = OR[AND\,(I_1, \bar{I}_4), \bar{H}_1]$ Extraction error = 0.1%

Given that the negation problem consists of three separate *xor* problems, we can see that these rules describe a solution. I_1 is the inversion control bit, passing straight through to O_1, while O_2, O_3, and O_4 implement *xor* functions between I_1

Table 5
Summary of the MONK's problems

Problem	Network topology	Hidden unit penalty term	Training set No of patterns	Performance of rules (%)	Test set No of patterns	Performance of rules (%)
Monks-1	17-3-1	Decay	124	100	432	100
Monks-2	17-2-1	Decay	169	100	432	100
Monks-3	17-0-1	–	122	93.4	432	97.2

and I_2, I_1 and I_3, and I_1 and I_4, respectively. The *xor* solutions are like those for *xor*-1 mentioned earlier.

Taken together, these simulations demonstrate the capabilities of our algorithm when applied to multilayer networks trained on the complete pattern space of a function. We turn next to the issue of generalization performance.

5.3. The MONK's problems

We tested the generalization performance on the MONK's problem, a set of three problems used to compare a variety of symbolic and connectionist learning algorithms (Thrun et al., 1991). These problems require binary classification over a six-attribute input space shown as follows:

x_1 : head_shape $\in$ {*round*, *square*, *octagon*}

x_2 : *body_shape* $\in$ {*round*, *square*, *octagon*}

x_3 : *is_smiling* $\in$ {*yes*, *no*}

x_4 : *holding* $\in$ {*sword*, *balloon*, *flag*}

x_5 : *jacket_color* $\in$ {*red*, *yellow*, *green*, *blue*}

x_6 : *has_tie* $\in$ {*yes*, *no*}

These attributes are based on an artificial robot domain and give rise to 432 possible robots. For each of the problems, a subset of 432 possible robots was used to provide the training data, and the complete set was used to test the generalization performance. The three MONK's problems are:

monks-1:
(*head_shape* = *body_shape*) or (*jacket_color* is *red*)
monks-2:
exactly 2 of the attributes have their first value
monks-3:
(*jacket_color* is *green* and *holding* is *sword*) or (*jacket_color* is not *blue* and *body_shape* is not *octagon*)

The *monks*-1 problem has 124 patterns in its training set, *monks*-2 has 169, and *monks*-3 has 122. Training sets for *monks*-1 and *monks*-2 contain no noise, but the training set for *monks*-3 contains six patterns that are misclassified, which amounts to approximately 5% training noise. In all cases, the test sets are noise-free.

We tested the MONK's problems using a $-1/+1$ input unit for each of the 17 possible attribute values. Training continued until the outputs for all patterns were within 0.05 of their target $-1/+1$ values. We made use of the grouped-input variant of our extraction algorithm described in Section 3.7.1. A group was associated with each of the six input attributes, having 3, 3, 2, 3, 4, and 2 weights, respectively. For *monks*-1 and *monks*-2, we also applied a selective weight decay to hidden weights, implemented using the additional error term

$$E_{\text{decay}} = \eta_{\text{decay}} \sum_{\text{g}} \sum_{i \neq i_{\text{g}}^{\max}} w_i^2,$$

where g is an index over weight groups and i is an index over all the weights within a group except the maximum absolute value weight, denoted $i_{\text{g}}^{\max}$. For *monks*-1, we used $\eta_{\text{decay}} = 0.1$. For *monks*-2, we used the following schedule:

$$\eta_{\text{decay}} = 50\left(1 - \text{e}^{-\text{epoch}/1000}\right).$$

The results are summarized in Table 5. Our performance was equal to or better than that of all of the systems tested by Thrun et al. (1991) for the *monks*-1 and *monks*-2 problems. Moreover, the rules extracted by our algorithm were very concise and easy to understand, in contrast to those produced by several other symbolic systems. (The two connectionist systems reported in Thrun et al., 1991, were opaque, i.e. no rules were extracted.) Consider the following output of our extraction technique for the *monks*-1 problem:

$H_1 = 2$ of (*head_shape square*, *body_shape square*, *jacket_color not red*)

$H_2 = 2$ of (*head_shape not round*, *body_shape not round*, *jacket_color red*)

$H_3 = 2$ of (*head_shape octagon*, *body_shape octagon*, *jacket_color not red*)

$O = 2$ of ($\bar{H}_1$, H_2, $\bar{H}_3$)

Recall that the target concept for this problem is: (*head shape* = *body shape*) or (*jacket color is red*). Given that there are only three possible values for head shape and body shape, the aforementioned rules can be seen to be a correct expression of the concept. All the rules extracted for this problem were simple variations on the set shown earlier.

Typical rules extracted for the monks-2 problem included

Table 6
Summary of performance on the promoter recognition task

Network topology	Training set No of patterns	Perf. of network	Perf. of rules	Test set No of patterns	Perf. of network	Pref. of rules
228-0-1	105	100.0%	95.9%	1	94.2%	87.6%

the following:

$H_1 = 2$ of (*head_shape round*, *body_shape round*, *is_smiling yes*, *holding sword*, *jacket_color red*, *has_tie yes*)

$H_2 = 3$ of (*head_shape round*, *body_shape round*, *s_smiling yes*, *holding sword*, *jacket_color red*, *has_tie yes*)

$O = AND\ (H_1, \bar{H}_2)$

All the rules extracted for this problem were similar in form to these. These rules demonstrate a rather elegant use of *n*-of-*m* expressions to describe the target concept. The rules express the idea of ''exactly 2'' by saying, in effect, ''at least 2 but not 3.'' This is as much as a human might describe the concept.

The *monks*-3 problem is difficult both because of the training noise and the nature of the underlying function. The function is a disjunction of two terms, but the terms are such that 420 of the 432 total patterns can be account for by the second disjunct: *jacket_color* is not *blue* and *body_shape* is not *octagon*. The other disjunct is required to map the remaining 12 patterns. The overall function is therefore similar in character to the *rule-plus-exception* problem discussed earlier. The training set for *monks*-3 includes only two of the 12 patterns mapped by the 'exception' disjunct. This makes for a difficult learning problem even in the absence of noise. While our networks had no difficulty learning the training patterns using hidden units, our generalization performance on the test set was poor, because the networks had learned the noise present in the training set. In his own experiments applying the connectionist networks to the *monks*-3 problem, Thrun et al. (1991) used weight decay to improve generalization. However, this still did not lead to interpretable weight configurations. Our solution was to use a network with no hidden units. Our network did not perfectly learn the training patterns, but it matched Thrun's generalization performance of 97.2%. It did this by picking the dominant disjunct out of the training noise; the rule extracted in every case was:

$O = AND$ (*jacket_color not blue*, *body_shape not octagon*).

The 12 patterns in the test set missed by this rule are precisely those patterns covered by the other disjunct: *jacket_color* is *green* and *holding* is *sword*.

The MONK's problems provide a good testing ground for exploring the generalization performance of our algorithm. The experiments in rule extraction described here are by no means an exhaustive treatment of these problems, but they do illustrate some of the general strengths and weaknesses of our approach. Results on the *monks*-1 and *monks*-2 problems are noteworthy because they show the power of *n*-of-*m* rules to capture equality of feature values and to express complex concepts such as ''exactly 2''.

A weakness of our method illustrated by these problems is the dependence on an extra error term for hidden units in multilayer configurations. (A similar dependence was illustrated by the *negation* problem of the previous section.) Weight decay was required for reliable rule extraction on *monks*-1 and *monks*-2. When we removed the weight decay, the network still learned the function, but the weights were not interpretable using our procedure. Interpretation problems also occurred when we used more hidden units. This experience points up the main disadvantage of our procedure: it is not effective when a network solves a problem using weight configurations that do not match those described by our weight templates. As demonstrated here, judicious use of extra error terms can often steer a network toward interpretable solutions, but in general the effectiveness of our procedure will be problem dependent.

5.4. Promoter recognition

The remaining three simulations address extraction performance on real-world databases from the University of California at Irvine machine learning repository (Murphy and Aha, 1994). The first of these is the promoter recognition task. This task was studied extensively by Towell and a good explanation of the problem may be found in Towell (1991). In brief, promoters are short DNA sequences that mark the beginnings of genes. Identifying the locations of genes (portions of DNA that are transcribed into proteins) is an important goal of molecular biology research. DNA sequences for which the promoter status is known make up the data set for this problem. The data includes 53 promoters and 53 non-promoters, for a total of 106 patterns. Each of the patterns represents a DNA sequence that is 57 elements long. Each element of the sequence may take on one of the values *A*, *C*, *G*, or *T*. In this problem the elements are indexed from -50 to $+7$, inclusive (An index of 0 is not used.) This allows subsequences within the overall sequence to be described using a convenient '@index' notation. For example, the description ''@-10 'A-TTG''' specifies the composition of the subsequence located between indices -10 and -14. (The ''$-$'' indicates an unspecified element.)

We coded each element of the sequences using a simple

Table 7
Summary of performance on the voting record task

Network topology	Training set No of patterns	Perf. of network	Perf. of rules	Test set No of patterns	Perf. of network	Pref. of rules
16-0-1	387	97.3%	96.2%	43	95.7%	95.9%

1-of-4 local representation, so there were 228 input units in our network for this task, divided into 57 natural groups of four. We achieved the best results with no additional penalty terms, no hidden units, and by restricting all weights to be positive. Training continued until all outputs were within 0.05 of their target values. We followed a leave-one-out cross-validation procedure: A total of 106 networks were trained, each on 105 patterns, with the remaining pattern in each case being used for generalization testing. Each network was trained 10 times using different initial weights, so a total of 1060 networks was tested.

Table 6 shows a summary of results, averaged across all 1060 simulations. It is interesting to note that a network with no hidden units can learn all the patterns. As shown by the table, such a network achieves 94.2% generalization performance without rule extraction. Rule extraction reduces the performance to 87.6%. This figure is less than the best performance (96%) of Towell (1991), but is nonetheless impressive when viewed in light of the simplicity and comprehensibility of our extracted rules. While Towell's results for this task included five rules like the one shown in Section 4.2, our single rule is quite simple:

promoter = 5 *of* (@−45 '*AA-------TTGA-A-----T------T-----AAA----C*')

This rule was extracted in 515 of the 1060 simulations we ran. A total of 28 unique rules was extracted over the course of the 1060 simulations. Most rules possessed many terms in common with the aforementioned 5-of-13 rule. For example, the next most frequently occurring rule (337 times) was:

promoter = 5 *of* (@−45 '*AA-------TTGA-A--T-T------T-----AA-----C*')

The results for this problem are a good illustration of the complexity and expressiveness differences between our method and Towell's method. Although our rule is less expressive, it is completely symbolic and very easy to understand.

5.5. Congressional voting records

The next task we studied was the congressional voting-record data base in the UCI machine learning repository. This task involved predicting the political party affiliation (Democrat or Republican) for members of the US House of Representatives based on 16 key votes taken during 1984. The votes are defined as follows:

$V_1 =$ Voted for handicapped-infants bill
$V_2 =$ Voted for water-project-cost-sharing bill
$V_3 =$ Voted for adoption-of-the budget bill
$V_4 =$ Voted for physician-fee-freeze bill
$V_5 =$ Voted for el-salvador-aid bill
$V_6 =$ Voted for religious-groups-in-schools bill
$V_7 =$ Voted for anti-satellite-test-ban bill
$V_8 =$ Voted for aid-to-nicaraguan-contras bill
$V_9 =$ Voted for mx-missile bill
$V_{10} =$ Voted for immigration bill
$V_{11} =$ Voted for synfuels-corporation-cutback bill
$V_{12} =$ Voted for education-spending bill
$V_{13} =$ Voted for superfund-right-to-sue bill
$V_{14} =$ Voted for crime bill
$V_{15} =$ Voted for duty-free-exports bill
$V_{16} =$ Voted for export-admin-south-africa bill

The attributes of this problem map directly into a Boolean input coding. As with the problem of the promoter, we achieved best results with no hidden units; our network was thus a 16-0-1 configuration. As a zero-hidden unit network cannot quite learn all the training patterns, training was stopped when error seemed to reach asymptote. Generalization testing was carried out using 10-fold cross validation. In this method, the data set is divided into 10 groups, with nine used for training and the remaining group for testing. As the experiment is performed 10 times, each group is used once as the test set. Additionally, the grouping itself is repeated 10 times to eliminate dependencies on the initial ordering of the patterns. Finally, in our case each network was trained and tested 10 times with different initial random weights. Thus 100 networks were trained and tested 10 times each for this experiment. To make the calculation easier, a subset of 430 of the 435 patterns in the database was used for all the experiments here. This means that each training set had 387 patterns and each test set had 43 patterns.

Table 7 summarizes the simulation results for this problem. The table indicates that a very good performance is achieved for this problem without the use of hidden units. The average value of p^* for these simulations was 1.105, and the average value of the extraction error was 13.15%. The rule most frequently extracted for this problem is:

$$Democrat = OR[5 \text{ of } (V_3, \overline{V}_7, V_9, \overline{V}_{10}, V_{11}, \overline{V}_{12}), \overline{V}_4].$$

This rule indicates a strong link between the party affiliation and the vote on physician fee freezes. In fact, an inverse dependence on V_4 was exhibited in all of the 49 unique rules extracted for this problem. This is not surprising, as the class predictiveness values for this attribute – the probability that the affiliation is Republican given a yes vote, and the probability that the affiliation is Democrat given a no vote – are

Table 8
Encoding of features for the breast-cancer problem

Attribute value	Inputs								
	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9
1	−1	−1	−1	−1	−1	−1	−1	−1	−1
2	1	−1	−1	−1	−1	−1	−1	−1	−1
3	1	1	−1	−1	−1	−1	−1	−1	−1
4	1	1	1	−1	−1	−1	−1	−1	−1
5	1	1	1	1	−1	−1	−1	−1	−1
6	1	1	1	1	1	−1	−1	−1	−1
7	1	1	1	1	1	1	−1	−1	−1
8	1	1	1	1	1	1	1	−1	−1
9	1	1	1	1	1	1	1	1	−1
10	1	1	1	1	1	1	1	1	1

0.92 and 0.99, respectively. The strength of the V_4 link is reflected by the fact that it stands alone as a term inside an augmented *n*-of-*m* expression. According to the template definitions for these expressions, it carries a weight five times that carried by the items in the 5-of-6 term of the augmented expression.

These results are noteworthy because the generalization performance of the rules is virtually identical to that of the raw network. This indicates that the extracted rules are capturing a significant portion of the computation being performed by the network.

5.6. Breast cancer diagnosis

Our final simulation involved the UCI repository breast cancer (data base courtesy by Dr. William H. Wolberg; see also Mangasarian and Wolberg, 1990). The goal of this task is to determine if a tumor is benign or malignant based on the following attributes: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses. In the UCI data base, each of these attributes is nominally coded with integer values between 1 and 10 inclusive. Instead of using a standard local representation for each attribute, we used a thermometer coding (Table 8). This coding is advantageous because it allows less-than or greater-than conditions for an attribute to be detected with a single weight in the attribute group. For example, for the network to learn the condition *clump thickness* > 6, it merely has to develop a positive weight associated with input number 6. (Under a purely local representation, this would require four weights, one for each of inputs 6–9.) This property is consistent with the template selection method based on input groupings. Additionally, using an input representation which facilitates less-than or greater-than conditions seems like an appropriate bias for a diagnosis task such as this.

Applying the aforementioned encoding to each of the nine attributes for this task results in 81 inputs. We obtained the best results by using three hidden units and by restricting the hidden weights to be positive. However, a penalty for hidden unit activations or weight decay was not required. Training continued until the training error appeared to asymptote. Generalization testing consisted of 10-fold cross validation as described for the voting-record task, using 630 patterns for training and 70 patterns for testing.

A summary of our results on the breast-cancer diagnosis task is given in Table 9. As an example of the rules extracted for this task, consider the following:

$H_1 = 4$ of (*thickness* > 3, *size* > 1, *adhesion* > 2, *epithelial* > 5, *nuclei* > 3, *chromatin* > 1, *normal* > 2, *mitoses* > 1)
$H_2 = 3$ of (*thickness* > 6, *size* > 1, *shape* > 1, *epithelial* > 1, *nuclei* > 8, *normal* > 9)
$O = AND\ (H_1, H_2)$

These rules were extracted with p^* values of approximately 1.8, 3.6, and 2.8, respectively. They account for 674 (or 96%) of the 700 total patterns. In approximately 20% of the simulations, including this run, H_3 did not figure into the overall rule. No single set of rules dominated for this problem, but many of the rules extracted over different training sets differed only slightly in their use of terms involving the various attributes. For example, instead of specifying *thickness* > 3, a rule might specify *thickness* > 4.

As with the voting-record task, these results are significant because the generalization performance of the rules is very close to that of the raw network, indicating that the

Table 9
Summary of performance on the breast-cancer diagnosis task

Network topology	Training set			Test set		
	No of patterns	Perf. of network	Perf. of rules	No of patterns	Perf. of network	Pref. of rules
81-3-1	630	98.5%	96.3%	70	95.8%	95.2%

rule-based description is capturing most of the computation being performed by the network.

5.7. Discussion of simulation results

We have presented a collection of simulations performed using our template-based rule extraction algorithms. The domains included: randomly-generated *n*-of-*m* and augmented *n*-of-*m* functions, simple logic problems, a set of learning benchmarks known as the MONK's problems, and three problems from the UCI machine learning repository. Together these problems represent a good range of learning tasks. For each problem, we gave examples of the kinds of rules extracted and discussed conditions under which the extracted rules provided an accurate description of network behavior.

Several conclusions can be reached from the results of our simulations. The algorithms appear to be very effective on single-layer networks. This is not surprising, as these networks must make use of Boolean activations at the outputs in order to solve the classification problems. The algorithms also seem effective on highly constrained multilayer networks (e.g. networks that have an appropriate number of hidden units and that make use of appropriate secondary error terms). Our approach breaks down in multilayer networks when the hidden units do not take on Boolean activations. This problem aside, the simulations demonstrate the feasibility of our approach to connectionist rule extraction, and provide compelling evidence that our approach can be used to interpret and understand the behavior of a neural network.

6. Conclusion and future directions

This article has presented a general approach for extracting various types of *n*-of-*m* symbolic rules from trained networks of sigmoidal units, assuming approximately Boolean activation behavior. Our approach makes several contributions to the study of neural network interpretation and rule extraction. First, we have made a principled extension to the template-based approach of McMillan, giving it a more rigorous mathematical treatment. Second, we have demonstrated the flexibility of our general approach by showing how the basic extraction algorithm may be adapted to the particular needs of an application domain. Third, we have compared our method of rule extraction to that presented by Towell, concluding that the two methods embody different choices in the trade off between complexity and expressiveness. The symbolic rules produced by our method are less expressive but also considerably less complex than the mixed symbolic and numeric rules produced by Towell's method. Simulations conducted with our algorithms demonstrated their overall usefulness as well as the general strengths and weaknesses of our approach.

The simulations reported in this work used rule extraction after training error reached asymptote. One promising possibility that we did not explore is the idea of extracting rules during training, and then reinjecting them back into the network for further training. McMillan (1992) reported the use of such an iterative extraction technique and determined that this method slightly improved the ultimate generalization performance of his system. The appeal of extracting and reinjecting rules during training is that the extraction operation provides a way of taking a step (through weight space) not based on the gradient of the error function. It thus has the potential of altering the network's path through weight space as well as its eventual destination. If extraction/reinjection is performed at the right times during training, it may help constrain a network, biasing it toward solutions that are for the most part symbolic. Determining exactly when to perform the extractions in such a scheme would be a subject for further investigation. One could extract at some uniform interval during training based on epoch counts. Another approach would be to extract based on how the value of the error function was changing during training. Yet another idea would be to perform an extraction fairly regularly, but only reinject the extracted rules if criteria for p^* and $E_{\text{extraction}}$ were met.

Jordan (1998) has suggested that the sigmoid belief networks of Neal (1991) may offer a principal alternative to introducing penalty terms when Boolean unit behavior is desired. An interesting direction would be to apply template-based extraction techniques to such belief networks. Jordan has also pointed out that our weight templates can be viewed as special cases of certain invariances (syzygies) in the field of symbolic algebra. Exploring this relationship in detail could provide a stronger theoretical basis for weight templates, and could lead to additional insights on their uses and limitations in connectionist systems.

We began this article with a discussion of opacity in the connectionist networks. In all likelihood, no one solution exists to this challenging problem, but template-based techniques provide a broad set of tools that can be tailored to a task domain. Advances are clearly being made in analyzing the behavior of networks through rule extraction. An important contribution of our work has been to show that rule-extraction techniques are approaching a level of sophistication, where they will be useful in interpreting the behavior of networks on challenging, real-world problems.

Acknowledgements

Our thanks to Michael I. Jordan, James Martin, Paul Smolensky, and Clayton McMillan for their comments and assistance with this work. JA was supported by a Resident Fellowship from the Hewlett-Packard Company. MM was supported by grant 97-18 from the McDonell-Pew Program in Cognitive Neuroscience, NSF award IBN-9873492, and by Sensory, Inc.

Appendix A

This appendix presents the two theorems referred in the body of the article. These theorems allow for large regions of the search space of ***n***-of-***m*** expressions to be pruned from further consideration based on simple attributes of the actual weight vector being interpreted. The theorems are important because they show that this pruning is mathematically justified rather than being based on a heuristic.

A.1. Theorem 1

This Theorem demonstrates how knowledge of the signs of the actual weights may be applied to the extraction process to reduce the number of templates that must generated.

Begin with an actual weight vector **w** and two generalized *n*-of-*m* templates **t** and **s**:

$$\mathbf{w} = w_1\ w_2\ \ldots\ w_k w_{k+1},$$

$$\mathbf{t} = t_1\ t_2\ \ldots\ t_k\ t_{k+1},$$

$$\mathbf{s} = s_1\ s_2 \ldots s_k\ s_{k+1}.$$

In this notation, items subscripted with '$k+1$' indicate bias values. Let each of the weights in **t** be represented as $t_i = u_i p$, where p is by definition positive. For **t** to be a valid *n*-of-*m* template, m of the u_i values ($i < k+1$) must be ± 1 and $k-m$ of the u_i values must be 0. In addition, u_{k+1} must be equal to $m+1-2n$. For each of the non-zero u_i, let $u_i = sign(w_i)$.

Similarly, let each of the weights in **s** be represented by $s_i = v_i p$, where each of the m non-zero v_i values (including the bias value v_{k+1}) are such that either $v_i = u_i$ or $v_i = -u_i$. Let j be an index over the cases where $v_i = -u_i$, and let these cases number at least one. Then **t** and **s** are identical except that some of the non-zero weights in **s** have the opposite sign of their counterparts in **t**.

The conjecture is that once the templates **t** and **s** are instantiated with values of p, **t** will always be closer to **w** than **s**. In mathematical terms, this relationship is expressed as

$$\|\mathbf{t}-\mathbf{w}\|^2 < \|\mathbf{s}-\mathbf{w}\|^2.$$

To show that this is indeed the case, note that

$$\|\mathbf{t}-\mathbf{w}\|^2 = d_{\mathbf{t}} = \sum_{i=1}^{k+1} (u_i p - w_i)^2 = \sum_{i=1}^{k+1} (sign(w_i)|u_i|p - w_i)^2,$$

and

$$\|\mathbf{s}-\mathbf{w}\|^2 = d_{\mathbf{s}} = \sum_{i=1}^{k+1} (v_i p - w_i)^2$$

$$= \sum_{i=1}^{k+1} (sign(w_i)|v_i|p - w_i)^2 + 4p \sum_j |v_j||w_j|.$$

The distances $d_{\mathbf{t}}$ and $d_{\mathbf{s}}$ can be minimized by setting their derivatives with respect to p to zero. This yields the following optimal values of p:

$$p_{\mathbf{t}} = \frac{\sum_{i=1}^{k+1} |w_i||u_i|}{\sum_{i=1}^{k+1} u_i^2},$$

$$p_{\mathbf{s}} = \frac{\sum_{i=1}^{k+1} |w_i||v_i| - 2\sum_j |w_j||v_j|}{\sum_{i=1}^{k+1} v_i^2}.$$

When these values of $p_{\mathbf{t}}$ and $p_{\mathbf{s}}$ are substituted into $d_{\mathbf{t}}$ and $d_{\mathbf{s}}$, respectively, the difference between $d_{\mathbf{t}}$ and $d_{\mathbf{s}}$ reduces to

$$d_{\mathbf{s}} - d_t = \frac{4\left(\sum_j |w_j||v_j|\right)\left(\sum_{i=1}^{k+1} |w_i||v_i| - \sum_j |w_j||v_j|\right)}{\sum_{i=1}^{k+1} u_i^2}.$$

As the weights indexed by j always form a subset of the complete set of $k+1$ weights, the difference $d_{\mathbf{s}} - d_{\mathbf{t}}$ is always positive,[2] meaning that **t** is always closer to **w** than is **s**. The Theorem is complete. Note also that the aforementioned equation for $p_{\mathbf{t}}$ guarantees a positive value, as required by the template definitions.

A.2. Theorem 2

This Theorem demonstrates how knowledge of the magnitudes of the actual weights may be applied to the extraction process to further reduce the number of candidate templates.

Begin with an actual weight vector **w** and two generalized *n*-of-*m* templates **t** and **s**:

$$\mathbf{w} = w_1\ w_2 \ldots w_k w_{k+1},$$

$$\mathbf{t} = t_1\ t_2 \ldots t_k\ t_{k+1},$$

$$\mathbf{s} = s_1\ s_2 \ldots s_k\ s_{k+1}.$$

As in Theorem 1, items subscripted with '$k+1$' indicate bias values. Let the weights in **t** and **s** be represented by $t_l = u_l p$ and $s_l = v_l p$, respectively. In accordance with Theorem 1, the signs of any non-zero u_l and v_l are equal to the signs of the corresponding w_l. Let the m non-zero u_l ($l < k+1$) in **t** correspond to the m highest absolute value actual weights in **w**, and let the m non-zero v_l in **s** correspond to any other set of weights in **w**. Also in accordance with Theorem 1, let $u_{k+1} = v_{k+1} = m+1-2n$ for some value n such that $sign(u_{k+1}) = sign(v_{k+1}) = sign(w_{k+1})$. Templates **t** and **s**

[2] Note that when *all* the weights in **s** are inverted from their counterparts in **t** (i.e. j spans the complete set of $m+1$ weights), the difference $d_{\mathbf{s}} - d_{\mathbf{t}}$ is zero. This is not surprising, as in that case $p_{\mathbf{s}} = -p_{\mathbf{t}}$, and when this value is substituted into **s** the templates **s** and **t** become identical.

thus reflect the same values of n and m, and they are both consistent with the results of Theorem 1. They differ in that their m terms involve different sets of inputs.

For convenience in what follows, we use four subscripts to index over subsets of the $k+1$ elements in **w**,**t**, and **s**:

i indexes over the $m+1$ non-zero weights in **t**;
j indexes over the $m+1$ non-zero weights in **s**;
i^* indexes over the $k-m$ zero weights in **t**;
j^* indexes over the $k-m$ zero weights in **s**.

In this scheme i and i^* together include all the $k+1$ elements, as do j and j^* together.

The claim we wish to prove is that once the templates **t** and **s** are instantiated with the values of p, the following relationship is guaranteed:

$$\|\mathbf{t}-\mathbf{w}\|^2 < \|\mathbf{s}-\mathbf{w}\|^2.$$

To show that this relationship holds, we follow a similar course of action as in Theorem 1. The distances $d_\mathbf{t}$ and $d_\mathbf{s}$ are given as:

$$d_\mathbf{t} = \sum_{l=1}^{k+1} (u_l p - w_l)^2 = \sum_{l=1}^{k+1} (|u_l| p - |w_l|)^2,$$

and

$$d_\mathbf{s} = \sum_{l=1}^{k+1} (v_l p - w_l)^2 = \sum_{l=1}^{k+1} (|v_l| p - |w_l|)^2.$$

Using the subscripts described earlier, optimal values of p for **t** and **s** are:

$$p_\mathbf{t} = \frac{\sum_i |w_i||u_i|}{\sum_i u_i^2},$$

$$p_\mathbf{s} = \frac{\sum_j |w_j||v_j|}{\sum_j v_j^2}.$$

As **t** and **s** reflect the same values of n and m, we can make the following simplification:

$$D = \sum_i u_i^2 = \sum_j v_j^2.$$

The values of $d_\mathbf{t}$ and $d_\mathbf{s}$ are then

$$d_\mathbf{t} = \sum_i (|u_i| p - |w_i|)^2 + \sum_{i^*} w_{i^*}^2,$$

$$d_\mathbf{s} = \sum_j (|v_j| p - |w_j|)^2 + \sum_{j^*} w_{j^*}^2.$$

When $p_\mathbf{t}$ and $p_\mathbf{s}$ are substituted into $d_\mathbf{t}$ and $d_\mathbf{s}$, respectively, the difference between $d_\mathbf{t}$ and $d_\mathbf{s}$ reduces to

$$d_\mathbf{s} - d_\mathbf{t} = \frac{\left(\sum_i |w_i||u_i|\right)^2 - \left(\sum_j |w_j||v_j|\right)^2}{D}.$$

This difference is always positive, by the following reasoning: The subscripts i and j index the same number of non-zero weights ($m+1$ in each case). For each non-zero weight indexed by i or j, $|u_i| = |v_j|$. By definition the sum of the $|w_i|$ values exceeds that of the $|w_j|$ values. The difference of the squares of these sums is then positive, and D is also positive. Thus, $d_\mathbf{s}$ is always greater than $d_\mathbf{t}$ and the claim is proven.

References

Bridle, J. S. (1989). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fogelman-Soulie & J. Herault (Eds.), *Neuro-computing: algorithms, architectures and applications*, (pp. 227). Berlin: Springer.

Dinsmore, J. (Ed.) (1992). *The symbolic and connectionist paradigms: closing the gap*. Hillsdale, NJ: Erlbaum (Lawrence).

Dolan, C. P., & Smolensky, P. (1989). Tensor product production system: a modular architecture and representation. *Connection Science*, *1*, 53–68.

Fu, L. M. (1989). Integration of neural heuristics into knowledge-based inference. *Connection Science*, *1*, 325–340.

Fu, L.M. (1991). Rule learning by searching on adapted nets. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 590–595).

Gallant, S. I. (1988). Connectionist expert systems. *Communications of the ACM*, *31*, 152–169.

Hayashi, Y. (1990). A neural expert system with automated extraction of fuzzy if-then rules and its application to medical diagnosis. In R. P. Lippmann & J. E. Moody & D. S. Touretzky (Eds.), (pp. 578). *Advances in neural information processing systems*, 3. San Mateo, CA: Kaufmann (Morgan).

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, *40*, 185–234.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). Induction: processes of inference, learning, and discovery. Cambridge, MA: MIT Press.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, *3*, 79–87.

Jordan, M.I. (1998). Personal communication.

Mangasarian, O. L., & Wolberg, W. H. (1990). Cancer diagnosis via linear programming. *SIAM News* , *23* (5), 18.

McMillan, C., Mozer, M. C., & Smolensky, P. (1991). Rule induction through integrated symbolic and subsymbolic processing. In J. E. Moody & S. J. Hanson & R. P. Lippmann (Eds.), (pp. 969). *Advances in neural information processing systems*, 4. San Mateo, CA: Kaufmann (Morgan).

McMillan, C. (1992). Rule induction in a neural network through integrated symbolic and subsymbolic processing. Unpublished Ph.D. Thesis. Department of Computer Science, University of Colorado, Boulder, CO.

Minsky, M. (1991). Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, Summer 1991, pp. 35–51.

Mozer, M. C., & Smolensky, P. (1988). Skeletonization: atechnique for trimming the fat from a network via relevance assessment. In D. S. Touretzky (Ed.), (pp. 107). *Advances in neural information processing Systems*, 1. San Mateo, CA: Kaufmann (Morgan).

Murphy, P.M., Aha, D.W. (1994). *UCI repository of machine learning databases*. [Machine-readable data repository.] Irvine, CA: University of California, Department of Information and Computer

Science. Monks data courtesy of Sebastian Thrun, promoters data courtesy of M. Noordewier, J. Shavlik, congressional voting data courtesy of Jeff Schlimmer, breast cancer data courtesy of Dr William H. Wolberg.

Neal, R. (1991). Unpublished doctoral dissertation.

Nowlan, S. J., & Hinton, G. E. (1991). Adaptive soft weight tying using gaussian mixtures. In J. E. Moody & S. J. Hanson & R. P. Lippmann (Eds.), (pp. 993). *Advances in neural information processing systems*, 4. San Mateo, CA: Kaufmann (Morgan).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland & the PDP Research Group (Eds.), (pp. 318). *Parallel distributed processing: explorations in the microstructure of cognition*, 1. Cambridge, MA: MIT Press Foundations.

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

Sun, R. (1991). Connectionist models of rule-based reasoning. *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*. Hilsdale, NJ: Erlbaum (Laurence) pp. 437–442.

Thrun, S.B. and 23 other authors (1991). *The MONK's problems – a performance comparison of different learning algorithms*. Technical Report CS-CMU-91-197. Carnegie Mellon University, Pittsburgh, PA.

Touretzky, D. S., & Hinton, G. E. (1998). A distributed connectionist production system. *Cognitive Science*, *12*, 423–466.

Towell, G. (1991). Symbolic knowledge and neural networks: Insertion, refinement and extraction. Ph.D. Thesis. Department of Computer Science, University of Wisconsion, Madison, WI.

Towell, G., & Shavlik, J. W. (1991). Interpretation of artificial neural networks: mapping knowledge-based neural networks into rules. In J. E. Moody & S. J. Hanson & R. P. Lippmann (Eds.), (pp. 977). *Advances in neural information processing systems*, 4. San Mateo, CA: Kaufmann (Morgan).