SYMBOLIC KNOWLEDGE AND NEURAL NETWORKS: INSERTION REFINEMENT AND EXTRACTION

by

Geoffrey G. Towell

Computer Sciences Technical Report #1072

January 1992

SYMBOLIC KNOWLEDGE AND NEURAL NETWORKS: INSERTION, REFINEMENT AND EXTRACTION

by
Geoffrey G. Towell

B.A., Hamilton College, 1983
M.S., University of Wisconsin — Madison, 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy
(Computer Science)

at the
UNIVERSITY OF WISCONSIN — MADISON
1991

To Betsy, a dedicated "morning person", for getting me up and keeping me going throughout this long process.

Contents

ACKNOWLEDGEMENTS

As with any document that is a long time in the writing, there are probably more people to thank than there are pages. So, rather than trying to thank everyone, I mention only three groups of people.

The first group includes Richard Maclin, Eric Gutstein, and Charles Squires Jr. Fellow grad students, they all took time out from their own work to review, and substantially improve, this thesis.

The second group consists of Mike Litzkow and everyone involved in the development of Condor [Litzkow88]. Condor is a system which allows jobs to run on any idle workstation in the department. (Even as I am writing this sentence, I have tasks executing on more than 40 machines.) Much of the work in this thesis was only possible because of Condor; during the final 6 months of work on my thesis, I used more than three CPU years of computer time through the Condor system.

Finally, my advisor, Jude Shavlik and my thesis committee: Richard Lehrer (reader), Charles Dyer (reader), Leonard Uhr (nonreader) and Olvi Mangasarian (nonreader). Having found the light at the end of the tunnel that is my thesis, I see that Jude consistently guided me along a course towards that light. To my amazement, the committee saw that same light despite my best efforts at obfuscation.

In addition to thanking individuals, this work is partially supported by Office of Naval Research Grant N00014-90-J-1941, National Science Foundation Grant IRI-9002413, Department of Energy Grant DE-FG02-91ER61129 and the University of Wisconsin Graduate School.

ABSTRACT

Explanation-based and empirical learning are two largely complementary methods of machine learning. These approaches to machine learning both have serious problems which preclude their being a general purpose learning method. However, a "hybrid" learning method that combines explanation-based with empirical learning may be able to use the strengths of one learning method to address the weaknesses of the other method. Hence, a system that effectively combines the two approaches to learning can be expected to be superior to either approach in isolation. This thesis describes a hybrid system called KBANN which is shown to be an effective combination of these two learning methods.

KBANN *(Knowledge-Based Artificial Neural Networks)* is a three-part hybrid learning system built on top of "neural" learning techniques. The first part uses a set of approximately-correct rules to determine the structure and initial link weights of an artificial neural network, thereby making the rules accessible for modification by neural learning. The second part of KBANN modifies the resulting network using essentially standard neural learning techniques. The third part of KBANN extracts refined rules from trained networks.

KBANN is evaluated by empirical tests in the domain of molecular biology. Networks created by KBANN are shown to be superior, in terms of their ability to correctly classify unseen examples, to a wide variety of learning systems as well as techniques proposed by experts in the problems investigated. In addition, empirical tests show that KBANN is robust to errors in the initial rules and insensitive to problems resulting from the presence of extraneous input features.

The third part of KBANN, which extracts rules from trained networks, addresses a significant problem in the use of neural networks — understanding what a neural network learns. Empirical tests of the proposed rule-extraction method show that it simplifies understanding of trained networks by reducing the number of: consequents (hidden units), antecedents (weighted links), and possible antecedent weights. Surprisingly, the extracted rules are often more accurate at classifying examples not seen during training than the trained network from which they came.

Chapter 1

INTRODUCTION

Suppose you are tying to teach someone (named Betsy) who has never seen some class of objects (e.g., cups) to recognize the members of that class. One approach is to tell Betsy everything about the category. That is, you could state a "domain theory" [1] that describes how to recognize individual, critical components of the class members and how those components interact. Using this domain theory, she could then distinguish between members and nonmembers of the class. For instance, in teaching Betsy recognize cups, you might have her learn: what a handle is, how material that a cup of made of affects its properties, and the utility of the picture on the side of some cups (i.e., none). Using this information, she could determine what is, and is not, a cup.

A different approach to teaching Betsy to recognize a class of objects is to show her lots of examples. As each example is presented, you would tell Betsy whether the example is, or is not a member of the class, and nothing else. After seeing sufficient examples, she could classify new examples by comparison to those already seen.

A more reasonable way of teaching is to combine these two approaches. That is, teach Betsy to recognize cups by showing examples and providing a domain theory. She could then use examples to fill gaps in the theory and theory to fill gaps in the set of examples.

The first two methods of teaching roughly characterize the two main approaches to machine learning: *explanation-based learning* [DeJong86, Mitchell86] and *empirical learning* [Michalski83, Quinlan86, Mitchell82, Rumelhart86, Holland86b]. Explanation-based learning corresponds to teaching by giving a person a domain theory without an extensive set of examples. Conversely, empirical learning involves giving a person lots of examples without any explanation of why the examples are members of a particular class. Unfortunately, for reasons described in the following section, neither of these approaches to machine learning is completely satisfactory.

[1] A *domain theory* is a collection of rules that describes the interactions of facts in a system. For classification problems, a domain theory can be used to prove whether or not an object is a member of the class in question.

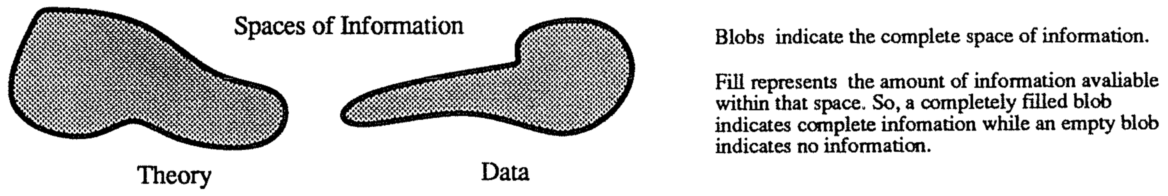

Spaces of Information
Theory
Data
Blobs indicate the complete space of information.
Fill represents the amount of information avaliable within that space. So, a completely filled blob indicates complete infomation while an empty blob indicates no information.

Figure 1.1: The two spaces of information.

They each suffer from flaws that preclude either from being a general method of learning.

The flaws of each learning method are, for the most part, complementary. Hence, a "hybrid" system that effectively combines learning from theory with learning from data might be like the hypothetical student Betsy taught using the combination of theoretical information and examples. Given both kinds of information, she would have been able to combine it such that the whole was greater than the sum of its parts. Similarly, a hybrid learning system can be expected to find synergies that make it more effective that either purely empirical or explanation-based learning systems.

The KBANN *(Knowledge-Based Artificial Neural Networks)* system described and evaluated in this thesis is such a system. Briefly, the approach taken by KBANN is to *insert* a set of symbolic rules into a neural network. The network is then *refined* using standard neural learning algorithms and a set of classified training examples. Finally, symbolic rules that reflect modifications to the initial rules as a result of neural learning are *extracted* from the network.

Empirical tests in Chapter 3 show that KBANN benefits from its combination of explanation-based and empirical learning. The result is that KBANN is better at classifying examples not seen during training than methods that learn purely from examples as well as methods which, like KBANN, learn from both theory and examples (see Section 3.2).

1.1 Learning from Theory and Data

Consider dividing information about a given task into two non-overlapping parts – theory and data – as represented by Figure 1.1. The theory part contains only rules that describe how pieces of information fit together. Thus, the theory part might contain rules that define what it is to be a cup. The data section contains only instances of objects labeled by their category. Hence, it might hold lots of examples of cups and non-cups. This theory/data split is exactly the division assumed by explanation-based and empirical learning systems.

1.1.1 Explanation-based learning

The information used by explanation-based learning (EBL) systems is shown in Figure 1.2. That is, EBL systems learn almost exclusively from theoretical knowledge of a problem —

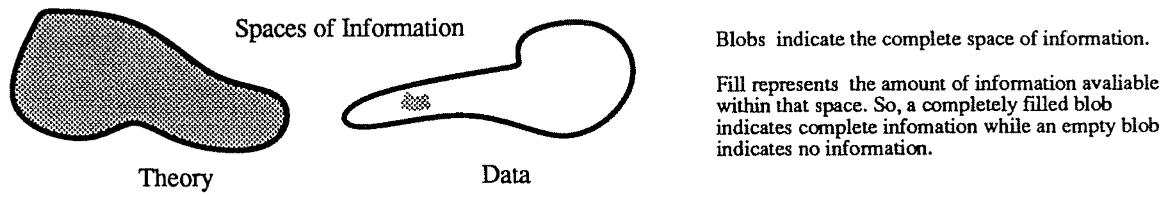

Spaces of Information
Theory
Data
Blobs indicate the complete space of information.
Fill represents the amount of information avaliable within that space. So, a completely filled blob indicates complete infomation while an empty blob indicates no information.

Figure 1.2: Information used by explanation-based learning systems.

knowledge is assumed to be complete and correct. (This assumption is made only by basic EBL algorithms.) The principle advantage of EBL is that it requires very little data; often a single example is sufficient for learning. However, EBL systems are fraught with difficulties, of which the following is a partial list:

1. *The basic EBL algorithms assume that the domain theory is both complete and correct* [Mitchell86]. Thus, anything not meeting the definition of a cup is not a cup and anything meeting the definition must be a cup. However, writing domain theories is very difficult because the task often requires formalizing "functional categories" (i.e., categories defined only in terms of their functions [Brunner56]). Furthermore, in many domains writing complete and correct domain theories may not be merely difficult, but impossible [Bennett88, Rajamoney87]. For instance, it is impossible to make a perfect model of motion for a robot because the world constantly changes.

2. *Basic EBL systems do not learn at the "knowledge level"* [Dietterich86]. The knowledge level is an abstraction which captures everything an entity knows how to do. (I.e., the knowledge level encompasses the deductive closure of an entity's knowledge.) Hence, when an entity learns a new way to represent something that it already knew, it has not learned at the knowledge level. The implication of this for EBL systems is that they cannot correct mistakes in their initial theory. So, they are forever doomed to repeat their mistakes.

3. *Domain theories can be "brittle"* [Holland86a]. That is, they only apply to a very narrow domain with very sharp boundaries. As a result, correct application of knowledge does not gracefully degrade as the boundaries are crossed, but drops immediately to zero. Unfortunately, it can be difficult to make EBL systems aware of having crossed the boundaries of their knowledge. As a result, systems that are normally reliable may provide incorrect answers without warning.

4. *Domain theories can be intractable to use* [Mitchell86, Rajamoney87]. Constructing proofs (which is essentially how EBL systems make decisions) using a domain theory may exceed time and/or memory bounds of the computer. Thus, an EBL system may

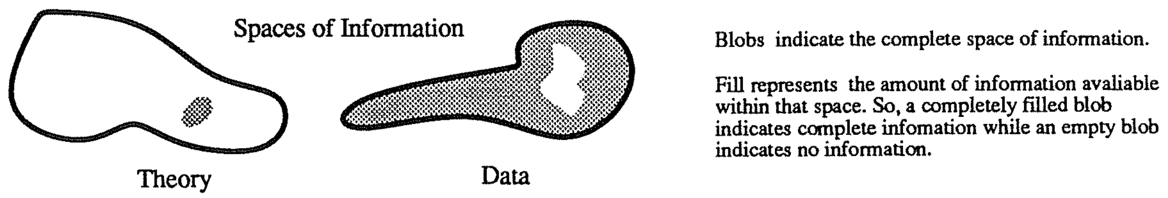

Spaces of Information
Theory
Data
Blobs indicate the complete space of information.
Fill represents the amount of information avaliable within that space. So, a completely filled blob indicates complete infomation while an empty blob indicates no information.

Figure 1.3: Knowledge assumed by empirical learning systems.

be unable to provide an answer despite having the knowledge to correctly answer the question at hand. For instance, given a board position and a proposed move, an EBL system for chess could potentially determine if is that move is optimal. However, doing so is computationally impossible.

5. *Domain theories must be supplied by some outside agency.* The creation of a domain theory suitable for use in an EBL system is itself, a significant learning problem [Pazzani88]. More generally, domain theory construction is one of the significant limiting factors in the development of "expert systems" [Waterman86].

1.1.2 Empirical learning

Empirical learning (EL) systems learn by generalizing from examples. Thus, as represented by Figure 1.3 they require little theoretical knowledge; instead they require a large, but possibly incomplete, library of examples. Their complete ignorance of theory has several disadvantages. Some of the most significant are:

1. *Spurious correlations in the examples can lead to incorrect classifications.* For instance, if all of the cups a learner has ever seen are colored red, it might conclude that all cups must be colored red. This is the classic problem of induction [Goodman83].

2. *Even when a large set of examples are available, small sets of exceptions may be either unrepresented or very poorly represented.* As a result, datasets with many "small disjuncts" tend to be poorly handled by EL systems [Holte89, Rendell90]. (Techniques are being developed to help empirical learning systems handle small disjuncts [Quinlan91].)

3. *Features relevant to classification are context dependent* [Schank86]. For example, that a $10 bill is made of flammable paper is unimportant when the bill is in your hand, but is quite significant when it is in a bank that is on fire.

4. *An unbounded number of features can be used to describe any object.* As a result, any two objects can appear similar or different by considering appropriate sets of features [Wantanabe69, the theorem of the ugly duckling]. This point implies that, at a minimum,

empirical learning systems require knowledge that indicates the relevant features. So, the small blob in the theory space of Figure 1.3 represents knowledge of the features that are probably necessary and sufficient for learning the solution to a problem.

5. *Irrelevant features in the description of examples can negatively affect learning* [Breiman84]. For example, consider the problem of seeing only red cups discussed above in the context of spurious correlations. The color of a cup is irrelevant, but spurious correlations between positive examples of cups and their color make red appear to be relevant. The presence of some irrelevant features in the description of an example is generally unavoidable since the set of relevant features may not be known. However, as the number of irrelevant features increases, so does the probability that there will be a spurious correlation making an irrelevant feature appear relevant.

6. *Complex features that can be constructed from the initial features may considerably simplify learning* [Rendell90]. EL systems must either independently discover these derived features or do without them. As most EL systems operate using some sort of hill-climbing heuristic, they may be unable to discover derived features that are only valuable when fully constructed.

1.1.3 Artificial neural networks

Artificial neural networks (ANNs), which form the basis of KBANN, are a particular method for empirical learning. ANNs have proven to be equal, or superior, to other empirical learning systems over a wide range of domains, when evaluated in terms of their ability to correctly classify examples not seen during training (i.e., their ability to generalize) [Shavlik91, Fisher89, Weiss89, Atlas89, Dietterich90, Tsoi90, Ng90]. However, they have a set of problems unique to their style of empirical learning. Among these problems are:

1. *Training times are lengthy.* Experiments [Shavlik91, Fisher89, Weiss89] indicate that ANNs require 100 to 1000 times as much training time as some symbolic learning algorithms (e.g., ID3 [Quinlan86]). That is, ANNs learn with the ponderous grace of a glacier.

2. *The initial parameters of the network can greatly effect how well concepts are learned* [Ahmad88, Kolen90]. Because training ANNs usually involves a hill-climbing algorithm, local minima may be a problem. For example, in some experiments involving the conversion of text to speech, correctness on a large dictionary of examples ranged from 1% to 31% [Shavlik91]. The only difference between the runs was the randomly-chosen initial set of weights.

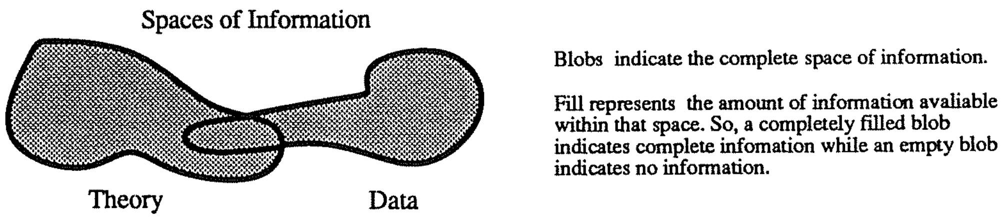

Spaces of Information
Theory
Data
Blobs indicate the complete space of information.
Fill represents the amount of information avaliable within that space. So, a completely filled blob indicates complete infomation while an empty blob indicates no information.

Figure 1.4: Interactions between the two spaces of information.

3. *There is not yet a problem-independent way to choose a good network topology.* The performance of an ANN can be greatly affected by the layout and number of hidden units, as well as the specification of connectivity. Many researchers have suggested methods for addressing parts of this problem [Honavar88, Diederich88, Chauvin88, Hanson88, Kruschke88, Fahlman89]. However, no method eliminates it.

4. *After training, a neural network is normally used as a "black box".* That is, given input, it produces output, but there is no explanation how they relate. Yet, without the ability to explain their decisions, it is hard to be confident in the reliability of a network that addresses a real-world problem. Moreover, the extraction of accurate, comprehensible, symbolic knowledge from networks is important if the results of neural learning are to used in related problems. Work has only just begun on the problem of shining light into the black box that is a trained neural network [Saito88, Hanson90, Fu91, Towell91].

1.2 Thesis Statement

Two of the disadvantages of empirical learning (i.e., that feature relevance is context dependent and that an unbounded number of features can be used to describe any object) point towards the conclusion that the distinction between theory and data is largely artificial. Theory and data are not independent as is suggested by Figure 1.1. Rather, as pictured in Figure 1.4, theory and data overlap. At a minimum, examples affect theory by pointing to its relevant parts (as well as identifying inconsistent parts). Likewise, theory affects examples by pointing out some of their relevant features.

Therefore, systems that make a hard distinction between theory and data are, in some sense, bankrupt. The distinction does not, and cannot, exist in the manner required by either explanation-based or empirical learning systems. *Instead learning systems must be able to learn from both theory and data.*

Moreover, the problems of explanation-based and empirical learning systems are complementary. *Hence, a hybrid learning system that is able to use strengths of one system to offset the weaknesses of the other may prove to be a powerful learning method.* Table 1.1 highlights the complementary nature of the weaknesses of explanation-based and empirical learning systems. This table lists each of the weaknesses of empirical and explanation-based learning discussed previously. With each item in the list is a brief description of why the other learning method is unaffected by the weakness.

The preceding discussion of the complementary nature of theory and data – as well as explanation-based and empirical learning – is the basis for the following:

> **Thesis:** *A system capable of learning effectively and efficiently must be able to draw on the available knowledge from both theoretical understanding of problems as well as examples of solved problems. A good approach to building such a system is to combine explanation-based learning with empirical learning. The resulting "hybrid" learning system should be able to profitably learn from theory and data; it should be able to use the strengths of one learning mechanism to overcome the weaknesses of the other.*

1.3 Review of Learning in Neural Networks

This section is a brief digression into a review of artificial neural networks; they are the basis of KBANN. Readers familiar with neural networks may skip this section without trepidation.

Artificial neural networks (ANNs) are a class of learning systems inspired by the architecture of the brain and theories of learning in the brain [Rosenblatt62, Hebb49, McCulloch43]. They are composed of cell-like entities (referred to as *units*) and connections between the units corresponding to dendrites and axons. (The connections are referred to as *links.*) Links are *weighted*; hence, the signal each link carries is the product of the link weight and the signal sent through the link. Abandoning the brain metaphor, an ANN can be thought of as a directed graph with weighted connections.

Units do only one thing – they compute a real-numbered output (known as an *activation*) which is a function of real-numbered inputs. Inputs either come from the environment or are received on links from other units. For instance, Figure 1.5 shows a single unit with three incoming links. The activation of the unit is shown to be a function of the sum of the incoming signals.

Units in neural networks are commonly sorted into three groups: *input*, *hidden* and *output*. These groups are shown graphically in Figure 1.6. Input units are so named because they receive signals from the environment. Similarly, output units are so named because their activation is available to the environment. (Generally the activation of the output units is the answer computed by the network.) Finally hidden units are so named because they have no direct interaction with the environment. They are purely internal to the network.

Table 1.1: Complementary strengths and weaknesses of learning systems.

Weaknesses of EBL are offset by strengths of EL.

- Basic EBL systems require a complete and correct theory,
  but EL requires little or no theory.
- Basic EBL systems do not learn at the "knowledge level",
  but EL systems do.
- Rules can be "brittle" in application,
  but EL systems such as neural networks gracefully degrade,
- EBL domain theories may be intractable to use,
  but the results of EL learning can often take little time to use.
- EBL systems must be provided with a knowledge base. However, they are incapable of its origination,
  but EL systems require little or no domain knowledge. Also, EL systems may be used to create domain knowledge.

Weaknesses of EL are offset by strengths of EBL.

- Spurious correlations may reduce the accuracy of empirical learning,
  but EBL systems do not rely upon correlations in the data.
- Small disjuncts can be difficult or impossible to learn accurately,
  but domain knowledge can specify arbitrarily small subsets.
- Contextual dependency of features is difficult to capture,
  but context may be a part of the domain knowledge.
- Every object can be described using an unbounded number of features,
  but domain knowledge may specify relevant features.
- Irrelevant features can interfere with learning,
  but domain knowledge may specify relevant features.
- Derived features, necessary for correctly acquiring a concept, may be impossible to learn,
  but intermediate conclusions in rules can specify arbitrarily complex derived features.

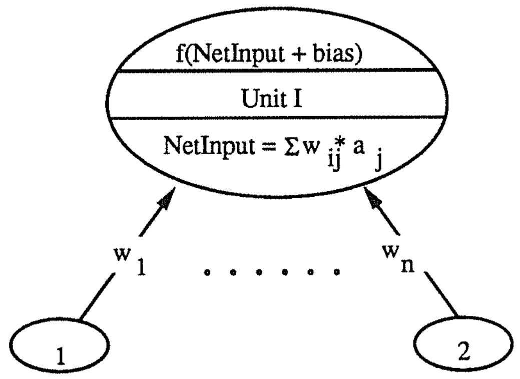


Figure 1.5: A single unit of an ANN.

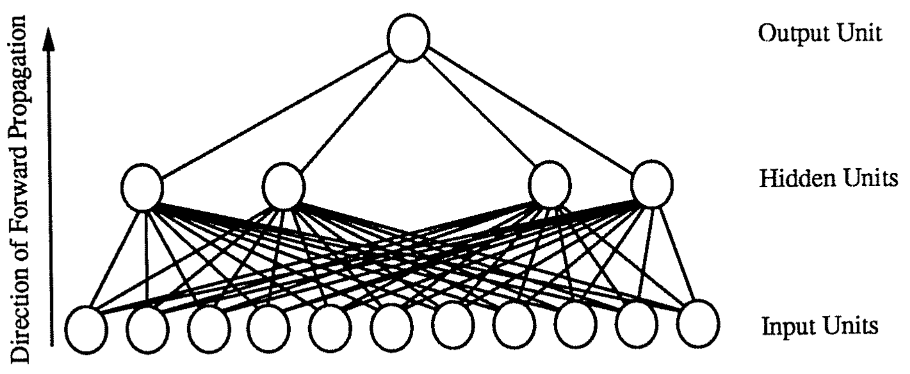


Figure 1.6: A prototypical artificial neural network.

Table 1.2: The backpropagation algorithm.

1. Activations of input units are set by the environment;
2. Activation is propagated forward along the directed connections (links) possibly through hidden units to the output units;
3. Errors are determined as a function of the difference between the computed activation of the output units and the desired activation of the output units;
4. Errors are propagated backward along the same links used to carry activations;
5. Changes are made to link weights to reduce the difference between the actual and desired activations of the output units.

The general process of learning in a neural network, using the standard backpropagation model [Rumelhart86], is given by the five-step algorithm in Table 1.2. The equations that underlie this algorithm appear in Table 1.3. These equations define how activations are computed, errors are propagated and weighted are changed following the standard approach of backpropagation.

Equation 1.2 defines the activation function for units in a backpropagation network. As illustrated in Figure 1.7, this standard "sigmoid" function squashes net incoming activation to a unit (Equation 1.1), which ranges over $[-\infty, +\infty]$, into the range [0, 1]. Note that the effect of θ_i, the "bias" term, in Equation 1.1 is to increase (or decrease) the net input to a unit. One way of looking at the bias is that it shifts the position of the sloping part of the curve in Figure 1.7. (In this case, the X-axis would be only the weighted sum of the incoming activations.) Hence, the bias acts like a threshold for the activation function.

Equations 1.3–1.6 define the learning mechanism of neural networks. Intuitively, this

Table 1.3: The mathematics underlying backpropagation.

$$NetInput_i = \sum_j w_{ji} * a_j + \theta_i \tag{1.1}$$

$$a_i = \frac{1}{1 + e^{-NetInput_i}} \tag{1.2}$$

$$E = \frac{1}{2}\sum_{i=1}^{n}(d_i - a_i)^2 \tag{1.3}$$

$$\begin{aligned} eOutput_i &= -\frac{\partial E}{\partial NetInput_i} \\ &= -\frac{\partial E}{\partial A} * \frac{\partial A}{\partial NetInput_i} \\ &= [d_i - a_i] * [(1 - a_i) * a_i] \end{aligned} \tag{1.4}$$

$$\begin{aligned} eHidden_i &= \frac{\partial A}{\partial NetInput_i} * \sum_k e_k * w_{ik} \\ &= [a_i * (1 - a_i)] * \sum_k e_j * w_{ik} \end{aligned} \tag{1.5}$$

$$\Delta w_{ij}(t) = \alpha * e_j * a_i + \mu * \Delta w_{ij}(t-1) \tag{1.6}$$

where:	i	is the index number of a unit
	$NetInput_i$	is the net incoming activation to unit i
	j	is a index ranging over every unit from which unit i receives activation
	w_{ij}	is the weight on a connection from unit j to unit i
	θ_i	is the threshold term associated with unit i
	A(x)	is the activation function
	a_i	is the activation of unit i
	E	is the total error of the network
	n	is the number of output units
	d_i	is the desired activation (training signal) for unit i
	$eOutput_i$	is the error of unit i, when the units is an output unit
	$eHidden_i$	is the error of unit i, when the units is a hidden unit
	k	is an index ranging over units to which unit i sends activation
	α	is the learning rate, a real number typically from [0...1]
	μ	is the momentum term, a real number typically from [0...1]
	t	is a time index

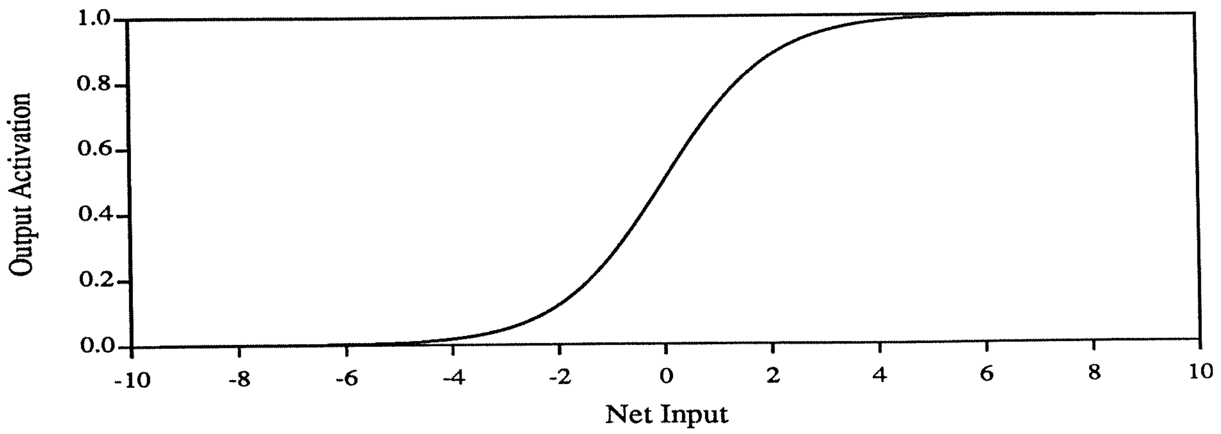


Figure 1.7: Output from a unit as a function of its net input.

method works by determining where the network went wrong, and making changes to address the mistake. Equation 1.3 defines the total error of the network as the sum of the squared differences between the actual and desired activations of the output units. Defining error as a sum of squared differences is common in statistics. For example, this measure underlies standard deviation.

Equations 1.4 and 1.5 define the propagation of error through a network. Equation 1.4 sets the error at an output unit. This function, the partial derivative of the error with respect to the net input, can be computed by applying the chain rule to Equations 1.2 and 1.3. Equation 1.5 defines the error at each hidden unit to be a function of the activation of that unit and the errors of all the units to which that unit sends activation. So, Equation 1.5 is the mechanism whereby credit or blame is assigned to units in the network. In other words, Equation 1.5 defines the way backpropagation addresses the "credit assignment problem" [Minsky63]. (The credit assignment problem is a classic problem of determining who should be rewarded for taking a correct action, or blamed for taking an incorrect action.)

Finally, Equation 1.6 shows that the adjustment of a link's weight is dependent upon the error at the receiving end of the link and the activation at the sending end of the link. In addition, this equation contains a "momentum" term that is dependent upon the previous change in the link weight. The momentum term is intended to damp out oscillations in the weight adjustments.

1.4 Overview of this Thesis

The rest of this thesis describes KBANN, a system that demonstrates the hypothesis that systems can learn from both theory and data by combining aspects of explanation-based and empirical learning systems. Chapter 2 contains a complete specification of KBANN. Briefly, KBANN has three, largely separate parts: a mechanism for inserting domain knowledge into a neural network, a mechanism for refining the network, and a mechanism for extracting refined rules from a trained network. In combination, these three parts form a powerful and general learning program.

Two real-world problems from the area of molecular biology are used to test the generality of learning in KBANN. (See Section 3.1 for their descriptions.) These problems are a small subset of the growing number of problems in computational biology due to the Human Genome Project [Alberts88]. In addition, KBANN has been used as the basis of a psychological model of the development of geometric reasoning in children (see Chapter 4).

The empirical results presented in Section 3.2 verify the hypothesis that a system that learns from both theory and data can learn more effectively than a system that learns from data alone. Further tests show KBANN makes more effective use of the combination of theory and data than other hybrid systems.

Tests in the remainder of Chapter 3 characterize the conditions under which it is useful to learn from both theory and data. For example, one set of tests adds noise to theories to determine the limits on how poor a domain theory can be while still providing useful information. A second set of tests investigates the properties of data sets which make theoretical information useful.

Chapter 4 continues the testing of KBANN, but in a very different way than the previous chapter. Rather than evaluating KBANN as a classifier, this chapter uses KBANN as the basis for a model of the development of geometric reasoning in children.

Chapter 5 describes two enhancements to KBANN. The first is DAID, a preprocessor that makes slight adjustments to the weight of links in a network created by KBANN. These adjustments, based upon the errors that the domain knowledge makes on a set of training examples, are shown to both improve generalization and reduce training effort. The second described in Chapter 5 augments KBANN with the ability to add hidden units not specified by the normal network construction method. These hidden units provide KBANN with the ability to grow beyond the vocabulary defined by the initial domain knowledge.

Overall, KBANN supports the thesis that a system which learns from both theory and data can outperform a system that learns from theory or data alone. The tests reported in this thesis show that this hypothesis is true under a wide variety of conditions on three different problems. Moreover, KBANN is shown to be an effective learning system by comparison to other hybrid approaches. Hence KBANN is a powerful and general approach to machine learning that empirically verifies this work's thesis.

Chapter 2

KBANN

This chapter describes the organization and algorithms of KBANN. Figure 2.1 depicts KBANN as a series of three algorithms. The first algorithm *inserts* approximately-correct, symbolic rules into a neural network. This step creates networks that make the same responses as the rules upon which they are based. Details of this first algorithm are given in Section 2.2.

The second algorithm of KBANN *refines* networks using the backpropagation learning algorithm [Rumelhart86]. While the learning mechanism is essentially standard backpropagation, the network it operates on is not standard. Instead, the network is constructed by the first algorithm of KBANN and thus, makes the same responses as the provided domain knowledge. This property of the networks created by KBANN has implications for training which are discussed in Section 2.3.

The final algorithm of KBANN *extracts* refined rules from trained networks. This algorithm makes it possible to explain the responses of trained networks. As a result, the information learned by the network is made available for human review. Moreover, the extracted rules can

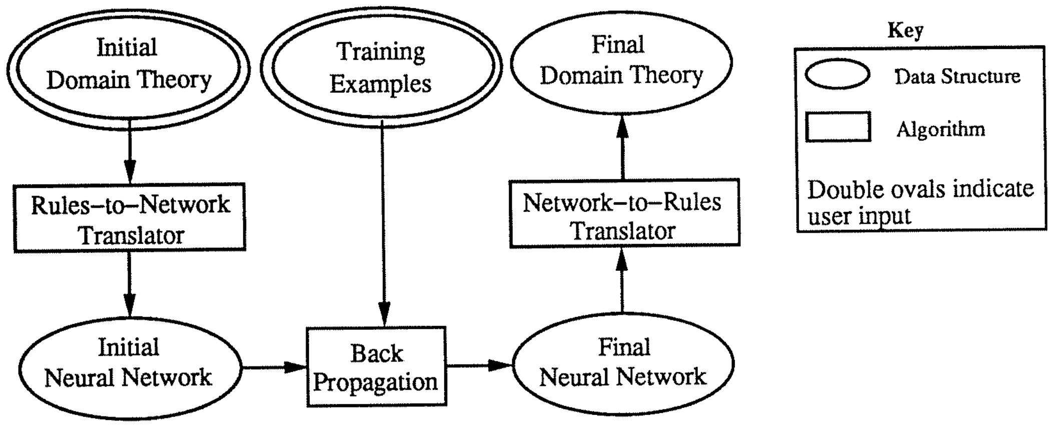


Figure 2.1: The flow of information through KBANN.

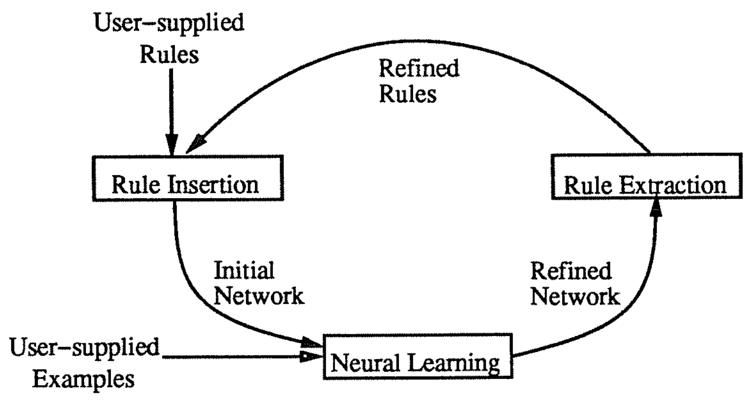


Figure 2.2: Information feeds back upon itself in KBANN.

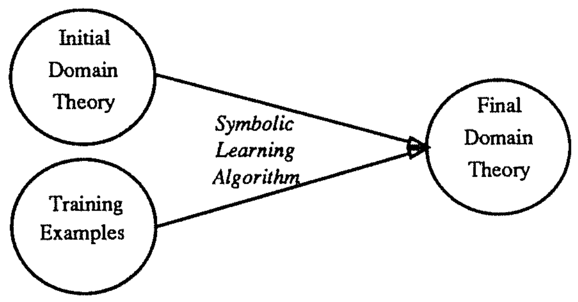


Figure 2.3: Flow chart of "all-symbolic" theory-refinement.

be used as a part of the initial knowledge for related problems. Hence, this algorithm allow KBANN to feed results back to itself in much the manner illustrated by Figure 2.2.

Before continuing with the description of KBANN, consider the difference between Figures 2.1 and 2.3. These figures present two alternative architectures for systems that learn form both theory and examples. As described above, Figure 2.1 shows the tri-algorithm architecture of KBANN. By contrast, Figure 2.3 represents the architecture is of EITHER [Ourston90] and Labyrinth-k [Thompson91], two "all-symbolic" hybrid learning systems. Whereas KBANN requires three algorithms, these all-symbolic systems require only a single algorithm because their underlying empirical learning mechanism operates directly upon the rules rather than their re-representation as a neural network. (See Chapter 6 for a descriptions of EITHER, Labyrinth-k, and other hybrid learning systems.) Tests reported in Chapter 3 show that this extra effort entailed by KBANN is well rewarded; KBANN consistently outperforms these all-symbolic systems.

The next section presents a high-level overview of KBANN which gives a feel for the approach. The subsequent three sections contain in-depth descriptions of each of the three algorithmic steps of the KBANN.

Table 2.1: Correspondences between knowledge-bases and neural networks.

Knowledge Base	*Neural Network*
Final Conclusions	Output Units
Supporting Facts	Input Units
Intermediate Conclusions	Hidden Units
Dependencies	Weighted Connections

2.1 Overview of KBANN

As shown in Figure 2.1, KBANN consists of three largely independent modules: a rules-to-network translator, a refiner, and a network-to-rules translator. Briefly, rules-to-networks translation is accomplished by mapping between a rule set and a neural network. This mapping, specified by Table 2.1, defines the topology of networks created by KBANN (KBANN-nets) as well as its initial link weights (see Section 2.2).

By defining KBANN-nets in this way, many of the problems inherent to neural networks and empirical learning are either eliminated or significantly reduced. For example, rules identify features that are very likely to be relevant. Hence, problems of spurious correlations and irrelevant features are significantly reduced. (See Section 3.4 for empirical evidence in support of this contention.) In addition, as KBANN admits rule sets that are not perfectly correct, it addresses several of the problems of explanation-based learning (EBL). For example, complete and correct rule sets are not required because the networks created by KBANN can (and do) learn at the knowledge level. This procedure also indirectly addresses other problems of EBL such as intractable theories because approximately correct theories are often quite brief.

The second major step of KBANN is to refine the KBANN-net using standard neural learning algorithms and a set of classified training examples. At the completion of this step, the trained KBANN-net can be used as a classifier that is more accurate than those derived by other machine learning methods. (Section 3.2 contains empirical evidence that supports this contention.)

The final, network-to-rules translation step of KBANN extracts a set of symbolic rules from the trained KBANN-net that retains its accuracy. This part of KBANN, described in Section 2.4, directly addresses problems that occur because trained neural networks are effectively "black boxes". Rule extraction shines light into the box, a process with three principle benefits: (1) the information learned by the KBANN-net during training is accessible for human review; (2) the rules can be used to give a semantically meaningful explanation for the responses of the network; (3) the modified rules can be used as part of knowledge bases of related problems. Hence, the extraction of rules allows information transfer to other neural networks [Pratt91] as well as other, symbolically-oriented learning systems.

2.2 Inserting Knowledge into a Neural Network

The first step of KBANN is to translate a set of approximately-correct rules into a knowledge-based neural network (KBANN-net). This translation has four important benefits. First, the translation specifies the features that are probably relevant to making a correct decision. Feature specification addresses problems inherent to empirical learning such as spurious correlations, irrelevant features, and the unboundedness of the set of possible features. Second, the translation of rules can specify important "derived" features, thereby simplifying the learning problem [Rendell90]. Moreover, these derived features can capture contextual dependencies in an example's description. Finally, the translated rules can refer to arbitrarily small regions of feature space. Hence, the rules reduce the need for the empirical learning system to learn about "small disjuncts." (Section 1.1 contains a definition of this problem.)

Rules to be translated into KBANN-nets are expressed as Horn clauses using the notation described in Appendix A. There are two constraints on the rule set. First, the rules must be propositional. This constraint results from the use of neural learning algorithms which are, at present, unable to handle variables. Second, the rules must be acyclic. (There can be no rule, or combination of rules, in which the consequent of the rule is an antecedent of that rule.) This constraint simplifies the translation of rules and training of the resulting networks. However, the constraint does not represent a fundamental limitation on KBANN as there exist algorithms based upon backpropagation which can be used to train networks with cycles [Pineda87]. (Section 7.2 describes plans for relaxing these constraints.)

In addition to these constraints, rule sets are usually hierarchically structured. That is, some rules do not map directly form inputs to outputs. Rather, the rules provide intermediate conclusions that describe useful conjunctions of the input features. These intermediate conclusions may be used by other rules to either determine the final conclusion or other intermediate conclusions. It is the hierarchical structure of a set of rules that creates derived features for use by the empirical learning system. Hence, if the domain knowledge is not hierarchically structured, then the networks created by KBANN will have no derived features to specify contextual dependencies or other useful conjunctions within example descriptions.

The rules-to-network translator is described in the next three subsections. The first of these subsections provides a high-level description of the translation; the second contains an example of the translation process; and the third contains a set of mathematical proofs of the correctness of the translator. Appendix A contains a complete specification of the information accepted by the translator along with descriptions of how information is translated into a network.

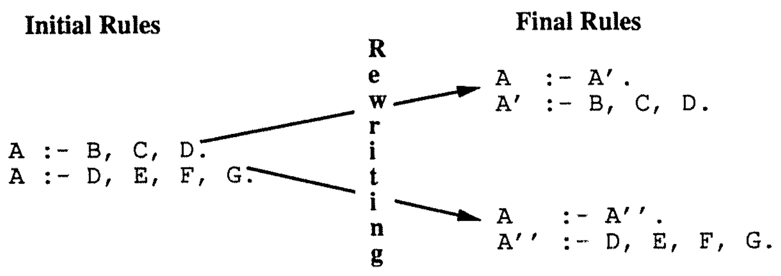


Figure 2.4: Rewriting rules to eliminate disjucts with more than one term.

2.2.1 The rules-to-network algorithm

Table 2.2 is an abstract specification of the rules-to-network translation algorithm. This algorithm initially translates a set of rules into a neural network. It then augments the network so that it is able to learn concepts not provided by the initial rules.

Table 2.2: The rules-to-networks algorithm of KBANN.

1. Rewrite rules so that disjuncts are in rules which have only one literal on the right-hand side.
2. Map the rule structure into a neural network.
3. Label units in the KBANN-net according to their "level".
4. Add hidden units to the network at user-specified levels. (*optional*)
5. Add units for known features not used in the rules.
6. Add links not specified by translation between all units in topologically-contiguous levels.
7. Perturb the network by adding near-zero random numbers to all link weights and biases.

Step 1, rewriting. The first step of the algorithm transforms the set of rules into a format which clarifies its hierarchical structure and makes it possible to directly translate the rules into a neural network. This step involves scanning every rule with the same consequent to determine if any have more than one antecedent. (The only form of disjuncts allowed by KBANN is multiple rules with the same consequent.) If there is more than one rule to a consequent, then every rule with more than one antecedent is rewritten as two rules. One of the rules has the original consequent and a single, newly-created term as an antecedent. The other rule has that same newly-created term as its consequent and the antecedents of the original rule as its antecedents. For instance, Table 2.4 shows the transformation of two rules into format required by the next steps of KBANN. (See Theorem 5 in Appendix C for a proof that this rewriting is required.)

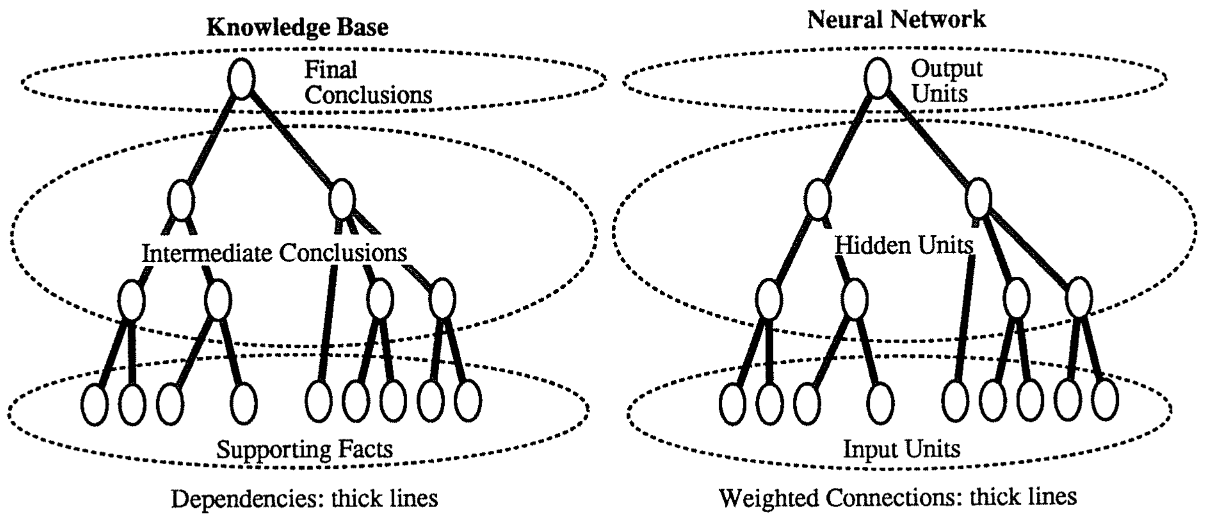


Figure 2.5: Correspondences between a rule set and a neural network.

Step 2, mapping. The second step of the rules-to-network algorithm establishes a mapping between a transformed set of rules and a neural network. This mapping, shown in Figure 2.5, creates a KBANN-net which has a one-to-one correspondence with elements of the rule set. Weights on all links specified by the rule set are set to $-\omega$ or ω depending on whether the antecedent is negated or not. Biases are set to $-\frac{\omega}{2}$ for disjunctive rules and $-\frac{2N-1}{2}\omega$ otherwise (where N is the number of unnegated terms).[1] Using this scheme for setting weights, units in a KBANN-net are active (i.e., have activation near 1) only when the corresponding consequent in the rule set is satisfied. Thus, the KBANN-net mimics the responses of the set of rules upon which it is based. (Section 2.2.3 contains proofs that this method of setting weights creates networks that accurately mimic the rules upon which they are based.)

At the completion of this step, a KBANN-net has been created that has the information from the set of rules concerning relevant, and derived, features. However, there is no guarantee that the set of rules refers to all of the relevant features or provides a significant collection of derived features. Hence the next four steps expand the KBANN-net by adding: links, inputs units, and possibly hidden units.

Step 3, labeling. In this step, units in the KBANN-nets are labeled by their "level". This label is not useful in itself, but is a necessary precursor to several of the following steps. Level may be defined in any of the following four ways:

1. Length of the *minimum* length path connecting a unit to an *input* unit. The upper-left graph in Figure 2.6 shows an example of this labeling method.

[1] KBANN uses $\omega = 4.0$, a number empirically found to work well.

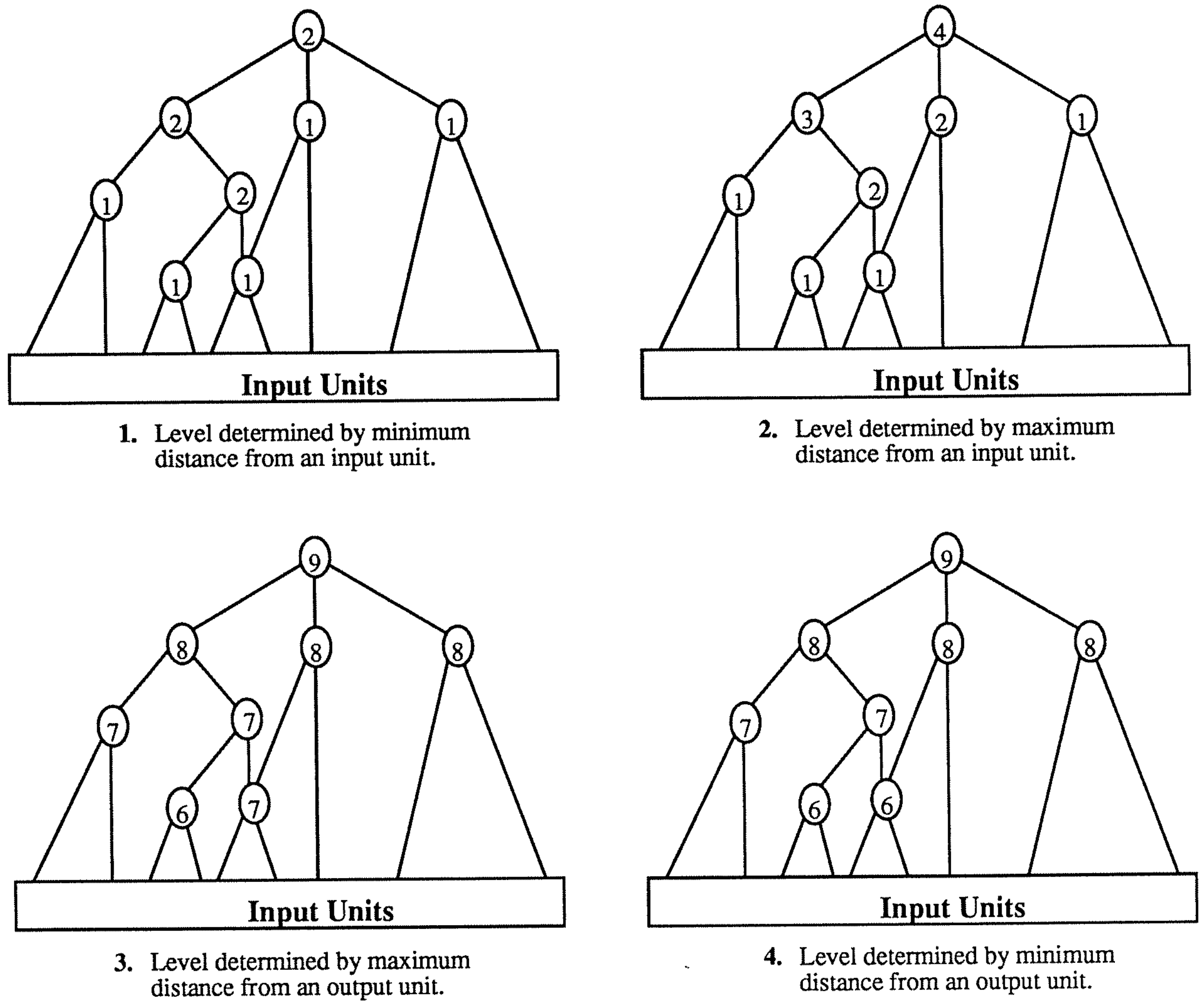

Input Units
1. Level determined by minimum distance from an input unit.
Input Units
2. Level determined by maximum distance from an input unit.
Input Units
3. Level determined by maximum distance from an output unit.
Input Units
4. Level determined by minimum distance from an output unit.

Figure 2.6: Four methods of labeling units in a KBANN-net.

2. Length of the *maximum* length path connecting a unit to an *input* unit. This method is used in the sample translation that appears later in this chapter. The upper-right graph in Figure 2.6 shows an example of this labeling method.

3. N less the length of the *minimum* length path connecting a unit to an *output* unit. The lower-right graph in Figure 2.6 shows an example of this labeling method ($N = 9$).

4. N less length of the *maximum* length path connecting a unit to an *output* unit. The lower-left graph in Figure 2.6 shows an example of this labeling method ($N = 9$).

Each of these labeling methods results associating a number with every hidden or output unit in the network. Every labeling method gives input units the same label.

All of these labeling techniques implicitly assume that every chain of reasoning is complete; that is, every intermediate conclusion is a part of a directed path from the one or more inputs to one or more outputs. However, there is no guarantee that every chain will be complete. Hence, the rules-to-network translator has the ability of recognize and handle incomplete chains. For instance, rules leading to a consequent may not reach the input units. In this case, the translator may attach the "dangling" consequents directly to every input unit (with low-weight links). Alternately, the translator may attach dangling consequents to one or more added hidden units. One the other hand, when a reasoning chain does not reach any output, the translator may attach the end of the reasoning chain directly to the output units or to some intermediate conclusions. For both types of incomplete theories, the defaults of the translator can be overridden by users.

Step 4, adding hidden units. This step adds hidden units to KBANN-nets thereby giving KBANN-nets the ability to learn derived features not specified in the initial rule set. In other words, added hidden units give KBANN-nets the ability to expand the vocabulary of the initial rules.

In many cases, the initial rules provide a vocabulary sufficient to obviate the need for adding hidden units. Hence, this step is optional. Because added hidden units are not always needed, they are only added upon specific instruction from a user. This instruction must specify the number and distribution among the levels established in the previous step of the added units.

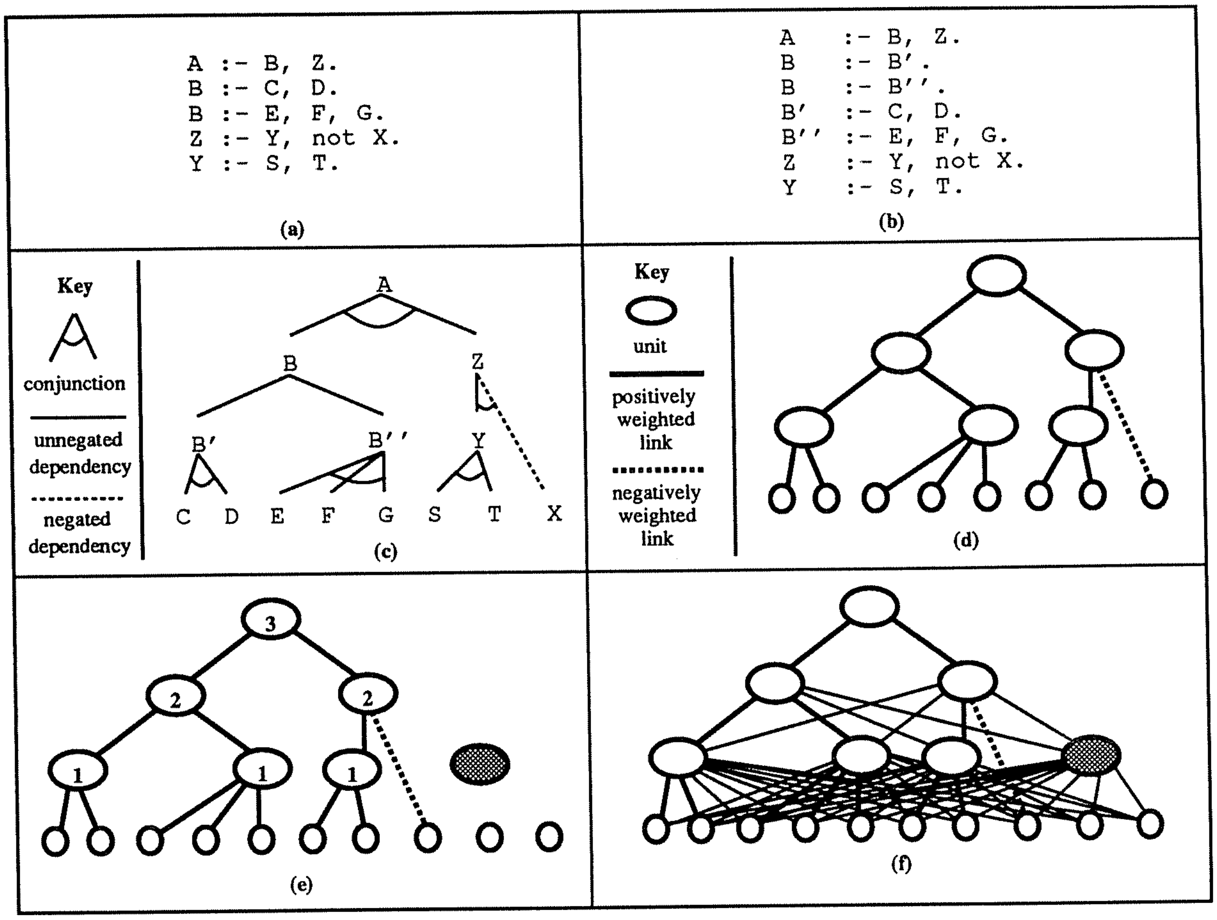
The addition of hidden units to KBANN-nets is a subject that has been only partially explored. Section 5.2 presents one method of hidden unit addition.

Step 5, adding input units. In this step, the KBANN-net is augmented with input features not referred to by the rule set. This addition is necessary because a set of rules that is only approximately correct may not identify every input feature that is required for correctly learning a concept.

Step 6, adding links. In this step links with weight zero are added to the network using the labeling of units established in step 4. Methods for adding links include:

1. Add links connecting each unit with label $n-1$ to each unit with label n.
2. Add links connecting each unit with label m to each unit with label n such that $n > m$.
3. Add links connecting every input unit to every hidden or output unit.

Adding links using the first method in conjunction with labeling according to the maximum distance form an input unit has proven slightly superior to the other methods on problems whose rule set has a good hierarchical structure. Note that neither addition of links nor addition of units have an immediate impact upon the KBANN-net because weights of the added links are so low that they do not affect the activation of any of the network's units.



```
A :- B, Z.
B :- C, D.
B :- E, F, G.
Z :- Y, not X.
Y :- S, T.
```

(a)

```
A    :- B, Z.
B    :- B'.
B    :- B''.
B'   :- C, D.
B''  :- E, F, G.
Z    :- Y, not X.
Y    :- S, T.
```

(b)

Figure 2.7: Sample rules-to-network translation.

Step 7, perturbing. The final step in the network-to-rules translation is to perturb all the weights in the network by adding a small random number to each. This perturbation is too small to have an effect on the KBANN-net's computations. However, it is sufficient to avoid training problems that occur in symmetrical networks [Rumelhart86].

2.2.2 Sample rules-to-network translation

Figure 2.7 shows a step-by-step translation of a simple set of rules into a KBANN-net. Panel (a) shows a set of rules in PROLOG-like notation. Panel (b) is the same set of rules after they have been rewritten in step 1 of the translation algorithm. The only rules affected by rewriting are `B :- C, D` and `B :- E, F, G` which together form a disjunctive definition of the consequent B.

Panel (c) is a graphical representation of the rules in panel (b) that shows the hierarchical structure of the rules. In this figure, dotted lines represent negated antecedents while solid lines represent unnegated antecedents. Arcs connecting antecedents indicate conjuncts (i.e., this a standard AND/OR tree).

The next step of the translation algorithm (step 2 in Table 2.2) is to create a neural network by mapping the hierarchical structure of the rules into a network. As a result, there is little visual difference between the representations of the initial KBANN-net in panel (d) and the hierarchical structure of the rules in panel (c).

Panels (e) and (f) illustrate the process whereby links, input units and hidden units not specified in the set of rules are added to the KBANN-net (steps 3–6 of the algorithm). Panel (e) shows units in the KBANN-net labeled by their "level" where level is defined to be the maximum length path to an input unit. In addition, panel (e) shows a hidden unit (it is shaded) added to the network at level one. (For the purposes of this example, assume that the user instructed the network-to-rules translator to add a single hidden unit at level one.)

Panel (f) shows the network after links with zero weight have been added to connect all units that are separated by one level. Note that in addition to providing existing units with access to information not specified by the domain knowledge, the low-weighted links connect the added hidden units to the rest of the network.

There is no illustration of the final step of the rules-to-network translation algorithm because the perturbation of link weights results only in minute changes that do not affect the decisions of the initial network.

2.2.3 Translation of rules into KBANN-nets

This section describes how KBANN translates rules containing the logical connectives AND, OR, and NOT into a KBANN-net. Recall that individual rules are assumed to be conjunctive, acyclic, and variable-free; disjuncts among the antecedents of rules are encoded (without loss of generality) as multiple rules.

To simplify discussion in this section, only binary-valued features are assumed to exist. Nominally-valued features are handled by simple recoding each nominal value as a binary decision. For example, consider a feature *color*, having three values *red*, *blue* and *green*. This feature would be encoded in a KBANN-net by three input units corresponding to the binary decisions: *color-is-red*, *color-is-blue*, and *color-is-green*. While this is far from the most parsimonious encoding scheme, Shavlik *et al.* suggest that this scheme improves generalization by some empirical learning algorithms [Shavlik91]. Methods for translating additional types of features into networks are presented in Appendix A.

The rules-to-network translator sets weights on the links and the biases of units so that units have significant activation (i.e., activation near one) only when the corresponding deduction can be made using the domain knowledge.[2] Likewise, when the deduction cannot be made

[2]The translation of rules into neural structures has been described by McCulloch and Pitts [McCulloch43] and Kleene [Kleene56] for units with threshold functions. Thus, the important and original contribution of this

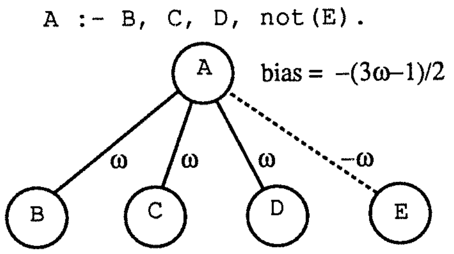


Figure 2.8: Translation of a conjunctive rule into a KBANN-net.

using the knowledge base, then the corresponding unit should be inactive (i.e., have activation near zero).

The descriptions and proofs in this section use the following terms:

C_a	the minimum activation for a unit to be considered *active*
C_i	the maximum activation for a unit to be considered *inactive*
$\mathcal{F}(x)$	$1/(1+e^{-x})$, the standard logistic activation function
w_p	the weight on links corresponding to positive dependencies
w_n	the weight on links corresponding to negated dependencies
P	the number of positive antecedents to the rule
N	the number of negated antecedents to the rule
K	the total number of antecedents to a rule ($K = P + N$)

Appendix C contains proofs of generalized versions of the theorems in this section.

Translation of conjunctive rules

Conjunctive rules are translated into a neural network by setting weights on all links corresponding to positive (i.e., unnegated) antecedents to ω, weights on all links corresponding to negated antecedents to $-\omega$, and the bias on the unit corresponding to the rule's consequent to $\frac{-2P+1}{2}\omega$. Using the terms defined above: $w_p = \omega$, $w_n = -\omega$, and $\theta = \frac{-2P+1}{2}\omega$. KBANN commonly uses a setting of $\omega = 4$, a value empirically found to work well. For example, Figure 2.8 shows a network which encodes:

```
A:- B, C, D, ¬E.
```

Intuitively, this translation method is reasonable. The input from the links plus the bias can only exceed zero when all of the mandatory antecedents are true and none of the prohibitory antecedents are true. (Units are only active when the net incoming activation plus

thesis is the idea of training networks that have been so constructed, rather than simply the construction of these networks.

the bias exceeds zero.) Hence, the unit will only be significantly active when all positively-weighted links carry a signal near one and all negatively-weighted links carry a signal near zero.

Theorem 1: *Mapping conjunctive rules into a neural network using:*

1. $w_p = -w_n = \omega > 0$
2. $\theta = \frac{-2P+1}{2}\omega$

creates a network that accurately encodes rules given that the rules have a "sufficiently small" number of antecedents. (The conditions on "sufficiently small" are given below.)

Proof of Theorem 1

This theorem is proved by showing that these mappings are correct in the following two situations:

A) The unit encoding the consequent of the rule is active when the rule's antecedents are satisfied.

B) The unit encoding the consequent of the rule is inactive when the rule's antecedents are not satisfied.

The proofs of each of these situations takes advantage of the monotonically-increasing property of the logistic activation function. This allows the analysis to focus on the boundary cases. This proof assumes only $C_i = 1 - C_a$ which provides nicely symmetric results. Also, $C_i = 1 - C_a$ means that $ln(\frac{1}{C_a} - 1) = -ln(\frac{1}{C_i} - 1)$.

Case A of Theorem 1 *Assume that the rule's antecedents are satisfied and show that the unit encoding the consequent is active.*

By assumption, the P positive antecedents are true and the N negative antecedents are false. Hence, the boundary condition given by Equation 2.1 obtains. This boundary condition expressed the minimum net incoming activation a unit could receive when all of its antecedents are satisfied. This occurs when all positive antecedents are at the minimum activation indicative of truth and all negative antecedents are at their maximum activation indicative of falsehood. Solving Equation 2.1 for the link weight yields Equation 2.2. Given the above assumptions about C_a and C_i and that $\omega > 0$, Equation 2.3 provides a bound on the relationship between the number of antecedents in a rule and the amount the activation of a unit may differ from 0 or 1. When this condition is met, the encoding scheme proposed by this theorem is correct for case A.

$$C_a \leq \mathcal{F}[PC_a\omega + NC_i\omega + \frac{-2P+1}{2}\omega] \tag{2.1}$$

$$\omega \geq \frac{ln(\frac{1}{C_a} - 1)}{K(1 - C_a) - \frac{1}{2}} \tag{2.2}$$

$$KC_i \leq \frac{1}{2} \tag{2.3}$$

Case B of Theorem 1 *Assume that the rule's antecedents are not satisfied and show that the unit encoding the consequent is inactive.*

A conjunctive rule is false when at least one positive antecedent is false (Equation 2.4) or one negative antecedent is true (Equation 2.5). These boundary conditions complement those of case A above as they capture the maximum net incoming activation a unit could receive when its antecedents are not satisfied. Equations 2.4 and 2.5, when solved for ω, both yield Equation 2.6 (again, given the assumptions about C_a and C_i and that $\omega > 0$). So, the proposed encoding scheme is correct from case B when ω satisfies Equation 2.6.

Note that the denominator of Equation 2.6 requires $C_a > \frac{1}{2}$ to be true for $\omega > 0$ to hold. Thus, this constraint enforces $C_a > C_i$ under the assumption that $C_i = 1 - C_a$. This is a reasonable (and minor) constraint given the definitions of C_a and C_i.

$$C_i \geq \mathcal{F}[(P-1)\omega + C_i\omega + \frac{-2P+1}{2}\omega] \tag{2.4}$$

$$C_i \geq \mathcal{F}[P\omega - C_a\omega + \frac{-2P+1}{2}\omega] \tag{2.5}$$

$$\omega \geq \frac{ln(\frac{1}{C_i} - 1)}{C_a - \frac{1}{2}} \tag{2.6}$$

The scheme for translating conjunctive rules into networks has been shown to correctly apply to both case A and case B. The final step to show that encoding scheme correct is to point out that the constraint on ω given by Equation 2.2 is compatible with the constraint given by Equation 2.6.

The only conditions on the correctness of this translation method are that K is "sufficiently small", where "sufficiently small" is given by Equation 2.3 and the very minor constraint that $C_a > 0.5$.

□

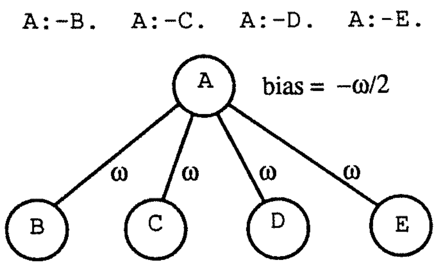


Figure 2.9: Translation of disjunctive rules into a KBANN-net.

Translation of disjunctive rules

To translate a set of rules encoding a disjunct, KBANN sets the weight of each link corresponding to a disjunctive antecedent to ω and the bias on the unit encoding the consequent to $-\frac{\omega}{2}$. For example, Figure 2.9 shows the network that results from the translation of four disjunctive rules into a KBANN-net. Intuitively, this is a reasonable strategy; the incoming activation overcomes the bias when one or more of the antecedents is true. The following theorem supports this intuition.

Theorem 2: *Mapping disjunctive rules into a neural network using:*

1. $w_p = \omega > 0$
2. $\theta = -\frac{1}{2}\omega$

creates a network that accurately encodes the disjunctive rules given that there are a "sufficiently small" number of rules. (Recall that each rule has a single antecedent. The conditions on "sufficiently small" are given below.)

Proof of Theorem 2

This theorem is proved by showing that this mapping is correct in the following two situations:

A The unit encoding the consequent of the set of rules encoding the disjunct is active when the one or more of the antecedents are satisfied.

B The unit encoding the consequent of the set of rules encoding the disjunct is inactive when the none of the antecedents are satisfied.

As in the proof of Theorem 1, the proofs of each of these situations takes advantage of the monotonically-increasing property of the logistic activation function. Also, as in Theorem 1, this proof assumes only $C_i = 1 - C_a$ (for symmetry). Recall that $C_i = 1 - C_a$ means that $ln(\frac{1}{C_a} - 1) = -ln(\frac{1}{C_i} - 1)$.

Case A of Theorem 2 *Assume that the consequent of the set of disjunctive rules is satisfied and show that the unit encoding that consequent is active.*

By assumption, at least one positive antecedent is true. (Disjunctive rules are assumed, without loss of generality, to have no negative antecedents. This assumption can be enforced by simply adding rules t0 invert negated antecedents.) Hence, the boundary condition given by Equation 2.7 obtains. As in case A of Theorem 1, this boundary condition captures the minimum activation that a unit encoding the consequent of a set of disjunctive rules could receive when the consequent is satisfied. Solving Equation 2.7 for the link weight yields Equation 2.8. Note that this is the same lower bound on ω as determined in case B of Theorem 1. Also, note that this solution requires $C_a > \frac{1}{2}$ which is quite reasonable given its definition.

$$C_a \leq \mathcal{F}[C_a\omega - \frac{1}{2}\omega] \tag{2.7}$$

$$\omega \geq \frac{ln(\frac{1}{C_i} - 1)}{C_a - \frac{1}{2}} \tag{2.8}$$

Case B of Theorem 2 *Assume that the consequent of a set of disjunctive rules is not satisfied and show that the unit encoding that consequent is inactive.*

The consequent of a set of disjunctive rules is not satisfied only when the lone antecedent of each rule in the set is not true. Equation 2.9 expresses the boundary condition. Solving this equation for ω, yields Equation 2.10 (given the assumptions concerning C_a and C_i and that $\omega > 0$). Finally, this solution for ω is only correct when Equation 2.11, which defines "a sufficiently small number of antecedents", holds.

This solution parallels that of Theorem 1, case A. Both cases provide a lower bound on ω that is a function of the allowed deviation of activations from 0 and 1 as well as the total number of antecedents to the rule. Also, both solutions place an upper bound on KC_i.

$$C_i \geq \mathcal{F}[KC_i\omega - \frac{1}{2}\omega] \tag{2.9}$$

$$\omega \geq \frac{ln(\frac{1}{C_a} - 1)}{KC_i - \frac{1}{2}} \tag{2.10}$$

$$KC_i \leq \frac{1}{2} \tag{2.11}$$

The scheme for translating disjunctive rules into networks has been show to correctly apply to both case A and case B. The final step to show that this encoding scheme correct is to point out that the constraint on ω given by Equation 2.8 is compatible with the constraint given by Equation 2.10.

□

2.3 Refining KBANN-nets

KBANN refines its networks using backpropagation [Rumelhart86], a standard neural learning method. Unfortunately, KBANN-nets create problems for backpropagation because they start with confident answers (i.e., the output units have activation near zero or one), regardless of the correctness of those answers. This causes problems because under the standard formulation of backpropagation, when answers are confident little change is made to the network regardless of the correctness of the answer. Rumelhart *et al.* argue this is a desirable property for a standard neural networks because it means that their learning tends to be noise resistant [Rumelhart86, page 329]. However, when the outputs of the network are incorrect, this property makes it very difficult to unlearn the aspects of the network that cause the errors. This problem with unlearning mistakes has been termed the "flat spot" of backpropagation [Fahlman88].

In the general context of neural networks, several solutions that require only minor adjustments to backpropagation have been proposed for the flat spot problem [Franzini87, Hinton89]. Solutions involving more significant changes to backpropagation have also been proposed [Fahlman88, Barnard89]. The results for KBANN-nets described herein all use the solution suggested by Hinton [Hinton89]. Hence, the rest of this subsection describes that approach.

Hinton's [Hinton89] approach to eliminating the flat spot in error propagation is given by the equations in Table 2.3. The approach uses a different error function so that large error signals are propagated through the network when the difference between the actual and desired output is large. Specifically, Hinton replaces the standard error function (Equation 2.12) with the *cross-entropy* function (Equation 2.15).

Statistically, the cross-entropy function operates by treating the actual and desired activation of each output unit as the probability that the activation value is actually one, independent of any other output units. This statement is equivalent to Equation 2.14. The independence assumption is enforced in Equation 2.14 by raising the terms to either d_i or $(1 - d_i)$. As d_i is either zero or one, raising terms to these powers results in either the term or one. Thus, when the desired activation is one, Equation 2.14 only depends upon the difference between the actual activation and one.

Under this assumption of independence among the output values, the likelihood of producing the desired outputs is maximized when the *cross entropy* between actual and desired outputs is minimized. That is, to minimize the error of the network, maximize Equation 2.14. Equation 2.15 is an algebraic restatement of Equation 2.14 which takes the log of Equation 2.14

Table 2.3: Revisions to error propagation.

$$E = \frac{1}{2}\sum_{i=1}^{n}(d_i - a_i)^2 \tag{2.12}$$

$$\begin{aligned} e_i &= -\frac{\partial E}{\partial NetInput_i} \\ &= -\frac{\partial E}{\partial A} * \frac{\partial A}{\partial NetInput_i} \\ &= [d_i - a_i] * [(1 - a_i) * a_i] \end{aligned} \tag{2.13}$$

$$C = p(\vec{\mathbf{d}}|network) = \prod_{i=1}^{n}(a_i(1 - d_i) * (1 - a_i)^{d_i}) \tag{2.14}$$

$$C = -\sum_{i=1}^{n}[(1 - d_i) * log_2 a_i) + (d_i * log_2(1 - a_i)] \tag{2.15}$$

$$\begin{aligned} e_i &= \frac{\partial C}{\partial NetInput_i} \\ &= log(2) * \left(\frac{d_i}{a_i} - \frac{1 - d_i}{1 - a_i}\right) * a_i * (1 - a_i) \end{aligned} \tag{2.16}$$

where:	E	is the standard definition of error output unit error
	A	is the activation function
	e_i	is the error of unit i
	a_i	is the activation of unit i
	d_i	is the desired activation for unit i
	n	is the number of output units
	$NetInput_i$	is the net incoming activation to unit i
	$\vec{\mathbf{d}}$	is the vector of desired output activations
	$p(\vec{\mathbf{d}}|network)$	is the probability of the desired outputs given the network
	C	is the cross-entropy function

to put it into a form that can be easily applied to neural learning.

Use of the cross-entropy error function changes only the error signal at output units, for which Equation 2.16 replaces Equation 2.13. Hidden units still use Equation 2.13. Equation 2.16 is considerably simpler when the desired activation is 0 and 1 (recall that this assumption underlies cross-entropy); it respectively reduces to $-o_i$ and $(1 - o_i)$ times a constant. (I.e., $\frac{\partial C}{\partial net_i} = (1 - o_i)$ when $d_i = 1$ and $\frac{\partial C}{\partial net_i} = -o_i$ when $d_i = 0$.) Hence, under the cross-entropy

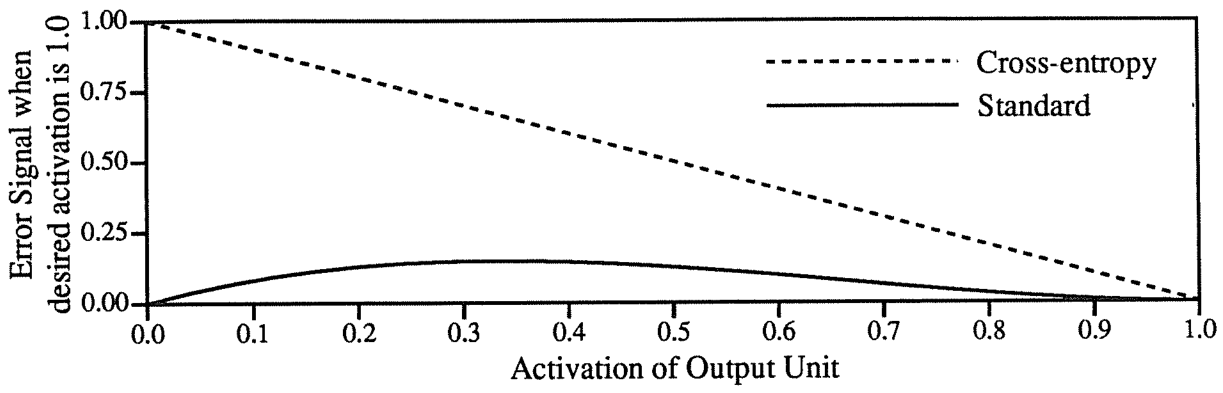


Figure 2.10: Error signal for two popular error functions.

error function, when the desired output is either zero or one, the error signal is proportional to the actual error.

Figure 2.10 graphs the error propagated backward from output units by cross-entropy and the standard error function when the desired activation is 1.0. Highlighted by Figure 2.10, the principle advantage of cross-entropy is that, unlike the standard error function, the magnitude of cross-entropy's error signal always varies with the difference between the desired and the actual activations.

In summary, cross-entropy makes more sense for KBANN-nets than the standard error function because it reduces problems that result from initially confident, but possibly-mistaken answers. Empirical comparisons (results not shown) of learning in KBANN-nets using both the standard and cross-entropy error functions show this analysis to be correct — cross-entropy results in shorter training times and slightly better generalization.

2.4 Extracting Rules from Trained Networks

After KBANN-nets have refined, they can be used as highly accurate classifiers. The results presented in the first parts of Chapter 3 demonstrate just that. However, trained KBANN-nets provide no explanation of how an answer was derived. Nor can the results of their learning be shared with humans or transferred to related problems. (Pratt's work on direct transfer between two neural networks is an exception to this [Pratt91].)

The extraction of symbolic rules directly addresses these problems. It makes the information learned by the KBANN-net during training accessible for human review and justification of answers. Moreover, the modified rules can be used as part of knowledge bases for the solution of related problems.

This section presents two approaches to the extraction of rules from trained KBANN-nets. The next section describes features shared by all approaches to the extraction of rules from neural networks and explains two assumptions about trained neural networks made by the algorithms described herein. The two algorithms appear in the subsequent sections.

2.4.1 Underpinnings of rule extraction

Assumptions

The methods of rule extraction described below make two assumptions about trained KBANN-nets. The first assumption is that training a KBANN-net does not significantly shift the meaning of its units. By making this assumption, the methods are able to attach labels to every consequent and antecedent of the extracted rules, thereby enhancing the comprehensibility of the rules. These labels correspond to consequents in the symbolic knowledge upon which the KBANN-net is based. The utility of the labels would be compromised if meaning shifted significantly.

Observation of trained networks indicates that meanings usually are quite stable. Yet, meanings can shift. Such shifts are most common at units associated with the least certain of the initial rules.[3] Hence, inappropriately labeled rules can be used as pointers to weak portions of the initial knowledge base.

The second assumption is that almost all of the units in a trained KBANN-net are either fully active (i.e., have activation near one) or inactive (i.e., have activation near zero). By making this assumption, each non-input unit in a trained KBANN-net can be treated as a step function or a Boolean rule. Therefore, the problem for rule extraction is to determine the situations in which the "rule" is true. Again, examination of trained KBANN-nets suggests that this assumption is valid.

This second assumption is not particularly restrictive; the standard logistic activation function can be slightly modified to ensure that units approximate step functions. In particular, Equation 2.17, in Table 2.4, is the logistic activation function to which a scaling parameter, σ, has been added[4] The standard value for this parameter is 1.0. However, as indicated by Figure 2.11, changing σ can make the logistic activation function resemble a straight line or a step function.

Adjusting σ so that the activation function resembles a step function must be done cautiously. Standard backpropagation looses effectiveness as the activation function steepens due to a widening of the "flat spot" [Fahlman88] in the error propagation mechanism. (The cross-entropy error-function discussed in the previous section only addresses this problem at the output units. When every unit resembles a step function, the flat spot is a problem throughout the network.) As a result, it is difficult and time-consuming to train networks that have

[3]Currently, certainty of rules is information known by the user but not provided to KBANN. Section 7.2 describes plans for adding the ability to handle assessments of rule confidence.

[4]Kruschke and Movellan refer to this parameter as the *gain* [Kruschke91].

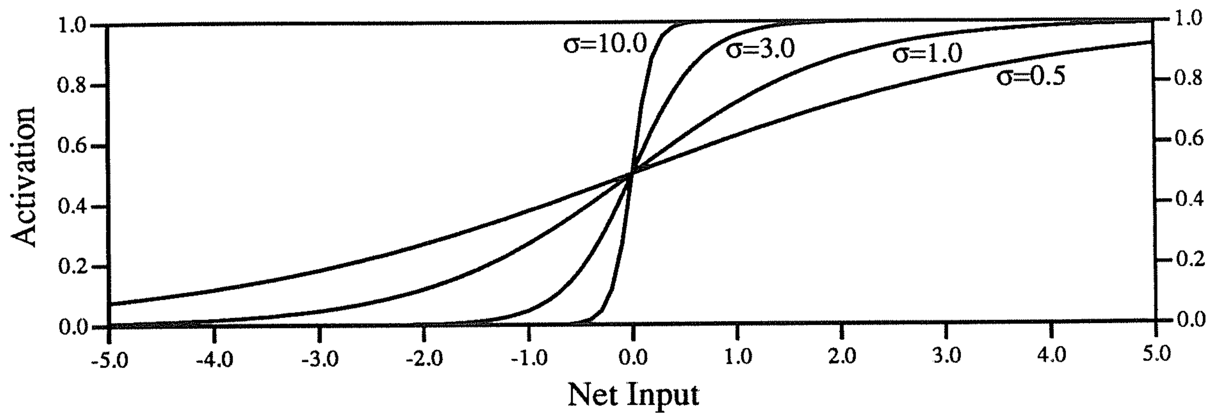


Figure 2.11: Activation of units when σ is varied.

activation functions which approximate step functions.[5] In addition, according to the unpublished lore of neural networks, some of the power of neural networks comes from their ability to make use of partially-activated units. Hence, adjusting the σ parameter so that units resemble step functions may impose some correctness penalty on the networks.

Commonalities

The methods described in this thesis for extracting rules from neural networks, as well as those in the literature, do so by trying to find combinations of the input values to a unit that result in it having an activation near one. Recall that units in neural networks normally have activations defined by Equations 2.17 and 2.18. Broadly speaking, the result of Equation 2.18 is that if the summed weighted inputs exceed the bias, then the activation of a unit will be near one. Otherwise the activation will be near zero. *Hence, rule extraction methods look for ways in which the weighted sum of inputs exceeds the bias.*

The second of the above assumptions, that all units in trained networks have activations near zero or one, simplifies this task by forcing links to carry a signal equal to their weight or no signal at all. That is, Equation 2.17 reduces to Equation 2.19 As a result, rule extraction need only be concerned with the weight of links entering a unit and may ignore the activation of the sending unit.

Rule extraction is further simplified by the fact that units always have non-negative activations. This property allows rule-extraction methods to take the sign of a link's weight as a perfect indicator of the way in which an antecedent will be used. That is, negatively-weighted links can only give rise to negated antecedents, while positively-weighted links can only give

[5]Kruschke and Movellan [Kruschke91] suggest that is possible to use backpropagation to train σ for each unit at the same time as standard training. Their technique would avoid some of the difficulties of training with very steep activation functions. However, their method does not guarantee that units assume a step-like character at the end of training.

Table 2.4: The logistic activation function.

$$a_i = f\left(\sum_j (w_{i,j} * a_j) + \theta_i\right) \tag{2.17}$$

$$f(x) = \frac{1}{1+e^{-\sigma x}} \tag{2.18}$$

$$a_i = f\left(\sum_{\{j|a_j \approx 1\}} w_{i,j} - \theta_i\right) \tag{2.19}$$

where: a_i is the activation of unit i
$w_{i,j}$ is the weight on a link from unit j to unit i
θ_i is the "bias" of unit i
σ is a paramenter affecting the slope of the sigmoid.

rise to unnegated antecedents. This considerably reduces the size of the search space [Fu91].

2.4.2 The SUBSET method

The first method for rule extraction is referred to in this work as the SUBSET algorithm. It is so named because it operates by attempting to find subsets of the units to which a unit is connected whose summed weighted activations exceed the bias of that unit. The method was developed independently, though it is fundamentally similar to approaches described by Saito and Nakano [Saito88], Fu [Fu91] and Masuoka [Masuoka90].[6] SUBSET represents the state of the art in the published literature.

The SUBSET algorithm is summarized in Table 2.5. It relies heavily upon the assumption that units are either active or inactive, as this assumption allows the method to look only at link weights. Hence, steps 2 and 3 of the algorithm ignore activations of the units at the sending ends of the links.

As an example, consider the unit in Figure 2.12a. Given that the link weights and the bias are as shown, the four rules listed in Figure 2.12b are extracted by the algorithm.

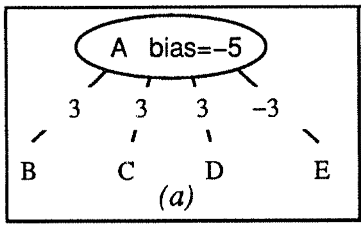
The major problem with the SUBSET method is that the cost of finding all subsets grows as the size of the power set of the links to each unit. Thus, the algorithm can only be expected to work on simple networks and toy domains. To sidestep these combinatorial problems, Saito and Nakano [Saito88] establish a ceiling on the number of antecedents in extracted rules.

[6]Empirical comparisons (results not reported) of the method described here to the algorithm described by Saito and Nakano indicate the algorithm described here is superior in terms of generalization accuracy.

Table 2.5: The SUBSET approach to rule extraction.

```
1. With each $U \in$ {all hidden and output units}
   Let $\theta$ be the bias of unit $U$

2. Find up to $\beta_p$ groups of positively-weighted links to unit $U$ such that
   $0 < \theta + \sum(\textit{link weights of group})$
   Call the set of groups found in this step $G_p$

3. For each $\mathcal{P} \in G_p$

     Find up to $\beta_n$ groups of negatively weighted links to unit $U$ such that
        $0 > \theta + \sum(\textit{link weights of } \mathcal{P}) + \sum(\textit{link weights of group})$
        Call the set of groups found in this step $G_n$
     For each $\mathcal{N} \in G_n$
        Create a rule with the following form:
           ``if $\forall\mathcal{P}$ and $\neg\forall\mathcal{N}$ then (name of $U$)''

4. Remove any duplicate rules.
```

Appendix B.2.1 contains pseudocode for this algorithm.



if B, C, and not(E) then A.
if B, D, and not(E) then A.
if C, D, and not(E) then A.
if B, C, D then A.
(b)

Figure 2.12: Rule extraction using the SUBSET algorithm.

However, the initial rules for the real-world domains studied in this work involve large numbers of antecedents. As a result, the lowest possible ceiling on the number of antecedents might require considering more than 10^5 sets of antecedents.

Therefore, the implementation of SUBSET described here explicitly bounds the number of positive and negative groups, rather than setting a bound on the number of antecedents. While this approach is advantageous because it allows rules with an unlimited number of antecedents, it may discover up to $\beta_p * \beta_n$ (β_p bounds the number of positively weighted groups, β_n bounds the number of negatively weighted groups) rules for each non-input unit in the network to be translated. In practice, far fewer rules are extracted. Yet, the worst case is a significant concern because the accuracy of the set of extracted rules is expected to increase as the size of β_p and β_n increases. Thus, when using the SUBSET method a tension can be expected between the accuracy and potential size of the set of extracted rules. This tension is empirically investigated in Section 3.5.

The SUBSET method typically extracts about 300 rules from networks trained for the problems studied in this work. While 300 rules is a large set, it is smaller than many handcrafted expert systems [Fozzard88, McDermott82]. Hence, SUBSET provides sets of rules that are, at least potentially, tractable. However, the rules tend to hide significant structures in trained networks. For instance, in Figure 2.12a, the links to B, C and D all have the same weight while the link to E has the negative of that weight. Looking at the problem in this way suggests the rules in Figure 2.12b could be rewritten as the following rule, which provides a clearer statement of the conditions on A:

```
if (3 of {B, C, D, not(E)}) then A.
```

This consideration of structure shared among several rules led to the development of the algorithm presented next.

2.4.3 The NOFM method

The second algorithm for rules extraction, referred to as NOFM, explicitly searches for antecedents of the form:

`if (`N` of the following `M` antecedents are true) then ...`

This approach was taken because, as discussed in the previous section, rule sets discovered by the SUBSET method often contain "N-of-M" style concepts. Furthermore experiments indicate that ANNs are good at learning N-of-M concepts [Fisher89] and that searching for N-of-M concepts is a useful inductive bias [Murphy91]. Finally, note that purely conjunctive rules result if $N = M$, while a set of disjunctive rules results when $N = 1$; hence, using N-of-M rules does not restrict generality.

The algorithm

The idea underlying NOFM, an abstracted version of which appears in Table 2.6, is that individual antecedents (links) do not have a unique importance. Rather, groups of antecedents form equivalence classes in which each antecedent has the same importance as, and is interchangeable with, other members of the class. This equivalence class idea is the key to the NOFM algorithm; it allows the algorithm to consider groups of links without worrying about the particular links within the group.

Step 1, clustering. Backpropagation training tends to group links of KBANN-nets into loose clusters rather than tight equivalence classes as assumed by the NOFM algorithm. Hence, the first step of NOFM is to group links into equivalence classes. This grouping can be done in either of two ways.

Table 2.6: The NOFM approach to rule extraction.

1. With each hidden and output unit,
   form groups of similarly-weighted links.
2. Set link weights of all group members to the average of the group.
3. Eliminate any groups that do not significantly affect whether the unit will be active or inactive.
4. Holding all links weights constant, optimize biases of all hidden and output units using the backpropagation algorithm.
5. Form a single rule for each hidden and output unit. The rule consists of a threshold given by the bias and weighted antecedents specified by the remaining links.
6. Where possible, simplify rules to eliminate weights and thresholds.

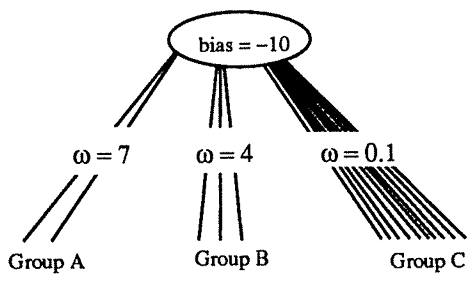
Appendix B.2.2 contains pseudocode for this algorithm.

First, clustering may be done using a standard clustering method such as the *join* algorithm [Hartigan75]. This method clusters by joining the two closest clusters starting with n clusters of size 1. Clustering stops when no pair of clusters is closer than a set distance (KBANN uses 0.25).

Alternately, clustering can be done by sorting all links and performing a single pass across the sorted links. Using this approach, items are added to the current cluster until either: (a) a new item differs from the mean of the cluster by more than a specified bound or (b) the addition of an item causes an item already in the cluster to differ from the cluster's mean by more than this same bound. As with the join approach to clustering, the bound used by KBANN for this approach is 0.25. (See Appendix B.2.2 for pseudocode of this clustering approach.)

Each of these two clustering methods work quite well for the NOFM algorithm. However, the second method is used almost exclusively as its complexity is $O(n * log(n))$ while the complexity of the *join* procedure is $O(n^2)$.

Step 2, averaging. After groups are formed, the second step of the algorithm is to set the weight of all links in each group to the average of each group's weight. Thus, the first two steps of the algorithm force links in the network into equivalence classes as required by the rest of the algorithm.



Possible Combinations of A and B	Total weight of combo
1B	4
1A	7
2B	8
1A+1B	11
3B	12
2A	14
1A + 2B	15
etc	etc

Figure 2.13: Link weight combinations that indicate a cluster should be dropped.

Step 3, eliminating. With equivalence classes in place, the procedure next attempts to identify and eliminate those groups that are unlikely to have any bearing on the calculation of the consequent. Such groups generally have low link weights and few members. Elimination proceeds via two paths one heuristic, and one algorithmic.

The first elimination procedure algorithmically attempts to find clusters of links that cannot have any effect on whether or not the total incoming activation exceeds the bias. This is done by calculating the total possible activation that each cluster can send (taking into account properties of units — e.g., that only one unit related to each nominal input feature may be active at any time). This total possible activation is then compared to the levels of activation that are reachable given link weights in the network. Clusters that cannot change whether the net input exceeds the bias are eliminated.

This procedure is very similar to SUBSET. However, the small number of possible link weights considerably reduces the combinatorics of the problem. For instance, consider Figure 2.13 which illustrates a unit with a bias of -10 and three clusters of links: (A) two links of weight 7, (B) three links of weight 4, and (C) ten links of weight 0.1. In this case, cluster (C)is eliminated. Its total possible activation is 1.0 and the only reachable activations that are less than the bias using the clusters (A) and (B) are 7 and 8. Neither of these activation levels, when combined with the 1.0 total from group C, exceeds 10.0. Hence, the links in cluster (C) can have no effect on the unit into which they feed. So, the cluster is safely eliminated.

The heuristic elimination procedure is based explicitly upon whether the net input received from a cluster ever is necessary for the correct classification of a training example. This procedure operates by presenting a training example to the clustered network and sequentially zeroing the input from each cluster. If doing so results in a change in the activation of the unit receiving activation from the cluster, then the cluster is marked as necessary. After doing this for every example, any clusters not marked as necessary are eliminated.

Step 4, optimizing. With unimportant groups eliminated, the fourth step of NOFM is to optimize the bias on the unit. This step is necessary because the averaging of the link weights in a cluster and elimination of links can change the times at which a unit is active. As a result, prior to optimization, networks on which the first three steps of the NOFM algorithm have been applied may have error rates that are significantly higher than they were at the end of training.

Optimization can be done by freezing the weights on the links so that the groups stay intact and retraining the network using backpropagation. To reflect the rule-like nature of the network, the activation function is modified by the addition of a slope term as was shown in Equation 2.17. Setting σ to a large positive value (e.g., 20) ensures that units in the network take on values near zero or one.

Step 5, extracting. This step of the NOFM algorithm form rules that simply re-express the network. That is, rules are created by directly translating the bias and incoming weights to each unit into a rule with weighted antecedents such that the rule is true if the sum of the weighted antecedents exceeds the bias. Note that because of the equivalence classes and elimination of groups, these rules are considerably simpler than the original trained network; they have fewer antecedents and those antecedents tend to be in a few weight classes. (See the extracted rules in Section 3.5 for examples of unsimplified rules.)

Step 6, simplifying. Finally, rules are simplified whenever possible to eliminate the weights and thresholds. Simplification is accomplished by scanning each restated rule to determine the possible combinations of group items that exceed the rule's threshold (i.e., bias). This scan may result in more than one rule. Hence, there is a tradeoff in the simplification procedure between complexity of individual rules and complexity resulting from a number of simple rules.

For example, consider the rule[7]:

```
A :- 10.0 < 5.1*number-true{B, C, D, E} +
            3.5*number-true{X, Y, Z}
```

The simplification procedure would simplify this rule by rewriting it as the following three rules:

```
A :- 2 of {B, C, D, E}
A :- 1 of {B, C, D, E} and 2 of {X, Y, Z}
A :- X, Y, Z.
```

If the elimination of weight and biases requires rewriting a single rule with more than five rules,[8] then the rule is left in its original state.

[7]The function number-true returns the number of antecedents in the following set that are true.

[8]When there are more than five disjuncts to a single consequent it is difficult to keep understand the conditions under which the consequent is satisfied.

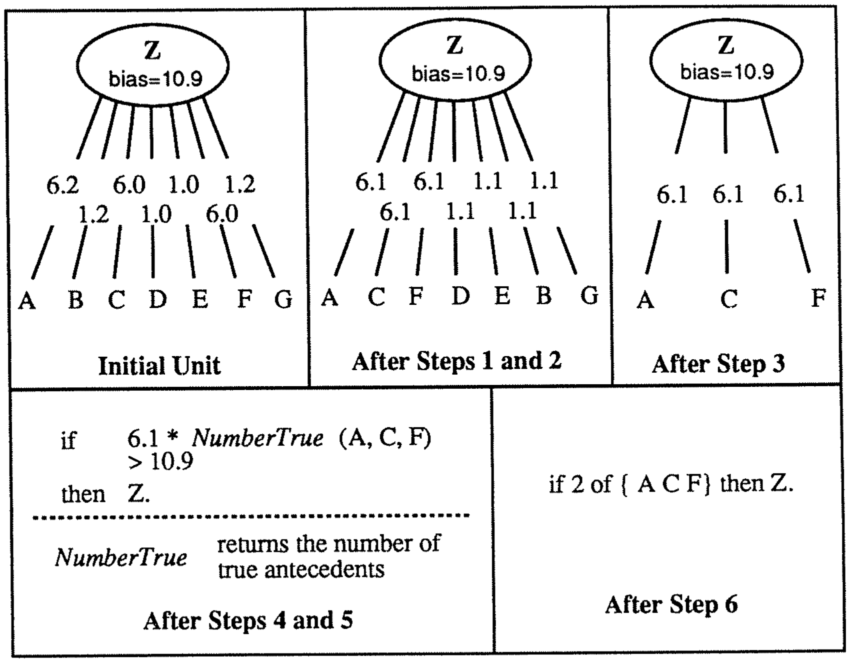


Figure 2.14: An example of rule extraction using NofM.

Example of the NofM method

As an example of NofM, consider Figure 2.14. This figure illustrates the process through which a single unit with seven incoming links is transformed by the NofM procedure into a rule that requires two of three antecedents to be true.

The first two steps of the algorithm transform the unit with seven unique inputs into a unit with two classes of inputs, one with three links of weight 6.1 and one with for links of weight 1.1. The next step of the algorithm eliminates the group with weight 1.1 from consideration because there is no way that these links – either alone or in combination with links in the other group – can affect whether or not the sum of the incoming activation to unit Z exceeds the bias on Z. (Note this step takes advantage of the assumption that units have activations near either zero or one is necessary here.)

Figure 2.14 does not illustrate bias optimization. The lower left panel of Figure 2.14 shows the re-expression of the simplified unit as a rule. The final step of the algorithm is illustrated by the bottom right panel of Figure 2.14, in which the rule with weighted antecedents and a threshold is transformed into a simple N-of-M style rule. (The Subset algorithm would find the three rules that are the expansion of the 2-of-3 rule found by NofM. However, to find these three rules Subset would have had to consider as many as 125 possibilities.)

Algorithmic complexity of NofM

The complexity of NofM is difficult to precisely analyze as the bias optimization phase uses backpropagation. However, the problem addressed in bias-optimization is considerably simpler than the initial training of the network. Usually, networks have more than an order of magnitude fewer links during bias optimization than during initial training. Moreover, only the biases are allowed to change. As a result, this step takes a reasonably short time.[9] Each of the other steps requires $O(m * n)$ (m is the number of units, n is the number of links received by each unit) time except for the initial clustering, which requires $O(m * n * log(n))$ time and cluster elimination which requires $O(x * m * n)$ (x is the number of training examples).

2.4.4 Subset and NofM: Summary

Both of these rule-extraction algorithms have strengths with respect to the other. For example, the individual rules returned by Subset are more easily understood than those returned by NofM. However, because Subset can be expected to return many more rules (which are often quite repetitive) than NofM, the rule sets returned by NofM may actually be easier to understand than those of Subset. More significantly, Subset is an exponential algorithm whereas NofM is approximately cubic. Finally, results presented in the next chapter indicate that the rule derived by NofM are more accurate than the rules derived by Subset.

2.5 Review of Kbann's algorithms

This chapter has described the three algorithms that together comprise Kbann. The first step is a rules-to-network translator. This step *inserts* domain information, in the form of sets of propositional rules, into a neural network. As a result, the network initially makes the same responses as the rules upon which it is based.

The second step of Kbann *refines* the networks created by the first step. Aspects of the insertion process render standard backpropagation less than efficient on the networks that Kbann creates. Hence, the refinement algorithm of Kbann uses a modified definition of error.

The final step of Kbann is the *extraction* of symbolic rules from networks trained in the second step. Extraction is performed by a network-to-rules translator for which two methods are described. The extraction of rules allows trained networks to explain their actions. It also make possible the reuse of the extracted refined rules on related problems. Hence, rule extraction "closes the loop" by allowing rules refined by Kbann to be fed back to the rules-to-network translator.

[9]Training neural networks has been proven NP-complete [Judd88, Blum88]. However, Hinton [Hinton89] suggests that in practice backpropagation usually runs in $O((m * n)^3)$ time.

Chapter 3

EMPIRICAL TESTS OF KBANN

This chapter details empirical tests that explore *how well* KBANN works as opposed to the previous chapter which describes *how* KBANN works. Four distinct sets of tests present evidence about *how well* KBANN works. The first three test only the insertion and refinement algorithms of KBANN. These tests use trained KBANN-nets directly; they do not extract rules. The first of the tests, in Section 3.2, compares the classification abilities of KBANN-nets to standard empirical learning methods as well as other methods for learning from both theory and data. The tests show KBANN-nets are very effective classifiers. The following tests attempt to identify the aspects of KBANN that make it superior to backpropagation, its underlying empirical algorithm (Section 3.3). The third set of tests characterize the conditions under which KBANN can be expected to provide results that are superior to standard empirical learning systems (Section 3.4).

The fourth, and final set of tests in this chapter (Section 3.5), explore the relative strengths and weaknesses of the two methods of extracting rules from trained KBANN-nets described in Section 2.4. These tests show the NOFM method to be consistently superior to SUBSET; moreover, the rules extracted by NOFM retain the accuracy of the networks from which they came.

The next chapter continues the empirical studies of this chapter, but has a very different goal. Whereas, this chapter can be characterized as proving the concept underlying KBANN, the experiment in the next chapter assumes that KBANN is effective and applies it to the modeling of learning in school children.

3.1 Experimental Datasets

Before going into the experiments, this subsection describes the two real-world datasets from the domain of molecular biology used for testing KBANN. These datasets are used throughout this chapter and Chapter 5 to test aspects of KBANN. Both datasets have been previously used to demonstrate the usefulness of the KBANN algorithm [Noordewier91, Towell90].

3.1.1 Molecular Biology 101

As much of the empirical testing in this work uses two problems from molecular biology, this section contains a brief overview of the topic and of why it is of interest in computer science. More details can be found in textbooks (e.g., [Watson87]) or [von Heijne87].

DNA is a linear sequence from the alphabet {A, G, T, C}. Each of these characters is referred to as a *nucleotide*. Human DNA consists of approximately $3 * 10^9$ nucleotides. By contrast, the DNA of *E. coli*, a common intestinal bacterial, contains about $5 * 10^6$ nucleotides. A DNA molecule contains a double-strand of these nucleotides arranged in a complementary fashion, so an A on one strand is always paired with a T on the complementary strand. While the property of complementarity is vital for cellular reproduction, it is unnecessary for the purposes of the experiments reported herein. Hence, the rest of this work treats DNA as a one-dimensional sequence of characters.

Almost all living organisms store a complete set of instructions for their life and reproduction using DNA.[1] However, DNA is not directly used in the day to day activities of a cell. Rather, DNA is transcribed into RNA which is, in turn, transcribed into proteins. Proteins are the actual drivers of cells. As proteins are created on the basis of instructions in DNA, mistakes in DNA can lead to incorrect proteins and any number of inheritable disorders (e.g., hemophilia, sickle-cell anemia). Thus, knowing the DNA sequence of humans is a step (albeit small) towards the ability to treat genetic disorders.

The Human Genome Project [Alberts88] is a world-wide effort to determine the sequence of characters in the DNA of humans and other organisms and the location of *genes* in these sequences. (A gene is a portion of the DNA sequence that is transcribed into a protein.) Unfortunately, knowing the sequence of nucleotides in DNA is not equivalent to understanding DNA. Instead, understanding the DNA sequence is roughly analogous to reading a foreign language for which the only information you have is its alphabet.

Currently, biologists have the time to intensively study small sections of the DNA sequence. As a result, translations of many short subsequences are available. However, at the completion of the Human Genome Project, there will be long runs of characters that have not been analyzed. So there is a need for the application of computer-based methods of sequence analysis. At a minimum, this computer-based work must support the human investigators. (For instance, computer-based analysis is currently being used to check DNA sequences for errors [Shavlik90a].) A larger goal is for computer-based analysis to automatically augment

[1]The exception are "retroviruses" such as the one responsible for AIDS which store their genetic information using RNA.

the knowledge of human researchers. To this end, there have been several attempts to apply machine learning to problems that can be studied given knowledge of the DNA sequence. Perhaps the most studied of these problems is the prediction of "protein secondary structure" [Holley89, Qian88, Maclin91, Hunter91].

One of the first problems in developing an understanding of the DNA sequence is to recognize where genes begin and end. The end of genes in DNA are easy to spot, they are marked by three character sequences known as "stop codons". The beginnings of genes are not so clearly marked. However, there are many examples of the beginnings of genes [Harley87, Hawley83]. Moreover, researchers have developed and understanding of the characteristic structure of these examples [O'Neill89]. Unfortunately, understanding the characteristic structure of the beginning of genes does not mean that there exist good techniques for their recognition. Hence, this is a problem that is ripe for an application of machine learning techniques. In fact, work has already been done on this problem [Lapedes89, Lukashin89] and it is one of the problems studied in this thesis.

There are many more problems in the analysis of DNA sequences for which there is both a growing body of examples and some theoretical understanding. For instance, the aforementioned prediction of "protein secondary structure". (For a more complete review of problems in DNA sequence analysis see [von Heijne87].) As the Human Genome Project increases the pace at which sequences of DNA are known, the number of problems with this characteristic will very likely grow. So, there is a present and increasing need for the application of computer-based analysis in molecular biology. The problems investigated in this work are but a first step.

3.1.2 Notation

The two biological datasets in this thesis use a special notation for specifying locations in a DNA sequence. The idea is to number locations with respect to a fixed, biologically-meaningful, reference point. Negative numbers indicate sites preceding the reference point (by biological convention, this appears on the left) while positive numbers indicate sites following the reference point. Figure 3.1 illustrates this numbering scheme.

When a rule's antecedents refer to input features, they first state a location relative to the reference point in the sequence vector, then a DNA symbol, or a sequence of symbols, that must occur. So `@3 'AGTC'` means that an `'A'` must appear three nucleotides to the right of the reference point, a `'G'` must appear four nucleotides to the right of the reference point, etc. By biological convention, position numbers of zero are not used. In rule specifications, `'-'` indicates that *any* nucleotide will suffice.

In addition to this notation for specifying locations a a DNA sequence, Table 3.1 specifies a notation for referring to any possible combination of nucleotides using a single let-

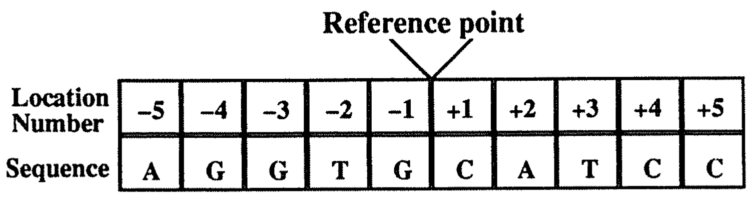


Figure 3.1: The numbering of nucleotide locations in DNA sequences.

Table 3.1: Ambiguity codes for DNA nucleotides.

Code	*Meaning*
M	A or C
R	A or G
W	A or T
S	C or G
Y	C or T
K	G or T
V	A or C or G
H	A or C or T
D	A or G or T
B	C or G or T
X	A or G or C or T

ter. These codes for "nucleotide ambiguity" conform to a standard adopted by the IUB. [IUB Nomenclature Committee85]. They are compatible with the codes used by the EMBL, GenBank, and PIR data libraries, three major collections of data for molecular biology.

3.1.3 Promoter recognition

The first problem studied in this thesis is that of prokaryotic[2] *promoter recognition*. Promoters are short DNA sequences that precede the beginnings of genes. Thus, an algorithmic method of promoter recognition would make it possible to precisely identify the starting locations of genes in DNA for which the sequence is known, but which have not otherwise been analyzed. Promoters can be detected in "wet" biological experiments as they are locations at which the protein *RNA polymerase* binds to the DNA sequence. In addition, several groups of biologists have reported attempts to use neural networks for promoter recognition [Lukashin89, Lapedes89].

The input features for promoter recognition are a sequence of 57 DNA nucleotides. Following biological convention, the reference point for promoter recognition is the site at which gene transcription begins (if the example is a promoter). The reference point is located seven

[2]Prokaryotes are single-celled organisms that do not have a nucleus (e.g., *E. coli* and blue-green algae).

Table 3.2: Rules for promoter-recognition.

```
promoter  :- contact, conformation.
contact   :- minus-35, minus-10.

minus-35 :- @-37 'CTTGAC-'.            minus-35 :- @-37 '-TTG-CA'.
minus-35 :- @-37 '-TTGACA'.            minus-35 :- @-37 '-TTGAC-'.

minus-10 :- @-14 'TATAAT--'.           minus-10 :- @-14 '-TA-A-T-'.
minus-10 :- @-14 '-TATAAT-'.           minus-10 :- @-14 '--TA---T'.

conformation  :-  @-45 'AA--A'.
conformation  :-  @-45 'A---A', @-28 'T---T-AA--T-', @-04 'T'.
conformation  :-  @-49 'A----T', @-27 'T----A--T-TG', @-01 'A'.
conformation  :-  @-47 'CAA-TT-AC', @-22 'G---T-C', @-08 'GCGCC-CC'.
```

nucleotides from the right. (Thus, positive examples contain the first seven nucleotides of the transcribed gene.)

Table 3.2 contains the initial rule set used in the promoter recognition task. According to the rules in Table 3.2, there are two sites at which the DNA sequence must bind to *RNA polymerase* – the minus 10 and minus 35 regions. (These regions are named for their distance from the reference point.) The `conformation` rules attempt to capture the three-dimensional structure of DNA, thereby ensuring that the minus 10 and minus 35 sites are spatially aligned. This set of rules was derived from the biological literature [O'Neill89] by M. Noordewier.

The rule set in Table 3.2 is translated by KBANN into a neural network with the topology

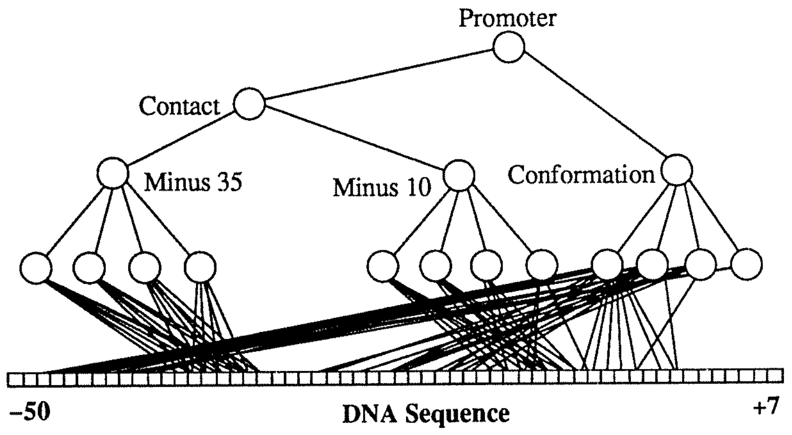


Figure 3.2: The initial KBANN-net for promoter recognition.

shown in Figure 3.2. Recall that KBANN adds additional low-weighted links (not shown) so that if additional sequence information is relevant, the algorithm can capture that information during training.

The set of examples contains 53 sample promoters and 53 nonpromoter sequences. The 53 sample promoters were obtained from a compilation produced by Hawley and McClure [Hawley83]. Negative training examples were derived by selecting contiguous substrings from a 1.5 kilobase sequence provided by Prof. T. Record of the University of Wisconsin's Chemistry Department [Record89]. This sequence is a fragment from *E. coli* bacteriophage *T7* isolated with the restriction enzyme *Hae*III. By virtue of the fact that the fragment does not bind *RNA polymerase*, it is believed to contain no promoters.

Prior to training, the rules in Table 3.2 do not classify any of the 106 examples as promoters. Thus, the rules are useless as a classifier. Nevertheless, they do capture a significant amount of information about promoters.

3.1.4 Splice-junction determination

The second dataset used in this thesis is that of eukaryotic *splice-junction determination.* Splice junctions are points on a DNA sequence at which 'superfluous' DNA is removed during the process of protein creation in higher organisms. The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between *exons* (the parts of the DNA sequence retained after splicing) and *introns* (the parts of the DNA sequence that are spliced out). This problem consists of two subtasks: recognizing exon/intron boundaries (referred to as E/I sites), and recognizing intron/exon boundaries (I/E sites).[3] Figure 3.3 illustrates how splicing occurs during the process of protein creation.

As with the promoter recognition, biologists have attempted to use neural networks from splice-junction determination. The work of Brunak *et al.* is a very complete treatment of the topic [Brunak91].

Figure 3.4 illustrates a prototypical DNA sequence showing both the I/E and E/I borders. This prototypical sequence is the basis of the rules for splice-junction determination contained in Table 3.3.

The dataset used to approach this problem was extracted from the biological literature by M. Noordewier. The full set of examples contains 3190 examples, of which approximately 25% are *I/E* , 25% are *E/I* and the remaining 50% are *neither.* Each example consists of a 60 nucleotide-long DNA sequence categorized according to the type of boundary at the center of the sequence; the center of the sequence is the reference location used for numbering nucleotides. The examples were obtained by taking the documented "split" genes from all

[3]In the biological community, I/E borders are referred to a "acceptors" while E/I borders are referred to as "donors".

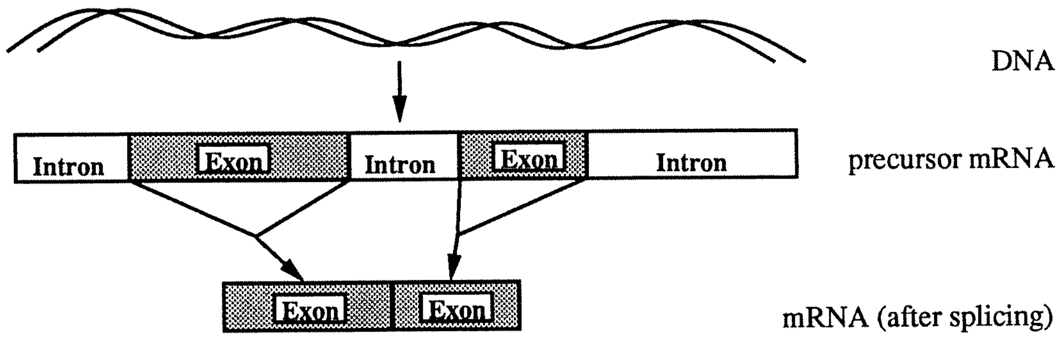
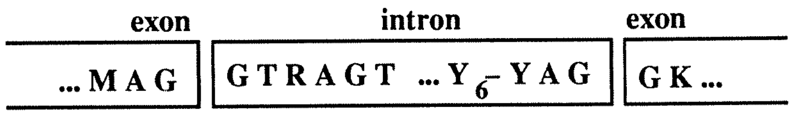


Figure 3.3: The organization of genes in higher organisms.



Y_6 means that either a 'Y' (i.e., a 'C' or a '*T*' — see Table 3.1) occurs in six consecutive locations.

Figure 3.4: "Cannonical" splice-junction.

primate gene entries in Genbank release 64.1 that are described as complete. This procedure resulted in 751 examples of *I/E* and 745 examples of *E/I*. Negative examples are derived from similarly-sized windows, which did not cross an intron/exon boundary, sampled at random from these sequences. (Due to processing constraints, most tests of this dataset use 1000 examples selected randomly from the 3190 examples available.) Networks are configured with output units for the categories *I/E* and *E/I*. The third category (*neither*) is considered true when neither *I/E* nor *E/I* exceeded a threshold. (All tests reported in this work use 0.5 as the threshold.)

In addition to the examples, information about splice-junctions includes the 23 rules in Table 3.3. (This count does not include the rules defined by the iterative construct ``For i from ...'' which define the meaning of `Y'.) This set of rules was derived from the biological literature [Watson87] by M. Noordewier.

Briefly, the splice-junction domain theory specifies generic sequences that appear near the two kinds of sites. In addition to these generic sequences, the I/E boundary is characterized by a *pyramidine-rich* region (i.e., a region where the majority of nucleotides are either C or T). Also, the rules ensure that the current location is not near the end of a gene by looking for certain sequences of letters that tell a cell to stop translating a gene into a protein. The "stop" rules disjunctively check for such indicators near the putative splice site.

The initial set of rules classifies 61% of the examples correctly. As for promoter recognition, the success rate of the splice junction rules initial rules is due largely to their tendency to "just

Table 3.3: Initial rules for splice-junction determination.

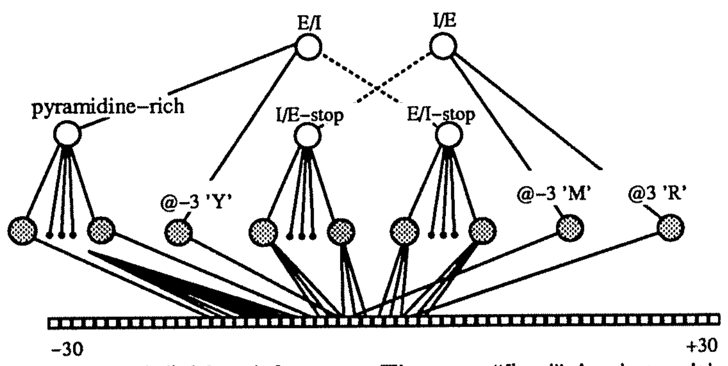
```
E/I :- @-3 'MAGGTRAGT', not(E/I-stop).

E/I-stop ::- @-3 'TAA'.    E/I-stop ::- @-4 'TAA'.    E/I-stop ::- @-5 'TAA'.
E/I-stop ::- @-3 'TAG'.    E/I-stop ::- @-4 'TAG'.    E/I-stop ::- @-5 'TAG'.
E/I-stop ::- @-3 'TGA'.    E/I-stop ::- @-4 'TGA'.    E/I-stop ::- @-5 'TGA'.

I/E :- pyramidine-rich, @-3 'YAGG', not(I/E-stop).
pyramidine-rich :- 6 of (@-15 'YYYYYYYYYY').
For i from ((-30 to -1) and (+1 to +30))
        {@<i> 'Y' ::- @<i> 'C'.  @<i> 'Y' ::- @<i> 'T'.}

I/E-stop ::- @1 'TAA'.    I/E-stop ::- @2 'TAA'.    I/E-stop ::- @3 'TAA'.
I/E-stop ::- @1 'TAG'.    I/E-stop ::- @2 'TAG'.    I/E-stop ::- @3 'TAG'.
I/E-stop ::- @1 'TGA'.    I/E-stop ::- @2 'TGA'.    I/E-stop ::- @3 'TGA'.
```

See Table 3.1 for meanings of letters other than A, G, T, C. The notation '::-' indicates a rule that is a definition. Hence, it is not to be altered during learning. The construct "For i ..." creates 120 rules that define a disjunct at each location in the input. Consequents with an antecedent of the form 'n of (...)' are satisfied if at least n of the parenthesized antecedents are true.

say no"; the rules correctly classify only 40% of the I/E and 3% of the E/I examples. Figure 3.5 depicts an abstracted version of the network that KBANN generates given these rules. Finally, note that this initial set of rules creates a KBANN-net that has only three trainable hidden units. Moreover, these three units are not shared among the two output units. Rather, the *I/E* output units has only a single trainable hidden unit so it is capable no more than perceptron-style learning [Rosenblatt62].

3.2 KBANN versus other Learning Systems

Tests in this section explore the hypothesis that KBANN is an effective and efficient learning tool in terms of its ability to generalize to examples not seen during training. To test this hypothesis, KBANN is compared in the following section to empirical learning systems using the splice junction and promoter datasets. In the subsequent section, KBANN is compared to other systems which learn from both theory and data.



Shaded units represent definitional features. They are "fixed" in that neither the weights of incoming links nor the bias may change during training.

Figure 3.5: The initial splice-junction KBANN-net.

3.2.1 KBANN and empirical learners

This section compares the generalization ability of KBANN to systems that learn strictly from training examples. The comparisons are run in three ways. First, KBANN is compared to six algorithms for its ability to extract information from the training examples. Second, KBANN is compared backpropagation — the most effective of the six empirical algorithms — for its ability to learn from small sets of examples. Third, KBANN and backpropagation are compared in terms of their relative training effort.

Seven-way comparison of algorithms

In this section, KBANN is compared to six empirical learning algorithms: standard backpropagation [Rumelhart86], ID3 [Quinlan86], "nearest neighbor", PEBLS [Cost90], perceptron [Rosenblatt62], and Cobweb [Fisher87]. The results of these tests will show that, in almost every case, KBANN is statistically significantly superior to these strictly-empirical learners.

Standard backpropagation [Rumelhart86] is used as a comparison algorithm for two reasons. First, several empirical studies have shown it to be at least as effective at generalizing to examples not seen during training as symbolic learning algorithms [Shavlik91, Fisher89, Weiss89, Atlas89]. Second, standard backpropagation is the empirical algorithm upon which KBANN is based. Hence, comparisons between KBANN and standard backpropagation provide an indication of the utility of the information which the rules-to-networks translator inserts into the networks it creates. Following Shavlik *et al.* [Shavlik91] the number of hidden units in standard ANNs is set to 10% of the number of input units.[4] Hidden units are arranged in a

[4]Thus, standard ANNs for promoter recognition and splice-junction determination had 23 and 24 hidden units, respectively. Unreported empirical testing verified that ANNs with these numbers of hidden units are about as effective at categorizing testing examples as networks containing anywhere from five to 60 hidden units.

single layer that is fully-connected with both the input and output layers. There are no direct connections between inputs and outputs.[5] All weights are randomly initialized to numbers within 0.5 of zero.

ID3 [Quinlan86] is a symbolic empirical learning algorithm. It uses training data to construct a "decision tree".[6] Briefly, each step of the algorithms adds a new node to the decision tree by partitioning the training examples based on their value for the single, statistically most-informative, feature. Partitioning stops when all examples in a partition are in the same category. ID3 is included in this comparison set because it is a widely-used algorithm that has been tested on a number of large datasets. Also, experimental comparisons with other symbolic learning systems show that ID3 generally performs as well or better than other systems [O'Rorke82, Rendell89]. In addition, ID3 is the underlying empirical algorithm for EITHER, a hybrid system compared to KBANN in Section 3.2.2.

Nearest neighbor represents an extremely simple – but often very effective (e.g., [Aha91]) – approach to machine learning. It compares the current instance to all known instances, locating exact matches or the k most similar matches. The classification of the instance is the classification of the majority of the k most similar neighbors. With distance defined as the number of mismatched nucleotides, $k = 3$ was found to work best on both biological datasets.

PEBLS [Cost90] is an extension on the idea of nearest-neighbor algorithms. The extension allows weights to be associated with both features and examples. Testing examples take the class of the single nearest example in the training set (the distance to each examples is a function of its weight as well as the weight on each feature). PEBLS is very similar to radial basis functions [Moody88], a nearest-neighbor algorithm expressed in terms of neural networks.

Perceptron [Rosenblatt62] is an early neural learning algorithm that differs from standard backpropagation in that it lacks hidden units. Hence, it is only able to learn concepts that are linearly separable [Minsky88]. Nevertheless, it is attractive by comparison to standard backpropagation for its relatively simple algorithm and because it is amenable to formal analysis [Rosenblatt62, Minsky88]. Moreover, recent tests that compare standard backpropagation to perceptron show that on real-world tasks the perceptron is only marginally inferior to standard backpropagation [Shavlik91]. Perceptron handles multiple categories by using one output unit per category. As output units in perceptrons are either "on" or "off", it is possible for Perceptron to indicate that any number from zero to the number of possible categories are true.

[5]Wieland and Leighton have show that a neural network with sufficient units in a single hidden layer is sufficient to learn most concepts [Wieland87].

[6]A decision tree is tree in which internal nodes are labeled with a feature-name. Branches from a node correspond to the possible values of that labeling feature. Leaves of the tree are labeled using class names. To determine the class of an example, start at the root of the tree and traverse down to a leaf following the features and values that label each node and branch.

Cobweb [Fisher87], the final algorithm investigated, is an incremental system for *conceptual clustering.* It operates by building a classification tree in which each node is a probabilistic concept that represents an object class. The incorporation of an object is a process of descending the tree and updating counts (statistics are maintained for each feature and each node in the tree) along the appropriate path. When the bottom of the tree is reached, an object is incorporated into the tree by either: (a) simple inclusion into an existing cluster, (b) creation of new clusters by splitting an existing cluster, or (c) the merger several existing clusters into a single cluster. The particular implementation of Cobweb, is specialized for classification tasks by the addition of mechanisms for manipulating the class of each example as a special kind of feature [Gennari90]. This algorithm is tested as it is the basis of Labyrinth-k [Thompson91], a hybrid learning method whose performance is compared to KBANN in Section 3.2.2.

Method. Following Weiss and Kulikowski's suggestion [Weiss90] the systems were evaluated using repeated ten-fold cross-validation for splice-junction determination and leaving-one-out testing for promoter recognition.[7] In N-fold cross-validation, the set of examples is partitioned into N sets of equal size. A system is trained using $N - 1$ of the sets and tested using the remaining set. This procedure is repeated N times so that each set is used as the testing set once. In 10-fold cross-validation the initial ordering of examples affects the split of training and testing examples. As a result, each system is tested on the splice-junction problem using eleven different example orderings.[8] The error rates reported are simple averages of the eleven trials.

"Leaving-one-out" is an extreme form of N-fold cross-validation in which N is equal to the number of examples in the set. So, for promoter recognition leaving-one-out requires 106 training passes in which the training set has 105 examples and the testing set has one example. Example order has no affect on leaving-one-out testing as the test set always consists of a single example. However, every algorithm other than ID3 and nearest neighbor is sensitive to the order of example presentation. Hence, each algorithm, other than ID3 and nearest neighbor, was tested using eleven different orderings of the 106 promoter examples. Rather than simply recording the average number of errors over the eleven trials, statistics were maintained for each example. Only those examples that were incorrectly classified by the majority of the eleven trials were regarded as incorrect. This error scoring method was used in an attempt to capture the best result of each algorithm; it always results in a decrease in the number of errors. For example, Cobweb is quite sensitive to presentation order. It misses many examples two or

[7]Due to processing limitations, results in Figure 3.8, and elsewhere, reflect testing on a 1000 example subset drawn randomly from the 3190 available examples.

[8]Eleven permutations of both the promoter and splice-junction examples were used throughout this thesis. Eleven was chosen as it represents a level that is computationally feasible, but provided a sufficient base-line for reliable statistical analysis.

three times, but misses few examples more than six times. Hence, the simple average error rate for Cobweb is more than double than the level reported in Figure 3.7.[9] Conversely, perceptron is relatively insensitive to presentation order. Therefore, this definition of incorrectness has little effect on perceptron; its error rate drops by less than one example. The effect on other algorithms is between these extremes, reducing the error rate by from one to two examples.[10]

For KBANN and standard backpropagation, classification of examples is made with respect to a threshold. If the activation of the output unit is greater than the threshold, then the example takes the category of the most active output unit. Otherwise, the example is assigned to the negative category. For instance, in the splice-junction problem, if the output activations of the E/I and I/E outputs are 0.8 and 0.6 respectively and the threshold is 0.5, then the example would be categorized as E/I. Conversely, if the activations of the E/I and I/E units are 0.1 and 0.2 and the threshold is still 0.5, then the category assigned by the network would be *neither*.

The initial randomized configuration of a neural network can affect learning. To compensate for this, an extra level of testing is required for both standard backpropagation and KBANN. Specifically, each of the eleven example permutations of each dataset is tested using ten different initial states. Hence, each test requires 110 separate runs of n-fold cross-validation.[11] This test design is illustrated in Figure 3.2.1. The classification derived from the ten trials varying the starting state is based on the average of the output unit activations. So, if the output unit of a promoter recognition network is 0.52 on 9 trials and 0.01 on the tenth trial, then the classification of the network is based upon the relationship between 0.46 and the threshold.[12]

Networks are trained until one of three following stopping criteria is satisfied:

1. On 99% of the training examples the activation of every output unit is within 0.25 of correct.

2. Every training example has been presented to the network 100 times. (I.e., the network has been trained for 100 *epochs.*)

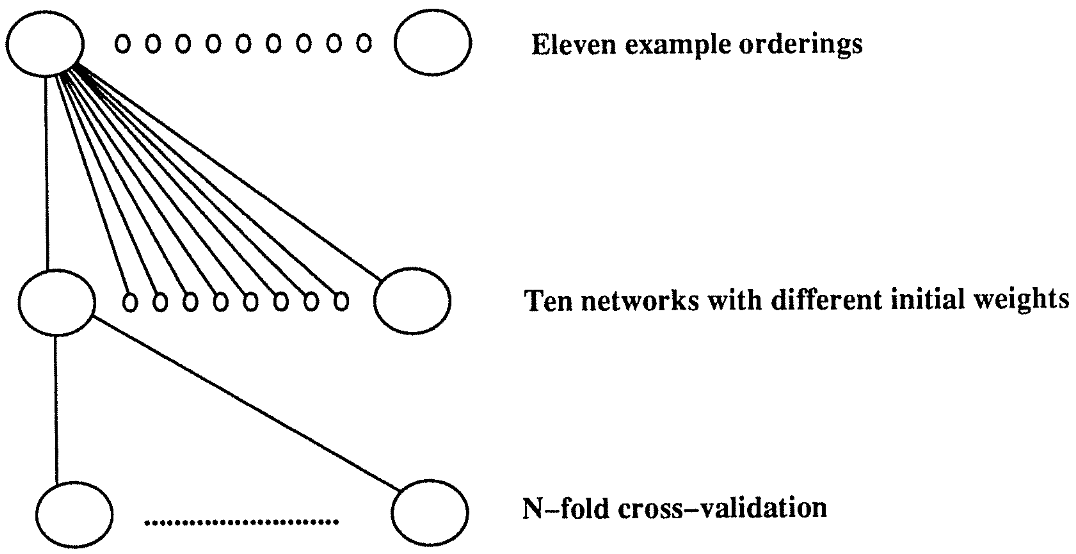
3. The network is classifying at least 90% of the training examples correctly but has not improved it ability to classify the training examples for five epochs. (This implements

[9]For an incremental system like Cobweb, it may not be reasonable to count promoter errors following the method described. The method implies that an error is only an error if it occurs most of the time, over different instance orderings. However, the hypothesis of incremental systems is that the user cannot do this sort of post-collection analysis. Instead, the algorithm is expected to execute and provide answers in real-time and provide answers while running. As a result, the simple average error rate is probably the best indicator of the true error rate for Cobweb. Nevertheless, numbers reported for Cobweb reflect the counting method used for the other algorithms to allow controlled comparisons.

[10]Due to processing time considerations, all subsequent tests of the promoter domain use simple average error rates and 10-fold cross-validation.

[11]For the promoter dataset in which leaving-one-out testing is used, this meant training 11,660 networks!

[12]For all tests, the threshold was set to 0.5.



Lines in this figure indicate tests. Lines are only drawn from the leftmost circle at each level.

Figure 3.6: The design of experiments to test the generalization accuracy of learning systems based on neural networks.

the *patience* stopping criteria suggested by Fahlman and Lebiere [Fahlman89] and which Squires [Squires91] showed is a useful method of preventing overfitting of the training data.)

In general, networks trained for promoter recognition stopped training on the first criterion – they are able to rapidly learn the training data. On the other hand, KBANN-nets for splice-junction determination often terminate on the second or third criterion because they are unable to perfectly learn the training data.

The results for the full comparison of algorithms uses the "relative information score" (RIS); a metric for evaluating learning efficiency suggested by Kononenko and Bratko [Kononenko91]. This information-theoretic measure is based upon a comparison of the information extracted by the learning algorithm versus the total information required to perfectly partition the training data.

Calculation of the relative information score is given by the Equations 3.1–3.4 in Table 3.4. Briefly, Equation 3.1 defines "entropy" for classification problems; entropy is the total amount of information required to perfectly partition the training examples. The information provided by the classification system is asymmetric (Equation 3.2), requiring a different function when it overpredicts a category than when it underpredicts a category. Equation 3.3 defines the total information provided by the classifier to be the average of the information derived from each example. Finally, Equation 3.4 defines the relative information score to be the average information provided by the classifier divided by the information required to perfectly partition

Table 3.4: Calculation of the Relative Information Score

$$E = \sum_{j}^{N} P(C_j) * log(P(C_j)) \tag{3.1}$$

$$I = \begin{cases} \text{-log(P(C)) + log(P'(C))} & \text{if P(C) < P'(C)} \\ \text{-log(1 - P'(C)) + log(1 - P(C))} & \text{if P(C) > P'(C)} \end{cases} \tag{3.2}$$

$$I_a = \frac{1}{T} \sum_{k}^{T} I(k) \tag{3.3}$$

$$RIS = \frac{I_a}{E} * 100 \tag{3.4}$$

where:	P(C)	is the given probability that an example is in class C
	P'(C)	is the probability of class membership estimated by the classifier
	T	is the number of training examples
	N	is the number of different categories

the examples.

The advantage of this measure is that it accurately reflects learning over domains in which the distribution of examples is skewed or in which there is more than one decision to be made. For instance, consider a domain in which 90% of the examples fall into a single class. Merely guessing the majority class results in a 10% error rate but an RIS of only 26. (When a two-category problem has an equal number of examples in each category, always guessing one category gives an RIS of 0.0.) Hence, RIS is used in this section to compare the effectiveness of each of the learning systems. Unfortunately, RIS requires considerably more detailed record keeping than simple error rates. Therefore, all subsequent tests in this thesis report simple error rates.

Results and discussion. Figures 3.7 and 3.8 present the comparisons of KBANN to the six empirical-learning algorithms described above. In addition to the above described empirical algorithms, Figure 3.7 contains the accuracies of two methods suggested by biologists for promoter recognition. The first method, labeled "Stormo", is a method which learns through the presentation of positive-only examples [Stormo90]. The second method, labeled "O'Neill" [O'Neill89], is a non-learning, hand-refined, technique for differentiating between promoters and non-promoters.[13]

[13]Stormo's technique for constructing consensus sequences is a learning algorithm. His approach collects a set of aligned examples of some category and counts how many times each nucleotide appears at each position. This information is used to build a scoring matrix, which is then applied to new sequences; those that score

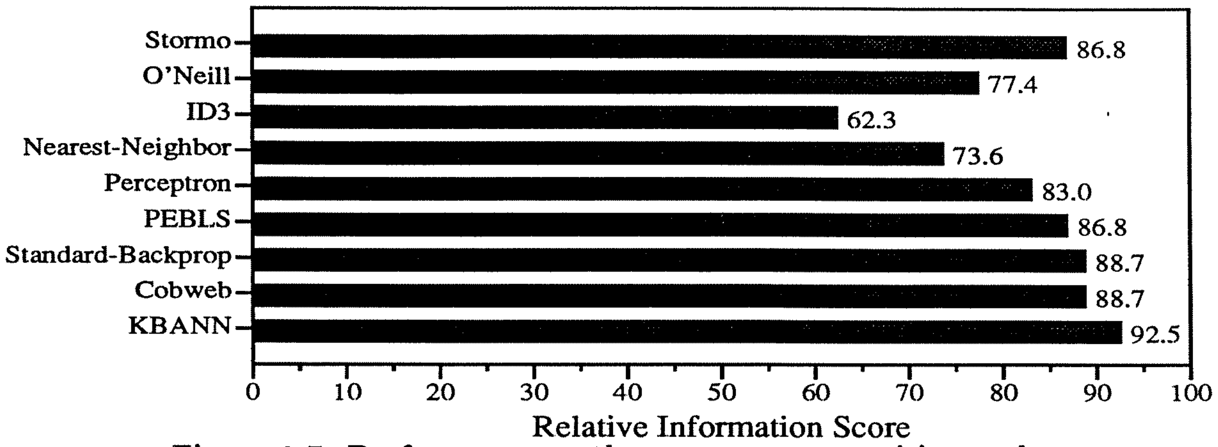


Figure 3.7: Performance on the promoter recognition task.

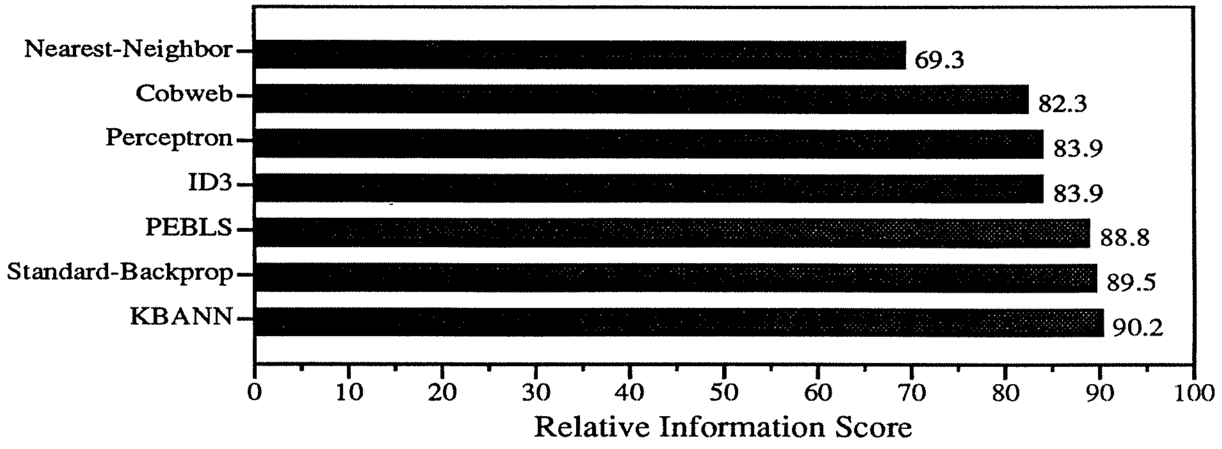


Figure 3.8: Performance on the splice-junction determination task with 1000 training examples.

KBANN is superior to every empirical learning systems tested on both of the biological domains. In most cases, differences are statistically significant with 99.5% confidence (see Tables D.1 and D.2 in Appendix D). The lone exception to this generalization is that KBANN is only marginally superior to standard backpropagation on the splice-junction problem. The comparatively poor performance of KBANN on this problem can be at least partially attributed to the sparseness of the initial theory of splice junctions (see Table 3.3 or Figure 3.5). Section 5.2 describes experiments in which the standard KBANN-net has been augmented with extra hidden units to improve the generalization performance of the KBANN-net.

above some threshold (we used a threshold of zero) are predicted to be members of the class being learned.

O'Neill analyzed the Hawley and McClure [Hawley83] promoters and produced a collection of filters to be used for promoter recognition[O'Neill89]. If a sequence properly matches his filters, it is classified as a promoter. We directly implemented his approach. (Note that the promoter domain theory was derived from O'Neill's paper; however, it is not identical to the promoter-finding technique he proposed.)

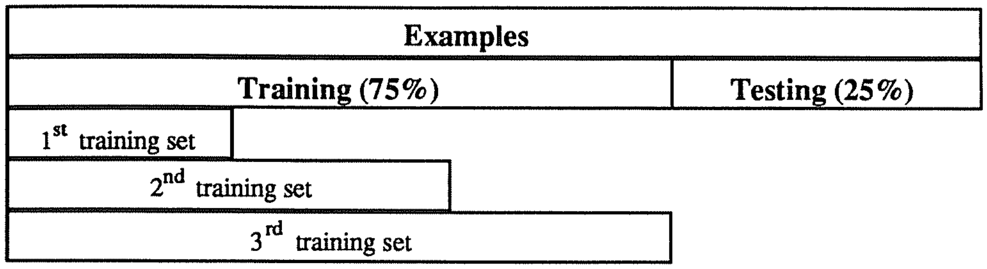


Figure 3.9: The partitioning of a set of examples to form a learning curve.

Learning from small sets of examples

One of the predictions in the thesis statement is that a system which learns using both theory and data should be more efficient than a system that learns from data alone. One aspect of this efficiency hypothesis is that a theory and data learning system should require fewer training examples than systems that learn only from data. Hence, the performance of algorithms when a large amount of training examples are available, as exemplified by the tests in the prior section, is only part of the story of the effectiveness of learning. The ability to learn from relatively few training examples is important because training examples are often hard to find or are expensive to collect. Thus, it is important for a learning system to be able to extract useful generalizations from a small set of examples.

Method. These tests compare KBANN to only standard backpropagation as the above tests show it to be the most effective of the systems for learning from examples only. "Learning curves" are built by splitting the set of examples into two subsets, one containing approximately 25% of the examples and the other containing the remaining examples. The 25% set is put aside for testing. Then, the 75% set is partitioned into sets of increasing size, such that smaller sets are always subsets of larger sets. Networks are trained using subsets of the 75% set and tested using the 25% set. Figure 3.9 illustrates the partitioning of the examples to form learning curves.

This partitioning is dependent upon the ordering of the examples. Hence, it is repeated eleven times with the same permutations of the examples used in the comparison of algorithms. As above, these tests are repeated using ten initial sets of network weights. (The ten initial weight sets used are the same as those in the seven-way comparison of algorithms.)

Results and discussion. The results presented in Figures 3.10 and 3.11 verify the hypothesis that KBANN efficient by comparison to standard backpropagation in terms of its ability to extract accurate generalizations from small sets of training examples. For both the promoter and splice junction datasets, the test set error rate, given a small set of training data, for KBANN is about 20% less than that of standard backpropagation. The difference steadily

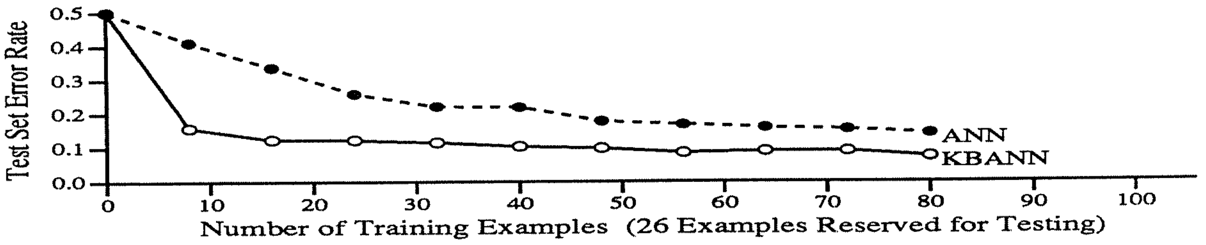


Figure 3.10: Learning curves for the promoter domain with 106 examples.

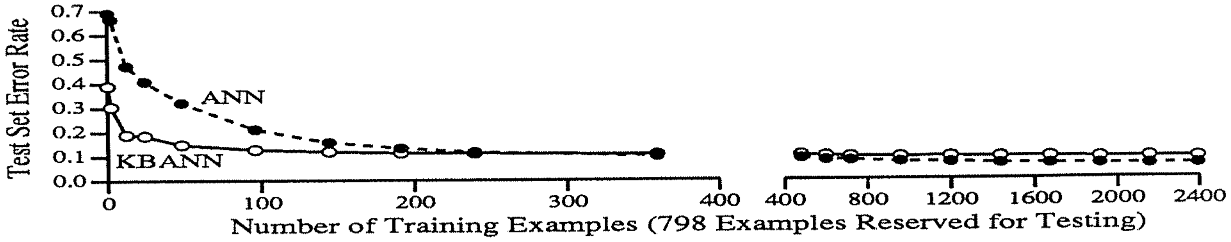


Figure 3.11: Learning curves for the splice-junction domain with 3190 examples.

declines on both datasets. On the promoter set, (Figure 3.10) KBANN is at an advantage of about 4% after training using the largest training set. On the other hand, KBANN is at a 2% disadvantage to backpropagation on the splice junction set after training on the largest available set of training examples.[14]

A different way of analyzing these learning curves is in terms of how quickly error rates reach the levels attainable using every training example. On the promoter dataset, KBANN-nets require only about 15 training examples to achieve an error rate within 5% of that achieved with the full set of 80 training examples. On the other hand, standard backpropagation requires more than 50 training examples to approach the levels it achieves with 80 training examples. The results on the splice-junction problem are similar. KBANN requires roughly one-third the number of training examples required by backpropagation to reach error rates on testing examples that are close the levels asymptotically attained by each system.

[14]At 900 training examples, in Figure 3.11, KBANN is at a 1.2% disadvantage to standard ANNs. This contrasts with results reported in the seven-way comparison that show KBANN superior to standard ANNs when trained with 900 examples. This dichotomy reflects a difference between the relative information score used in the seven-way comparisons and the simple error rates used here.

Learning and generalization as a function of training effort

The previous section, which showed that KBANN effectively generalizes from small sets of training examples, tested one aspect of the hypothesis that KBANN is an efficient learning method. This section tests a second aspect of the efficiency hypothesis. Specifically, this section tests the hypothesis that the learning rate of KBANN is superior to that of standard backpropagation. This issue is important because one of the weaknesses of standard backpropagation (listed in Section 1.1) is that its learning rate is orders of magnitude slower than that of symbolic learning systems such as ID3 [Shavlik91]. Therefore, methods for enhancing the learning speed of systems based upon backpropagation are desirable.

Method. To test the hypothesis that KBANN speeds learning, datasets are partitioned into two subsets: a testing set containing ten percent of the examples and a training set containing ninety percent of the examples. The training set is then used to train both a standard backpropagation network (configured as described in the seven-way empirical test) and a network created by the rules-to-network translator. Each network is trained until every example is correctly classifies or until every example has been presented 50 times. For the purposes of training, an example is considered correctly classified if every output is with 0.25 of correct. After each training epoch (i.e., after each example in the training set has been presented once), networks are tested for their ability to correctly classify each example in both the training and testing sets. For testing, a "best-guess" criterion is used. Under this criterion, an example is considered correctly classified if the output unit with the maximum activation is with 0.5 of the correct activation. (These training and testing methods replicate the procedures of the seven-way comparisons above with the exception that training is not terminated using the patience criterion.)

This procedure is repeated 50 times to smooth fluctuations due to randomness in split of examples and in the assignment of initial weights to the networks.

Results and discussion. Figures 3.12 and 3.13 plot the error rates for KBANN-nets and standard backpropagation on training and testing examples (according to the best-guess criterion) at intervals of one training epoch on the promoter and splice-junction datasets. Curves in both figures start after one training epoch. Prior to training, KBANN-nets and standard backpropagation both have a 0.5 error rate on the promoter set. (While these error rates occur for different reasons; standard ANNs randomly guess whereas KBANN-nets always guess no.) On the splice-junction problem, the initial error rate for KBANN-nets is 0.39 while it is 0.69 for standard backpropagation.

The two plots have several points in common. First, after one training epoch, standard ANNs and KBANN-nets have about equal accuracy on the training set. Second, when measures

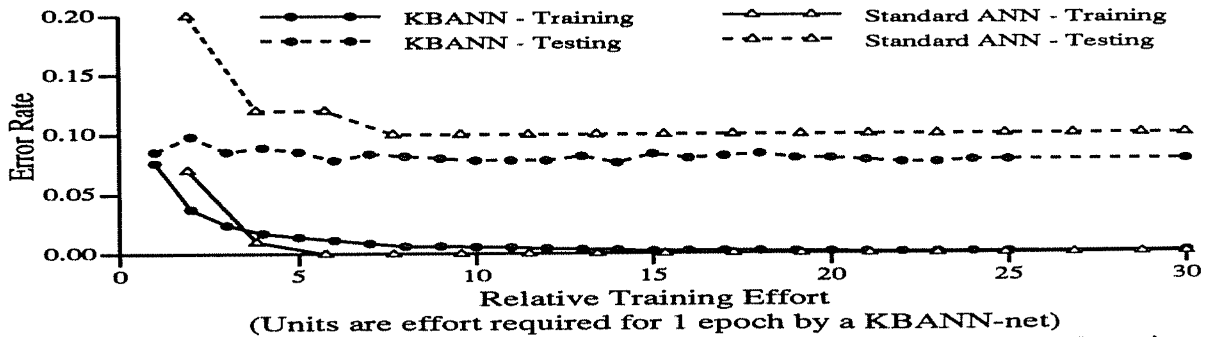


Figure 3.12: Learning as a function of training effort on the promoter domain.

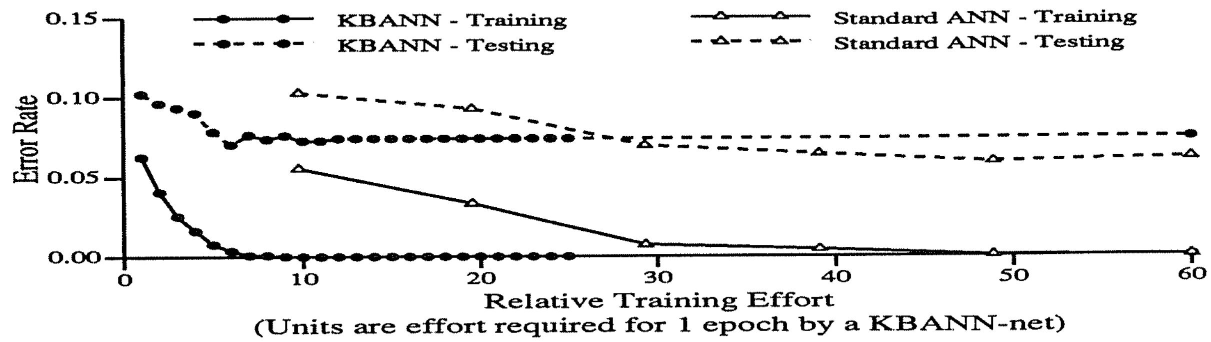


Figure 3.13: Learning as a function of training effort on the splice-junction domain with 1000 examples.

in terms of training epochs, standard ANNs rapidly converge to correctly classifying every training example correctly. (After convergence, networks are not trained further. Hence, the test set correctness for standard backpropagation is a strait line beyond ten epochs on both figures.) Third, KBANN-nets converge only after seeing the examples many more times than standard ANNs. On the promoter dataset, 25 epochs are required for every of the 50 KBANN-nets to converge.

Beyond addressing learning rate, these figures present evidence about a second important point. Specifically they show little evidence that the networks overfit the training set. The expectation of overfitting is that learning the training set to a high level of accuracy reduces the ability of the network to generalize correctly on the testing set. Hence, overfitting can be expected result in a slow decline in test set accuracy as accuracy on the training set approaches its maximum level. As this slow decline in test set accuracy is not present in the either Figure 3.12 or 3.13, it is reasonable to conclude that overfitting is not a serious concern on either dataset.

Finally, recall that these tests do not use a *patience* criterion [Fahlman89, Squires91] to terminate training. Had this criterion been used, no KBANN-net for either promoter recognition of splice-junction determination would have received more than ten epochs of training.

Discussion of the comparison of KBANN to empirical learning algorithms

The results reported in Figures 3.7 and 3.8 describe only the asymptotic performance of each system. On the basis of these comparisons, KBANN is the most effective learning algorithm.

Many problems are characterized by a paucity of data. Hence, the performance of systems when there is little data available is at least as important as the performance when a large collection of examples is available. Thus, the learning curves in Figures 3.10 and 3.11 that compare the performance of backpropagation to that of KBANN are at least as important as the results in Figures 3.7 and 3.8. Both Figures 3.10 and 3.11 clearly show the advantage of KBANN-nets when there are few training examples available. Therefore, it is reasonable to conclude this section with a paraphrase of an old axiom, namely "A domain theory is worth 100 examples."

3.2.2 KBANN and theory & data learners

In this section, KBANN is compared to two systems which learn from both theory and data. The first system is Ourston and Mooney's EITHER [Ourston90, Mooney91b, Ourston91]. This system uses ID3 to make minimal corrections to domain theories. The second system is Thompson, Langley, and Iba's Labyrinth-k [Thompson91], an extension of Labyrinth to allow the insertion of domain knowledge. (Labyrinth is, in turn, based on Cobweb [Fisher87].) Like KBANN, these systems are able to use initial knowledge that is both incorrect and incomplete as the basis for learning. (These systems are described in more detail in Chapter 6.)

Testing methodology

Results for both Labyrinth-k and EITHER were supplied by the authors of the respective systems, but are based on published sources [Ourston90, Thompson91]. Fortunately, the systems were tested using virtually identical procedures. The testing of KBANN follows, as closely as possible, the method used by these systems.

The method used was to produce learning curves for only the promoter dataset. (Neither Labyrinth-k nor EITHER have been tested on the splice-junction problem. EITHER cannot currently be applied to the splice-junction problem as it is unable to handle negated antecedents.) Learning curves were created using a methodology very similar to that described in the experiments above. Specifically, the initial step was to partition the promoter dataset into a test set of 26 examples and a training set of 80 examples. The training set was further partitioned into sets of 10, 20, ..., 80 examples in which smaller sets are subsets of larger sets. These subsets were used for training. To smooth statistical fluctuations, this procedure was repeated

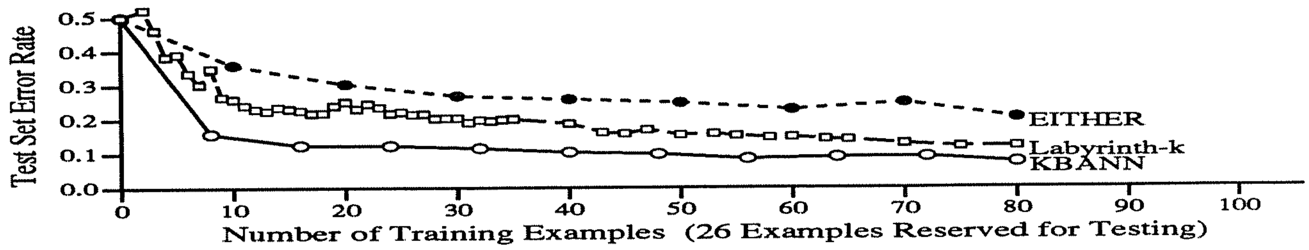


Figure 3.14: Accuracy of Theory/Data systems for Promoter Recognition.

five times.

Results

Figure 3.14 plots the test set correctness of each of the three hybrid systems. KBANN is consistently at least 5% better than both of the other systems. Aside from an initial blip in the performance of Labyrinth-k, EITHER consistently has the poorest performance.

3.2.3 Discussion of the empirical comparisons

In summary, these results indicate that KBANN is superior to both (a) systems that learn only from examples (i.e., empirical learners) and (b) systems that, like KBANN, learn from both theory and data.

KBANN's relative weakness on the splice-junction dataset has been attributed to two factors. First, the splice-junction domain theory is relatively poor. It allows for little more than perceptron-like learning. Hence, a richer domain theory might be expected to improve the performance of KBANN-nets. This hypothesis cannot currently be tested as there is no better domain theory for splice junctions.

A second hypothesis to explain the relatively poor performance of KBANN on the splice-junction problem is that standard backpropagation is better able to make use of the large dataset available in the splice-junction domain. An alternate view of this hypothesis is that the domain theory prevents the network from learning a solution to the problem that is as accurate as a solution learned without knowing the domain theory. This hypothesis suggests that standard backpropagation should outperform KBANN only when large amounts of training data are available. The learning curve in Figure 3.11 supports this hypothesis.

While the results presented in this section demonstrate that KBANN is an effective learning technique, they fail to precisely identify the aspects of KBANN that lead to its generalization ability. This topic is the subject of the tests in the next section.

3.3 Sources of KBANN's Strength

Results presented in the prior section indicate that KBANN is generally superior to both systems 'that learn from theory and data (Figure 3.14) as well as systems that learn from data alone (Figures 3.7 and 3.8). This section further explores these results. In particular, the first subsection tests (and rejects) the hypothesis that the difference between KBANN and other hybrid learning systems is due only to differences in the learning algorithms underlying each system. The following subsection tests three hypotheses that attempt to explain why KBANN is superior to backpropagation, its underlying empirical learning algorithm.

3.3.1 Why is KBANN superior to other theory & data learners?

One hypothesis to account for KBANN's superiority to EITHER and Labyrinth-k is that the difference is due wholly to the superiority of KBANN's underlying learning algorithm. In other words, KBANN is superior to both EITHER and Labyrinth-k because backpropagation is superior to both ID3 and Labyrinth. Data presented in Figures 3.7 and 3.8 support this contention; these figures show backpropagation to be superior to both ID3 and Cobweb[15] on both the promoter and splice-junction datasets. Figure 3.15, which compares backpropagation, ID3 and Cobweb using the same methodology as Figure 3.14, lends further credence to this hypothesis. In addition, a number of empirical studies suggest that, over a wide range of conditions, backpropagation is as good or better than other empirical learning systems at classifying examples not seen during training [Shavlik91, Fisher89, Weiss89, Atlas89]. Therefore, it is reasonable to conclude that some of the power of KBANN is due simply to the superiority of its underlying empirical learning algorithm.

Yet, Figure 3.16 indicates that there is more to KBANN than simply backpropagation. This figure plots the improvement of each hybrid system over its underlying empirical algorithm as a percentage of the total possible improvement. It shows that KBANN proportionally improves more on its underlying learning algorithm than both EITHER and Labyrinth-k. In other words, KBANN makes more effective use of the domain knowledge than both EITHER and Labyrinth-k. This result refutes the hypothesis raised in this section that KBANN's effectiveness is due *wholly* to the relative superiority of its underlying learning algorithm.

3.3.2 Why is KBANN superior to standard backpropagation?

The results just presented indicate that KBANN's superiority to other hybrid systems is due partly, but not wholly, to the superiority of backpropagation. However, the results do not address why KBANN improves upon backpropagation. That is the topic of this section.

[15]Labyrinth slightly outperforms Cobweb [Thompson91]

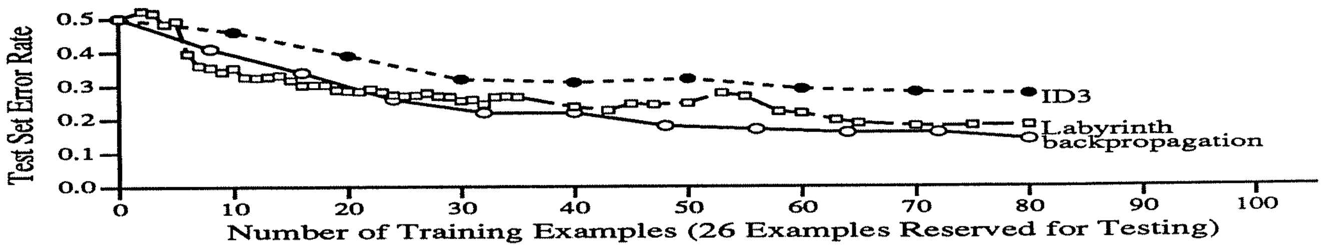
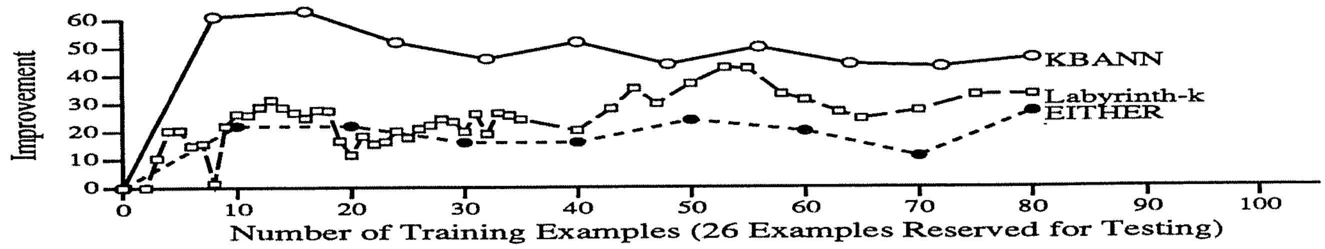


Figure 3.15: Accuracy of empirical algorithms underlying Theory/Data systems on the promoter data.



$$improvement = 100 * \frac{< error\ rate\ of\ underlying\ alg > - < error\ rate\ of\ hybrid\ alg >}{< error\ rate\ of\ underlying\ alg >}$$

Figure 3.16: Improvement of hybrid learning systems over their underlying empirical algorithms.

Three hypotheses may account for KBANN's abilities with respect to backpropagation:

1. *Structure is responsible for* KBANN *'s strength.* That is, the topology of a KBANN-net is better suited to learning in the particular domain than a standard ANN (i.e., an ANN with a single layer of hidden units and complete connectivity between layers).

2. *Initial weights are responsible for* KBANN *'s strength.* That is, the initial weights of a KBANN-net select critical features of the domain, thereby simplifying learning and reducing problems resulting from spurious correlations in the training examples.

3. *Both structure and initial weights are required to account for* KBANN *'s strength.* Neither the initial structure nor the initial weights alone are sufficient to account for the superiority of KBANN. It is the combination of structure and weight that account for KBANN's strength.

The following subsections test each of these hypotheses using the Promoter domain.

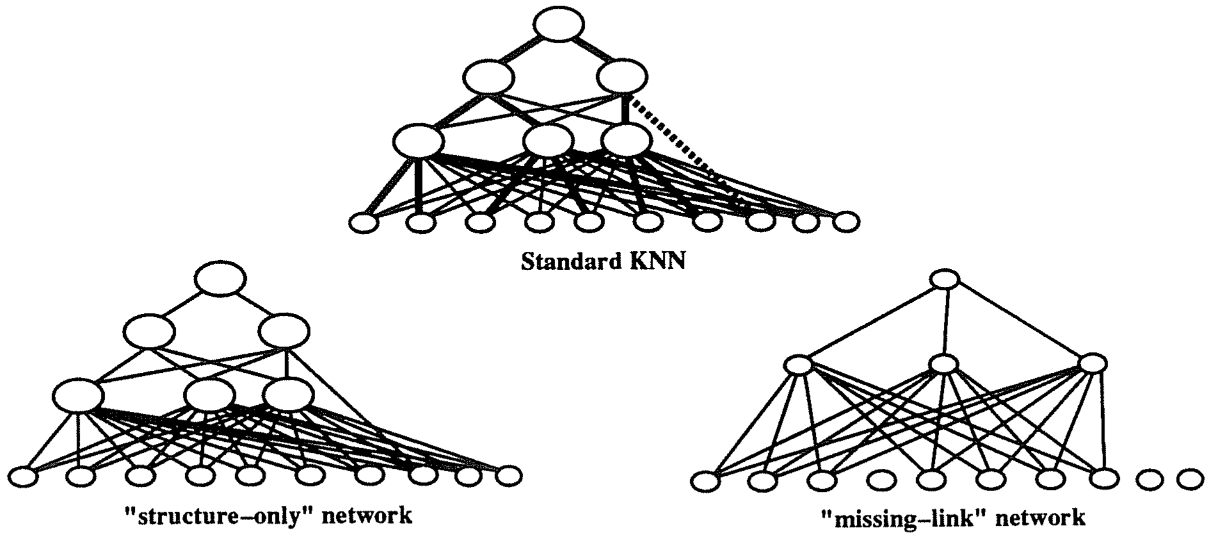


Figure 3.17: Standard, structured, and weighted networks.

Structure is responsible for KBANN's strength

If the first hypothesis is correct, then a network which has the structure given by the rules, but initially random weights, should be as effective at generalizing to testing examples as a standard KBANN-net.

Method. This hypothesis can be tested by creating networks that have the exact structure of a KBANN-net but in which all link weights and biases are randomly distributed around zero. The learning abilities of these "structure-only" networks can then be compared to that of standard KBANN-nets. Figure 3.17 graphically depicts the difference between standard and structured networks.

To control this experiment as tightly as possible, the following experimental design is used. First, a KBANN-net is created. The network is duplicated except that all link weights set by the domain knowledge are reset to a randomly selected value within 0.5 of zero. Likewise, all biases are set randomly selected value within 0.5 of zero. This pair of networks is tested using eleven repetitions of ten-fold cross-validation. (The example ordering in the eleven repetitions are the same as those used in previous experiments.)

To smooth any differences resulting from slightly different initial weight settings, this procedure is repeated ten times. Hence, the structure-only networks and standard KBANN-nets are compared using 110 repetitions of ten-fold cross-validation.

Results. Table 3.5 presents the results of these tests. The results clearly refute the hypothesis that the strength of KBANN arises solely from the determination of network topology. Not only

Table 3.5: Learning by standard KBANN-nets and structure-only networks.

Network type	Error Rate	Training Epochs
Promoter:		
Standard KBANN-net	6.49%	15.0
Structured	11.83%	16.9
Standard ANN	10.94%	4.3

is the structured network worse at generalizing to promoter testing examples, it is slower to train than standard KBANN-nets on both domains. The differences are statistically significant using a one-tailed, paired sample *t*-test with 99.5% confidence.[16]

Note that, the structured network is also slower to train and worse at generalizing to testing examples than standard ANNs. The difference in generalization between standard ANNs and structured networks is not statistically significant.

Initial weights are responsible for KBANN's strength

This section tests the second hypothesis, that the strength of KBANN results from the weights on links in KBANN-nets identifying relevant features of the environment. An exact test of this hypothesis is probably impossible to construct. A reformulation of this hypothesis that is testable is: if feature identification is the source of KBANN's strength, then an ANN connected to only the features identified by the domain knowledge theory should be better at generalizing than a fully-connected ANN.

Method. To test this hypothesis, standard ANNs were used except that the single layer of hidden units is initially connected to only those input units explicitly mentioned by the domain knowledge (as illustrated in Figure 3.17. In the promoter domain, this creates a network with about $\frac{1}{4}$ of the number of links in a fully-connected network.

To this network, the missing links between the input and hidden layers were added (by random selection) until the hidden and input layers were fully connected. At each 5% step of the missing links were added, the network was copied, and the copy was put aside for testing. The link addition procedure is illustrated in Figure 3.18 using a step size of 33%.

[16]A *t*-test is a standard method for statistics for determining whether the difference between two samples is significant. For instance, consider a class of school children for which a teacher want to determine the effect of a sequence of instruction. To do so, the teacher could test the children both before and after the instruction. Then by applying a *t*-test, the instructor could whether or not the instruction had a significant effect. In this situation, the instructor can make statements beyond merely that the difference was significant. Specifically, using a "one-tailed" *t*-test the instructor can determine the significance of the direction of the difference. Also, the instructor can use a "paired-sample" design for the *t*-test. In a paired-sample test, the changes of each student and compared rather than the changes in the overall class statistics. Hence, a paired sample *t*-test may discover that the change as a result of instruction is significant while a simple *t*-test cannot reach a similar conclusion. Any quality book on statistics can provide more information on these, and other forms, of *t*-tests.

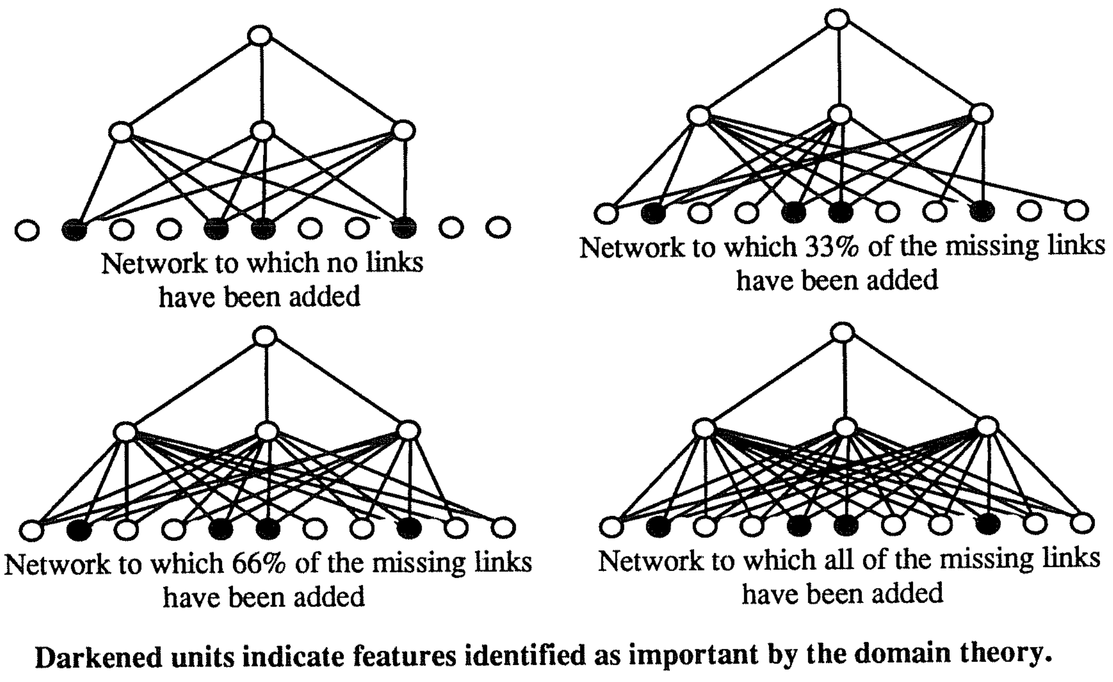

Network to which no links have been added
Network to which 33% of the missing links have been added
Network to which 66% of the missing links have been added
Network to which all of the missing links have been added
Darkened units indicate features identified as important by the domain theory.

Figure 3.18: The creation of networks to test the hypothesis that initial weights are a significant contributor to KBANN.

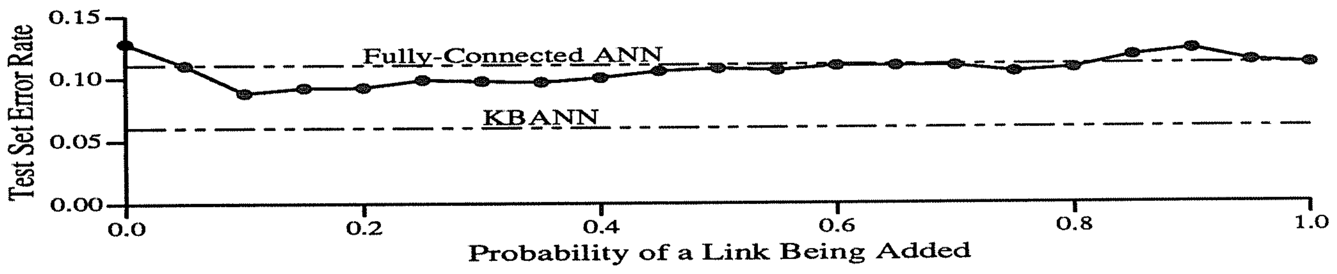


Figure 3.19: The classification performance of standard ANNs that initially have links to only those features specified by the promoter domain theory.

To smooth fluctuations resulting from the addition of particularly important links, this procedure was repeated three times, thereby creating 60 networks. These 60 networks were trained using ten-fold cross-validation and the standard eleven permutations of the training examples.

Results. Figure 3.19 plots the average error rate of these "missing link" networks. Networks which have only links identified by the initial domain knowledge are worse at generalizing to testing examples than both KBANN-nets and standard ANNs. However, by the time 10% of the missing links have been added, the missing-link networks are superior to standard ANNs. Missing-link networks remain superior to standard ANNs until 80% of the missing links have been added.

From 80% to 95% of the links added, the missing-link networks are slightly inferior to fully-connected networks. These differences appear to be anomalies; they are not statistically significant.

The performance of missing-link networks peaks when 10–15% of the missing links have been added. This best-case performance is significantly inferior to that of KBANN (with 99.5% confidence using a one-tailed *t*-test).

These results indicate that some, but not all of the improvement of KBANN over standard backpropagation, results from its identification of important input features.

Both structure and initial weights are required to account for KBANN's strength

The tests in the two prior sections indicate that neither of the above hypotheses is sufficient to account of the superiority of KBANN-nets to standard ANNs. Therefore, the third hypothesis, that it is a combination the structure and the weights that give KBANN its advantage over backpropagation, must be true.

An analysis of the rules-to-network translation algorithm makes it less than surprising that the combination of structure and weighting are necessary for the success of KBANN. The combination of structure and weight both focuses the network on significant input features and provides the set of derived features that are likely important. Structure alone provides only the potential for the derived features and weight alone provides only the significant inputs. Its is the combination of structure and weight that supplies the derived features that give KBANN its power.

3.3.3 Discussion of the sources of KBANN's strength

Tests in this section were designed to determine the source of KBANN's classification ability. The initial tests support the hypothesis that at least some of KBANN's strength, by comparison to other systems for learning from both theory and data, can be traced directly to the underlying empirical learning algorithms.

While this hypothesis is confirmed, its confirmation does not tell the whole story as further tests show that KBANN gains more from its theory/data hybrid than do other hybrid systems. Hence, a subsequent suite of tests investigate the reasons why KBANN is able to improve so greatly upon its underlying algorithm. These tests reveal that KBANN's strength is based on its ability to focus empirical learning on the relevant features *and* the creation of useful derived features; neither alone is sufficient.

3.4 When is KBANN superior to backpropagation?

Tests in this section address two hypotheses about when networks created by KBANN will have an advantage over standard neural networks. These hypotheses are tested by varying aspects of the information provided by the user and observing the effects of these changes on the generalization ability of KBANN-nets.

The first hypothesis is that KBANN-nets are relatively insensitive to noise in domain theories. Hence, the domain theory need only be approximately correct for it to supply useful information. This hypothesis is tested by systematically adding noise to domain theories. Tests of this hypothesis, in addition to verifying the hypothesis, suggest bounds on the correctness of the initial domain theory within which KBANN can be expected to outperform standard ANNs.

The second hypothesis tested is that KBANN-nets are insensitive to irrelevant features. This hypothesis is tested by adding features to networks that initially contain only those features specified by the domain knowledge.

3.4.1 Noise in domain theories

There has been little or no previous work in the addition of noise to domain theories. Hence, the types of noise described here, and the methods used to approximate that noise, represent but a first attempt to identify and test important types of "domain theory noise".

Domain theory noise can be split into two categories: rule-level noise and antecedent-level noise. Each of these categories has several subcategories. The discussion of each type of noise assumes, for the sake of clarity of explanation, a two-category (i.e., positive and negative) domain and that the rules use negation-by-failure.[17]

Rule-level noise affects only whole rules. It appears in either of the following two guises:

- *missing rules* Rules required for the correct operation of the domain theory are missing; the rule set is incomplete. The effect of missing rules is to render some proofs impossible. As a result, a rule set which is missing some rules underpredicts the positive class.

- *added rules* Rules not required for correct operation of the domain theory are present. The effect of added rules is to make possible proofs that should not occur. (Assuming that the extra rules are incorrect, not merely redundant.) Hence, rule sets with extra rules overpredict the positive class.

Antecedent-level noise is independent of whether rule set is missing rules or has unnecessary

[17]Rule sets using negation-by-failure make predictions in a two-category domain by attempting to prove an example is a member of the positive category. If the proof is successful, then the example is in the positive category. Otherwise, the example is assumed to belong to the negative category.

rules. Rather, antecedent-level noise looks at the problems that result from the improper statement of the antecedents of individual rules. This impropriety can occur in any of three forms: extra antecedents, missing antecedents or inverted antecedents.

- *extra antecedents* Extra antecedents are those not required in the correct rule. Their effect is to overconstrain the application of a rule. Hence, extra antecedents result in a rule set underpredicting the positive class.
- *missing antecedents* Missing antecedents are antecedents that should be part of a rule, but are not. The effect of missing antecedents is to underconstrain the application of a rule. As a result, missing antecedents, much like added rules, result in overpredicting the positive class.
- *inverted antecedents* Inverted antecedents should appear in a rule, but in the opposite way. That is, a negated antecedent should actually be unnegated (or vice versa). The effect of an inverted antecedent is to make the rule apply in exactly the wrong situation. This may result in both labeling positive examples as negative or the converse.

Method

Tests for antecedent-level noise start with the initial domain theory then probabilistically add noise to it (the probability is based on the total number of antecedents in the domain theory).[18] As with all other experiments involving increasing quantities, noise is added incrementally. That is, to a domain theory with no noise, the smallest noise fraction is added. Then, the slightly noisy rule set has further noise added to it so that the total noise is equal to the second smallest fraction. This procedure is repeated four times for each of the three kinds of antecedent noise.

Antecedent-level noise is added by scanning each antecedent of each rule. As each antecedent is scanned, the noise-addition procedure probabilistically determines whether or not to add noise. If the decision is to add noise, and the kind of noise to be added is either deletion or inversion, then the appropriate action was taken on the antecedent being scanned (the scanned antecedent is either inverted or dropped). If antecedents are to be added, then an antecedent is randomly selected from among the input features as well as any consequent "below" the consequent of the rule being considered[19]. The selected antecedent is then added to the rule of which the scanned antecedent is a member.

A similar incremental procedure was used to add rule-level noise to the domain theory. Again, noise is added probabilistically (where the probability is based upon the number of

[18] All distributions are uniform.

[19] "Below" is defined using the first labeling procedure defined in Section 2.2.1.

rules in the initial domain theory). Details about the insertion of rule-level noise are given in the next two paragraphs.

Rules are added by creating new rules that are simple conjunctions of randomly-selected input features. The number of antecedents to each added rule is a random number in the range [2...6].[20] Rules so created are added to the domain theory only to existing disjuncts in the domain theory. The reason for this restriction is that such additions require minimal changes to the structure of the rule set. Two other kind of additions are possible but not used. First, rules could be added to the conditions of a conjunct. This sort of addition requires adding antecedent-level noise in addition to rule-level noise. Second, rules could be added that create new ways to prove a consequent that is not disjunctive. This sort of addition requires significant alterations to the hierarchical structure of the rules.

All rules other than the rule leading to final conclusions are subject to deletion. As with antecedent noise, this procedure is repeated four times for each type of noise.

Results and discussion

Figures 3.20 and 3.21 present the results of adding noise to the promoter domain theory. As might have been expected, inverting antecedents has the largest effect of the approaches to inserting antecedent-level noise on the efficacy of KBANN. After the addition of 30% of this type of noise to the domain theory, the resulting KBANN-net generalizes worse than a standard ANN. On the other hand, irrelevant antecedents have little effect on KBANN-nets, with 50% noise (that is for every two antecedents specified as important by the original domain theory, one spurious antecedent was added) the performance of KBANN-nets is still superior to that of a standard ANN. Not appearing in Figure 3.20 are tests in which noise of all three types was added to the domain theory. Not surprisingly, the resulting KBANN-nets perform at about the average of the three type of noise individually.

Results were much the same for rule-level noise. In this case, up to 10% of the initial rules can be deleted while leaving KBANN-nets superior to ANNs. The addition of randomly created rules has little effect on the accuracy of KBANN-nets. The inability of added rules to influence the correctness of KBANN-nets is not surprising. The added rules have an average of four, randomly-chosen, antecedents. Hence, these rules can be expected to match only one in every 256 patterns.[21] As a result, the chances are good that the rules never match the training or testing patterns. Had the introduced rules matched the training patterns, the effect would have

[20]The range [2...6] is used because it contains the number of antecedents in most of the rules in the promoter and splice-junction domain theories.

[21]The number 256 assumes a DNA sequence analysis domain where each feature has four possible values. Hence given four antecedents looking a four different features, the number of possible combinations is 4^4. Also, these odds assume that DNA is randomly assembled from its four-character alphabet. However, DNA is non-random

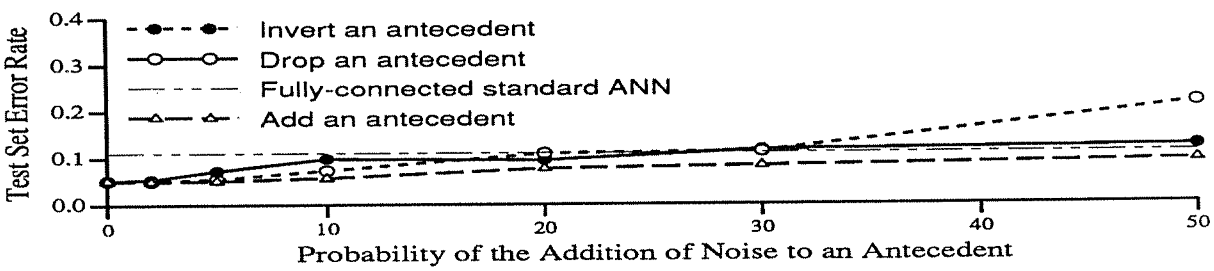


Figure 3.20: The effect of antecedent-level noise on classification performance of KBANN-nets in the promoter domain.

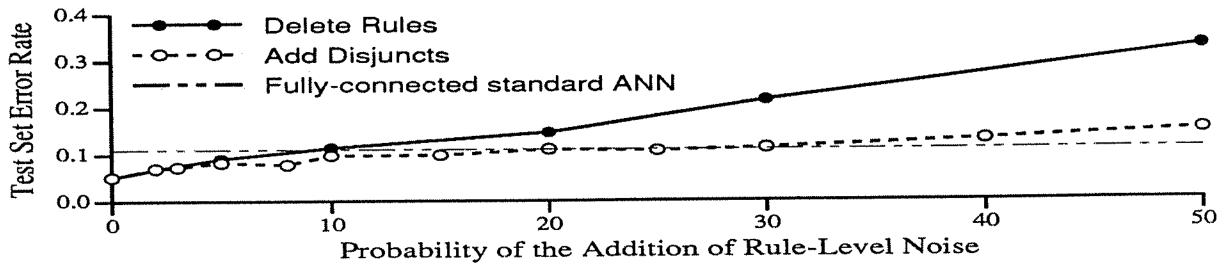


Figure 3.21: The effect of rule-level noise on classification performance of KBANN-nets in the promoter domain.

been similar to adding a single highly-weighted link to the input unit. As Figure 3.20 indicates that KBANN-nets are relatively insensitive to this type of noise, the effect of the irrelevant rules can be expected to be minimal.

These results show the networks created by KBANN are superior to standard ANNs through a wide range of domain theory quality. The theory of promoters could have been up to 30% more incorrect than it was originally and still provide a benefit over standard ANNs.

The moral of these tests is clear (if somewhat overstated): if a rule or antecedent might be useful, include it in the domain theory. It is better to say too much than too little. On the other hand, it is better to say nothing than something that is absolutely false.

3.4.2 Irrelevant features

Another hypothesis of KBANN-nets is they are insensitive to the addition of irrelevant features. By contrast, adding irrelevant features to empirical learning algorithms can negatively affect generalization [Rendell90]. Hence, as the number of irrelevant features in the input set increases, the relative advantage of KBANN-nets should also increase.

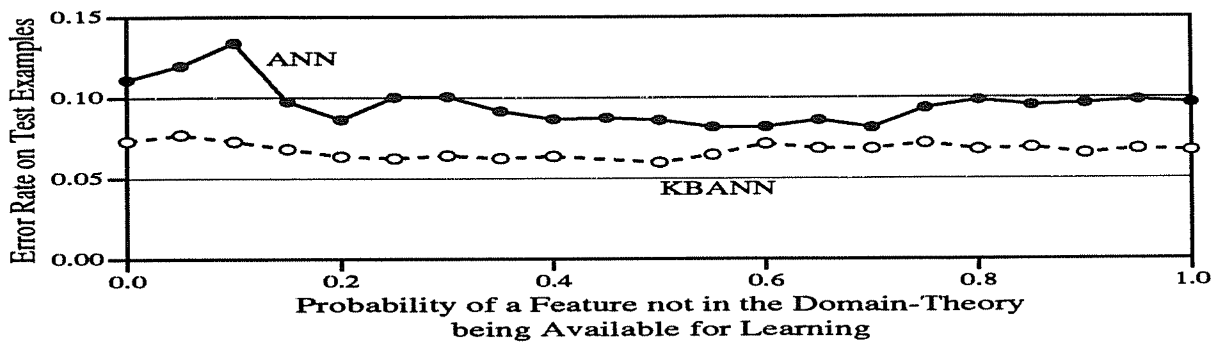


Figure 3.22: The effect of the presence of irrelevant features on classification performance in the promoter domain.

To test this hypothesis, the methodology of the "missing link" study described earlier in this chapter is duplicated with one difference. In the previous test, links were added to networks; in these tests whole features are added. Looking at standard ANNs, this means that when a feature is added, it is connected to every hidden unit. By contrast, in the standard ANNs of Section 3.3.2, input units could be connected to any number of the hidden units. Note that there is a problem with using the promoter domain, or any approximately-correct domain theory, for this study – it is not certain which features are relevant. However, as features are added the network, it is virtually certain that some of those features will be irrelevant. Hence, a network with connections to every input unit is certainly connected to more irrelevant inputs than a network with fewer connections.

Networks to test this hypothesis are created by initially forming a network fully connected to only those features included in the promoter domain theory. To this network, features are added and fully connected, 5% of the unreferenced ones at a time. As with other tests of increasing quantities, features are added so that networks with 20% of the missing features are subsets of networks with 25% of the missing features added. This process is repeated four times for both standard ANNs and KBANN-nets thereby creating 80 networks of each type.

Results and discussion

The results in Figure 3.22 clearly demonstrate that KBANN-nets are less sensitive to the presence of irrelevant features than are ANNs. After an initial decrease in accuracy, KBANN-nets slowly improve with the addition of features so that the most accurate KBANN-nets are connected to about 50% of the input features. The accuracy of these networks is statistically indistinguishable from those with access to every feature. The most accurate ANNs are those connected to approximately 60% of the features not mentioned in the domain theory. The difference in accuracy between ANNs connected to 60% of the missing features and ANNs connected to every feature is statistically significant with 99% confidence.

3.5 Experiments in the Extraction of Rules

This section presents a set of experiments designed to determine the relative strengths and weaknesses of the two rule-extraction methods described in Section 2.4. These rule-extraction techniques are evaluated using two measures: *quality*, which is measured both by the accuracy of the rules and their fidelity to the network from which they were extracted; and *comprehensibility* which is measured both by characterizing whole sets of rules and by looking at individual rules. The final part of this section, which examines the meanings of individual rules, includes samples of the rules extracted from trained KBANN-nets by both SUBSET and NOFM.

Prior to proceeding with experiments that compare SUBSET and NOFM, this section briefly investigates only the SUBSET method. In particular, the tests in this section empirically determine good settings for the parameters that control the size of the space searched by SUBSET.

A hypothesis advanced when SUBSET was introduced is that there is a tradeoff between the number of rules extracted by SUBSET and the accuracy of those rules. Figure 3.23 shows this tradeoff between size and accuracy of the extracted set of rules on the promoter domain. The figure shows that, as expected, the number of extracted rules increases with β_p.[22] Also as expected, the accuracy of the extracted rules increases with the size of the rule set. Significantly, the accuracy of the rules does not steadily approach the accuracy of the network. Instead, the rules improve rapidly until reaching a level about 2% worse than the network. After this point, the rate of improvement in accuracy slows considerably. Thus, increasing β_p beyond 70 results in very little gain in accuracy but a large gain in the number of rules. Although it is not shown, the asymptotic behavior of the extracted rules also occurs on the splice-junction dataset. All subsequent experiments use $\beta_p = 50$ and $\beta_n = 5$, values which are chosen as a good tradeoff between size and accuracy. At this level, SUBSET discovers an average of 300 rules in the promoter domain.[23]

3.5.1 Testing methodology

Testing methodology largely follows that of Section 3.2. Briefly, following Weiss and Kulikowski [Weiss90], networks are trained using repeated 10-fold cross-validation for assessing the quality of learning in both domains. (Note that this is a change in methodology; earlier in this chapter (Section 3.2) promoter recognition was tested using "leaving-one-out".) This change results

[22]Recall that β_p is the beam with for subsets of positive only links. That is, during the process of rule formation, SUBSET begins by searching for rules with only positive antecedents.This search is constrained to look at no more β_p possibilities.

[23]To a certain extent, picking an optimal value for β_p represents cheating. To be perfectly fair, values should be determined without ever seeing the problems on which they will be evaluated. However, fairness is not the primary concern of these tests. Rather, these tests give every possible advantage to the SUBSET algorithm, based upon ideas already in the literature [Saito88]. Hence, these tests attempt to prop up the "straw man" that is the SUBSET algorithm.

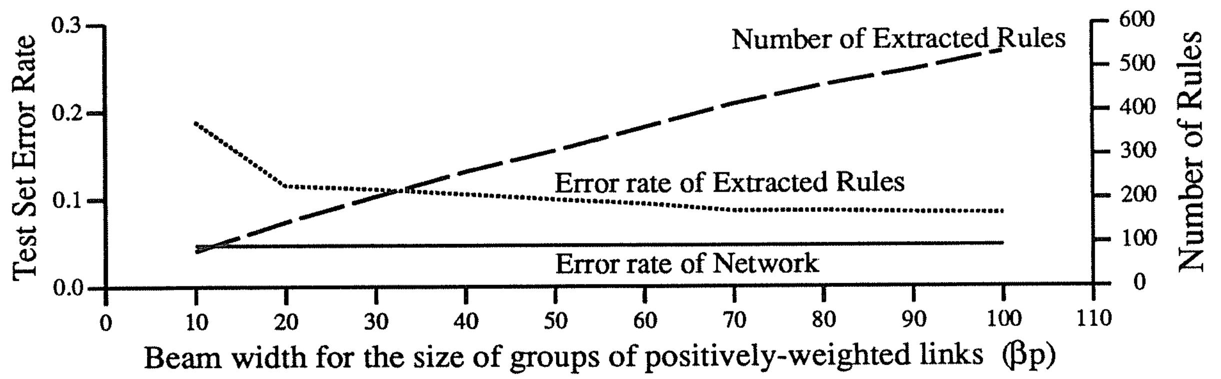


Figure 3.23: The tradeoff between number of rules and accuracy in the promoter domain using the SUBSET algorithm.

from processing time considerations.) Networks are trained until 99% of the training examples are correctly classified or every example has been presented 100 times. Following Hinton's [Hinton89] suggestion for improved network interpretability, all weights are subject to gentle "decay" during training.[24] Finally, as in the previous chapter, networks are trained using the *cross-entropy* error function [Hinton89]. This methodology is used, except when noted, in all experiments described in this section.

3.5.2 Rule quality

The issue of overriding importance to this work is the quality of the extracted rules. As a measure, *quality* is at least two dimensional. First, the rules must accurately categorize examples that were not seen during training. If the rules lose the advantage in accuracy that KBANN-nets provide over most symbolic learning algorithms, then there is little value in rule extraction. It would be simpler to use an "all symbolic" method (e.g., EITHER [Ourston90]) to develop rules, while using KBANN-nets only to develop highly-accurate, but not understandable, classifiers. Second, the extracted rules must capture the information contained in the KBANN-net. This is necessary if the extracted rules are to be useful for determining that a trained KBANN-net is likely to behave correctly in the future. On each of these measures, the results in this section show that the NOFM method is superior to the SUBSET method. More significantly, the NOFM method extracts rules that are superior to those of all-symbolic systems, such as EITHER, and are at least equivalent to the networks from which they came at classifying testing examples.

[24]That is, to the standard weight change function a term ϕ is added ($0 < \phi < 1$). Thus, the weight change formula becomes: $w_{ij}(t) = \phi * (w_{ij}(t-1) + \Delta_{ij})$ where Δ_{ij} is the standard weight adjustment [Rumelhart86]. Weights change after each example presentation, so ϕ is set to a very gentle 0.99999. Weight decay is not used in previously reported experiments simply because it added unnecessary complexity to these experiments. It has little or no measurable effect of the generalization ability of KBANN-nets.

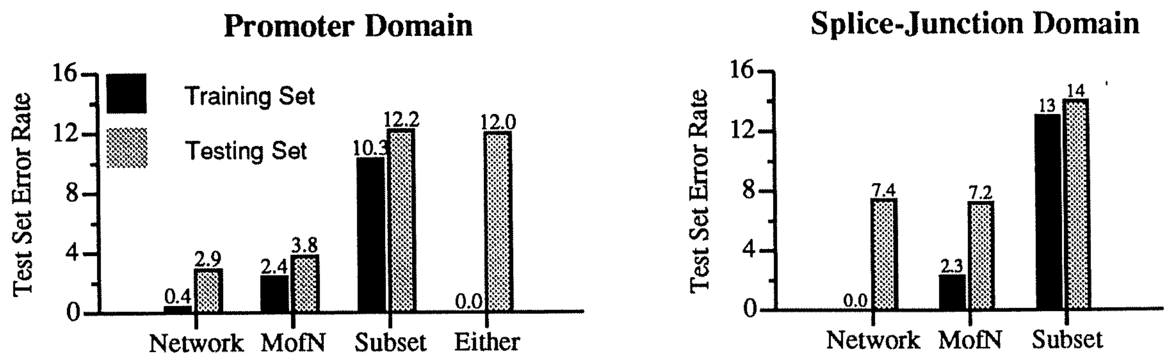


Figure 3.24: Error rates of extracted rules.

Accuracy

Figure 3.24 addresses the issue of the accuracy of extracted rules. It plots percentage of errors on the testing and training sets, averaged over ten repetitions of 10-fold cross-validation, for both the promoter and splice-junction tasks. For comparison, Figure 3.24 includes the accuracy of the trained KBANN-nets prior to rule extraction (the bars labeled "Network"). Also included in Figure 3.24 is the accuracy of the EITHER system, an "all symbolic" method for the empirical adaptation of rules which has been tested using the promoter dataset [Ourston90]. The numbers for EITHER were supplied by the authors of that system [Mooney91a]; they reflect a slightly different testing method. Labyrinth-k, [Thompson91] another system for the symbolic refinement of rules, has also been tested on the promoter problem. Its results are superior to those of EITHER but inferior to KBANN. (Sadly, the differences in testing methodology between Labyrinth-k and the tests in this section are sufficient to preclude comparisons. Neither EITHER nor Labyrinth-k have been tested on the splice-junction problem.)

Recall the initial rule sets for promoter recognition and splice-junction determination correctly categorized 50% and 61%, respectively, of the examples. Hence, each of the systems plotted in Figure 3.24 improves upon the initial rules. Comparing only the systems that produce refined rules, the NOFM method is the clear winner. On the training examples, the error rate for rules extracted by NOFM is slightly worse than EITHER, but superior to the rules extracted using SUBSET. On the testing examples the NOFM rules are much more accurate than both EITHER and SUBSET. (One-tailed, paired-sample t-tests indicate that for both domains the NOFM rules are superior to the SUBSET rules with 99.5% confidence. Statistical comparisons to EITHER are not made due to lack of the appropriate data for EITHER.)

Perhaps the most significant result in this section is that, on the testing examples, the error rate of the NOFM rules is superior to that of the networks from which the rules are extracted on the splice-junction problem. (One-tailed, paired-sample t-tests indicate that the difference is statistically significant with 90% confidence (t=1.6, d.f.=9).) Earlier tests showed the error

Table 3.6: Fidelity of extracted rules to trained KBANN-nets on training examples.

Rule Extraction Method	Overall Percent Agreement	Probability of agreement between KBANN-net and extracted rules given: KBANN-net correct	KBANN-net incorrect
Splice Junction			
SUBSET	86%	0.87	0.38
NOFM	93%	0.95	0.31
Promoter			
SUBSET	87%	0.90	0.24
NOFM	99%	0.99	0.10

rate of the extracted rules on the promoter problem was also below that of the networks from which the rules were extracted. However, alterations to the training method improved the accuracy of the networks without significantly affecting the accuracy of the extracted rules.

Conversely, the error rate of the SUBSET rules on testing examples is statistically worse than the networks in both problem domains. Discussion at the end of this section analyses reasons why NOFM rules are superior to the networks from which they are extracted.

Fidelity

Fidelity is the ability of the extracted rules to mimic the behavior of the network from which they are extracted. Fidelity is important in two circumstances: (1) if the extracted rules are to be used as a tool for understanding the behavior of the network, and (2) if the extracted rules are to be used as a method of transferring the knowledge contained in the network. Table 3.6 indicates that NOFM is superior to SUBSET at reproducing the behavior of the network on the examples seen during training. For instance, on the splice-junction data, rules extracted using NOFM give the same answers as trained KBANN-nets 93% of the time, while rules extracted by SUBSET match only 86% of the time. In addition, Table 3.6 shows that both methods of rule extraction are much better at mimicking the behavior of the network when the network generates the correct answer, than when it generates an incorrect answer. Note that the relatively poor fidelity of the NOFM rules when the networks generate an incorrect answer results from the superior classification performance of the NOFM rules. That is, the NOFM rules often fail to repeat the errors of the networks. The fidelity of the extracted rules to the networks on testing examples (data not shown) is similar.

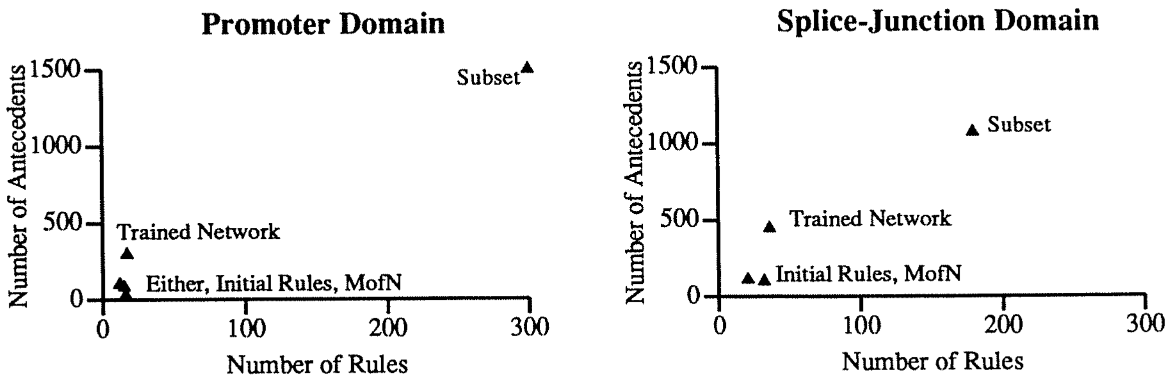


Figure 3.25: Comprehensibility of extracted rules.

3.5.3 Comprehensibility

To be useful, the extracted rules must not only be accurate, they also must be *understandable.* While an ill-defined concept, there are several ways in which "understandability" might be measured. One approach is to look at statistics describing whole sets of rules. An alternative is to look at individual rules and try to determine if they are meaningful. The following sections present results on both of these measures.

Global comprehensibility

This section contains results that characterize the comprehensibility of sets of rules using three measures: *size*, *number of antecedents per rule*, and *consistency.*

Size. Certainly size is of major concern in determining whether or not a rule set is comprehensible, for sets with large number of rules can be difficult, if not impossible, to understand by humans. Figure 3.25 addresses the issue of rule-set size by plotting each rule-extraction method in a space spanned by the number of extracted rules and total number of antecedents in those rules. Counting rules and antecedents for the rules extracted by NofM is less than straightforward, as simplification may increase the number of rules and require the reuse of some antecedents. The data in both Figures 3.25 and 3.26 blithely ignore these complications, reflecting only the rule and antecedent counts of unsimplified rule sets. For trained networks, the count of antecedents includes each link whose weight is within two orders of magnitude of the bias on the unit receiving the link. Weights of lesser size are not counted because they have little effect. Also included in Figure 3.25 are the initial set of rules in each domain and the set of useful rules produced by Either after training on all 106 promoter examples [Mooney91a].

Taking the initial rule sets in each domain as a standard for interpretability, the rules refined by Either are likely to be easy to understand. (Recall, however, that these rules are

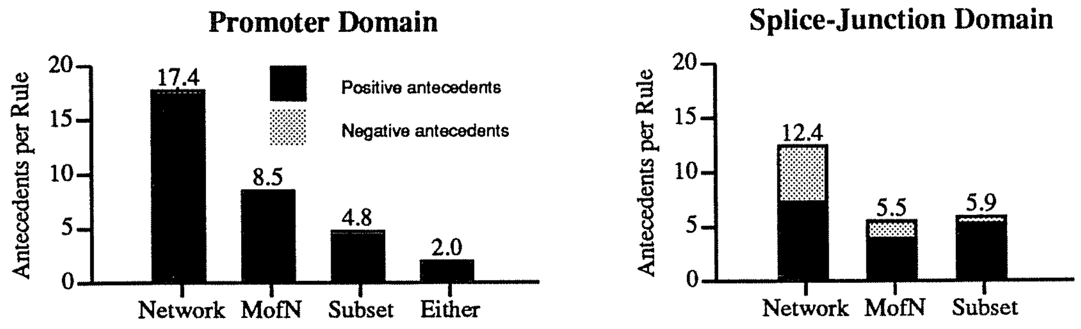


Figure 3.26: Average number of antecedents per rule.

not particularly accurate.) Rules extracted by NOFM are also likely to be easily understood; they have virtually the same number of rules and antecedents as the initial rule sets. On the other hand, the rules SUBSET extracts are much less likely to be comprehensible. (See Tables 3.8 and 3.9 for examples of rules extracted by NOFM and SUBSET, respectively.)

Antecedents per rule. A second important global statistic is the number of antecedents per rule. If this value is large, individual rules are unlikely to be understandable [Brunner56]. Furthermore, negated antecedents add to the difficulty of evaluating rules [Nessier62]. However, the effects of negated antecedents are difficult to quantify. Weighted antecedents, such as those appearing in NOFM rules and which implicitly appear in trained networks, cloud the picture further. Thus, Figure 3.26 takes a very simple approach to judging rules on the basis of antecedents. Each bar consists of a stack that represents the simple sum of the negative and positive antecedents to an average rule.

On this measure, as with rule-set size, EITHER is the clear winner. The NOFM rules are slightly larger than SUBSET rules and contain more negative antecedents. Still, both methods return rules that are well within the limits of human comprehensibility [Miller56] (but recall that NOFM extracts many fewer rules).

Consistency. Finally, the "consistency" of rules is important because if the extracted rules vary considerably between training sessions, it is difficult to ascribe any significance to a particular rule set. There are several ways to determine the consistency of two or more sets of rules. Arguably the best way, and certainly the most thorough, is to test each rule set with every possible input and compare the actions of each set. However, this approach is impossible for it could require testing up to 10^{34} combinations for the promoter problem and about 10^{36} combinations for the splice-junction problem.[25]

[25]Let $\Omega = 15 * 10^9$years (approximately the age of the universe). Then, assuming that you could evaluate one million combinations per second, it would take $2.8 * 10^{12}$ Ω to evaluate all the possible splice-junction combinations.

Table 3.7: Number of classification differences among the rules extracted from three trained networks.

Problem	Number of Examples	NoFM	Subset
Promoter	106	1	3
Splice-Junction	1000	76	193

Because of these combinatoric problems, consistency is measured by training three networks, differing only by slight randomness in initial link weights, with all of the examples. Hence, the only differences between trained networks are (a) the initial starting configurations of the networks (which vary only by small random perturbations to the network) and (b) the order in which examples are presented. Rules extracted from these networks are tested against the examples, and the number of examples for which the three sets of extracted rules do not agree is counted. This measure is blind to whether or not the rules are correct, it is concerned only with whether the rules make the same decision. The results of this test appear in Table 3.7. On both domains, NoFM extracts rules that are considerably more consistent than Subset, averaging about one inconsistency for every three made by the Subset rules.

A slightly different approach to determining consistency is to look at the errors made during repeated 10-fold cross-validation runs. Specifically, if the rule-extraction method is consistently pulling out the same information, then the extracted rules should make the same number of errors on each cross-validation run. Hence, a consistent method should have a low standard deviation in the number of errors. This is exactly what is observed for the NoFM method; its standard deviation is lower than both Subset and trained Kbann-nets.

Individual comprehensibility

Global statistics are sufficient to indicate whether or not a whole rule set is likely to be comprehensible. Once it has been determined that a rule set is potentially comprehensible, it is necessary to look at individual rules and assess their meaning. To make this assessment, rule sets extracted by the NoFM and Subset are critically examined.

Tables 3.8 and 3.9 present the rules NoFM and Subset extracted from a Kbann-net trained for promoter recognition. Because rule sets extracted by Subset can be very large, β_p and β_n (beam widths used during the search of rule space) are respectively set to five and one. These values are too small to deliver reasonably accurate rule sets; the extracted rules have a 25% higher error rate than the networks from which they are extracted.

While the rules extracted by NoFM in Table 3.8 are somewhat murky, they are vastly more comprehensible than the network of 3000 links from which then came. Moreover, the rules in this table can be rewritten in a form very similar to one used in the biological community

Table 3.8: Promoter rules NofM extracts.

```
Promoter :- Minus35, Minus10.

Minus-35
 :-10 < 4.0 * nt(@-37 '--TTGAT-') +
        1.5 * nt(@-37 '----TCC-') +
        0.5 * nt(@-37 '---MC---') -
        1.5 * nt(@-37 '--GGAGG-').
Minus-35
 :-10 < 5.0 * nt(@-37 '--T-G--A') +
        3.1 * nt(@-37 '---GT---') +
        1.9 * nt(@-37 '----C-CT') +
        1.5 * nt(@-37 '---C--A-') -
        1.5 * nt(@-37 '------GC') -
        1.9 * nt(@-37 '--CAW---') -
        3.1 * nt(@-37 '--A----C').
Minus-35 :- @-37 '-C-TGAC-'.
Minus-35 :- @-37 '--TTD-CA'.

Minus-10 :- 2 of @-14 '---CA---T' and
        not 1 of @-14 '---RB---S'.
Minus-10
 :-10 < 3.0 * nt(@-14 '--TAT--T-') +
        1.8 * nt(@-14 '-----GA--') +
        0.7 * nt(@-14 '----GAT--') -
        0.7 * nt(@-14 '--GKCCCS-').
Minus-10
 :-10 < 3.8 * nt(@-14 '--TA-A-T-') +
        3.0 * nt(@-14 '--G--C---') +
        1.0 * nt(@-14 '---T---A-') -
        1.0 * nt(@-14 '--CS-G-S-') -
        3.0 * nt(@-14 '--A--T---').
Minus-10 :-  @-14 '-TAWA-T--'.
```

See Table 3.1 for meanings of letters other than A, G, T, and C.

"nt()" returns the number of enclosed in the parentheses antecedents that match the given sequence. So, nt(@-14 '- - - C - - G - -') would return 1 when matched against the sequence @-14 'AAACAAAAA'.

[Stormo90], namely weight matrices.

One major pattern is apparent in the rules extracted by both NofM and Subset. Specifically, the network learned to disregard `conformation`. The `conformation` rules are also dropped by Either [Mooney91a], which suggests that dropping these rules is not an artifact of Kbann but rather that DNA bases outside the *minus35* and *minus10* regions are less important than the conformation hypothesis [Koudelka87] suggests. Hence, machine learning methods can provide valuable evidence confirming or refuting biological theories.

In general, the rules NofM extracts confirm the importance of the nucleotides identified in the initial rules. However, whereas the initial rules required matching every base, the extracted rules allow a less than perfect match. In addition, the extracted rules point to places in which changes to the sequence are important. For instance, in the final *minus10* rule, a `'T'` in position 8 is a strong indicator that the rule is true. However, replacing the `'T'` with a `'G'` prevents the rule from ever being satisfied.

It is more difficult to get a clear picture of a promoter from the Subset rules (Table 3.9). The first three *minus35* encode a simple 2-of-3 concept while the *minus10* rules to approximate a 3-of-7 concept. Note that these patterns support the idea implemented in the NofM method that a bias towards N-of-M style concepts is useful. Other than these patterns, it is difficult

Table 3.9: Promoter rules extracted by SUBSET.

```
Promoter :- Minus35, Minus10.

Minus35   :- Minus35b, Minus35d.            Minus10  :- @-14 ‘-ATA----’.
Minus35   :- Minus35a, Minus35b.            Minus10  :- @-14 ‘-T--A-A-’.
Minus35   :- Minus35a, Minus35d.            Minus10  :- @-14 ‘--A-T-T-’.
                                            Minus10  :- @-14 ‘-TA-----’.
Minus35   :- @-37 ‘-T-T--A’.                Minus10  :- @-14 ‘-T------’.
Minus35   :- @-37 ‘-TT---A’.                Minus10  :- @-14 ‘---A---T’.
Minus35   :- @-37 ‘---G-CA’.                Minus10  :- @-14 ‘--T----T’.
Minus35   :- @-37 ‘--T--C-’.                Minus10  :- @-14 ‘--TA----’.
Minus35   :- @-37 ‘-T---CA’.                Minus10  :- @-14 ‘T-T-A---’.
                                            Minus10  :- @-14 ‘-AT--T--’.
Minus35a :- @-37 ‘CT--A--’.                 Minus10  :- @-14 ‘--TAA---’.
Minus35a :- @-37 ‘C--G-C-’.                 Minus10  :- @-14 ‘--TA-T--’.
Minus35a :- @-37 ‘C-T--C-’.
Minus35a :- @-37 ‘CT---C-’.
Minus35a :- @-37 ‘C---AC-’.
Minus35b :- @-37 ‘---GA--’.
Minus35b :- @-37 ‘--T--CA’.
Minus35b :- @-37 ‘-T---C-’.
Minus35b :- @-37 ‘---G-CA’.
Minus35b :- @-37 ‘----ACA’.
Minus35d :- @-37 ‘-TT--C-’.
Minus35d :- @-37 ‘-T-G-C-’.
Minus35d :- @-37 ‘--T-AC-’.
Minus35d :- @-37 ‘---GAC-’.
Minus35d :- @-37 ‘-T--AC-’
```

This abbreviated set of rules extracted by SUBSET has a test set accuracy of 75%. Sets whose statistics are reported previously in this section contain about 300 rules.

to say anything about the SUBSET rules.

The rules extracted by NOFM and SUBSET for the splice-junction domain appear in Tables 3.10 and 3.11, respectively. As in the promoter domain, several combinations of the SUBSET rules appear to approximate N-of-M style rules. For instance, the rules for the output unit `E/I` are approximated by:

```
E/I :- (4 of ‘AG-TRAG’), not(E/I-stop).
```

The `E/I` rule of NOFM captures this restated rule, but with some added possibilities and conditions on its satisfaction. Similarly, the SUBSET rules appear to encode an N-of-M style concept for `I/E`. In this case four rules SUBSET extracts could be rewritten:

```
I/E :- (3 of @-06 ‘Y--YAG’), pyramidine-rich, not(I/E-stop8).
```

Again, the rule NOFM extracts explicitly encodes this rule (along with some additional conditions).

Comparing the rules extracted by NOFM (in Table 3.10) to the initial rules (Table 3.3), there are three patterns to what KBANN-net learn. First, not all of the features mentioned in the initial rules are important. For instance, the initial `I/E` rule asserted the importance of `@1 'G'`. The extracted rules never mention this nucleotide. Likewise, the extracted rules for `E/I` do not mention `@-3 'M'`, a value whose importance the initial rules assert.

Second, some positions contain more information than others. For instance, the `pyramidine-rich` rule learns that a pyramidine at position -12 is more important than a pyramidine anywhere else in the region -14 through -6. Similarly, the extracted rules show that trained KBANN-nets learn to place almost twice as much value upon `@3 'T'` as any other nucleotide for the determination of `E/I`.

Third, the network learned to specialize its highly disjunctive rules for the recognition of "stop codons". Both the `I/E` and `E/I` rules initial have nine ways of recognizing stop codons. In both cases, the network learns to prefer stop codons that are immediately adjacent to the splice site. The `E/I` consequent also shows a marked preference for stop codons with the sequence `'TAG'`. The `I/E` rules also show a preference, but it is for the sequence `'TGA'`.

3.5.4 Discussion of rule extraction

The results presented in this section indicate that, upon any of the measures investigated, the NOFM method is able to extract a good set of rules from trained networks. Specifically, with the NOFM method, not only can comprehensible, symbolic rules be extracted from trained KBANN-nets, the extracted rules can be superior to the KBANN-net from which they are extracted at classifying testing examples. Additionally, the NOFM method produces refined rule sets whose accuracy is substantially better than EITHER, an approach that directly modifies the initial set of rules [Ourston90, see Figure 3.24]. While the rule set produced by the NOFM algorithm is slightly larger than that produced by the "all symbolic" approach of EITHER, the sets of rules produced by both of these algorithms is small enough to be easily understood. Hence, although weighing the tradeoff between accuracy and understandability is problem and user-specific, the NOFM algorithm for network-to-rules translation offers an appealing mixture.

Two hypotheses explain the superiority of NOFM over rule-refinement method that directly modify rules. First, the re-representation of a rule set as a neural network allows for more fine-grained refinement. When cast as a neural network, rules can be modified in very small steps. Hence, it may be possible to more closely fit a target concept than when taking the large steps required by direct rule-refinement. Second, DNA sequence-analysis problems have aspects that can be perspicuously captured by N-of-M style rules. For instance, in the promoter problem, there are several potential sites at which hydrogen bonds can form between DNA and a protein; if enough of these bonds form, promoter activity can occur. As neither EITHER

Table 3.10: The splice-junction rules NOFM extracts

```
I/E :- 10 < 2.8 * nt(@-06 'Y--YAG', pyramidine-rich) +
             1.5 * nt(@-10 'Y', I/E-stop1, E/I-stop7) +
            -1.5 * nt(I/E-stop3, E/I-stop5) +
            -2.8 * nt(@-12 'Y', I/E-stop4, E/I-stop4, E/I-stop6, E/I-stop) +
            -4.7 * nt(E/I-stop3) +
            -7.5 * nt(I/E-stop8).

pyramidine-rich :- @-14 '--YY----', 1 <= nt(@-14 'YY---Y-Y').
pyramidine-rich :- @-14 'YY-Y-Y-Y'.
pyramidine-rich :- @-14 '--Y-----', 2 <= nt(@-14 'YY---Y--').

I/E-stop  :- 1 <= nt(I/E-stop1, I/E-stop2, I/E-stop3, I/E-stop4,
                     I/E-stop5, I/E-stop6, I/E-stop9).
I/E-stop1 :- @1 'TAA'.                I/E-stop4 :- @2 'TAA'.
I/E-stop2 :- @1 'TAG'.                I/E-stop5 :- @2 'TAG'.
I/E-stop3 :- @1 'TGA'.                I/E-stop6 :- @2 'TGA'.
                     I/E-stop9 :- @3 'TGA'.

E/I  :- 10.0 < 4.2 * nt(@-02 '---T----') +
               2.2 * nt(@-02 '-G--R-G-') +
               1.2 * nt(@-02 'AA---M-T') +
               0.6 * nt(@-02 'G--A---C') +
              -0.6 * nt(@-02 '-C---T--') +
              -1.2 * nt(@-02 '---S----') +
              -2.2 * nt(E/I-stop).

E/I-stop :- 1 <= nt(E/I-stop1, E/I-stop2, E/I-stop5, E/I-stop8).
E/I-stop1 :- @-3 'TAA'.
E/I-stop2 :- @-3 'TAG'.              E/I-stop5 :- @-4 'TAG'.
                     E/I-stop8 :- @-5 'TAG'.

For i from ((-30 to -1) and (+1 to +30))
           {@<i> 'Y' :- @<i> 'C'.  @<i> 'Y' :- @<i> 'T'.}
```

"nt()" returns the number of named antecedents that match the given sequence. So, nt(@-14 '- - - C - - G - -') would return 1 when matched against the sequence @-14 'AAACAAAAA'.

nor Labyrinth (nor the SUBSET method) can easily express N-of-M rules, these algorithms may be at a disadvantage on the molecular biology test domains.

The superiority of the NOFM rules over the networks from which they are extracted occurs because the rule-extraction process reduces overfitting of the training examples. The principle evidence in support of this hypothesis is that the difference in ability to correctly categorize testing and training examples is smaller for NOFM rules than for trained KBANN-nets. In other words, the rules that the NOFM method extracts are only slightly better at classifying

Table 3.11: The splice-junction rules SUBSET extracts

```
E/I :- @-02 'A--TR-G', not(E/I-stop).
E/I :- @-02 '---TRAG', not(E/I-stop).
E/I :- @-02 'AG-TR--', not(E/I-stop).
E/I :- @-02 '-G-TRA-', not(E/I-stop).
E/I :- @-02 '-G-TR-G', not(E/I-stop).

E/I-stop  :- E/Istop9, E/Istop6, not(E/Istop7).
E/I-stop  :- E/Istop5, not(E/I-stopA).
E/I-stop  :- E/Istop2, not(E/Istop7).
E/I-stop  :- E/Istop1, not(E/Istop7).
E/I-stop  :- E/Istop8, not(E/Istop7).
E/I-stopA :- E/Istop7, E/Istop4, E/Istop3.
E/Istop1 :- @-3 'TAA'.    E/Istop4 :- @-4 'TAA'.    E/Istop7 :- @-5 'TAA'.
E/Istop2 :- @-3 'TAG'.    E/Istop5 :- @-4 'TAG'.    E/Istop8 :- @-5 'TAG'.
E/Istop3 :- @-3 'TGA'.    E/Istop6 :- @-4 'TGA'.    E/Istop9 :- @-5 'TGA'.

I/E  :- @-06 'Y--YA-', pyramidine-rich, not(I/E-stop8).
I/E  :- @-06 'Y--YAG', not(I/E-stop8).
I/E  :- @-06 'Y---AG', pyramidine-rich, not(I/E-stop8).
I/E  :- @-06 'Y--Y-G', pyramidine-rich, not(I/E-stop8).
I/E  :- @-06 '---YAG', pyramidine-rich, not(I/E-stop8).

pyramidine-rich  :- @-14 '-YY--Y-Y'.
pyramidine-rich  :- @-14 'YYY--Y--'.
pyramidine-rich  :- @-14 'Y-YY----', not(pyramidine-rich0).
pyramidine-rich  :- @-14 '--YY-Y--'.
pyramidine-rich  :- @-14 '-YYY----'.
pyramidine-rich0 :- @-14 '----Y-Y-'.

I/Estop8 :- @3 'TAG'.

For i from ((-30 to -1) and (+1 to +30))
          {@<i> 'Y' :- @<i> 'C'.  @<i> 'Y' :- @<i> 'T'.}
```

This abbreviated set of rules extracted by SUBSET has a test set accuracy of 75%. Sets whose statistics are reported elsewhere in this section contain about 300 rules with a test set accuracy of 92%.

training examples than at classifying testing examples. (While the differences between training and testing set performance are smaller, they are statistically significant with 99.5% confidence using a one-tailed, paired-sample, *t*-test.) The rules SUBSET extracts also have this property, but they are worse on the training set than both NOFM rules and trained KBANN-nets on the testing set.

Another piece of evidence in support of the overfitting hypothesis comes from the work in pruning of neural networks to eliminate superfluous parts [Mozer88, Le Cun89, Weigand90]. Rule extraction is, in some sense, an extreme form of pruning in which links and units are pruned and actions are taken on the remaining network to transform the network into a set of rules. Pruning efforts have also led to gains in test set performance. Generally the reseachers attribute these gains to reduced overfitting of the training examples.

3.6 General Discussion of KBANN's Effectiveness

Results in this section suggest a fairly concise definition of the abilities of KBANN. Namely, KBANN is more effective at generalizing to examples not seen during training than either empirical learners or systems that learn from both theory and data. This superiority has at least two causes. First, KBANN is built upon a more effective empirical learning system than other systems that learn from both theory and data. Second, KBANN gets more out of the combination of theory and data than other systems. It was further shown that the strength of the hybrid in KBANN is due to both the identification of relevant features in the environment and the reaction of derived features that simplify the learning problem.

A separate set of tests indicate that reasonably-accurate domain theories are sufficient to allow KBANN to create a network that is quite effective at generalizing to examples not seen during training. Specifically, the results show that KBANN is most effective at learning from domain theories that have too many rules with too many antecedents (i.e., domain theories that are overly specific). At the very least, this result is interesting for its serendipity as both the promoter and splice-junction domain theories are overly specific. However, both domain theories were extracted directly from the biological literature by a person who was unfamiliar with this property of KBANN.[26] Hence, some of the effectiveness of KBANN by comparison to EITHER and Labyrinth may be due to the promoter domain theory better fitting the biases of KBANN than those systems.

The final tests of this section evaluate NOFM and SUBSET, two methods for implementing the network-to-rules translator of KBANN. The result show the NOFM method to be superior to SUBSET on every measure. Moreover, the rules extracted by NOFM retain the accuracy of the networks from which they came. Hence, these rules are more accurate than methods that directly refine rules.

Almost all of the tests in this chapter should be repeated with an artificial domain in which a classification theory and the relevant features are known with certainty. An artificial problem would allow very tightly controlled experiments that are not possible in the real-world problems studied in this chapter. For instance, an artificial problem would allow an analysis of rule extraction methods to look beyond classification accuracy, asking instead how well the

[26]At the time the theories were extracted, I too was unaware of this property.

extracted rules match the true theory. Similarly, tests of sensitivity to irrelevant features would benefit from the absolute knowledge about which features are important.

Yet, even without the tests made possible by an artificial problem, the results in this chapter show KBANN to be a robust learning algorithm. (The ideas underlying KBANN have also been successfully applied to other domains including protein secondary structure prediction [Maclin91] and the refinement of "PID" controllers [Scott91].) In summary, KBANN provides an accurate, understandable method for classification, and the explanation of classification decisions, under a wide range of conditions.

Chapter 4

PSYCHOLOGICAL MODELING USING KBANN

This chapter further demonstrates the utility of KBANN by showing its effectiveness on a different type of problem. Rather than trying to determine a good classification method as in the promoter and splice-junction problems, this chapter investigates the use of KBANN as a model of human learning. In particular, the knowledge given to KBANN captures the reasoning of children prior to formal instruction in geometry. Networks are then trained using two different sets of material reflecting two different pedagogical approaches. The responses of the trained networks are shown to closely match the responses of children after similar instruction. Hence, this chapter shows that KBANN has utility as a pedagogical testbed.[1]

Prior to describing the KBANN-based model, the first section of this chapter briefly describes the van Hiele model of geometry learning [van Hiele-Geldof57] – the most widely used model of the development of geometric reasoning. This description points out flaws in the van Hiele model which led to the development of a model based upon KBANN. Subsequent sections present details of the KBANN-based model and results of an initial test of the model.

4.1 The Van Hiele Model

Recent research about the development of skilled performance in geometry suggests that knowledge of fundamental spatial schemes is essential to the efficient construction of geometric proofs [Koedinger90]. This view suggests the need for spatial spadework prior to deduction, in which students have opportunities to develop adequate descriptions of spatial properties and their relationships. An understanding of the development process can serve to guide instruction,

[1]This section results from a collaboration with Professor Richard Lehrer of the Education Psychology Department at the University of Wisconsin. In particular, he suggested the problem, aided in the development of the initial model, and tested the children.

thereby maximizing the effectiveness of instruction during this period.

One model of the development of spatial properties and their relationships was suggested over 30 years ago by D. van Hiele [van Hiele-Geldof57] and subsequently elaborated by a variety of researchers [Fuys88, van Hiele86]. Briefly stated, the van Hiele model proposes that a series of levels of understanding evolve as children engage in geometry. At the first level (level 0 in most accounts), children develop constructions about space that are closely tied to their informal knowledge. For example, they may sort shapes on the basis of contour. As a result, their thought process is often characterized as "visual." Thus, squares that are similar in appearance to prototypical squares in the child's experience are identified as squares, but squares that depart significantly from the prototype's size or orientation are not classified appropriately.

At the second level (level 1) of the van Hiele model, thinking about space becomes more symbolic. Just as children's use of numeric symbols extends the range of their thought in addition and subtraction problems, so too does the use of symbolic properties extend their ability to reason about objects in space. Thus, "it looks like" is replaced with "it has all right angles", and the like. Practically, this means that children can describe commonalities and differences among objects that may not "look" alike. For example, squares may be characterized as figures with four congruent sides and right angles. Recognition of these characteristics leads to an increase in classification accuracy even for squares that are markedly different from prototypical examples.

At the third level (level 2), children construct relationships among symbolic descriptions (properties). For example, they recognize that if the opposite sides of a quadrilateral are parallel, then the opposite sides must be congruent. Similarly, children develop ideas about relationships among figures; a square is a rhombus because it has the properties of a rhombus. The model from here progresses to describe two higher levels of axiomatic reasoning and deduction.

Although the van Hiele model has generated much research and continues to attract many adherents, there is good reason to question its adequacy as an explanation of the transitions observed in children's reasoning about geometry.

First, the van Hiele model describes general benchmarks of thought. Although this type of benchmark can be useful to teachers in their design of classroom instruction and for assessment, it suffers from too much generality. To say that a student's understanding is limited to a "visual" analysis tells little about how such an analysis is conducted.

Second, the van Hiele model fails to specify any mechanisms for transition between levels (stages). For instance, there is not any principled account of why some types of contrasts among examples may be more conducive to learning than others.

Third, there is little empirical evidence to substantiate the demarcations of children's think-

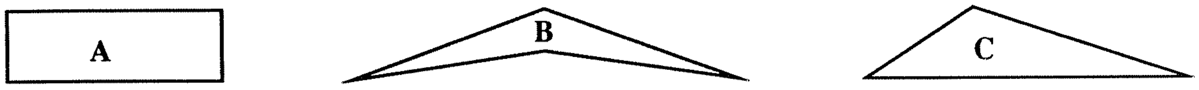


Figure 4.1: Sample triad.

ing into discrete and nonoverlapping levels, even by adherents of the theory.

Lastly, a series of studies with second, fourth, and fifth grade children suggest that children's descriptions of space use many of the same constructs used by experts, but that they also include features without diagnostic significance. These studies, conducted with triads of geometric shapes such as those in Figure 4.1, questioned children about similarities and differences among these shapes [Lehrer89]. For example, the majority of children at each of these grades report that shapes B and C are most similar, either because "they are both pointy", or because "you can move this line down here [the lower segments of B] and then they [B and C] would look alike." When questioned further, many children will assert that the shape C is a triangle and shape B "isn't really a triangle." Nevertheless, children go on to reassert their similarity. Thus, shape names do not imply a necessary set of properties for children the way they do for adults.

Hence, instead of following the ideas of van Hiele, a model of the development in children of spatial features and their relationships should have the properties listed in Table 4.1. This chapter describes a model based upon KBANN and provides an initial suggestion that this model has these properties. Subsequent sections describe a test of this model and the implications of this test on instruction. The section concludes with a description of current research issues for the model.

4.2 Pre-Instructional Geometry in Children

In the KBANN-based model, geometric reasoning develops by learning to identify the most similar pair of shapes in triads akin to Figure 4.1. This section describes the base state of the model; it presents both the initial theory provided to KBANN and details about the model's implementation.

4.2.1 Level Ø geometric theory

The model is initialized with a set of rules that roughly correspond to the geometric knowledge of children at level 0 on the von Hiele scale. This set of rules, designated LØ, implements a *feature counting* strategy in which similarity is judged according to the number of visual features shared by a pair of shapes. Seven visual features are counted by LØ: *tilted* (rotation from a standard orientation) *slanty-lines* (presence of lines other than vertical and horizontal), *physical description* (e.g., skinny, fat), *pointy* (the figure appears to point in some direction),

Table 4.1: Necessary properties of a useful model of the learning of geometry.

1. *Show "mixture" between von Hiele levels.* As discussed above, when children explain their decisions, they often cite factors that are associated with more than one of the van Hiele levels.
2. *Have a mechanism for skill development via incremental learning.* That is, the model should be dynamic. It should be able to learn from the same sort of instruction given to children.
3. *Allow for a combination of "visual" and ordinary features.* Children (and adults) commonly mix features like "pointy" with more sophisticated features like parallel lines. A successful model must also be able to do so.
4. *Treat shape names initially as features and later have them acquire diagnostic significance.* Children are often able to label shapes correctly (e.g., calling any three-sided figure a triangle) before they use those labels to make decisions. Hence, a model should be able to reproduce the transititon in reasoning from the mere labeling of shapes to the use of, and reliance on, those labels.
5. *Combine explanation (rules) and pattern recognition/identification.* Children often combine sophisticated rule-based reasoning with simple pattern identification. For instance, a child might reason that a square and a diamond are really the same, but still assert that a diamond and a triangle are similar because they are both "pointy".

point direction, *area*, and *two long and two short sides* (typical of rectangles).

A subset of LØ appears in Table 4.2A.[2] The first rule in this table, rule 4.2.1 (i.e., rule 1 in Table 4.2), says that if the area of a shape ?OBJ1[3] has some value (e.g., large) and the area of another shape ?OBJ2 has the same value, then the two shapes have the same area. Rule 4.2.4 is an example of a feature-counting rule.[4] Using the rules in Table 4.2A, the most that can be said about shapes B and C of Figure 4.1 is that they are *very-similar* since they are the same in terms of *pointiness*, *point-direction*, and *area*. On the other hand, shapes A and C are at most are *less-similar* as they only have the same *area*.

In addition to the rules characterized in Table 4.2A, LØ includes shape-naming rules (Table 4.2B). These rule are included because children at van Hiele level 0 are able to name objects as triangles, squares, etc. These rules match shapes against one or more prototypical instances for a shape name, rather than against symbolic descriptions of the shape names. Thus, a rule

[2]Tables E.1 – E.5 in Appendix E contain a complete listing of LØ.

[3]The notation ?X denotes a variable.

[4]Feature counting rules are actually expressed in the form "A is true if N out of these M antecedents are true."

Table 4.2: Sample rules in LØ.

(1)	same-area(?OB1, ?OB2)	:- area(?OB1, ?AREA),	area(?OB2, ?AREA).
(2)	same-pdir(?OB1, ?OB2)	:- point-direction(?OB1, ?PD),	point-direction(?OB2, ?PD).
(3)	same-pointy(?OB1, ?OB2)	:- pointy(?OB1, ?PTY),	pointy(?OB2, ?PTY).
(4)	very-similar(?OB1, ?OB2)	:- same-pdir(?OB1, ?OB2),	same-pointy(?OB1, ?OB2),
		same-area(?OB1, ?OB2).	
(5)	similar(?OB1, ?OB2)	:- same-pdir(?OB1, ?OB2),	same-pointy(?OB1, ?OB2).
(6)	similar(?OB1, ?OB2)	:- same-pdir(?OB1, ?OB2),	same-area(?OB1, ?OB2).
(7)	similar(?OB1, ?OB2)	:- same-pointy(?OB1, ?OB2),	same-area(?OB1, ?OB2).
(8)	less-similar(?OB1, ?OB2)	:- same-pdir(?OB1, ?OB2).	
(9)	less-similar(?OB1, ?OB2)	:- same-area(?OB1, ?OB2).	
(10)	less-similar(?OB1, ?OB2)	:- same-pointy(?OB1, ?OB2).	

A: A small, feature-counting theory of geometry.

(11) triangle(?OB) :- area(?OB, medium), pointiness(?OB, very), point-direction(?OB, up).
(12) triangle(?OB) :- number-of-sides(?OB1, 3).

B: Rules to recognize triangles.

(13)	most-similar(?OB1, ?OB2)	:- very-similar(?OB1, ?OB2),	not very-similar(?OB1, ?OB3),
			not very-similar(?OB2, ?OB3).
(14)	most-similar(?OB1, ?OB2)	:- similar(?OB1, ?OB2),	not similar(?OB1, ?OB3),
			not similar(?OB2, ?OB3).
(15)	most-similar(?OB1, ?OB2)	:- less-similar(?OB1, ?OB2),	not less-similar(?OB1, ?OB3),
			not less-similar(?OB2, ?OB3).

C: Rules for combining similarity judgments.

to recognize a triangle would look more like rule 4.2.11 than the more commonly accepted rule 4.2.12. As suggested by property (3) of Table 4.1, the shape naming rules participate in similarity judgments in LØ in the same manner as the visual features. So, the recognition that shapes A and B in Figure 4.1 are both have four sides would count no more than shapes B and C having the same area.

To make explicit the relation between pairs of shapes and between levels of similarity as determined by the feature counting rules, LØ must also have rules similar to those in Table 4.2C. Using the rules in Table 4.2 it would be possible to conclude that, for Figure 4.1, the pair BC is the *most similar* as BC is *very similar* while neither AB nor AC is *very similar.*

4.2.2 Details of the implementation

The rules presented in Table 4.2 are expressed using variables for the sake of brevity. However, KBANN currently cannot handle variables.[5] This has two effects. First, rules must be explicitly repeated for each pair of shapes. Hence, LØ actually contains about 250 rules. Second, it is

[5]See Appendix A for a description of a way in which KBANN handles variables.

possible for a KBANN-net based upon the LØ rules to learn to treat each pair of shapes differently. That is, a KBANN-net could learn a different rule for determining the similarity of the BC pair than is used for the AC pair. To prevent this, KBANN-nets are given explicit instructions that all pairs are to be looked at in the same way. Within a KBANN-net, this forces corresponding links to have equal weights. This is a standard technique in the application of neural networks to problems such as object recognition [Honavar88].

In addition to the instruction to treat all pairs equally, the model is instructed to only make changes to the KBANN-net in places that correspond to rules that consider the shapes. That is, in Table 4.2 the KBANN-net can change neither the feature-counting (part A) nor combining rules (part C). These two instructions constrain the problem and make the learning problem for the model very similar to the problem faced by school children.

In addition to the seven visual features, the model uses 13 symbolic features from proof geometry such as *number of sides*, *number of right angles*. The 20 features have an average of 4.2 possible values. (See Table E.6 in Appendix E for a listing of all features and values.)

4.3 An Application of the KBANN-based model

To test this model, it was trained using two different sets of materials. This experiment highlights the differences between the KBANN-based model and the van Hiele model. Specifically, the van Hiele model provides no basis for distinguishing the effectiveness of two instructional sequences. By contrast, the KBANN-based model can test the effectiveness of multiple instructional sequences through the independent presentation and testing of each sequence.

For this experiment, two sets of training shapes were developed by R. Lehrer. From these sets, triads were selected and used for training KBANN-nets. The effectiveness of each training set was tested using a set of triads drawn from a third collection of shapes.

4.3.1 Training data

The first set of shapes, consisting of 36 items, was derived from the geometry section of a fifth-grade textbook [Sobel87]. The shapes (some of which appear in Figure 4.2) are almost all oriented so that one side is horizontal. Following the presentation format of the textbook, shapes with different number of sides were never presented in the same triads.

The second set of shapes (some of which appear in Figure 4.3) consists of 81 items which might be produced by a child using a slightly-modified version of LOGO. This "LOGO" set lacks the orientation bias of the "textbook" set and allows for a free mixture of shapes in triads. Hence, the second set of shapes captures the kind of experience that a child might derive from using LOGO.

A collection of 33 triads were selected from each set to train the model. Using an equal

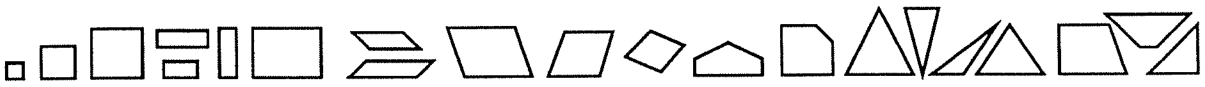


Figure 4.2: Representative textbook shapes.

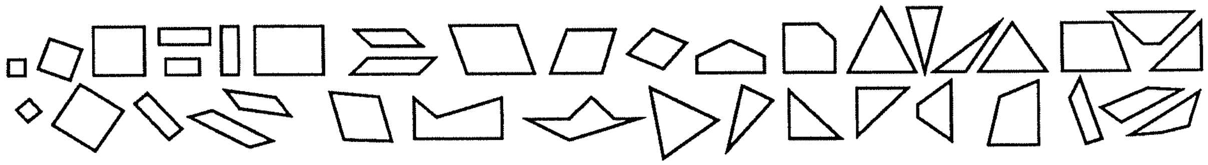


Figure 4.3: Representative shapes encountered using LOGO.

number of triads in the LOGO and textbook sets is probably unfair since LOGO affords students the opportunity to see not only a more diverse set of shapes, but also to make more comparisons between shapes. However, holding the number of training examples constant makes it possible to judge solely the effect of the increased diversity that LOGO makes possible.

4.3.2 Testing data

The test set consists of 27 items each used in a single triad. The nine resulting triads depicted in Figure 4.4 have been used to test the geometry knowledge of second and fifth-grade students [Lehrer89]. The second-grade sample consisted of 47 students who were presented these triads and asked to indicate which two were most alike and why. The fifth-grade sample consisted of 28 children who received two weeks of geometry instruction with LOGO.

4.4 Results and Discussion

Table 4.3 presents the answers generated by the model and the modal choices of children at each grade level. Responses by the model of "AB / AC" indicate that the model considers the pair AB and AC to be equally similar. This occurs when the similarity valuation of two pairs of shapes differs by less than 5%. For the children's responses "AB / BC" indicates that AB and BC were, respectively, the first and second most common responses. The notation "???" indicates that all responses were equally common.

Using only the conception of geometry expressed in LØ (i.e., the "no-training" condition), the model failed to uniquely identify the "correct" most similar pair for all of the testing triads. After training on the textbook examples, the model correctly classified three of the nine testing triads. After training on the LOGO examples, the model was correct on five of the nine test triads and uncertain between the correct answer and an incorrect answer on one other triad.

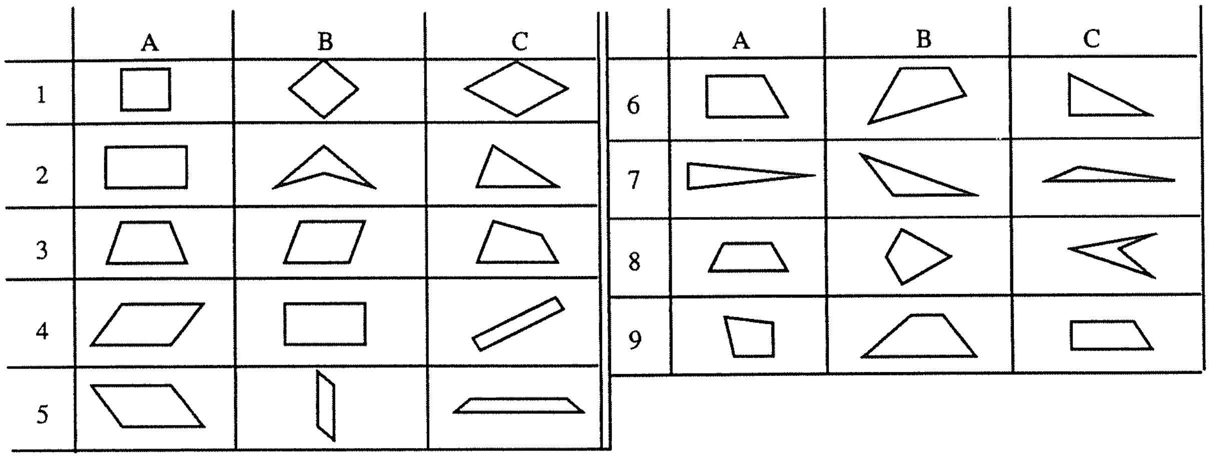


Figure 4.4: **Triads used to test learning.**

Table 4.3: Responses of the KBANN-based model and children on the nine test triads.

		Response of Model After			Children's Response	
Triad	"Correct" Response	No Training	"Textbook" Triads	"LOGO" Triads	Second Grade	Fifth Grade with LOGO
1	AB	BC	BC	BC	BC	**AB/BC**
2	AB	BC	BC	**AB**	BC	**AB**
3	AB	AC	AC	**AB**	AC	AC/**AB**
4	BC	**BC/AB**	**BC**	**BC/AB**	**BC/AB**	**BC**
5	AB	BC	BC	**AB**	BC	**AB**
6	AB	BC	**AB**	BC	???	**AB/BC**
7	BC	AC	AC	AB/AC	AC	AC
8	AB	BC	**AB**	**AB**	BC	BC/**AB**
9	BC	AC	AC	**BC**	AB	**BC**

Entries in boldface indicate that the response agrees with the "correct" response.

4.4.1 Responses of the model and children

As might be expected, before training the model, LØ matched the modal choices of the second grade sample for nearly all nine triads. This indicates that the proposed starting point for the model has a solid empirical basis.

Table 4.3 shows a similarity between the answers generated by the model after LOGO training and the responses of fifth-graders after using LOGO. On the seven triads for which the model provided definite answers, that answer agrees with the most common response of the fifth-graders on three triads, and with their second most common response on the remaining four triads. When the model did not provide a single answer, one of the two pairs selected by the model was the most common response of the fifth graders.

It should be noted that the LOGO training data are being compared to a group of students who used LOGO for a comparatively brief duration. Closer matches might occur with a longer exposure (e.g., the training set may be overestimating what the children actually generated, especially for triangles). On the other hand, the textbook examples may be overrepresented because they were arranged in a context for training that was probably more extensive than that provided by the text. (This would not be true for LOGO because one can generate multiple instances on the screen for comparison.)

4.4.2 Analysis of learning by the model

The KBANN-based model allows deeper comparisons than mere number of test examples correct because the KBANN-net following learning can be understood through comparison to its starting state (see work on rule extraction in Chapters 2 and 3). One direction for future work to use the sort of analysis made possible by the network-to-rules translator as the basis for detailed comparisons between the KBANN-based model and children. However, the analysis presented here looks only to explain why the model acts as it does after learning.

For example, after LOGO training the model was incorrect on triad 1 because, although it learned to ignore *slanty lines* and *pointiness*, the model continued to rely upon *tilted.*[6] To correct this error, triads could be added to the training set which highlight the importance *all angles congruent* and *all sides congruent* and the irrelevance of *tilted.* Also after LOGO training, the model was correct on triad 8 because it applied a newly-learned feature *convexity* while ignoring features such as *pointiness* and *area.* The model was unable to make unique decisions on triads 6 and 7 because it mixed visual with symbolic features when making its decisions. Thus, the model acts as if is at van Hiele level 0 on triad 1, level 1 on triad 8, and somewhere between on triads 6 and 7.

Further analysis of the model after LOGO training suggests that it had begun to discover the sort of relationships characteristic of van Hiele level 2. For instance, consideration of the number of sides was clearly linked to consideration of the number of angles. In other words, the network learned that these two features, which have no necessary relation, are the same for closed figures. So, depending on the problem, the model might give a response characteristic of any of van Hiele levels 0, 1, or 2 or some combination of these levels.

Analysis of the model after training on the textbook triads reveals that these triads largely failed to move the model towards level 1. While the model learned to ignore the visual features *area* and *point direction*, it still relied upon other visual features such as *pointy* and *2 long and 2 short sides.* These changes can be traced directly to the characteristics of the textbook

[6]This analysis was done prior to the development of the network-to-rules translator. It is based upon a mind-numbing examination of link weights in trained networks. However boring this was, the exercise was important as it established the foundations of the rule-extraction method described earlier.

training set. *Point direction* and *area* are eliminated because the textbook set makes this contrast quite effectively. Conversely, *pointy* remained significant because triangles which are *pointy* were never presented in a triad with *pointy* non-triangles. Hence, the model never learned that a pair of figures can be pointy and yet not be similar in a meaningful way.

Despite the inability of the textbook examples to move the model towards level 1, the model learned rules characteristic of level 2. For instance, a rule developed for recognizing squares which used a combination of *number of right angles*, *all angles equal* and *number of lines of symmetry*. Unfortunately, while this rule indicated that the pair AB for test triad 1 was the most similar, it was overshadowed by the judgments of visual features.

4.5 Conclusions About the Model

At the beginning of this chapter, Table 4.1 listed the necessary properties of an accurate and useful model of geometry learning. The van Hiele model has none of these. The results presented in this chapter indicate that the KBANN-based model already has four of the properties and can be extended to include the fifth property.

Moreover, the van Hiele model is pedagogically neutral. It gives no basis for distinguishing between effective and ineffective instructional sequences. In contrast, the experiment described in this section showed that the KBANN-based model can be used for determining the utility of instruction.

This chapter concludes the empirical verification of the ideas upon which KBANN is based. It has shown that KBANN can be used to address a problem far removed from the classification tasks of the previous chapter. The next chapter turns away from strictly empirical studies; it instead analyzes the KBANN algorithm and describes approaches to addressing some of its areas of weakness.

Chapter 5

EXTENSIONS TO KBANN

The preceding chapters have described the KBANN system and attempted to identify its relative strengths and weaknesses in comparison to other machine learning algorithms. This chapter presents two enhancements of KBANN which build upon the understanding developed in the previous chapters.

The first enhancement is a symbolic preprocessor to KBANN referred to as DAID. The general idea behind this algorithm to provide to KBANN information about important features that are not mentioned by the ruleset. This is motivated by the tests in Section 3.4.1 that showed KBANN to be relatively weak at that part of learning. The first section of this chapter describes how DAID addresses this weakness of KBANN and provides empirical evidence that the proposed method is effective.

The second enhancement to KBANN is a mechanism for the addition of hidden units not specified by the domain theory to the networks created by KBANN. Frequently, domain theories do not provide a sufficient vocabulary for a network to accurately learn a target concept. When this is true, networks will either fail to learn or be forced to adapt existing units to serve multiple needs. In the first case, the network does not learn the training examples to the desired level of accuracy. In the second case, the result of learning is an uninterpretable network. Both cases are less than desirable. Hence, it may be useful to add hidden units to KBANN-nets so as to provide them with natural places for vocabulary expansion. The second section of this chapter describes the addition of hidden units to KBANN-nets for splice-junction determination.

5.1 Symbolic Preprocessing of Domain Theories

The experiments in Section 3.4.1 indicate that KBANN is most effective when the domain theory specifies more antecedents than are required for solving the problem. That is, it is easier for KBANN to unlearn antecedents that are useless than it is for KBANN to learn the importance

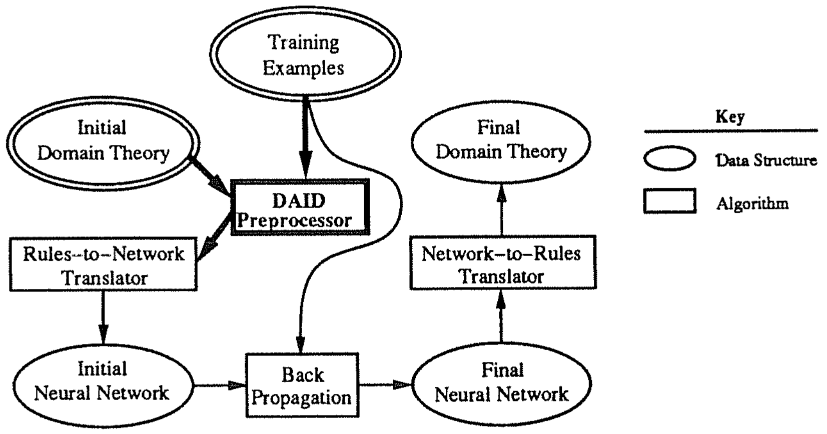


Figure 5.1: The flow of information through KBANN-DAID.

of antecedents initially believed to be useless. Hence, a method for providing information to KBANN about potentially useful features that are not used by the domain theory might be expected to improve the learning abilities of KBANN-nets. This section explores this hypothesis using DAID *(Desired Antecedent IDentification)*, a symbolic preprocessor for initial knowledge. Briefly, DAID uses the training examples to identify input features which may help to correct errors in the domain knowledge provided to KBANN.

The addition of the DAID preprocessor adds a fourth algorithmic step to KBANN, as Figure 5.1 illustrates. (Thick lines indicate the additions to KBANN entailed by DAID.) This figure shows DAID sitting between the user-provided domain theory and the rules-to-network translator. DAID uses the initial domain theory and the training examples to supply information to the rules-to-network translator that is not available in the domain theory alone. As a result, the output of the rules-to-network translator is not simply a recoding of the domain theory. Instead, the initial KBANN-net is slightly, but significantly, shifted from the state it would have assumed without DAID.

The next subsection describes the DAID algorithm and provides a simple example of its operation. Subsequently, a series of tests show that DAID significantly enhances the learning ability of KBANN.

5.1.1 The DAID algorithm

The assumption underlying DAID is that errors occur primarily at the lowest levels of the domain theory.[1] That is, DAID assumes the only rules that need correction are those corresponding to units in a KBANN-net to which the rules-to-network translator would add low-weight

[1]This idea has a firm philosophical foundation in the work of William Whewell. His theory of the *consilience* of inductions suggests that the most uncertain rules are those which appear at the lowest levels of a rule hierarchy and that the most certain rules are those at the top of the hierarchy [Whewell89].

connections from the input units. DAID's assumption that errors occur solely at the bottom of the rule hierarchy is significantly different from that of backpropagation (and other neural learning methods). These methods assume that error occurs along the whole learning path. As a result, it can be difficult for backpropagation to correct a KBANN-net that is only incorrect at the links connecting hidden units to inputs. Thus, one of the ways in which DAID provides a benefit to KBANN is through its different learning bias.

This difference in bias can be important in networks with many levels of connections between inputs and outputs (as is typical of KBANN-nets). In such networks, backpropagated errors can become smoothly distributed across the network. The result of a smooth distribution of error is that the low-level links all change in approximately the same way. Hence, the network learns little. DAID does not face this problem; its error-determination procedure is based upon Boolean logic so errors are not distributed needlessly.

Algorithm specification

To determine useful correlations between inputs and intermediate conclusions, DAID must first determine the "correct" truth-value of each intermediate conclusion. Doing this perfectly would require DAID to address the the full force of the "credit-assignment" problem [Minsky63]. However, DAID need not be perfect because its goal is simply to identify *potentially* useful input features. Therefore, the procedure used by DAID to track errors through a rule hierarchy simply assumes that every rule which can possibly be blamed for an error is to blame. DAID further assumes that all the antecedents of a consequent are correct if the consequent itself is correct. These two ideas are encoded in the recursive procedure *BackUpAnswer* that is outlined in Table 5.1.

BackUpAnswer works down from any incorrect final conclusions, assigning blame for incorrect conclusions to any antecedents whose change could lead to the consequent being correct. When *BackUpAnswer* identifies these "changeable" antecedents, it recursively descends across the dependency. As a result, *BackUpAnswer* visits every intermediate conclusion which can be blamed for an incorrect final conclusion. Table 5.2 summarizes how *BackUpAnswer* assigns blame.

Given the ability to qualitatively trace errors through a hierarchical set of rules, the rest of DAID is relatively straightforward. The idea is to maintain, for the consequent of each of the lowest-level rules, counters for each input feature-value pair (i.e., for everything that will become an input unit when the rules are translated into a KBANN-net). These counters must cover the following conditions:

- is the feature present with that value?
- is the consequent satisfied?
- is the consequent correct? (as determined by *BackUpAnswer*)

Table 5.1: Summary of the DAID algorithm.

Daid:

`;; Find input features relevant with respect to the corrected low-level conclusions.`

1. Establish eight counters (see text) associating each feature-value pair with each of the lowest-level rules.
2. Cycle through each of the training examples and do:
   - Compute the truth value of each rule in the original domain theory.
   - Use **BackUpAnswer** to guess the correct value of each lowest-level consequent.
   - Increment the appropriate counters
3. Compute pseudo-correlations between each feature-value pair and each of the lowest-level consequents.

BackUpAnswer:

`;; Determine the ''correct'' value of each intermediate conclusion.`

- For each antecedent of the rule being investigated
  - Determine the correctness of the antecedent (see Table 5.2)
  - Recursively call BackUpAnswer if the antecedent is incorrect

See Appendix B.3 for pseudocode of the algorithm.

Table 5.2: Assignment of blame by the recursive *BackUpAnswer* procedure.

Antecedents to blame for an incorrect consequent		
	Blame antecedent if	
Consequent should be	antecedent is negated and	antecedent is unnegated and
true	antecedent is true	antecedent is false
false	antecedent is false	antecedent is true

Hence, each of the lowest-level consequents must maintain eight counters for each input feature. DAID goes through each of the training examples, running *BackUpAnswer*, and incrementing the appropriate counters.

After the example presentations, the counters are combined to form numbers that are similar to the correlation between the presence, or absence, of each feature-value pair and errors at each of the lowest-level rules. This procedure results in link weight suggestions in the range [-1, ..., +1]. For the large, real-world problems tested, the majority of suggestions are for weights very near zero. Suggestions for link weights greater than 0.5 are quite rare. The process for keeping these counts and determining the pseudo-correlations is summarized as the DAID procedure in Table 5.1.

Table 5.3: Calculation of the pseudo-correlation.

$$\frac{\rho[true]\varrho[correct]\phi[?]}{\rho[true]\varrho[?]\phi[?]} * \left(\frac{\rho[true]\varrho[incorrect]\phi[present]}{\rho[true]\varrho[incorrect]\phi[?]} - \frac{\rho[false]\varrho[?]\phi[present]}{\rho[false]\varrho[?]\phi[?]}\right) \quad (5.1)$$

where:	$\rho[]$	refers to the consequent to of a rule
		true or false is **BackUpAnswer**'s guess at the correct value
	$\varrho[]$	refers to the consequent to of a rule
		correct or incorrect is whether or not the truth-value determined by the rules agrees with the guess.
	$\phi[]$	refers to an input feature-value pair
		present or absent is whether the feature-value pair is present in the example.
	?	means that all values are acceptable.

Recall that the link-weight suggestions that are the end result of DAID, are not an end of themselves. Rather, they are passed to the rules-to-network translator of KBANN where they are used to initialize the weights of the low-weighted links that are added to the network. By using these numbers to initialize weights, the KBANN-net produced by rules-to-network translator does not make the same errors as the rules upon which it is based. Instead, the KBANN-net is moved away from the initial rules, hopefully in a direction that proves beneficial to learning.

Computation of pseudo-correlations

The idea underlying the computation of pseudo-correlation that are the result of DAID is to produce a value that uses the representation of rules in KBANN-nets. Specifically, "true" is encoded as a value near one while "false" is encoded as a value near zero. Hence, increasing the weight on a link from an input unit only has an effect on the network when the input unit is true. So, the pseudo-correlations attempt to find places at which a true input is strongly correlated with an incorrect output. The formula for computing the pseudo-correlation is given by Equation 5.1 in Table 5.3.

To make this equation concrete, consider a single feature-value pair (call it $\mathcal{V}$) and a single consequent (call it $\mathcal{C}$). Steps 1 and 2 of DAID set up and increments eight counters recording the relationship between $\mathcal{C}$ and $\mathcal{V}$. If $\mathcal{C}$ is occasionally false when it should be true, then adding positively weighted link from $\mathcal{V}$ to $\mathcal{C}$ (in a KBANN-net) will reduce the number of times $\mathcal{C}$ is incorrectly false. However, a positively-weighted link cannot be indiscriminately added because $\mathcal{V}$ may often be present when $\mathcal{C}$ is correctly false. Hence, a positive link connects $\mathcal{V}$ to $\mathcal{C}$ should only be added when the probability that the link corrects errors at $\mathcal{C}$ is greater than the probability that the link causes errors that did not previously exist.

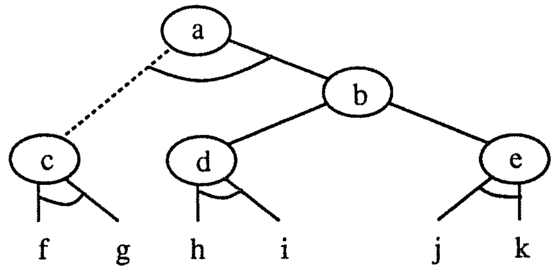


Figure 5.2: Hierarchical structure of a simple rule set.

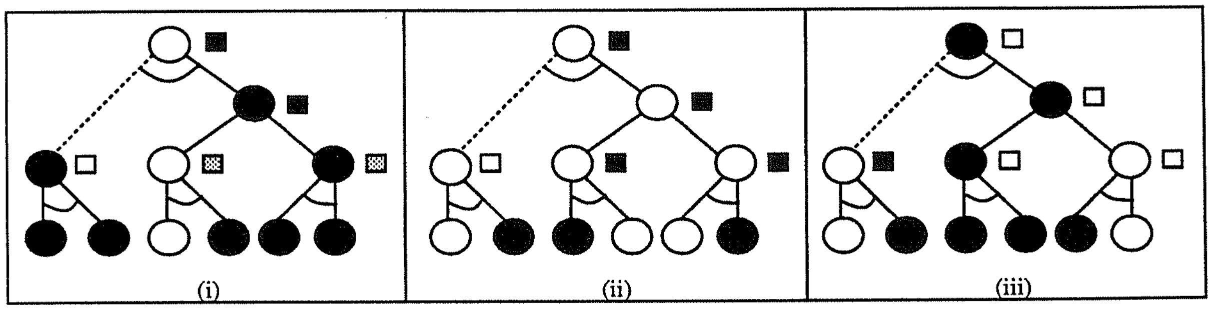


Figure 5.3: The state of the rule set after presenting three examples.

Table 5.4: Set of classified training examples.

example	input features present	correct final consequent
(i)	f, g, i, j, k	true
(ii)	g, h, k	true
(iii)	g, h, i, j	false

Finally, the pseudo-correlation is multiplied by the probability that the consequent actually has an error. So, a consequent that is almost always incorrect will receive higher link weight suggestions from DAID than a consequent that is rarely incorrect.

Example of the algorithm

As an example of the DAID algorithm, consider the rule set whose hierarchical structure is depicted in Figure 5.2. In this figure, solid lines represent mandatory dependencies while dashed lines represent negated dependencies. Arcs connecting dependencies indicate conjuncts, while the absence of arcs indicates disjuncts.

Table 5.4 contains a set of three classified examples that will be used to demonstrate the algorithm. Figure 5.3 depicts the state of the rules after the presentation of each of the three examples in Table 5.4. In this figure, lines represent dependencies, circles at the intersections of lines represent the values computed for each rule – filled circles represent true while empty of lines represent the values computed for each rule – filled circles represent true while empty represent false. The square to the right of each circle represents the desired truth value of each consequent as calculated by the *BackUpAnswer* procedure in Table 5.1. (Lightly-shaded squares indicate that the consequent may be either true or false and be considered correct. In *BackUpAnswer*, once the value of a consequent has been determined to be correct, then all of its dependencies are considered to be correct.)

Consider, for example, Figure 5.3(i) which depicts the rule structure following example (i). In this case the final consequent (a) is incorrect. On its first step backward, BackUpAnswer determines that (c) has an incorrect truth value while the truth value of (b) is correct. (Note that desired truth values invert across negative dependencies.) Because (b) is correct, all its supporting antecedents are considered correct regardless of their truth values. Hence, both (d) and (e) are correct despite their having opposite values.

After seeing the three examples in Figure 5.3 and Table 5.4, DAID would recommend that the initial weight from (f) to (d) and (e) be set to 0 while the initial weight from (f) to (c) be set to -1. However, the suggestion of a -1 weight from (f) to (c) would be ignored by the rules-to-network translator of KBANN because the domain theory already specifies a dependency in that location (even though the link specified by the domain theory has the opposite sign).

5.1.2 Results of using DAID

Results in this section demonstrate the effectiveness of the DAID preprocessor for KBANN along three lines: (1) generalization, (2) effort required to learn, and (3) accuracy of extracted rules. To a large extent, the tests reported here repeat tests described in Chapter 3. As for virtually all tests in this thesis, the data reported in this section are based upon the average of multiple ten-fold cross-validation runs.

Figure 5.4 shows that DAID improves generalization by KBANN-nets in the promoter domain five percentage points. The improvement is significant with 99.5% confidence according to a one-tailed t-test. DAID only slightly improves generalization for splice-junctions. This improvement is not statistically significant.

Equally important is the affect of the DAID preprocessor on the ability to extract rules from trained KBANN-nets. As with generalization performance, DAID enhances accuracy of the extracted rules (Figure 5.5). Specifically, on both datasets, the extracted rules are more accurate with DAID than without. Moreover, the rules extracted from KBANN-nets created using DAID are more accurate than the network from which they came on the splice-junction problem. (This result is statistically significant with 97.5% confidence (t=2.4, d.f. = 10).

Finally, and of lesser performance, is the effort required to learn the training data. Recall that the data in Section 3.2 indicate that KBANN-nets require somewhat less training effort – measured in terms of arithmetic operations – than standard ANNs. If DAID in fact makes

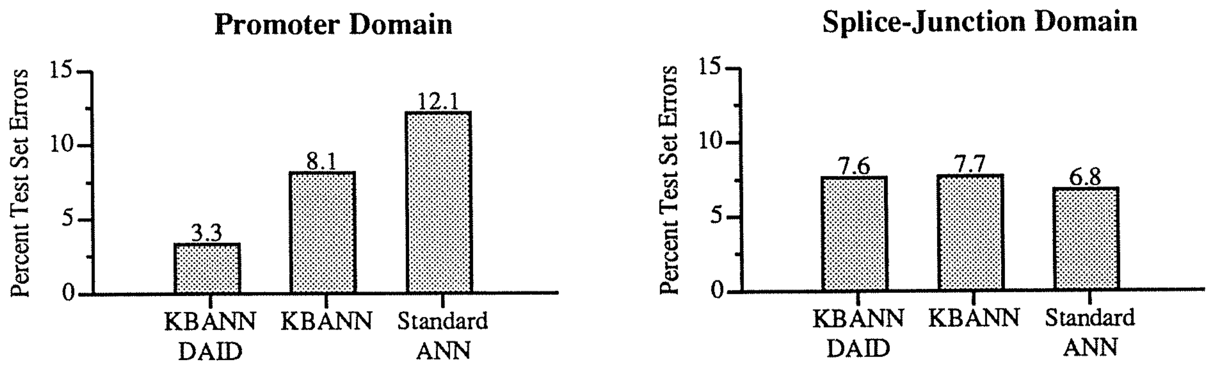


Figure 5.4: Generalization using DAID (measured using 10-fold cross-validation).

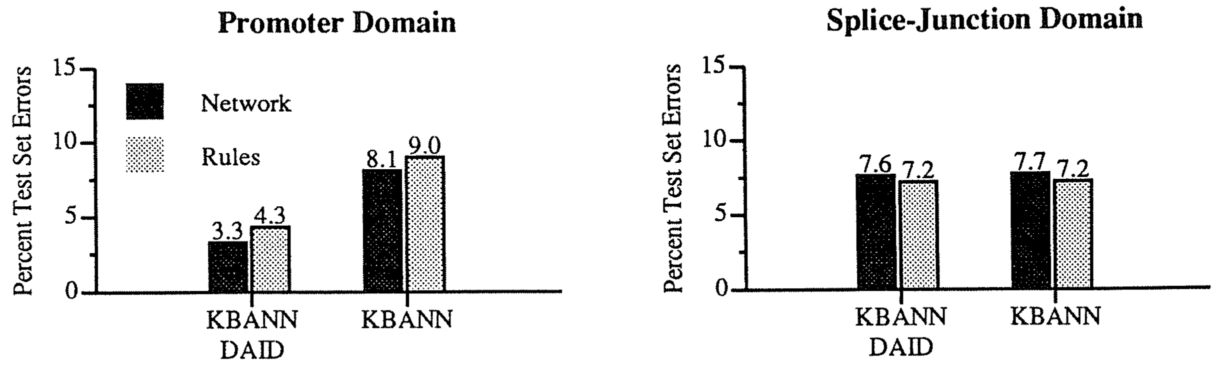


Figure 5.5: Accuracy of extracted rules using DAID (measured using 10-fold cross-validation).

learning easier for KBANN-nets, then it might be expected to appear in the training effort as well as the correctness reported above. As a result, preprocessing the rules using DAID might be expected to result in networks that learn more rapidly than those without such preprocessing.

Figure 5.6 plots the speed of learning on both splice-junctions and promoters. (DAID requires about the same number of arithmetic operations in a single training epoch.) Learning speed is measures in terms of the number of arithmetic operations required to learn the training set. Rather than using absolute training effort, both plots are scaled by the number of arithmetic operations required by KBANN-nets without DAID.

The results show that DAID dramatically speeds learning on the promoter problem. (This result is statistically significant with greater than 99.5% confidence.) As a result, the speed of learning by KBANN-nets on the promoter problem might be likened to a three-toed sloth rather than a glacier. The difference in learning speed when using DAID on the splice-junction problem is not statistically significant.

In summary, these results show that DAID is effective on the promoter data along the desired dimensions. DAID significantly improves both the generalization abilities, and learning

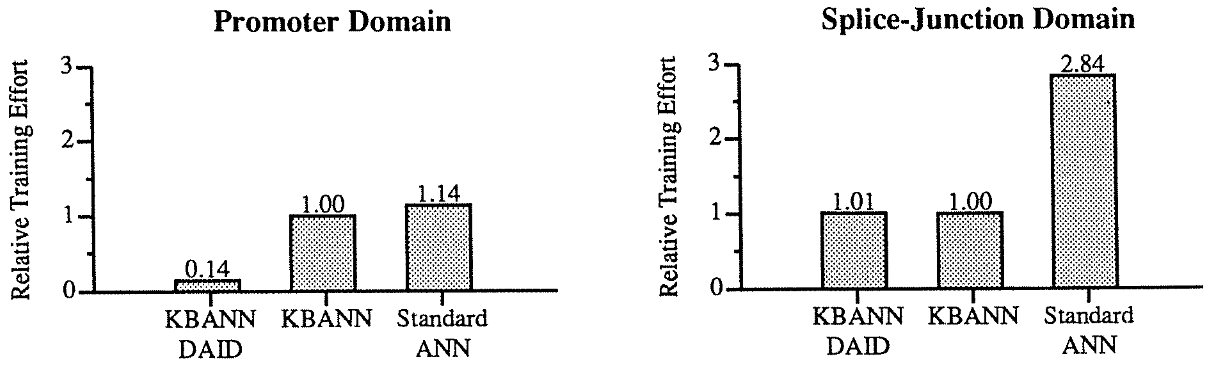


Figure 5.6: Training effort required when using DAID.

speed of KBANN-nets. Conversely, on the splice-junction dataset, DAID has little effect. The difference in the effect of DAID on the two problems is almost certainly due to the nature of the respective domain theories. Specifically, the domain theory for splice-junction determination provides for little more than perceptron-like learning. Hence, the learning bias that DAID contributes to KBANN – to changes at the lowest level of the domain theory – is not significant because there is only one level to the theory. On the other hand, the promoter domain theory is deep enough that there is a significant difference in the learning biases of the DAID and backpropagation.

5.2 The Addition of Hidden Units to KBANN-nets

This section describes an initial investigation into the addition of hidden units to KBANN-nets. Recall from Section 2.2 that two things are added to a KBANN-net to enhance its ability to learn: low-weight links and hidden units not specified by the domain knowledge. The addition of low-weight links is absolutely required for learning, so methods for the addition of links have been thoroughly studied. Additional hidden units provide a KBANN-net with the ability to expand its vocabulary beyond that provided by the initial domain theory. As a result, their addition can improve the accuracy attainable by a trained KBANN-net and improve the comprehensibility of the extracted rules.

The addition of hidden units to KBANN-nets has not been thoroughly studied because it is often unnecessary. For instance, in the promoter domain KBANN-nets are able to learn a very accurate decision procedure without needing additional hidden units

On the other hand, in the splice-junction domain KBANN-nets are less accurate than standard ANNs given large number of training examples (see Section 3.2). These results suggest that the splice-junction domain theory is impoverished. It lacks a vocabulary sufficient to allow a KBANN-net based upon the theory to learn to distinguish between the three possible categories as effectively as a standard two-layer, fully-connected ANN. Moreover, lacking the proper vocabulary, the KBANN-net is forced to adapt the vocabulary that is available to purposes for which it is not intended. As a result, the splice-junction rules extracted from trained KBANN-nets are difficult to understand. The addition of hidden units not specified by the domain theory should allow the KBANN-net to learn the vocabulary it is lacking. For this reason, a KBANN-net augmented with additional hidden units might be expected to outperform a standard ANN and be easier to interpret than a standard KBANN-net.

While accurate generalization by trained KBANN-nets is a valuable goal in and of itself, the additional of hidden units presents difficulties for KBANN. This section describes two significant problems that the addition of hidden units causes. The first is the topology question. Specifically, how many hidden units should be added, and how should they be connected to the ANN? The second question involves the interpretation of the units after training. Can these units be interpreted using the NOFM method like standard KBANN-nets or is their learning incomprehensible?

The following section describes a solution to the topology question. Subsequent sections discuss the interpretation of added hidden units and present the results of experiments which demonstrate that the solutions are effective. The presentation in this section is organized as a case study rather than a thorough investigation of the problem. Hence, the proposed solutions discussed in this section represent only one possibility. Many other approaches are possible; they are neither tested nor discussed.

5.2.1 Adding hidden units

While the performance of a neural network is dependent on its topology [Kolen90, Shavlik91], how to organize the topology of a network in order to best learn a particular task is an open question. By using existing domain knowledge about a problem to specify network topology, the rules-to-network translator of KBANN directly addresses this issue and eliminates the need to search network-topology space. However, adding extra hidden units to capture features not expressed in the domain theory reopens the problem of topology determination. Specifically, the number and connectivity of the added hidden units must be determined.

The guiding principle for determining the number and connectivity of the added hidden units is to, whenever possible, make decisions about topology to simplify the problem of interpreting trained KBANN-nets. A second idea guiding topology determination is an analogy between vision and the scanning of DNA sequences. This analogy suggests the use of *cones* [Honavar88] for scanning a DNA sequence. That is, individual hidden units should be connected only to input units representing a short, contiguous subsequence of the DNA strand. Hence, these hidden or "cone" units are similar to cells in the visual cortex that look for specific features in particular locations within the visual field. The analogy to recognition cones is supported by the biology of DNA, which suggests that certain localized regions on a DNA

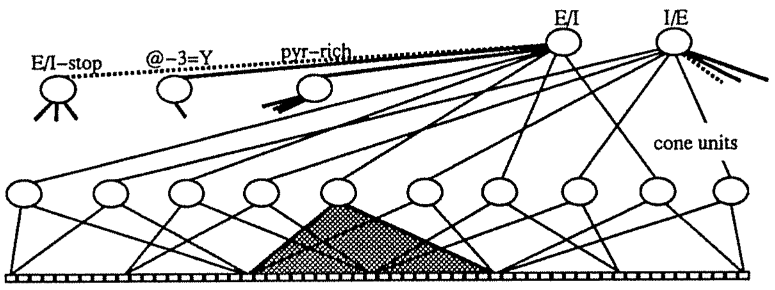


Figure 5.7: Splice-Junction Network with Cone Units

sequence are key in recognizing important biological signals. Moreover, this analogy at least partially resolves both the number and connectivity issues raised by the added units.

In the experiments reported below, each cone unit covers a sequence 20 nucleotides long. This cone size is slightly longer than several DNA features specified by the splice-junction domain theory. Cones are given 50% overlap with their neighbors. Hence, except at the ends of the sequence, where important information is least likely to be found, every input unit is covered by two cones. Networks with more cones of shorter length were tried, as were architectures which varied the amount of overlap between cones. None of the variations had a significant effect on network performance.

Each cone in the network is connected directly to only one of the output units rather than to an intermediate layer of hidden units. (Figure 5.7 is a schematic representation of the splice-junction network with cones added.) This simplifies interpretation because it does not allow multiple output units to encode different features with a shared cone. Supporting the decision to connect cones units to only one output unit is Dietterich's suggestion that sharing of hidden units is not an important contributor to the classification abilities of ANNs [Dietterich90].

The input and output weights and the biases of the cone units were initialized to random, near-zero values. The near-zero values of these parameters reflect the fact that, unlike the units that are specified by the domain theory, the roles of the cone units are completely unknown before training.

5.2.2 Interpreting added hidden units after training

Three problems arise in trying to apply the NOFM interpretation algorithm to KBANN-nets which include cone units. First, units in KBANN-nets are labeled as a result of their initial correspondence to parts of a domain theory. The cone units, however, have no similar labels. Until the biological significance of a feature measured by a cone unit can be determined, the boundaries of the DNA sequence measured by the unit are used as the consequent name in extracted rules (e.g., I/E[0..20]).

The second problem is that the rules extracted from cone units often contain unnecessary double negatives. This occurs because the biases of the cone units and the links into and out of the units are initialized to near-zero values. As a result, the signs on the links and biases of the cone units after training are meaningful in relation to each other, but arbitrary with respect to the concepts with which the units are associated. To alleviate this problem, cones with negative biases are inverted prior to interpretation. That is, the signs of the bias and all input and output links were reversed. The biases at all receivers of links from these reversed units are adjusted to reflect the changed signal.

The third problem is related to the way in which the NofM algorithm approximates each network unit with a linear threshold unit (LTU). As discussed in Sectionkbann.interp, the NofM method assumes that the units in a KBANN-net tend have activations near zero or one. When this condition of activation bifurcation is true, NofM is able to accurately approximate each unit with an LTU. While activation bifurcation almost always holds true after training in those KBANN-net units specified by the domain theory, it is not as likely to hold true for cone units, since they are not initially configured to approximate LTUs. Therefore, it was necessary to force the cone units to approximate LTUs by steepening the logistic activation functions of these units during training.

With each of these issues addressed, the NofM method can be used to interpret the rules encoded by the added hidden units, thereby making the constructive inductions of KBANN-nets available for human review.

5.2.3 Experiments in the addition of hidden units

The rules extracted from networks that include cone units fall into two categories: refinements of the original domain theory, and rules encoding regularities discovered by the cone units. It is the latter set of rules that represent the new results of this section. Thus, these rules will be the focus of the following experiments.

As in Section 3.5, the extracted rules can be evaluated both in terms of their quality and their comprehensibility. Not surprisingly, the results reported in this section will largely parallel those of the previous section. However, tests characterizing the global comprehensibility of the rules are not repeated because they provide no new information.

Quality

The rule sets containing rules extracted from cones units must be successful on two distinct measures of quality. The first measure is the same as that addressed in Section 3.5.2. Specifically, are the extracted rules accurate at classifying testing examples and do the extracted rules reproduce the behavior of the network? The second measure is: are the sets of rules that include the added units more accurate than those which do not? The former measure is necessary, for if the set of extracted rules fails on this measure, then it is of little use. The latter

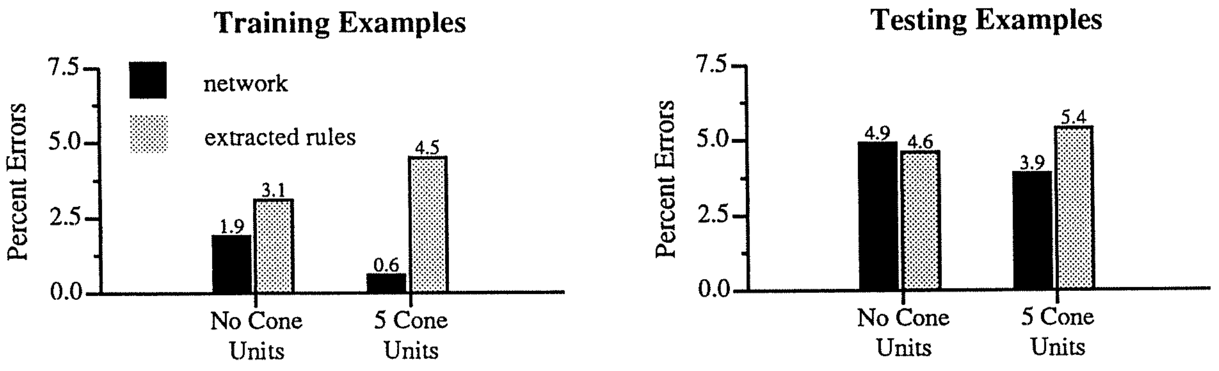


Figure 5.8: Error rates on training and testing txamples.

Table 5.5: Fidelity of rules extracted from KBANN**-nets with additional hidden units to the original networks.**

Rule Extraction Method	Overall Percent Agreement	Probability of agreement between KBANN-net and extracted rules given:	
		KBANN-net correct	KBANN-net incorrect
with cones units	90.7	0.92	0.31
without cone units	93.2	0.95	0.32

measure is desirable but not necessary, as the constructed rules may be useful in improving the comprehensibility of the rule set.

These three question are all addressed by the data in Figure 5.8 and Table 5.5. To simplify comparisons, data for networks that do not include cone units is included in both Figure 5.8 and Table 5.5. Several patterns are immediately apparent in the data. First, Figure 5.8 shows that the networks with cone units are superior to those without cone units at classifying examples in both training and testing sets. (The differences are significant with 99.5% confidence according to a one-tailed *t-test.*) Second, the rule sets extracted from the networks with cone units are less accurate at classifying both training and testing examples than either the networks or the rules extracted from networks without cone units. (All of the differences are significant at with 99.5% confidence using a one-tailed *t-test.*) Finally, while the rules extracted from networks with cone units are less accurate than the networks, they are quite accurate both at categorizing testing examples and at reproducing the behavior of the network. So, the rule set including the cone units pass the first measure of quality (accuracy and fidelity) but fail the second (superiority). Hence, the new rules must improve the comprehensibility of the rule sets to be judged useful.

Table 5.6: Typical Rules Extracted From Cone Units

```
E/I[1..20]      :-   @1 'G-A-G-----G'.
E/I[-10..10]    :-   @-5 '(g,c)---(t,G)(a,G)tTcG'.
I/E[10..30]     :-   @11 't--g', @25 'a'.
I/E[1..20]      :-   @2 'ta-G'.
I/E[-10..10]    :-   @-10 'a', @-3 'CAG----G-(t,a)'.
I/E[-20..-1]    :-   @-17 (A,C), @-3 'CAG'.
```

The notation E/I[-20..-1] signifies a cone connected to the E/I output unit that looks at a subsequence starting 20 nucleotides before, and ending at, the splice-junction. Antecedents in upper-case appear seven or more times during a single 10-fold cross-validation test. The notation '(g,c)' indicates that either a 'g' or a 'c' is acceptable.

Comprehensibility

As in Section 3.5, there are two aspects to the comprehensibility of constructive inductions: are the rule sets likely to be comprehensible, and are individual rules extracted from a network understandable? Results in the Section 3.5.3 are sufficiently conclusive that most of the tests reported therein need not be repeated. Instead, this section will focus on three questions:

1. Are the constructed rules consistent?
2. Are the constructed rules understandable?
3. What is the effect of the constructed rules upon the original rules?

Table 5.6 presents a representative set of rules extracted from cone units by NofM from one of the trials of a ten-fold cross-validation test. The table contains information relevant to the first two of these issues. Looking first at the comprehensibility of individual rules, the largest of these rules has ten antecedents, only five of which appear frequently (frequently appearing nucleotides are denoted by capital letters). Recall that the number of antecedents does not tell the whole comprehensibility story, as each antecedent is weighted. While weights are not shown here for clarity, rules rarely had antecedents with more than two different weights. Hence, the rules extracted from a *single* training episode are normally quite comprehensible.

Moreover, Table 5.6 indicates that the extracted rules are quite consistent. Nineteen of the 33 antecedents in the table appear in at least six of the other nine trials of a single ten-fold cross-validation experiment. Ten-fold cross-validation provides a good way to approach the latter issue, because each training set has 89% of the examples in common with each of the other training sets. As the majority of antecedents appear in most of the trials, it is reasonable to conclude that the networks learn to recognize consistent features with the cone units and that the NofM method is able to correctly extract this information.

Table 5.7: Rules extracted from network with five cone units.

```
E/I[-10..10]  :- @1 'G', E/Ia.
E/Ia          :- 10.0 < +5.0 * nt(@-5 'C---G-TACG') +
                        -5.0 * nt(@-5 '----TW---G').
E/I[-10..10]  :- 10.0 < +5.0 * nt(@-5 'C---G-TACG') +
                        -5.0 * nt(@-5 'G---TW').
E/I[1..20]    :- @1 'G',  1 of {@3 'G-A', @18 'G'}.
E/I[1..20]    :- @3 'G-A', @18 'G'.

I/E[-30..-10] :-  @25  'T', not(I/E[-30..-10]n).
I/E[-30..-10]n:-  @25 'A', @13 'A'.
I/E[-20..0]   :- @-17 'A', @-2  'A'.
I/E[-10..10]  :- 10.0 < +5.0 * (@-3 'GAG', @5 'C-A') +
                        -5.0 * (@-10 'A', @7 'T').
I/E[0..20]    :- @2 'TA-G'.
I/E[0..20]    :- not(@-17 'T').
I/E[10..30]   :- 2 of {@11 'T--G', not(@25 'A')}.
I/E[10..30]   :- @-17 'T'.

I/EstopA :- @2 'TAA'.             I/EstopB :- @2 'TAG'.
E/Istop  :- @-3  'TAG'.

E/I :- 10.0 < +3.0 * nt(@-3 '----T---', E/I[-10..10], E/I[1..20]) +
              +1.6 * nt(@-3 '--G--R-G', I/EstopA) +
              +0.9 * nt(@-3 'M-------', I/EstopA, I/EstopB) +
              +0.7 * nt(@-07 'Y', E/Istop) +
              -0.9 * nt(@-14 'Y-----Y').

I/E :- 10.0 < +3.0 * nt(@-10 '--------AG',
                        I/E[-20..-1], I/E[-10..10]) +
              +1.9 * nt(@-10 'Y---Y--Y--', E/Istop) +
              -1.9 * nt(I/EstopA) +
              -3.0 * nt(I/E[0..20], I/E[10..30], I/EstopA).
```

See Table 3.1 for meanings of letters other than A, G, T, and C.
"nt()" returns the number of named antecedents that match the given sequence.

Less obvious is that the rules in Table 5.7 resulting from constructive inductions simplify the rule sets for splice-junction determination. Yet, a close analysis of these rules suggests patterns that are obscured in Table 3.10. For example, the rule `E/I[-10..10]` is focused around recognizing the stop codon `'TAC'` starting at position 2. Much of the complexity of the rule results from negating antecedents which specify base substitutions that prevent the rule from being satisfied. For example, substituting a G for a C in position -5 is much worse than substituting an A.

Furthermore, the rule set with constructive inductions shows much less reliance upon direct connections to the DNA sequence. Instead, the networks learn to effectively use many of the derived features that are ignored in the rules extracted from networks without the cone units. This tendency further enhances comprehensibility, as the derived features specify combinations of bases that frequently-extracted rule sets recognize as important but refer to directly.

5.2.4 Discussion of hidden unit addition

On the majority of the measures described above, the rules extracted from the cone units are successful. The rules are both short and consistent, and hence quite comprehensible. While the extracted rules are less accurate than the networks from which they are derived and less accurate than the rules extracted from networks without cone units, the rules sets are nevertheless reasonably accurate. Furthermore, the rules constructed by the cone units seem to identify biologically-significant structures, although a more detailed assessments of the "interestingness" of the constructed rules is required.

The reasons why KBANN-nets augmented with hidden units are not as accurate as KBANN-nets without the addition are still under study. One hypothesis is that, despite steepening the activation function as described in Section 2.4, cone units continue to behave unlike linear threshold units. This violates a key assumption of the rule-extraction procedures. Hence, the rules extracted from networks containing cone units cannot achieve the accuracy of the networks. If this hypothesis is correct, further steepening the activation function may eventually eliminate the problem. However, further steepening causes problems for backpropagation.

5.3 General Discussion

This chapter presents two extensions to KBANN. The first extension is DAID, a symbolic learning adjunct to KBANN. This addition, operating as a preprocessor of the domain theory, was shown to enhance generalization, speed learning, and improve the quality of the rules extracted from trained KBANN-nets.

The second extension involves modifying the structure of KBANN-nets. Specifically KBANN-nets for the splice-junction domain are augmented with several hidden units not specified by the domain theory. The hypothesis behind adding these units is that they would enhance the splice-junction domain theory, thereby allowing more accurate answers. In addition, the added hidden units were expected to make the rules extracted from trained KBANN-nets more comprehensible by providing natural locations for learning.

Chapter 6

RELATED RESEARCH

This chapter discusses work that is similar in its goals to KBANN. This first section describes four prototypical approaches to learning from both theory and data. Subsequent to this overview is an attempt to codify approaches to learning from theory and data and to provide a way of understanding how future systems fit in with existing systems.

Following this review of hybrid systems is a discussion of systems that are similar in approach to KBANN in that they are based upon neural networks and make some recourse to symbolic reasoning. While this covers a wide range of possibilities, the review focuses on systems that are similar in spirit to KBANN.

The final part of this chapter reviews systems designed to aid the understanding of neural networks. This topic spans a range of approaches from efforts on the extraction of rules from trained networks to attempts to make trained networks amenable to visual analysis. Discussion in this part of this chapter samples the complete range of methods. However, only methods for the extraction of rules are covered in detail.

6.1 Hybrid Systems

A hybrid learning system is one that considers both theoretical information and examples during learning. Several trends have made the development of such systems an active area in machine learning. First, as discussed in Chapter 1, researchers are coming to the realization that the boundaries in training information assumed by both empirical and explanation-based learning do not exist. Moreover, synergies from using both theory and data may result in powerful learning systems.

Second, there is an abundance of psychological evidence that people rarely, if ever, learn purely from theory or examples [Wisniewski91]. For instance, Murphy and Medin suggest that "feature correlations are partly supplied by people's theories and that the causal mechanisms

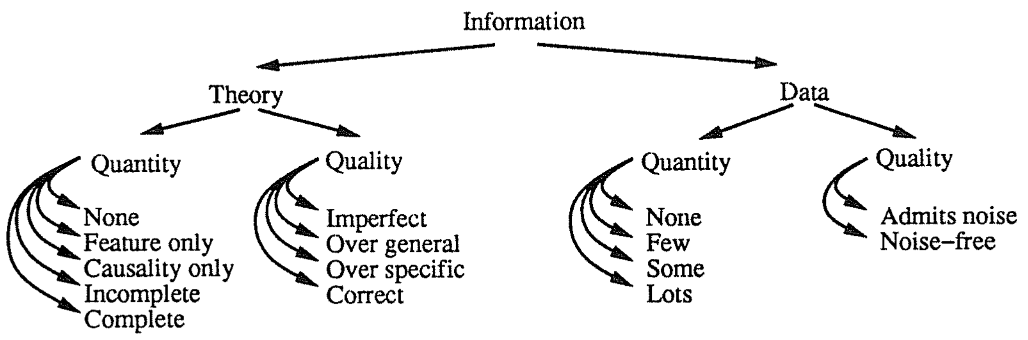


Figure 6.1: Partitioning the space of information available to machine learning algorithms.

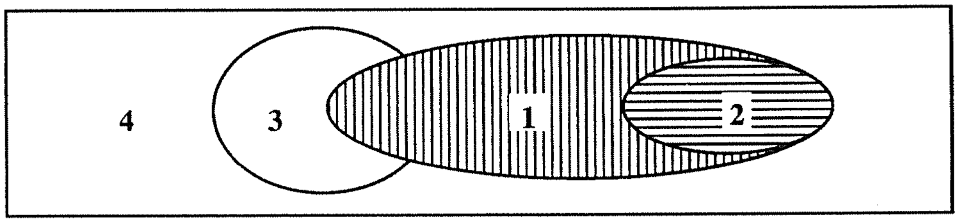


Figure 6.2: Possible classification decisions of a knowledge-based system.

contained in theories are the means by which corelational structure is represented" [Murphy85, page 294]. That is, theory and examples interact closely during human learning. While it is clear that people learn from both theory and examples, exact ways in which the interaction occur is still cloudy. Hence, it has been the subject of much research [Pazzani89, Wisniewski89, Wattenmaker87], all of which affects work in pure machine learning.

Finally, there is a purely practical consideration; hybrid systems actually seem to work.

6.1.1 Classifying hybrid learning systems

There are many dimensions along which hybrid learning systems can be described. Perhaps the most significant of these dimensions is the required information. In particular, what sort of theory and what sort of examples are required. Each of these categories can be further broken down to consider the quantity and quality of the information.

These four divisions can be further partitioned as illustrated in Figure 6.1. The subcategories on the four divisions are ordered by increasing amount of information. So, a *complete* theory contains more information than a theory that describes only the important *features.* With the exception of theory quality, the subcategories are all self-explanatory. In this partition, at least "overly general" and "overly specific" need to be defined. These terms can best be defined using the Venn diagram of Figure 6.2.

Consider the problem of deciding whether or not objects are members of some concept.

Table 6.1: Hybird learning systems.

Reference for System	Theory Quality	Theory Quantity	Data Quality	Data Quantity	Empirical Component
explanation-based	correct	complete	noise-free	one	—
empirical	overly-general	feature only	admits noise	lots	various
[Lebowitz86]	correct	complete	noise-free	some	unimem — [Lebowitz80]
[Hall88]	overly-specific	incomplete	noise-free	one or more	special
[Oliver88]	correct	complete	noise-free	some	ANNs — [Rumelhart86]
[Pazzani88]	correct	causality only	noise-free	few	special
[Flann89]	overly-general	complete	noise-free	few - pos.	special
[Mooney89]	overly-specific	incomplete	noise-free	few	ID3 — [Quinlan86]
[Hirsh89]	imperfect	zero or more	noise-free	some	Version Space — [Mitchell82]
[Danyluk89]	overly-specific	incomplete	noise free	some	special
[Redmond89]	imperfect	incomplete	noise-free	some	case-based — [Koton89]
[Katz89]	imperfect	incomplete	noise-free?	some	ANNs — [Rumelhart86]
[Mooney91b]	imperfect	zero or more	noise-free?	some	ID3 — [Quinlan86]
[Thompson91]	imperfect	zero or more	noise-free?	some	Cobweb — [Fisher87]
KBANN	imperfect	zero or more	admits noise	some	ANNs — [Rumelhart86]

The 'theory' and 'data' all indicate the minimum levels required by the system. So, a system that needs "few" examples can also use "many". Conversely, a system that requires an "overly-specific" theory cannot learn using an "imperfect" theory.

The shaded zones of Figure 6.2 represent the space in which members of a category are found, while the unshaded zones represent the space in which non-members of the concept are found. A *complete* and *correct* knowledge base describes only and all of the items in zones 1 and 2. Knowledge bases are frequently either too specific on some aspects, or too general on some aspects, or both. An *overly-general* knowledge base would include items in zone 3, thereby incorrectly classifying some negative examples as positive. An *overly-specific* knowledge base would exclude items in zone 2, with the result that some positive examples would be incorrectly classified as negative. Finally, an imperfect knowledge base would make both of these types of errors.

Table 6.1 describes thirteen hybrid learning systems in term of the partitions of Figure 6.1. The table is not complete, it is merely indicative of the use of the criteria described above. In addition, Table 6.1 contains information about a third partitioning dimension – the underlying empirical component. Frequently, hybrid learners are based upon a well known and well understood learning algorithm. As a result, this information can be very useful in determining the likely strengths and weaknesses of the system.

The hybrid systems described in Table 6.1 are sorted by date of reference, with the most recent systems at the end of the list. Scanning down the list, two trends are clear. First, more recent systems tend to be built on top of an established empirical learning system. Second, constraints on the quality of information have been eased. Whereas early system such as EXTENDED UNIMEM and OCCAM require correct theoretical information, more recent systems allow arbitrary imperfections in theories. Similarly, earlier systems required that data be noise free while more recent systems allow noisy examples. However, these reductions in quality requirements for both theory and data are generally accompanied by an increasing requirements of data quantity.

6.1.2 Approaches to hybrid learning

EXTENDED UNIMEM [Lebowitz86], OCCAM [Pazzani88], and IOE [Flann89] represent three major approaches to hybrid learning systems which take three distinctly different paths and address three distinctly different problems. EITHER [Ourston90] and Labyrinth-k [Thompson91] represent a still different approach to the problem. The next four subsections briefly describe each of these approaches, pointing out how the systems use theory and data and explaining which of the weaknesses of empirical and explanation-based learning that each address.

Extended UNIMEM

EXTENDED UNIMEM [Lebowitz86] addresses two problems; one of empirical and one of explanation-based learning. Its goals are: (1), to reduce problems caused by spurious correlations in the training examples, and (2), to reduce the number of explanations required of its explanation-based component, thereby addressing problems resulting from theory intractability. To make these reductions, EXTENDED UNIMEM uses its empirical component to initially form clusters of examples that have some common features. After the empirical component has processed all the examples, the explanation-based component examines each cluster, and attempts to explain why the features shared by the examples in the cluster are significant. If an explanation of the significance of the features can be built, the cluster is retained, otherwise it is discarded. Thus, the explanation-based component of EXTENDED UNIMEM acts as a filter for the results of the empirical component: it eliminates clusters founded upon spurious correlations.

Notice that this is a completely different use of domain knowledge than that of KBANN. Whereas KBANN uses domain knowledge to bias empirical learning, EXTENDED UNIMEM uses domain knowledge only after empirical learning is complete.

Using explanation as a filter significantly reduces the need for explanation, because explanations are built for summarized clusters of examples rather than for individual examples. In complex or ill-specified domains, Lebowitz suggests that building explanations may be extremely computationally intensive (i.e., intractable or nearly intractable), so reducing the number of explanations and specifying the features that should be explained can significantly improve the efficiency of the system [Lebowitz86, page 227]. This approach addresses none of the other problems which beset explanation-based and empirical learning systems.

OCCAM

Pazzani's OCCAM [Pazzani88] requires no initial knowledge specific to the problem domain. However, OCCAM does have a knowledge base containing general information about causality which is, by itself, insufficient to build detailed explanations. As it initially lacks specific knowledge, the explanation-based component OCCAM is initially unused. Instead the empirical component incrementally processes examples of a concept in order to form rules which can be used by the explanation-based component.

The general knowledge of causality acts on the examples in conjunction with an empirical component to eliminate unimportant features by giving OCCAM the ability to consider the meaning of features as well as correlations. Hence, OCCAM addresses three of the problems of explanation-based and empirical learners. Its causal knowledge makes it insensitive to unimportant features. It is able to build a domain theory from examples, partially addressing the knowledge-origination problem. (Knowledge of causality must still be given to the system.) Finally, the process of knowledge-base formation in OCCAM makes it unlikely that the theories it constructs will be intractable to use.

As for EXTENDED UNIMEM, the principle difference between KBANN and OCCAM is the way in which the two systems use background knowledge. OCCAM assumes that its general knowledge of causality is correct. So, it uses that knowledge as a very strong filter on examples. As a result, it is able to learn from few examples. By contrast, KBANN treats domain knowledge as a weak filter because it assumes the knowledge is only partially correct. Hence, KBANN needs more examples to learn than does OCCAM.

IOE

In their IOE *(Induction Over Explanations)* system, Flann and Dietterich assume that there exists an overly-general domain theory which must be specialized [Flann89]. To do this specialization, explanations are constructed and stored for each training example. When all explanations have been constructed, induction is performed on the explanations rather than on the examples. The induction process has two effects. First, the size of explanations may be reduced, as IOE eliminates portions of the structure which are not shared by many explanations. Second, 'equality' and 'constant' constraints may be introduced into the explanation structure [Flann89, page 11]. These constraints narrow the range of problems to which learned rules may be applied by recognizing consistencies within a single explanation structure and mandating that future examples share these consistencies.

IOE directly address several of the problems of empirical and explanation-based learning. It requires only a small initial training set because examples are needed only to generate explanations and there may be few unique and useful explanations. It passes to its empirical component a small number of explanations, and no actual examples. Hence, the problems of feature selection and spurious correlation are reduced. Writing a knowledge base is simplified by allowing it to be overly general. Finally, Flann and Dietterich indicate that IOE may reduce the problem of knowledge base brittleness by accepting a broader range of knowledge base designs that other approaches [Flann89, page 4].

Perhaps the most significant of the differences between IOE and KBANN is their assumptions about domain knowledge. By assuming the domain knowledge is only overly general, IOE is able to formulate learning as purely specialization. This rather strong bias allows IOE to learn from few examples. KBANN does not make this assumption because in real world problems it is difficult, if not impossible, to ensure that useful knowledge is only overly general. Therefore, KBANN avoids the assumption of the domain theory made by IOE. The penalty to KBANN for avoiding this assumption is that it requires more examples for learning than does IOE.

EITHER and Labyrinth-k

EITHER [Ourston90, Mooney91b, Ourston91] and Labyrinth-k [Thompson91] take a similar approach to combining empirical and explanation-based learning that is distinct from the three systems reviewed above. Both of these systems uses the initial knowledge to start the learning process. The underlying empirical learner then directly modifies the initial knowledge.

Labyrinth-k uses the initial knowledge to create a structure which Cobweb[1] [Fisher87], its underlying empirical algorithm, might create if given an appropriate set of examples. This structure is then acted on by Cobweb almost exactly as if Cobweb had created the structure in the first place.

By contrast, EITHER uses the initial knowledge to sort examples into those correctly classified and those incorrectly classified. The incorrect group is further broken down to reflect the part of the initial knowledge whose failure resulted in the misclassification. The underlying learning algorithm, ID3 [Quinlan86], is then executed using the particular incorrect examples and all the correct examples to determine a modification of the knowledge that makes the examples in both groups correct.

EITHER and Labyrinth-k principally address problems of explanation-based learning. Both systems allow the initial knowledge to have arbitrary types of errors. Moreover, the systems are able to create knowledge (in the form of rules) when supplied with no initial information. In addition, both systems use the initial knowledge to focus learning on relevant features, thereby simplifying the learning problem. Hence, they can be expected to require fewer examples and be less affected by spurious correlations than standard empirical learning systems.

These systems are very similar in approach to KBANN. The principle differences between

[1]Labyrinth-k actually uses an extension of Cobweb that allows the Cobweb algorithm use "structured" domains [Thompson91].

the approaches is their choice of learning algorithms. Whereas KBANN uses a "subsymbolic" algorithm, both EITHER and Labyrinth-k use symbolic algorithms. This results in the difference in the complexity of the approaches discussed at the beginning of Chapter 2.

6.2 Neural-Learning Systems Similar to KBANN

This section reviews several systems very much in the spirit of KBANN in that they use symbolic knowledge in some way to help define a neural network.

6.2.1 Gallant

Gallant described the first system combining domain knowledge with neural learning [Gallant88]. The principle similarity between his "Connectionist Expert System" (CES) and KBANN is in the definition of structure. CES is supplied with dependency information from which it builds a structured neural network with only feedforward connections. All of these connections initially have weight 0. Likewise, the bias initially has weight 0. No other connections are made, thus dependencies not specified can never be discovered.

By contrast, KBANN is given a set of rules which, in addition to specifying dependencies, specify the direction of the dependencies. Hence, KBANN creates a feedforward neural network with connections of weight ω or $-\omega$ depending upon the type of dependency and sets the bias of each units as necessary to satisfy the rule that the unit encodes. While CES stops at this point, KBANN adds links to the network on the observation that it is likely that some dependencies are not specified in the supplied rules. Thus, KBANN creates network with much the same structure as CES. However, KBANN networks have more links and the links initially have more meaning.

Both CES and KBANN train their networks to make them perform correctly on a set of examples. However, their methods of training differ considerably. KBANN uses standard backpropagation. Hence, the only information required during training is the input to output mapping for the learning examples. On the other hand, CES uses the "pocket" algorithm, a variant on perceptron [Rosenblatt62]. To train CES using the pocket algorithm, the correct activation of every unit in the network must be known. In the medical domain used to explain CES, this is a reasonable assumption as hidden units encode diseases. Hence, their presence or absence can be known with certainty. However, in the molecular biology domains in which KBANN has been tested, it is impossible to specify ahead of time the correct activation for each hidden unit. Therefore, networks can not be trained using the pocket algorithm.

There are a host of other small differences between CES and KBANN; however they are all minor by comparison to differences in the initial information and training method.

6.2.2 Oliver and Schneider

Oliver and Schneider [Oliver88] describe a method for using rules to decompose problems into smaller, more easily handled chunks. They report an improvement of several orders of magnitude improvement in the time required to train a neural network as a result of decomposition. The exact method of problem decomposition is not described, but their discussion of rules suggests that the decomposition is at a finer grain than the 'what vs. where' decomposition described by Rueckl [Rueckl88]. A secondary focus of the work by Oliver and Schneider is on improved speed of knowledge access and reduced brittleness of knowledge application, both of which result naturally from the use of neural networks for rule application.

The approach used by Oliver and Schneider appears to tolerate neither incorrect nor incomplete rules. Thus, this approach does not address the problems of writing a knowledge base. Nor does their approach address the any of the problems faced by empirical learning as the assumption of a correct and complete knowledge base obviates the need for empirical learning. Examples appear to be used merely for minor optimization of weights to improve rule execution.

6.2.3 Katz

Katz [Katz89] describes the *ILℵ* system which, much like KBANN, uses a knowledge base which has been mapped (apparently by hand) into a neural network. The focus of the work by Katz is on improving the time required to classify an object rather than generalization. This speed increase is achieved by building new connections which allow his *ILℵ* system to bypass intermediate processing stages. As a result, the initial resemblance of the network to the knowledge base is rapidly lost during learning. Thus, although the network is able to make corrections to its initial knowledge, it is impossible to interpret those corrections as symbolic rules which could be used in the construction of future networks.

6.2.4 Fu

Fu [Fu89] described a system very similar to KBANN in that it uses symbolic knowledge in the form of rules to establish the structure and connection weights in a neural network. However, Fu's system uses non-differentiable activation functions to handle disjunctions in rule sets. To tolerate this non-differentiability, Fu carefully organizes his networks into layers in which all of the units at a given layer are either differentiable or not. Then during training, different learning mechanisms are applied to the different types of units.

An empirical study in medical diagnosis showed this technique to be quite effective. However, the method is closely tied to its particular learning mechanism. Hence, developments in the methods of training neural networks such as conjugate gradient techniques [Barnard89] or functions based upon Bayesian reasoning [Buntine91] cannot be applied.

6.2.5 Jones and Story

Like KBANN, the networks created by Jones and Story [Jones89] have their topology and connections weights defined by sets of propositional rules. However, the intent of the network is not learning and the refinement of incorrect knowledge, but rather its use as a parallel platform for inheritance reasoning. Hence, although KBANN creates networks that are almost identical to the those created by Jones and Story (given the same set of rules), there are some practical differences. Principle among the differences is that all links in the networks created by Jones and Story are bi-directional. This allows activation to flow throughout the network, thereby making it useful for investigations of non-monotonic reasoning — the focus of Jones and Story's work.

6.2.6 Berenji

Berenji [Berenji91] describes a system in which approximately correct, "fuzzy" rules are translated into a neural network. By so doing, Berenji shows that the pole-balancing problem, a classic in control theory, can be relatively easily solved using a neural network that learns. While details of the translation of the fuzzy rules into networks are lacking, the approach appears to be fundamentally similar to KBANN. The technique is more general than the basic algorithm for rules-to-networks translation in KBANN as it is able to handle fuzzy rules, a superset of the propositional rules admitted by KBANN. (See Section 7.2 for a simple extension that allows KBANN to handle fuzzy rules.)

6.3 Understanding Trained Neural Networks

A major weakness of neural approaches to empirical learning is that the representations learned are opaque. There have been several approaches to clarifying trained neural networks. In order of increasing algorithmic complexity, the first method is to present networks in some graphical form that makes it easier for humans to see what the network is doing. A second approach is to algorithmically prune excess links, and/or units from the network, thereby making the important points of the network stand out. A third approach is to limit the values that link weights an assume. This approach forces links with approximately equal influences to have exactly the same weight; understanding is simplified as the complexity of the network is significantly reduced. A final method for the understanding of networks is the extraction of symbolic rules. When the set of extracted rules is small, it can be much more easily understood than a neural network.

6.3.1 Visualization

The algorithmically simplest approach to enhancing the comprehensibility of a neural network is to present it in a compact, graphical form that enables a user to visually analyze what learning. As people tend to be more comfortable analyzing the relative thickness of lines (or sizes of squares or shades of grey) than the relative size of real numbers, such graphical presentation can significantly improve comprehension. "Hinton diagrams" are the most common effort in this direction [Hinton89]. The problem with this approach is that presenting a reasonably complex network in human-comprehensible form requires either a considerably simplified network [Hinton89] or interactive methods for presenting small portions of the network [Wejchert89, Craven92]. For example, to make "Hinton diagrams" of reasonable size, networks are configured with few or no hidden units [Tesauro89]. Alternately, each hidden unit may be individually viewed.

A closely-related approach is to use statistical clustering software to identify commonalities in the input/output mapping (or the weights of links) of ANNs [Sejnowski87, Dennis91]. This approach only appears to be useful for understanding the "high-level" decisions of the network.

The advantage of the visualization approach is its flexibility. It can be used in combination with all of the techniques described below.

6.3.2 Pruning

A second approach to making trained neural networks comprehensible is to prune away parts of the networks that are unimportant to the calculations of the network. In so doing, the complexity of the problem can be significantly reduced, thereby making the network more easily understood. For instance, Le Cun *et al.* [Le Cun89] describe the *OBD* method that eliminates as much as 30% of the number of free parameters in a network without increasing error. Experiments with OBD on the promoter domain indicate that up to 90% of the links can be removed from KBANN-nets without increasing error on this problem [Craven90]. A related technique is described by Mozer and Smolensky [Mozer88]; their "skeletonization" method eliminates whole units rather than just individual free parameters.

6.3.3 Extraction of rules from trained neural networks

This section uses several case studies of the extraction of rules from neural networks to describe this third approach to the understanding of trained neural networks. The methods fall into two loose groups. First are methods that will work on any network (although all examples are for networks with a single layer of hidden units). Second are methods which, like NOFM, only work on specially constructed and trained networks.

Techniques making no architectural assumptions

Several researchers have reported attempts to extract rules from ANNs with a single layer of hidden units and links that are initially randomly-weighted. Saito and Nakano [Saito88] and Fu [Fu91] describe methods similar to the SUBSET algorithm described in Section 2.4.2. Saito and Nakano looked at the input/output behavior of trained networks to form rules that map directly from inputs to outputs. To limit the combinatorics inherent to algorithms like SUBSET, Saito and Nakano limited rules to four antecedents. However even with this limitation, they extracted more than 400 rules for just one output unit in an ANN with 23 outputs. Thus, while their method is potentially useful for understanding networks, it may drown researchers in a sea of rules. Moreover, the methods of both Saito & Nakano and Fu find only rules that map directly from inputs to outputs.

Hayashi describes a method for extraction of extracts fuzzy rules from trained networks [Hayashi90]. Like the above rule-extraction algorithms, the technique reduces its otherwise combinatoric search by limiting the number of antecedents per rule. While the number of rules extracted by this method is quite small (48) in the reported test domain, the algorithm appears to be subject to the same combinatorics that make the techniques of Saito & Nakano and Fu less than appealing.

The rule-extraction algorithm described by Masuoka *et al.* extracts fuzzy rules in much the same manner as the other algorithms in this section [Masuoka90]. However, it is able to extract a hierarchical rule set, thereby reducing the number of rules extracted by the system. This gain comes not without a penalty, as their method requires a second training pass (not unlike the bias adjustment training of the NOFM algorithm) after the initial phase of rule extraction. In addition, the technique has much in common with those described in the next section in that it seems to rely upon assumptions about the architecture of the network being translated.

Techniques requiring special network architectures

Sestito and Dillon [Sestito90] avoid the combinatorial problems of the SUBSET algorithm in their approach to rule extraction by using a special network form. They transform a network with J inputs and K outputs into a network with $J + K$ inputs. After training, they look for links from the original J inputs that have a similar weight pattern to one of the K additional inputs. When similarities are found, rules are created. As this method can also look for similarities within the links from the K additional units, it is possible for this method to discover that some of the outputs provide information about other outputs. Hence, the method can discover hierarchical rule sets. As such, it might be useful as a method for constructive induction from neural networks. However, for this method to discover a hierarchy, it must be implicit in the outputs. That is, to discover the relationship (robin *isa* bird *isa* animal) the network must have outputs for robin, bird and animal. So, for the promoter domain, their approach would have to extract 64 rules and have five independently trainable output units to form a set equivalent to the 14 rules in the initial theory for promoters.

Another approach to rule extraction in this class is the *Connectionist Scientist Game* of McMillan, Mozer and Smolensky [McMillan91]. They propose a special network structure in which specific units are designed to learn rules that might be used in production systems. That is, there are multiple rules, all of which scan the exact same input, but only one of which can act at a given time. While the learned rules may be either conjunctive of disjunctive, they are limited to three antecedents, none of which may be negated. Hence, the type of rules that their system is capable of learning is tightly constrained (nevertheless, the set of possible rules is quite large). The most successful of the several methods of rule formation that McMillan *et al.* report involved up to ten cycles of: (a) training a network, (b) extracting rules from that network, and (c) inserting those rules into a new network. While computationally expensive, simulations on a simple problem domain indicate that the method is able to extract a set of rules which closely resembles the correct set (the system never sees the correct set of rules). The connectionist scientist game method appears to be closely tied to the problem. The network structure from which rules are extracted is so problem dependent that it may not be applicable to a wide range of problems. Moreover, the method makes no provision for problems naturally represented by hierarchical sets of rules. Hence, while the technique has impressive performance, it may not be able to handle the types of problems tackled by KBANN.

Finally, note that the both of the algorithms described in the network-to-rules translator of KBANN falls into this category.

6.3.4 Methods of neural learning similar to NOFM

The idea that antecedents fall into equivalence classes - which underlies NOFM - is also a fundamental assumption of "limited precision neural networks" [Hoehfeld91, Hollis90]. In limited precision networks, link weights are constrained to fall into a small number of weight classes by boundaries on the accuracy with which numbers are specified. Hence, training using this style of network could result in the same sort of groupings of links that result from the clustering step of the NOFM procedure.

There are, however, several important differences between NOFM and limited-precision networks. Most significant is that NOFM is mot limited to a preselected range of possibilities. Typically limited precision networks set some bound on the number of bits used to express weights. For example, Hollis reports using from four to ten bits [Hollis90]. The problem with this approach is that the choice must be made prior to training. Hence, search through precision space may be required to find optimal precision settings for each problem. Moreover, a large number of the bits may be wasted in dead space if the links fall into widely spaced, but tight, clusters. Finally, narrow precision bounds may force link weights into clusters whose weights are not optimal. Hence, the NOFM method can be expected to result in more accurate extracted rules than an application of limited precision networks to the same problem.

A second difference between limited-precision networks and the NOFM algorithm lies in the motivation of the procedures. Whereas NOFM is motivated by a desire for understandability, limited-precision networks are motivated by constraints of hardware implementations of neural networks. Hence, the similarities between the approaches result from similar solutions to very different problems.

Nowlan and Hinton [Nowlan91] describe very different system that has an effect on neural networks similar to that of NOFM. Their system uses standard neural learning, but with an addition to the error definition. This addition raises the error of systems that have many links with differing weights. Hence, their system attempts to learn a network that is both accurate and has a limited number of distinct weight clusters. As a result, networks trained using this method may look very much like networks after the application of the NOFM method. The principle difference between the two types of networks is that the weights Nowlan and Hinton's networks may be in loose clusters whereas NOFM forces weights into perfect clusters.

6.4 Summary of Related Work

This chapter has described work that is related to one or more aspects of KBANN. The first part of the chapter described systems that, like KBANN, are able to learn from both theory and data. A review of these systems reveals that, over the past three years, there is a clear trend towards trading off theory quality against data quantity. That is, more recent systems accept a broader range of mistakes in the domain knowledge but require more training examples.

The second half of this chapter reviewed systems for enhancing the human comprehensibility of trained neural networks. The first method – visualization – merely attempts to put the network into a simple-to-understand graphical form. It makes no attempt to reduce the overall complexity of the network. Conversely, the second method – pruning – strives to increase comprehensibility by algorithmically eliminating unnecessary parts of the network.

The third method of improving network comprehensibility is the extraction of rules. These approaches take two courses. First, some techniques work on arbitrarily-configured networks. A problem common to these methods is that they must either extract huge numbers of rules or use heuristics to drastically cut the number of rules that the methods may find. Even some draconian heuristics (such as limiting to three the number of antecedents in a rule) fail to keep the number of extracted rules down to a reasonable level.

The second course is to require a special topology of the networks from which rules are to be extracted. This is the approach taken by KBANN. This topology requirement of KBANN is not particularly restrictive, for it does not preclude learning any class of problems. Yet, it is quite helpful for it allows systems to learn reasonably-sized rule sets. Hence, it is an appealing approach.

In conclusion, there is a large body of work related to various aspects of KBANN, but there is other no work that presents an integrated system that uses neural networks to learn from both theory and data.

Chapter 7

CONCLUSIONS

The central thesis of this work is that an effective and efficient learning system must be able to draw upon information both in the form of rules about how to classify instances and classified training examples. The KBANN system (described in Chapter 2) supports this thesis.

Briefly, KBANN combines *explanation-based* with *empirical* learning. KBANN **inserts** a set of propositional rules into a neural network, thereby establishing its topology and initializing its link weights. The resulting network is **refined** using a set of classified training examples and standard neural learning algorithms (e.g., backpropagation [Rumelhart86]). After training, symbolic rules are **extracted** from the refined network; this allows human review of the learned rules and transfer of knowledge to related problems.

Empirical tests in Chapter 3 indicate this process is effective. KBANN is able to use sets of rules that are only approximately correct to create networks that are very effective at classifying example not seen during training. Moreover, these networks reach acceptably low error rates after seeing far fewer training examples than other systems require. Hence, KBANN creates networks that are both effective and efficient.

7.1 Contributions of this Work

Beyond supporting the thesis, the work reported herein makes several contributions to machine learning. The following lists the major contributions.

- This work sheds light onto the distinction between "subsymbolic" and "symbolic" levels of learning. (The subsymbolic level is associated with neural learning algorithms, the symbolic with algorithms like ID3 [Quinlan86].) In particular, Smolensky argues that learning at the subsymbolic level cannot be duplicated at the symbolic level [Smolensky88]. On the other hand, Langley suggests that there is no meaningful distinction between symbolic and subsymbolic learning [Langley89]. Instead, Langley notes the important difference between neural and symbolic learning is *granularity*. Specifically, neural algorithms allow more finely-grained corrections.

If there is a significant distinction between symbolic and subsymbolic levels of learning, then KBANN, which uses both levels, should not profit from the combination. As the results reported herein clearly show that KBANN is effective, they support Langley's position that the distinction between symbolic and subsymbolic learning is a red herring.

- KBANN demonstrates that machine learning techniques can contribute to real-world problems. In particular, the tests in Chapter 3 apply KBANN to two difficult problems in molecular biology related to the "Human Genome Project". The results show that machine learning techniques can be used to address problems for which no perfect solution is known.

  In addition, the analysis of the rules extracted from trained networks suggests that the `conformation` portion of the promoter domain theory is useless. This analysis is supported by EITHER [Ourston90], another machine learning program. This suggests machine learning programs can be used in two ways for real-world problems: (1) they can help to find good solutions, and (2) they can empirically test the quality of solutions suggested by human researchers.

- KBANN is the basis of a psychological model of leaning. The experiment described in Chapter 4 shows that a model based on KBANN is more useful than the extant model.

- The NOFM algorithm (Section 2.4) for the network-to-rules translator of KBANN is a method for developing an understanding of trained neural networks. Empirical results indicate that, by comparison to the SUBSET algorithm (the currently best available algorithm), NOFM is more efficient and delivers sets of rules that are both smaller and more accurate.

- Both the promoter and splice-junction datasets were initially developed as a testbed for KBANN. The promoter problem (developed prior to the splice-junction problem) has become one of the standard test problems in the fledgling field of hybrid learning. A partial list of the places in which this problem has been studied includes: University of Texas - Austin, NASA - Ames, Cornell, Rutgers, and Johns Hopkins. (Much of the credit for the development of these datasets goes to M. Noordewier who recognized the potential of KBANN for these biological problems and extracted the information from the biological literature.)

A different way of looking at the contributions of this work is in terms of the weaknesses of explanation-based learning, empirical learning, and empirical learning using neural networks (described in Chapter 1) that are addressed by KBANN. The empirical tests reported in Chapter 3 show that KBANN resolves many of these weaknesses. The following list summarizes these results.

- Weaknesses of explanation-based learning (EBL)
  - *Basic EBL requires a complete and correct domain theory.* KBANN is able to learn from theories that are neither complete nor correct. In particular, neither of the theories describing the real-world problems studied in this thesis are correct. Yet, KBANN was able to adapt them to form very effective classifiers (Section 3.2). In addition, experiments involving the addition of noise to domain theories showed that KBANN is robust even in the presence of large amounts of domain-theory noise (Section 3.4).
  - *EBL systems do not learn at the "knowledge level".* (Note that this problem is intimately related with the prior problem.) After the insertion of a domain theory into a neural network, that domain theory is modified so that it performs correctly on a set of training examples (Section 3.2). Hence, KBANN does learn at the "knowledge level."
  - *EBL may result in knowledge that is "brittle" in application.* No results presented in this work address this issue. However, trained neural networks are one of a class of algorithms that have been touted for their graceful degradation from certainty to uncertainty [Holland86a].
  - *EBL systems may be intractable to use.* This issue results from the problems of applying large, complex domain theories to many problems. Hence, it may be necessary to simplify the theory to make its solutions tractable. The propositional domain theories used in this work are simple enough that tractability is not an issue.
  - *Domain theories must come from somewhere.* This work does not fully address the issue of knowledge origin, as techniques for the extraction of rules from standard neural networks have not yet been developed. (See Section 7.2.3 for additional discussion of this issue.) However, insofar as KBANN is able to use incorrect and incomplete knowledge, the problem of domain theory origination is significantly eased.
- Weaknesses of empirical learning
  - *Spurious correlations may reduce the accuracy of learning.* This problem is especially significant when there is a paucity of available training examples; in this case spurious correlations are likely. Results in Section 3.2 show that KBANN-nets are

most effective, relative to backpropagation, when there are few training examples. (See also Table 3.16.)

- *Examples can be described by an unbounded number of features.* KBANN forces users to identify the features believed to be important. Further, the domain theory upon which a KBANN-net is based focuses KBANN on the relevant features. Hence, KBANN is insensitive to the presence of irrelevant features. (Section 3.4 empirically tests this point.)
- *Small disjuncts can be difficult or impossible to learn.* To the extent that the creator of the domain theory is aware of small disjuncts, KBANN can address this problem as the domain theory can provide methods for classifying disjuncts with only one example. Also, derived features (intermediate level consequents in the domain theory) may recode the inputs so as to reduce the number of small disjuncts [Rendell90]. Hence, the provision of domain knowledge addresses this problem in two ways. However, no empirical tests address this issue.
- *Features may be important or unimportant depending upon their context.* Domain theories can capture context dependencies by specifying context as a feature of the environment. Therefore, a feature whose importance is context dependent could be used in rules only in conjunction with the appropriate context. Hence, this issue is addressed in principle by KBANN. However, no experiments test this issue.

• Weaknesses of neural learning methods

- *Training times are lengthy.* One might expect that the provision of knowledge to a KBANN-net would make learning easier, thereby reducing training times. However, networks created by KBANN actually take longer to train, when measured in terms of example presentations, than standard neural networks. The DAID preprocessor speeds the training of KBANN-nets, but they are still not a whole lot faster than standard backpropagation (Section 5.1).

  Note that previously-reported results [Shavlik89] indicate that KBANN learns faster than standard backpropagation. The results are for small, artificial problems for which the initial domain theory was reasonably close to the target domain theory. So, the difficulty of correcting the initial knowledge on these artificial problems was considerably less than the difficulty of deriving an answer from scratch. As a result, KBANN required less training time than standard backpropagation.
- *Initial weights significantly affect learning.* Networks created by KBANN are quite consistent in what they learn. The standard deviation of the generalization ability for KBANN-nets is smaller than any of the empirical learning systems studied (see Tables D.2 and D.1).

- *There is no problem-independent way to choose a good network topology.* The rules-to-network translator of KBANN directly address this issue. Specifically, it uses domain knowledge to establish both the topology and links weights of neural networks. Networks created this way are shown to be effective at generalizing to examples not seen during training (Section 3.2).
- *Trained neural networks are difficult to understand.* The NOFM algorithm, part of the network-to-rules translator of KBANN, supplies a method for understanding trained KBANN-nets. This rule-extraction method has been shown to produce sets of rules that are comprehensible and which accurately reproduce the behavior of the network from which they came (Section 3.5).

So, KBANN addresses the majority of the problems of explanation-based learning, empirical learning, and neural networks.

7.2 Limitations and Future Work

While KBANN is an effective machine learning algorithm, it is unfinished. This section describes plans for future improvements to KBANN. These plans fall into two categories: (a) work to address limitations in the algorithm; and (b) ideas which lacked the time to come to fruition. Along with a description of each of the plans is a brief sketch of an approach to addressing the issue raise by the plan. The following three subsections describe future work involving each of the three modules of KBANN. The subsequent sections describe plans for improvements to the KBANN-based model of geometry learning (described in Chapter 4) and enhancements to the DAID preprocessor (described in Section 5.1).

7.2.1 Rules-to-network translator

As described in Section 2.2, the rules-to-network translator is limited to handling purely propositional, non-recursive domain theories. While these restrictions appear quite limiting, they are sufficiently broad to admit a large class of problems of which the two problems from molecular biology studied in this thesis are members. However, many real-world problems can not be handled under these assumptions. (For instance, most planning problems require both the ability to bind variables and recursion.) This section presents techniques that expand the sorts of information in domain theories that the rules-to-network translator can handle.

Input features

Appendix A describes how KBANN translates five types of input features into KBANN-nets. While these techniques are a part of the rules-to-network translator, only the technique for translating nominal (and binary) features has been thoroughly tested. An area for future work is the development of test domains that explore the full space of problems that KBANN can address.

Domain theories

There are several ways in which the allowable types of domain theories can be extended from the restriction to propositional, non-recursive theories. Perhaps the most significant expansion is to allow recursive domain theories. Other possible expansions include allowing rules with certainty factors (either on consequents or antecedents), and quantified variables. Each of these possible enhancements is discussed below.

Handling uncertainty. The rules-to-network translator assumes that all rules are equally certain and that all antecedents should all be given equal consideration. Yet, in the real world many conclusions can not be made with 100% certainty. Moreover, not all antecedents are equally indicative of a given conclusion. To address these issues, the designers of the Prospector expert system phrased rules in terms of Bayesian decision trees [Duda79]. The developers of Mycin [Shortliffe84] used "certainty factors" to address this issue. Another approach to this issue is to use "fuzzy logic" [Zadeh83] to reason over predicates that are less than certain.

Mycin-like certainty factors can be integrated into KBANN by expanding the domain theory vocabulary to allow factors (ranging from 0 to 1) to be associated with each rule and each antecedent. These factors would be used to adjust link weights, with factors on antecedents adjusting weights of receiving links while factors on consequents adjust weights on sending links. For example, a rule with a certainty factor of λ would be linked to higher level rules with a connection of weight $\omega * \lambda$ (where ω is the standard weight on a link). This sort of alteration to the setting of link weights requires corresponding modifications to the setting of the bias of receiving units.

Variables. The limitation of KBANN to proposition domain theories is a major constraint on the system. This limitation is due, in large part, to the inability of neural learning approaches to handle variables. Several attempts have been made to create neural networks that are able to handle quantified variables [Touretsky88, Smolensky87]. These attempts are limited in scope. For instance, Touretsky and Hinton in their DCPS system allow a network to bind only a single variable at a time [Touretsky88]. As a result, their system must sequentially investigate each possible variable binding. Their system is, at best, a first attempt at variable handling.

If a general system for handling variables in neural networks is discovered, it can almost certainly be adapted to KBANN. However, the prospects for such a discovery are not bright.

Recursion. Although not allowed by KBANN, in many cases it is useful to allow rules to refer back to their consequents. For instance, in the "blocks world" determining whether or not a block can be moved is solved by simply determining the mobility of every block on top of the block in question. A set of recursive rules to address this problem is easy to write. Solving the question in the absence of recursion requires unique rules for every possible number of stacked blocks. Hence, recursive solutions can considerably simplify the knowledge encoding process [Shavlik90b].

Unfortunately, allowing fully-recursive domain theories requires allowing variables as well. As a result, full recursion must wait for a solution to the variable binding problem (despite the presence of algorithms to train recursive networks [Pineda87]). A subset of full recursion is possible within he framework of KBANN. Specifically, problems with a sequential structure in which the solution of one problem affects the solution to the next problem in the series can be handled by modifying KBANN to use the outputs of one solution as inputs to the next. This approach is currently being investigated by R. Maclin in the context of work on protein secondary structure prediction [Maclin91].

7.2.2 Refinement of KBANN-nets

Currently KBANN uses standard backpropagation [Rumelhart86] (see Section 1.1.3) for training networks, except that cross-entropy is used rather than the standard error function (see Section 2.2.3). KBANN is committed to neither backpropagation nor the cross-entropy function. The next section discusses modifications of backpropagation as used in this thesis and the subsequent section, an alternative to backpropagation.

Modifications to backpropagation

Error function. The cross-entropy error function, described in Section 2.2.3, is based on the assumption that each output unit is independent of the others. For instance, in medical diagnosis, the presence of one disease is independent of the presence of other diseases. In many categorization tasks, this assumption is invalid. Rather, the correct assumption is that exactly one of the outputs is true and all of the others are false. Algebraically, this statement is equivalent to Equation 7.1 in Table 7.1. An error function based on the correct assumption of dependence among the outputs might be expected to perform better than cross-entropy.

Rumelhart has suggested such an error function [Rumelhart91] – the normalized exponential. The math underlying this function appears in Table 7.1. Specifically, by forcing the sum of the activations of the output units to equal one (Equation 7.2), it is possible to interpret each output activation as the probability that the output is equal to one. Under this assumption, the probability that the network generates the correct answer is simply given by Equation 7.3. Finally, let the activation of the output units be the normalized exponential (given by Equa-

Table 7.1: **The math of the normalized exponential error function.**

$$\sum_{i}^{n} d_i = 1,\ d_i \in \{0,1\} \tag{7.1}$$

$$\sum_{i}^{n} a_i = 1 \tag{7.2}$$

$$p(\vec{\mathbf{d}}|network) = \sum_{i}(d_i * a_i) \tag{7.3}$$

$$z_i = e^{\sum_j w_{ji} * a_j + \theta_i} \tag{7.4}$$

$$a_i = \frac{z_i}{\sum_k^n z_k} \tag{7.5}$$

$$e_i = \frac{\partial C}{\partial NetInput_i} = d_i - o_i \tag{7.6}$$

n	the number of output units in a network
d_i	the desired activation of output unit i
a_i	the actual activation of output unit i
$\vec{\mathbf{d}}$	the vector of desired activations
$p(\vec{\mathbf{d}}\|network)$	the probability that a given network computed the desired output
z_i	an intermediate activation of an output unit
w_{ji}	the weight on a link from unit j to unit i
e_i	the error of unit i
C	the function defining the error at output units
$NetInput_i$	the net activation received by unit i

tions 7.4 and 7.5). Given these conditions, the error signal from the output units is simply the difference between the desired and actual activation of each unit (Equation 7.6).

The advantage of this formulation is that it exactly captures the ideas underlying categorization tasks. Therefore, its use should result in faster training and more accurate generalization. Unfortunately, neither the promoter nor the splice-junction problem, as described in Section 3.1, satisfy Equation 7.1 because both problems are formulated so that output activations of zero are interpreted as evidence for the "no" condition.

Reformulating these problems to fit this restriction would require adding an output unit to each network to capture the "no" condition. This reformulation could be made in the promoter domain by merely adding the following rules:

```
non-promoter :- not(contact).
non-promoter :- not(conformation).
```

These rules simply state that a failure of either conformation or contact is sufficient to ensure that something is not a promoter. Similar rules can be added to the splice-junction domain to create a "negative" output unit. Alternately, "negative" output units can be added by simply adding the output unit and connecting it to several newly-added hidden units thereby allowing the network to create its own negative domain theory.

Revised weight decay. To improve the interpretability of trained networks, weights is "decay" towards zero [Hinton89] Forcing all weights in the network towards zero may not be the optimal strategy for improving the interpretability of KBANN-nets. Instead, it may be better to decay each weight towards its initial value [Hinton91]. Using this decay-to-original strategy, links corresponding to dependencies in the domain theory will tend to stay at their initial weight rather than being constantly dragged towards zero. Only when there is considerable evidence that a link should have a weight other than its initial value, will its weight tend to shift.

Gaussian collectors. Nowlan and Hinton describe a technique for training networks so that their weights to fall into a small number of Gaussian clusters [Nowlan91]. The idea behind this approach is that weights get pulled towards areas in which there are already weights. The effect of forcing weights into clusters is to require the network to search for simple solutions. Simple solutions are preferred because they tend to generalize better than more complex solutions.

This idea might be adapted to training KBANN-nets by specifying the starting values of the Gaussians so that they cover the initial weights of the links in a KBANN-net. During training, these Gaussians will tend to encourage link weights to stay near their initial values. However, when differentiation is required, the Gaussians can adapt to allow it.

Note that training with pre-specified Gaussians is quite similar to the revised weight decay idea described above. The principle difference between the idea is that the Gaussians give more flexibility in training because they do not always pull weights back towards their initial values. Instead, the Gaussians pull weights towards nearby clusters which may, or may not, stay near the initial weights.

Conjugate-gradient optimization

Even using cross-entropy (or the normalized-exponential described above), backpropagation is somewhat ill-suited to training KBANN-nets because units initially have activations near zero or one. Hence, errors propagated from the hidden units are reduced to very near zero by the term $[o_i * (1 - o_i)]$ — see Equation 1.5.

"Second-order" techniques such as conjugate-gradient optimization [Barnard89] are able to overcome the effects of small error terms because they are not limited to a single learning rate.

As a result, they may be very effective at training KBANN-nets. It should be trivial to modify KBANN to use conjugate-gradient optimization. This step has not been taken due to a lack of time.

Smart nominal features

The refinement of KBANN-nets is "dumb" in that it does not take advantage of the properties of nominal features. Specifically, nominal features are encoded as a set of N input units, only one of which is on at any given time. This *1-of-N* property of nominal features is information that a smart network should use. One method for doing so might be based upon the above discussion of error functions. Specifically, the normalized exponential formulates an error function that takes advantage of the *1-of-N* property of outputs in classification tasks. It may be possible to adapt those ideas to apply to links originating from input units.

7.2.3 Network-to-rules translator

The results presented in Section 3.5 indicate that the NOFM method is able to effectively extract rules from trained KBANN-nets. However, the method also has several limitations that have received scant attention. The following five sections briefly describe and propose solutions for some of the limitations of the NOFM algorithm.

Networks must be knowledge-based

The NOFM method is based upon the assumption that link weights fall into clusters. However, this is not the case for standard neural networks. Instead, weights tend to be distributed about zero, with the majority of links having weight very close to zero. For instance, the upper-left graph in Figure 7.1 is a histogram of link weights in a standard ANN for promoter recognition. The histogram clearly shows that standard ANNs are not amenable to interpretation using NOFM.

To make the NOFM method effective on standard ANNs, it is necessary to modify the way in which networks are trained to encourage more differentiation in link weights. The graphs in Figure 7.1 other than that of the standard ANN show three approaches to achieving differentiation of link weights.

The simplest approach, represented by the upper-right graph, is to gradually steepen the slope of activation function during training. This has the effect of forcing all units in the ANN to take on values of either zero or one. As a result, it may tend to force link weight differentiation. Unfortunately, the histogram reveals a weight distribution virtually identical to the standard ANN.

A second method for attempting to get link weight differentiation is illustrated by the

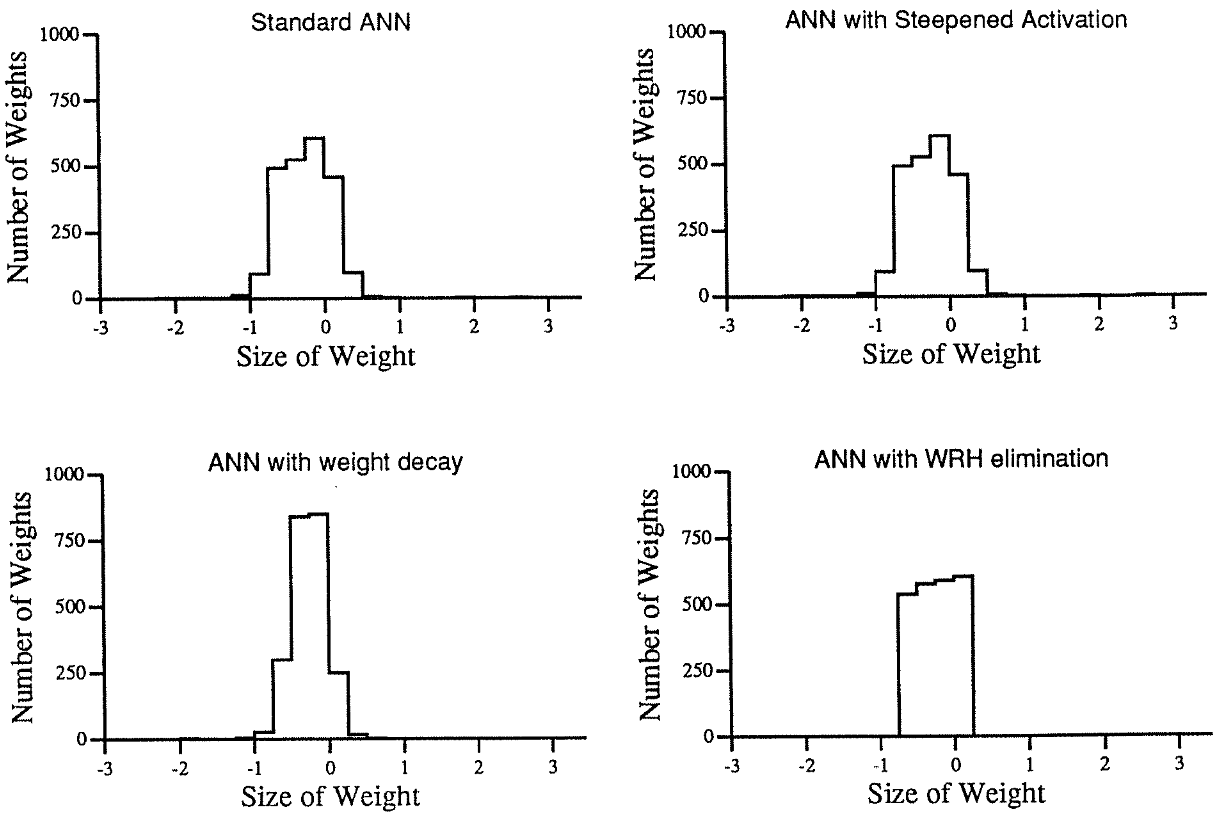


Figure 7.1: Histogram showing distribution of weights in ANNs trained for promoter recognition.

lower-left graph of Figure 7.1. This histogram plots link weights in a network trained using weight decay [Hinton89]. Sadly, weight decay does not have the hoped-for effect. Rather than encouraging differentiation among link weights, it clusters weights even more tightly around zero.

A third approach to encouraging weight differentiation is illustrated by the graph in the lower-right corner of Figure 7.1. This histogram results from training a network using the weight elimination idea of Weigand, Rumelhart, and Huberman [Weigand90]. Their technique pushes links into two groups, one of useless links with weight near zero and the other of links with weight greater than some user-defined level. Hence, this technique should begin to put links into the format required by NoFM. However, the histogram of link weight frequency shows no such differentiation.

A final approach (not appearing in Figure 7.1) to encouraging weight differentiation is to use the earlier-described technique of Nowlan and Hinton for encouraging weights to fall into a collection of Gaussian clusters [Nowlan91].

Shifts in meaning

Large shifts in the meanings of units as a result of training can make extracted rules difficult to comprehend. At a minimum, the system should flag such rules for review. A better solution would be to give KBANN some way of analyzing the extracted rules. Such a method might be based upon the "feature evaluation" system described by Matheus [Matheus90].

Insufficient domain theories

Domain theories may not provide a sufficiently-rich vocabulary to allow a KBANN-net to accurately learn a concept. When a KBANN-net is missing terms that are required to express a concept, it will modify existing terms to cover the vocabulary shortfall. This often leads to large shifts in the meaning of terms (discussed above). In such cases it may be necessary to augment the initial network with additional hidden units. However, adding hidden units opens many of the issues raised in the above discussion of extracting rules from networks that are not initialized with a domain theory. The discussion and results presented in Section 5.2 are but a first attempt at adding hidden units to KBANN-nets. The tests explore only one of the many possible unit addition methods.

Timing of rule extraction

Currently, the NOFM method operates only on fully-trained networks. As a result, the meanings of units can drift to the extent that they are difficult, if not impossible, to interpret (see above). A rule-extraction algorithm that operates *during* the training of a KBANN-net might tend to obviate this problem.

Rather than allowing link weights to freely take on arbitrary values, the algorithm would periodically round each link weight into a small set of values. (These values might be predetermined by the user, or set on the fly following the spirit of NOFM.) Training would thus consist of alternating cycles of (a) standard weight adjustment and (b) rounding. The rules produced by this method would be similar to those of NOFM. However, the search path through weight space taken by this algorithm would be quite different than that of NOFM. Instead of undergoing the transitions from an interpretable set of rules to a black-boxish KBANN-net and back to an interpretable set of rules, the rounding algorithm would preserve the comprehensibility of the knowledge base during training. In addition, this process might provide insight into the training process by allowing the inspection of the KBANN-net's knowledge base at various points during training. In this extreme, this method resembles the "connectionist scientist game" [McMillan91].

7.2.4 DAID

As it is described in Section 5.1, DAID is simply a preprocessor that slightly shifts initial weights in a KBANN-net. However, DAID can be used in several other ways.

DAID during backpropagation

One possible enhancement of DAID is to use it during training rather than just as a preprocessor. Specifically, DAID could be used to localize backpropagation. Doing so could improve the interpretability of trained networks by making changes happen only where necessary. This approach would also be valuable when using steepened activation functions, as when the activation function is steep, units tend to take on values very near zero or one. As a result, errors tend to get eliminated by the $[o_i * (1 - o_i)]$ term in the error function. By using DAID to localize error propagation, the cross-entropy function could be used to define the error signal from hidden units rather than just for output units, thereby reducing the problem of washed-out signals.

Creation of links

A second area in which DAID could be used is to alter the addition of links to KBANN-nets. Currently, adjacent levels are fully connected because there is no principled reason for making fewer connections. However, DAID establishes the probable utility of each link before neural training on the basis of the training examples. Hence, rather than using DAID to set link weights, these probabilities could be used to identify links that should be added to a KBANN-net. Doing so would eliminate the need to initially add the majority of near-zero weight links. Such reductions may both improve generalization of KBANN-nets, by preventing KBANN-nets from overfitting the training data, and reduce computation required for learning.

DAID and standard ANNs

Another possible use of DAID is to adapt it to initialize standard ANNs. For instance, it is possible to collect statistics relating each input unit to each desired output. Using these statistics, it would be possible to initialize weights as DAID does for KBANN-nets or to select connections as described above.

7.2.5 Geometry learning

The KBANN-based model of geometry learning is a first attempt to address a real problem in educational psychology. Hence, perhaps of equal significance to the results of the work described in Chapter 4 are the ideas for the future that this work generated. The following sections summarizes many of the important ideas.

Explicit training of shape-naming rules

Rules such as rule 11 in Table 4.2 are intended to let the model recognize shapes by relationship to prototypes and to treat these shape names as equivalent to other visual features. However, these shape-naming rules are never generalized. As a result, shape names do not acquire diagnostic significance with either the LOGO or textbook training examples. Nevertheless, after LOGO training, the model learned relationships independent of the shape-naming rules that are specific to pentagons, hexagons, and octagons. Thus, the model was able to generalize shape definitions, but was unable to do so from the supplied prototypes. Hence, a major topic for future work is the investigation of how the model can be modified to encourage the use and adaptation of the shape prototypes.

One approach is to directly train the hidden units related to the shape-naming rules. Through this explicit training, the shape-naming rules should generalize from the supplied prototypes. This shape training also allows altering the training style for textbook shapes so that it more closely reflects the content of textbooks. Realistically, textbooks do not present triads; instead single shapes are presented and their most specific name is given. The type of training implied by textbook presentation style is not possible for the model as described in this thesis.

Improved comparison of model and children

Significantly missing from the initial work with the KBANN-based model is a detailed comparison of the development of geometric reasoning in the model and in children. This analysis was missing both due to a shortage of information from children and the difficulty of interpreting the trained network.

Steps have been taken to address both of causes. R. Lehrer has collected a more detailed set of information about the reasoning of children than was available at the time of the initial study. On the other side of the problem, the development of the NOFM algorithm considerably eases the interpretation of trained networks. Hence, in future work using the KBANN-based model it should be possible to make detailed comparisons of the reasoning underlying decisions as well as the actual decisions.

New instruction during training

Most classroom instruction is a combination of seeing examples and being told rules. Hence, children must be able to integrate information from both sources during the learning process. As it currently stands, the KBANN-based model is unable to duplicate this style of learning as

KBANN is unable to assimilate rules after the initial translation phase.

One method of overcoming this problem is to augment KBANN with the ability to accept new rules while being trained by a set of examples. By so doing, the model will be able to more accurately reproduce classroom style instruction. Augmenting the model with the ability to add new rules after training has commenced. This addition will make possible tests of the combined effect of direct instruction with example presentation.

7.2.6 Additional empirical work

In parallel with many of the algorithmic developments discussed above, additional empirical work is needed to better understand and validate KBANN.

One of the necessary areas of future work is the development of an artificial domain of sufficient complexity to test the complete capabilities of KBANN. Artificial domains are necessary because complex, real-world problems which have no known perfect solution present problems for the accurate testing of algorithms. In particular for real-world problems no one knows which features are necessary to solve the problem and no one knows a set of rules that perfectly solves the problem. Lacking these two pieces of information, tightly-controlled experiments and quantitative measures of answer quality are impossible. For instance, the true set of rules for promoter recognition could be vastly more complex than the rules extracted by NoFM or considerably simpler. As the true solution is not known, the rules extracted by the network-to-rules translator can only be assessed in terms of their test-set accuracy.

Similarly, the true solution for the promoter problem could use all of the features in the 57-nucleotide wide window used in this work, plus some features not in this window. On the other hand, the true solution could require only a small fraction of the features in the window. Not knowing the true solution, the tests of the effect of irrelevant features in Chapter 3 assumed that the addition of features must introduce some irrelevant ones. The use of an artificial domain would allow a precise assessment of the effect of irrelevant features because the set of relevant features is known by definition.

Nevertheless, real-world domains make a good testbed for proving the validity and utility of a concept. In addition, real solutions to real problems can firmly ground otherwise ethereal ideas. So, while real-world problems are an excellent starting point and a necessary touchstone in the development and testing of algorithms for machine learning, there is a real need for testing on artificial domains as well.

7.3 Final Summary

This work describes the KBANN system, a hybrid method for learning using both domain knowledge and training examples. The method uses a combination of "subsymbolic" and "symbolic" learning to defend the thesis that an algorithm which uses learns from examples and rules can be both more effective and more efficient than algorithms that make use of only one of these sources of information. The principle results in support of this thesis appear in Chapter 3. In that chapter, KBANN is shown to more effective at classifying examples not seen during training than six empirical learning algorithms and two hybrid algorithms. Moreover, results presented elsewhere in Chapter 3 show that the method is effective under a wide variety of conditions ranging from very few training examples to poor domain knowledge.

The results in Chapters 3 and 4 show that KBANN is effective at learning in three domains. Two of the domains are in the field of molecular biology. These domains, promoter recognition and splice-junction determination, are problems from a field that is growing rapidly as a result of the "Human Genome Project". The third domain upon which KBANN is tested in this thesis is a psychological model of the development of geometric reasoning. A simple test of this model shows that it accurately reproduces the effects of instruction on children's understanding of geometry. The idea underlying KBANN have also been used by R. Maclin for protein secondary structure prediction [Maclin91], G. Scott for process control [Scott91] and T. Mitchell [Mitchell91] for appointment scheduling.

In addition to its success as a highly accurate classification method, KBANN has the ability to explain its decisions by extracting rules from trained neural networks (Section 2.4). This ability makes it possible to use KBANN as a rule-refinement system. When evaluated in this way, KBANN derives rules that are somewhat more complex than those resulting from "all-symbolic" methods of rule refinement, but which are significantly more accurate.

Much work remain to be done on KBANN, as well as the more general topic of learning from both theory and data. This thesis represents only a first step along a significant research path.

Appendix A

SPECIFYING EXAMPLES AND RULES

The rules-to-network translator (Section 2.2) requires two sets of information from a user to describe each problem. The first set of information is a specification of the features that are used to describe the examples. The second set of information is a set of rules that encode the knowledge of the problem which is supplied to the system. The following two sections define the syntax of each of these information types, and how they are handled by KBANN's rules-to-network translator.

A.1 Information about Features

The features used to describe an example take the general form:

```
(feature_name feature_type feature_values)
```

where:

`feature_name` is simply the name for a feature (e.g., size, color, etc),

`feature_type` is one of the following types in common usage in machine learning:

nominal	from among a named list (e.g., red, green, blue)
boolean	values must be either T or F
hierarchical	values exist in an ISA hierarchy
linear	values are numeric and continuous
ordered	values are non-numeric but have a total ordering

`feature_values` is a specification of the allowable values of a feature. Each type of feature has a different format in which values are expected.

The following subsections supply four pieces of information about each feature type:

1. a brief definition,
2. an example of how the feature type can be used,
3. an example of how the feature type is written for presentation to KBANN,

4. a description of how the feature is encoded by KBANN.

Missing values are handled in a consistent manner across the feature types. Namely, when the value of a feature is not known, all of the units used to encode it are given an equal activation. In addition, the sum of the activations of the units is set so that the total activation of the units is equal to the total activation of the units had the value been known. In this way, the network cannot learn to distinguish missing values merely on the basis of activation. Empirical tests [Shavlik91] have shown that this method of encoding missing values results in better generalization than other methods (such as setting all activations to zero.)

A.1.1 Nominal

This is the simplest type of feature. All the possible values of the feature are named. For the feature `color` of a coffee cup might be given by:

```
    (color nominal (red blue green))
```

KBANN would translate this feature into three units. The sole criteria for the activation of nominal features is that the summed activation of the units corresponding to a feature is equal to one. Hence, when there is information about a nominal feature, one of the units will have an activation of one and the others will have an activation of zero. When there is no information, each unit in normally given an activation equal to $\frac{1}{N}$ where N is the number of values of that feature. Alternately, the activations of each unit could be set to the prior probability of the feature having the value.

A.1.2 Binary

Binary-valued features are a special subclass of nominal features that have values of only two values (e.g., T, F). To reduce the number of input units, binary-valued features are given only a single input unit. The following is a binary feature:

```
    (temperature-is-hot binary (T F))
```

When there is no information available about a binary feature, its activation is 0.5. Note that this is a departure from the consistent total activation approach to handling missing variables.

A.1.3 Hierarchical

Hierarchical features are features defined in the context of an *ISA* hierarchy. For example, the following creates a feature which uses seven units to encode the information appearing in Figure A.1:

```
(material hierarchy (material (insulating (styrofoam)
                                          (open-cell-foam))
                              (non-insul  (paper)
                                          (ceramic))))
```

Hierarchical features act much like nominal features in that exactly one of the base-level units can be active. When no information is available concerning a hierarchical feature, then the activation of units at each level in the hierarchy is the reciprocal of the number of units in that level. Hence, in the above example, `styrofoam`, `open-cell-foam`, `paper` and `ceramic`

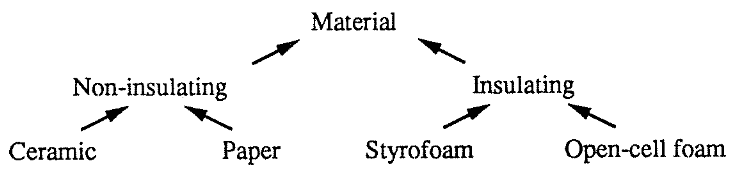


Figure A.1: A hierarchy of cup materials.

Table A.1: Equations used to encode a linear feature.

$$z_i = \frac{1}{1 + exp(\frac{abs(v - [upB - lowB]/2)}{upB - lowB})} \tag{A.1}$$

$$a_i = \frac{(z_i)^2}{\sum_j (z_j)^2} \tag{A.2}$$

where	upB	is the top of the subrange
	$lowB$	is the bopttom of the subrange
	v	is the exact value of the feature
	z_i	is an intermediate stage in the calculation of the activation
	a_i	is the activation of the unit
	i,j	are indices ranging over the units used to encode a feature

would all have activations of $\frac{1}{4}$; `insulating` and `non-insul` would have activations of $\frac{1}{2}$, and `material` would have an activation of 1.

A.1.4 Linear

Linear features have numeric values. Normally these values are continuous, but any feature with numeric values may be described using an linear feature. For example, the following feature describes the price of a cup in terms of three subranges.

```
    (price linear ((low-price 0 3)
                   (med-price 3 7)
                   (high-price 7 10)))
```

KBANN uses one unit to encode subrange. The activation of units is determined by Equations A.1 and A.2. Figure A.2 plots the activation of each feature over the whole range for `price`. So, if the value of the `price` feature is 6, then `low-price` would have and activation of 0.10, `med-price` would have an activation of 0.61 and `high-price` would have an activation of 0.29. When there is no information about a linear feature, each unit is activated as if the value was just outside of the range covered by the feature.

This encoding scheme allow users to write rules that refer to ranges rather than specific prices. Hence, this encoding creates a convenient shorthand for writing rules. However, the weights on the links entering into the consequent may be changed during backpropagation training.

A.2.2 Antecedents

In addition to simply listing positive antecedents to form a simple conjunctive rule, KBANN allows the following special predicates in the antecedents of its rules.

Gnot	Negate an antecedent.
Greater-than	Only for "linear" and "ordered" features.
Less-than	Only for "linear" and "ordered" features.
N-true	Allows compact specification of "n-of-m" concepts.

The specification of antecedents is recursive. Hence, the user may nest any of these types of antecedents within each other.

The format for including special predicates in rules, and the situations in which special predicates can be used are described below. In addition, each of these descriptions includes a picture showing the translation of a rule containing one of the special antecedents into a network.

Gnot

Gnot[1] allows rules to state that an antecedent must not be true for the consequent to be true. When multiple antecedents must not be true, each one should be in its own *Gnot* clause. Thus, the rule:

```
A :- B ∧ ¬ C ∧ ¬ D
```

would be written for KBANN as:

```
(<- A B (GNOT C) (GNOT D))
```

Figure A.3a shows how KBANN translates this rule into a neural network. Briefly, this figure shows that the negated antecedents are attached to the consequent with a weight that is the negative of the weight on the unnegated antecedents. Section 2.2.3 contains a proof that this encoding of negated antecedents is correct.

Greater-than and less-than

Greater-than and *less-than* both are defined only for "linear" and "ordered" features. (See the previous section (A.1) for descriptions of these feature types.) For instance, given a linear feature defined by:

```
(size linear ((breadbox 0 10) (car 10 50)
              (bus 50 100) (house 100 200)))
```

then a rule that states "an object must be smaller than a bus" would be written:

```
(<- goodsize (less-than size bus))
```

Figure A.3b shows how KBANN translates this rule into a neural network. The figure shows that the network has negative weight links from bus and house and positive weight links from breadbox and car. Hence, is the example is either car or breadbox sized, the net activation

[1]The term "Gnot" is used rather than simply "Not" simply to differentiate it from a function in the language used to implement KBANN.

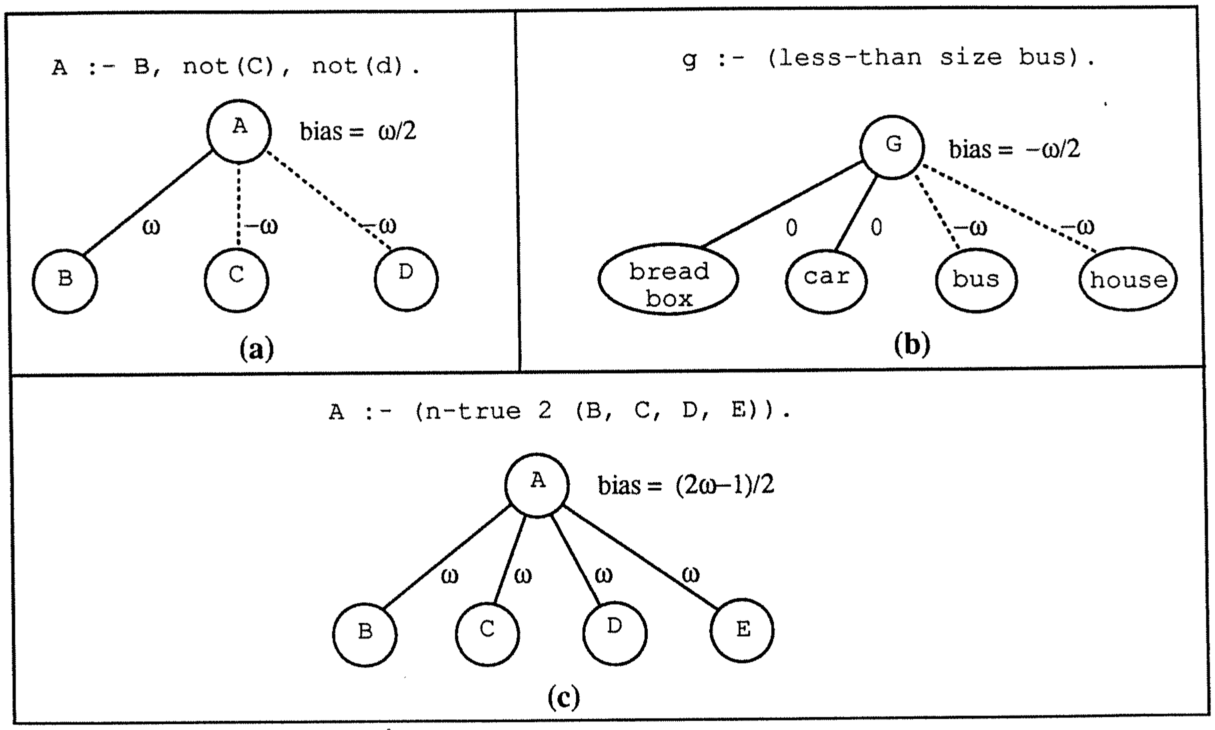


Figure A.3: Encoding rules in a neural network.

to the unit encoding the consequent will be greater than the bias. As a result, the unit will be active. Otherwise, the net activation will be less than the bias so the unit will be inactive.

N-true

N-true is a way of compactly specifying concepts characterized by requiring that n of m antecedents ($m > n$) be true. The antecedents are written in the form:

`(N-TRUE` n `(antecedent1 ...antecedentM))`

Figure A.3b shows how KBANN translates an n-of-m rule into a neural network. This figure shows that the network to encode an n-of-m rule is exactly equivalent to a network to encode a conjunct except for the bias which is reduced to reflect the fact that not every antecedent must be true for the consequent to be satisfied.

A.2.3 Variables

Recall that KBANN only allows the user to specify propositional knowledge. Hence, predicate-calculus variables are not allowed in rules. However, KBANN is able to use variables in a very limited way to reduce the number of rules that the user needs to write. The essential idea is to replace the *value* in a *(feature value)* antecedent with a variable. (Variables are denoted by a '?' prefix. Thus, *?name* is treated by KBANN as a variable.) KBANN expands all variables for which there is a known set of possible values by creating multiple rules, each with one value from the known set. When multiple values of a single rule are replaced by the same variable name, then KBANN expands the variable with values that appear in both sets.

For example, consider the problem of comparing two objects to determine whether or not they are similar. One of the requirements for similarity might be that the objects are the same color. If the possible colors for the objects are {`red`, `green`, `blue`} then the user could encode the comparison of the two objects using the following three rules:

```
(<- samecolor (obj1 red) (obj2 red))
(<- samecolor (obj1 green) (obj2 green))
(<- samecolor (obj1 blue) (obj2 blue))
```

Alternately, these three rules could be written using the following single rule by merely using variables in the antecedents appropriately.

```
(<- samecolor (obj1 ?y) (obj2 ?y))
```

Appendix B

PSEUDOCODE

This appendix contains pseudocode for the major parts of KBANN. Following this pseudocode, readers should be able to re-implement the code used in this thesis. In combination with the datasets available from the collection at UC-Irvine, this makes possible the reproduction of every experiment described in this thesis.

Note: in all of the following pseudocode, lines beginning with semicolons are comments.

B.1 Pseudocode for the Rules-to-Network Translator

The pseudocode in this section is for a basic implementation of the rules-to-networks translator. It assumes that all features are nominal and all antecedents are positive. This translator is sufficient for the promoter recognition problem and needs only a small expansion (to handle n-of-m style predicates) for the splice-junction determination problem.

```
Rules-to-Network()
Let: $\mathcal{U}$ be a data structure for units with the following fields
                linksFrom - an initially empty list
                ruleCount - initially 0
                bias - initially 0
                level - initially 0
                name - no initial value
     $\epsilon$ be a very small number
     $\omega$ be the initial weight for links specified by knowledge
Given: $\mathcal{R}$ - a list of rules with no internal disjunctions and
           no external disjunctions with more than one antecedent
           ;; See Appendix A.2 and Chapter 2
       $\mathcal{F}$ - a list of nominal features where each item in the list is in
           the form (featureName value1 value2 ... ):
           ;; See Appendix A.1
;; First create all the input units for the KBANN-net by going through each nominal
;; feature and making a unit with the name $fearureName.valueName$
;; Note that $\mathcal{U} : name$ refers to the name field of unit $\mathcal{U}$
$\forall (f \in \mathcal{F})$
  $\forall (v \in <values\ of\ f>)$
    create a unit $\mathcal{U}$ with
      $\mathcal{U} : name = f.v$
      $\mathcal{U} : level = 1$
;; Next create hidden and output units by creating a unit that corresponds
;; to every unique consequent and antecedent in the set of rules. Also,
;; link each consequent to its antecedents and increment the bias of each
;; consequent on the assumption that the rule is conjunctive.
;; This code above assumes that all disjunctive rules with more than one
;; antecedent originally in $\mathcal{R}$ have been rewritten according to Section 2.2.
$\forall (r \in \mathcal{R})$
  Let $\mathcal{U}_r$ be the unit with $\mathcal{U}_r.name =< consequent\ of\ r >$
  if $\mathcal{U}_r$ does not exist then create it
  Increment $\mathcal{U}_r.ruleCount$
  $\forall (a \in < antecedentsofr >)$
    Let $\mathcal{U}_a$ be the unit with $\mathcal{U}_a.name = a$
    if $\mathcal{U}_a$ does not exist then create it
    Create a link from $\mathcal{U}_a$ to $\mathcal{U}_r$ with weight $\omega$ and put it in $\mathcal{U}_r.linksFrom$
    Add $\omega$ to $\mathcal{U}_r.bias$
;; Now adjust the bias of each unit so that it is either $\frac{2P-1}{2}\omega$
;; for conjunctive units or $\frac{\omega}{2}$ for disjunctive units
;; If a unit encodes a disjunct, then its rulecount will be greater then one.
$\forall u \in < listofunits >)$
  if $(u.rulecount > 1)$ then $u.bias = \frac{\omega}{2}$
  else Decrement $u.bias$ by $\frac{\omega}{2}$
Establish levels of units according to one of the schemes described in Section 2.2.
;; Add low weight links to the network by fully connecting all units
;; separated by one level
$\forall u \in < listofunits >)$
  $\forall uu \in < listofunits >)$
    if $((u.level = uu : level + 1)$ and (there is no existing link between $u$ and $uu$))
      THEN create a link from $uu$ to $u$ with weight 0
;; Finally, slightly perturb the network by adding to each weight a number close to zero.
;; This prevents problems resulting from symmetries in the network.
$\forall (u \in < listofunits >)$
  $\forall (l \in u : linksFrom)$
    add a random number in the range $[-\epsilon \ldots + \epsilon]$ to the weight of $l$
  add a random number in the range $[-\epsilon \ldots + \epsilon]$ to $u : bias$
```

B.2 Pseudocode for the Network-to-Rules Translator

B.2.1 The SUBSET algorithm

```
SUBSET():
GIVEN: $U$, a set initially containing each output unit
         $Z$, a positive number (typically 1.0)
         $\beta_n$, a beam width for subsets of negatively weighted links
         $\beta_p$, a beam width for subsets of positively weighted links
;; Form rules for only the units which participate in other rules or lead to a final conclusion
WHILE $U \neq \{\}$
  ;; begin by finding combinations of the positively weighted links that exceed the bias
  LET $v = pop(U)$
      $\phi =< bias\ on\ v > +Z$
      ExceedingSubsets = BBSEARCH(positively weighted links to $v$, $\phi$,$\beta_p$)
;; then find combinations for the negatively weighted links that drive
;; each positive combination below the bias
  $\forall(es \in ExceedingSubsets)$
    ADD to $U$ the units in $es$ that have not previously been seen
    LET $\Phi =< sum\ of\ link\ weights\ in\ es > +Z- < bias\ on\ v >$
        NegatingSubsets = BBSEARCH(negatively weighted links to $v$, $\Phi$, $\beta_n$)
    $\forall(ns \in NegatingSubsets)$
      ADD to $U$ units in $ns$ that have not previously been seen
      FORM a rule: if $es$ and (not $ns$) then $v$.
```

```
BBSEARCH($links$, $threshold$, $beamWidth$)
;; this algorithm simply implements a beam search
;; with a branch-and-bound component
LET $branches$ be every link in $links$
$\forall(b \in branches)$
  LET branchSum be weight of link
LOOP
  Eliminate all branches for which $threshold$ is greater than
    <sum of the weights of all links of lesser weight than any link in the branch>
      + branchSum
  KEEP only $beamWidth$ of $branches$ by eliminating branches with smallest branchSum
  IF every member of $branches$ has $branchSum > threshold$ then Terminate LOOP
  IF there are no ways to expand any member of $branches$ then Terminate LOOP
  WITH every branch
    Form new branches using every link of lesser weight than any link in the branch
    Increment branchSum of new branches by the weight of then added link
Return all members of $branches$ with $branchSum > threshold$
```

B.2.2 The NoFM algorithm

```
N-of-M()
GIVEN $U$, a set initially containing each output unit
        $exs$, the training examples
;; $U$ is a set containing all consequents that have not yet been expanded into rules
WHILE $U \neq \{\}$
  LET $v = pop(U)$
      $i = 0$, $sum = 0$
      $\phi$ = bias on $v$
      $\gamma$ be a set of groups found using GROUPER(links to $v$)
      $\mathcal{R}v$ be a rule corresponding to the unit $v$;
          initially $\mathcal{R}v$ has no antecedents and a threshold = $\phi$
  $\gamma = DELETE\_GROUPS(\gamma, \phi, exs)$
  ;; With only important antecedents remaining, rewrite the result as a single rule
  $\forall(g \in \gamma)$
    $\forall(ga \in g)$
      Add to $\mathcal{R}v$ an antecedent $ga$ with weight $\gamma.weight$
      Add to $U$ $ga$ unless $ga$ is an input units or has already been added to $U$

Regularize the rules by making the threshold on every rule the same
  and adjust antecedent weights accordingly
Re-express all of the rules as a neural network using the rules-to-network translator.
Freeze weights on all links
Optimize thresholds using training examples and backpropagation
Simplify rules by eliminating weights on antecedents and thresholds
```

```
DELETE_GROUPS(groups, threshold, examples)
;; This function heuristically eliminates clusters of links that are not useful
;; for the classification of training examples
Associate with each group a counter $need\_group$ which has an initial value of 0.
$\forall(e \in examples)$
   Compute the activation of every unit
   If the current example is correctly classified
      ;; eliminate the activation traveling along each group to see if that group
      ;; is important for the correct classification of the current example
      $\forall(g_i \in groups)$
         Set the weight on all links in $g_i$ to 0.
         Compute the activation of every unit
         if the state of any consequent changes
            then increment $need\_group_i$
         Reset the weight of the links in $g_i$ -
   $\forall(g_i \in groups)$
      if ($need\_group_i = 0$ then remove every link in $g_i$ from the network
      Return $groups$
```

```
**GROUPER**(*links*)
;; The group finding algorithm for NOFM that runs in $O(n * lg(n))$ time
Sort *links* in descending order by weight magnitude
Delete all links whose magnitude is below a threshold $\tau$   ; typically $\tau = 0.1$
LET $\gamma = \{\}$
      $g = \{\}$
      $g\_avg = 0$
      $\mathcal{T}$ = a user defined threshold from [0...1] (Typically 0.75)
$\forall(L \in links)$
  LET $lw = < magnitude\ of\ weight\ on\ L >$
  IF $(g = \{\})$ THEN
  ;; the current group is empty so start a new one
    $g\_max = lw$
    PUSH $L$ onto $g$
  ELSE
    IF $(lw > \mathcal{T} * g\_avg)$ THEN
      ;; add link to the existing group as it is close enough to the group's average
      PUSH $L$ onto $g$
      $g\_avg = average($<magnitude of link weights in g>$)$
    ELSE
      ;; The link is too small to be added so close the current group and start a new one
      PUSH $g$ onto $\gamma$
      $g\_avg = lw$     $g = \{ L \}$     $g\_max = lw$
    IF $(g\_max > g\_avg * \frac{1}{\mathcal{T}})$ THEN
      ;; the addition of a link drags the average down so that the highest weight link is
      ;; too large. So eliminate the current link from the current group, close the current
      ;; group and start a new group
      REMOVE $L$ from $g$
      PUSH $g$ onto $\gamma$
      $g\_avg = lw$     $g = \{ L \}$     $g\_max = lw$
 RETURN $\gamma$
```

B.3 Pseudocode for the DAID Algorithm

In the pseudocode of this section, the notation $\rho[true]\rho[incorrect]\phi[present]$ is used to identify a single counter from among the eight set up at each input unit from each low level consequent. Replacing a name inside '[]' with a ? indicates that either value is acceptable. Hence, $\rho[?]\rho[correct]\phi[present]$ equals $\rho[true]\rho[correct]\phi[present] + \rho[false]\rho[correct]\phi[present]$.

```
DAID()
$\forall(\phi \in InputFeatures)$          ;; First initialize counters
  $\forall(\rho \in Rules)$
    Set up the counters for each of the following 8 situations:
      $\rho[True|False]\rho[Correct|Incorrect]\phi[Present|Absent]$
FOREACH $(\rho \in Rules)$
  Establish a state-variable $\rho\_consequentCorrect$
$\forall(\epsilon \in Examples)$     ;; scan each example incrementing the appropriate counters
  Compute truth value of each rule
  $\forall(fc \in FinalConclusions)$
    BackUpAnswer($fc$, correctAnswer, false)
  $\forall(\phi \in InputFeatures)$
    $\forall(\rho \in Rules)$
      Increment the appropriate counter from among
        $\rho[True|False]\rho[Correct|Incorrect]\phi[Present|Absent]$
;; with counters complete, now compute a number that captures the contribution
;; of each feature-value pair to correcting errors of each of the
;; lowest level antecedents $\forall(\phi \in InputFeatures)$
  $\forall(\rho \in Rules)$
    $on = \frac{\rho[true]\rho[correct]\phi[?]}{\rho[true]\rho[?]\phi[?]} * (\frac{\rho[true]\rho[incorrect]\phi[present]}{\rho[true]\rho[incorrect]\phi[?]} - \frac{\rho[false]\rho[?]\phi[present]}{\rho[false]\rho[?]\phi[?]})$
    $off = \frac{\rho[false]\rho[correct]\phi[?]}{\rho[false]\rho[?]\phi[?]} * (\frac{\rho[true]\rho[?]\phi[present]}{\rho[true]\rho[?]\phi[?]} - \frac{\rho[false]\rho[incorrect]\phi[present]}{\rho[false]\rho[incorrect]\phi[?]})$
    $on = max(0, on)$
    $off = min(0, off)$
    Replace the 8 counters of $\phi$ with: $\rho\_useweight$ = if ($on >$ abs($off$)) $on$ else $off$

BackUpAnswer(consequent, correctAnswer, currentlyCorrect)
;; a recursive procedure for stepping backward through a rule set, following
;; trails of correct and incorrect reasoning by the rules.
$\rho\_consequentCorrect$ = (currentlyCorrect or ValueOfConsequent = correctAnswer)
;; first determine if the truth-value of the consequent is correct.
;; It is correct if either its parent is correct or the truth-value is the same as $correctAnswer$.
FOREACH $(d \in DependenciesOfConsequent)$
  IF <at a negative dependency>
    dd = not(correctAnswer)
    ELSE dd = correctAnswer
  BackUpAnswer(d, dd, $\rho\_consequentCorrect$)
```

Appendix C

KBANN TRANSLATION PROOFS

This appendix contains detailed proofs that the scheme used by the rules-to-network translator accurately translates both conjunctive and disjunctive rules. These proofs complement the proofs appearing in Section 2.2.3. Specifically, Theorems 3 and 5 restate Theorems 1 and 2 but use a significantly different approach in the proof.

The following terms are used in the proof of Theorem 3:

C_a	the minimum activation for a unit to be considered *active*
A	a number such that $C_a \leq A \leq 1.0$
C_i	the maximum activation for a unit to be considered *inactive*
I	a number such that $0.0 \leq I \leq C_i$
$\mathcal{F}(x)$	$1/(1+e^{-x})$, the standard logistic activation function
w_p	the weight on links corresponding to positive dependencies
w_n	the weight on links corresponding to negated dependencies
θ	the bias
P	the number of positive antecedents to the rule
N	the number of negated antecedents to the rule
K	the total number of antecedents to a rule ($K = P + N$)
p	a number such that $1 \leq p \leq P$
n	a number such that $1 \leq n \leq N$.

Theorem 3: *(Repeats Theorem 1) Setting:*

1. $w_p = -w_n = \omega > 0$
2. $\theta = -\frac{2P-1}{2}\omega$

correctly encodes conjunctive rules into networks that reproduce the behavior of the rules under the conditions specified below.

The proof of this theorem uses Lemmas 3A and 3B.

Lemma 3A: *Part 1 of Theorem 3, that $w_p = -w_n = \omega$, results in an accurate encoding of conjunctive rules.*

Proof of Lemma 3A:

The method for mapping conjunctive rules into a KBANN-net is correct when Inequalities C.1 – C.4 are satisfied. Briefly, Inequality C.1 describes the only condition under which the consequent of a conjunctive rule is true; namely that all of its positive dependencies are true and none of its negative dependencies are true. Inequality C.1 simply translates this statement into a network, saying that for a unit to be active all of its positive antecedents must be active and none of its negated antecedents can be active. Inequality C.2[1] states the condition that a unit is inactive when fewer than all of its positive antecedents are active. Similarly, Inequality C.3 states the condition that a unit is inactive when at least one of its negated antecedents in active. Inequality C.4 gives the fourth possible condition on a unit; a unit is inactive when fewer than all of its positive antecedents are active and more that one of its negated antecedents are active. Hence, these inequalities describe all of the conditions which a conjunctive unit created by KBANN may experience.

$$C_a \leq \mathcal{F}[Pw_pA + Nw_nI + \theta] \tag{C.1}$$

$$C_i \geq \mathcal{F}[(P-p)w_pA + pw_pI + Nw_nI + \theta] \tag{C.2}$$

$$C_i \geq \mathcal{F}[Pw_pA + (N-n)w_nI + nw_nA + \theta] \tag{C.3}$$

$$C_i \geq \mathcal{F}[(P-p)w_pA + pw_pI + (N-n)w_nI + nw_nA + \theta] \tag{C.4}$$

The interesting conditions on these four inequalities are when they are at their boundaries. For instance, the interesting case for Inequality C.1 is when the net incoming activation to a unit is minimized. This occurs when each of the positive antecedents has an activation of exactly C_a and each of its negative antecedents has an activation of exactly C_i. If the resulting activation at this boundary condition only equals C_a then Inequality C.1 will always be satisfied because the every change to the network for which the rule remains true must increase the net incoming activation. Hence, any change must increase the activation of the unit. Equation C.5 expresses this boundary condition on Inequality C.1.

For units that should be inactive – because the consequent of the underlying rule is not satisfied – the converse of the preceding discussion applies. Thus, interesting boundary condition on Inequality C.2 occurs when the net incoming activation to a unit is maximized. This occurs exactly one of positive antecedents inactive, the activation of that antecedent is C_i, the activation of the remaining $(P-1)$ positive antecedents is 1.0, and the activation of every negative antecedent is 0.0. If, at this maximum, the unit has an activation of C_i, then Inequality C.2 will always be satisfied. Equation C.6 expresses this boundary condition on Inequality C.2.

[1]Inequality C.2 should be written

$$C_a \leq \mathcal{F}[\sum_{j=1}^{P} w_pA_j + \sum_{i=1}^{N} w_nI_i + \theta]$$

to be precisely correct. The complete statement is not made simply for brevity.

The final interesting boundary condition (on Inequality C.3) is very similar to the boundary condition on Inequality C.2. It occurs when exactly one of the negative antecedents is active. Specifically, the net incoming activation is maximized when the activation of each of the positive antecedents is 1.0, the activation of the active negated antecedent is C_a, and the activation of every negative antecedent is 0.0. Again, at this maximum, the unit must have an activation no greater than C_i. Equation C.7 expresses this boundary condition on Inequality C.3.

Equations C.6 and C.7 capture both of the interesting boundary conditions that might result from Inequality C.4. Hence, Equations C.5 – C.7 capture all of the interesting boundary conditions on the activation of a unit that encodes a conjunct. Equations C.8–C.10 simply restate these equations in more algebraically tractable forms.

$$C_a = \mathcal{F}[Pw_pC_a + Nw_nC_i + \theta] \tag{C.5}$$

$$C_i = \mathcal{F}[(P-1)w_p1.0 + w_pC_i + Nw_n0.0 + \theta] \tag{C.6}$$

$$C_i = \mathcal{F}[Pw_p1.0 + (N-1)w_n0.0 + w_nC_a + \theta] \tag{C.7}$$

$$-lg(\frac{1}{C_a} - 1) = Pw_pC_a + Nw_nC_i + \theta \tag{C.8}$$

$$-lg(\frac{1}{C_i} - 1) = (P-1)w_p1.0 + w_pC_i + \theta] \tag{C.9}$$

$$-lg(\frac{1}{C_i} - 1) = Pw_p1.0 + w_nC_a + \theta] \tag{C.10}$$

Finally, let $C_i = 1 - C_a$, $(C_a > C_i)$; this is reasonable as it yields a nice symmetric solution. Simply subtracting Equation C.10 from Equation C.9 results in $w_p = -w_n$. Hence, Lemma 1A is correct.[2]

□

Lemma 3B: *The maximum number of antecedents for which a network can consistently encode a conjunctive rule is given by* $K < \frac{C_a}{1-C_a}$.

Consider the following equations:

$$Pw_pC_a + Nw_nC_i + \theta > (P-1)w_p1.0 + w_pC_i + \theta \tag{C.11}$$

$$P\omega C_a - (K-P)\omega(1-C_a) + \theta > \omega(P-1) - \omega(1-C_a) + \theta \tag{C.12}$$

$$K < \frac{C_a}{1-C_a} \tag{C.13}$$

Inequality C.11 states the relationship between the right sides of equations Equations C.8 and C.9. This inequality simply states that the net incoming activation to a unit that is

[2]Note that the strict equality of w_p and $-w_n$ results for the assumption that Equations C.6 and C.7 both equal C_i. Relaxing this assumption to results in defining a range for w_n given a selection for w_p. Specifically: $-w_p - C_i \leq w_n \leq -w_p + C_i$. As $w_p = -w_n$ is at the center of this range, the results in the rest of this section will use these values.

active must be greater than the net incoming activation to a unit that is inactive. Substituting $1 - C_a$ for C_i (as in Lemma 1A), ω for w_p and $-\omega$ for w_n (the result of Lemma 1A) and $K - P$ for N yields Inequality C.12. This inequality reduces to Inequality C.13. Thus, Lemma 1B, which bounds the total number of antecedents admissible in a rule, is correct.
□

Proof of Theorem 3:

Lemma 3A proves that first part of Theorem 3.

One last step before completing Theorem 3; in addition to the work they have already done, the above equations can be used to determine values for the link weights. Specifically, substituting Lemma 3A that $w_p = -w_n$ into Equations C.8 and C.10, subtracting Equation C.10 from C.8 and solving for w_p yields Equation C.14. (The denominator of this equation re-expresses the bound defined by Lemma 3B.) This equation shows that the weight on links to encode a conjunctive rule is a function of both the amount a unit is allowed to differ from one (i.e., C_a) or zero and the number of antecedents in the rule.

$$w_p = \frac{lg(\frac{1}{C_i} - 1) - lg(\frac{1}{C_a} - 1)}{C_a + K(C_a - 1)} \tag{C.14}$$

At long last, the second part of Theorem 1 can be approached. This part of the theorem, which states that $\theta = -\frac{2P-1}{2}\omega$, is not algebraically correct. However, it is very close to correct. The following establishes bounds on the error for this method of setting for the bias.

Equation C.15 is Equation C.8 into which has been substituted Lemma 1A that $w_p = -w_n$. Solving Equation C.15 for θ (by substituting the setting for w in Equation C.14) yields Equation C.18. (Equations C.16 and C.17 show two steps of the simplification).

So, the formula for setting the bias given by part 2 of Theorem 1 is exactly correct when $1 = C_a + KC_i$. Unfortunately, this situation only occurs when $K = 1$. As rules rarely have only a single antecedent, the setting proposed in part 2 of this theorem is rarely exactly correct. However, substituting Lemma 1B that $K < \frac{C_a}{1-C_a}$ into $C_a + KC_i$ yields the result that $C_a + KC_i$ must always be in the range $[1\ldots2]$. Therefore, $\theta = -\frac{2P-1}{2}\omega$ is never far from correct.

$$\theta = -lg(\frac{1}{C_a} - 1) - PwC_a + NwC_i \tag{C.15}$$

$$\theta = -lg(\frac{1}{C_a} - 1) - [K(C_a - 1) + P]\frac{-2lg(\frac{1}{C_a} - 1)}{C_a + K(C_a - 1)} \tag{C.16}$$

$$\theta = \frac{[-lg(\frac{1}{C_a} - 1)][C_a + K(C_a - 1)] - [K(C_a - 1) + P] - 2lg(\frac{1}{C_a} - 1)}{C_a + K(C_a - 1)} \tag{C.17}$$

$$\theta = \frac{-2P + C_a + KC_i}{2}w \tag{C.18}$$

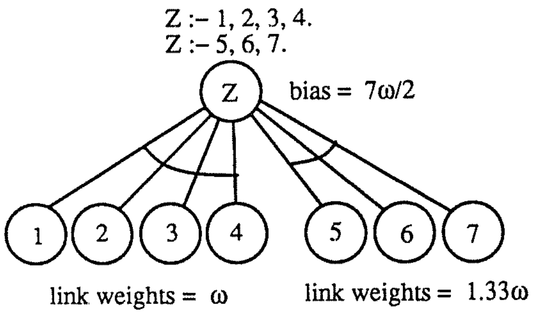


Figure C.1: The encoding of two disjunctive rules with multiple antecedents.

Moreover, while there will always be differences when using the simple function $\theta = -\frac{2P-1}{2}\omega$ to set the bias, these differences are too small to have an important effect on the resulting encoding of a conjunctive rule. Hence, $\theta = -\frac{2P-1}{2}\omega$ results in an approximately correct setting for the bias.

Therefore, given that $K < \frac{C_a}{1-C_a}$, setting $w_p = -w_n = \omega$ and $\theta = -\frac{2P-1}{2}\omega$ yields an encoding of conjunctive rules that is very close to correct.

□

Recall that the first step of the rules-to-network algorithm (Table 2.2) rewrites disjuncts so that all rules with the same consequent have exactly one antecedent. Figure C.1 illustrates the need for this rewriting. The figure shows two rules, one with four antecedents (named 1, 2, 3, and 4) and one with three antecedents (named 5, 6, and 7) all feeding unit Z. The bias on unit Z, and the links on to all of the antecedents are set so that Z will be active if either rule is true. However, Z will also be active when neither rule is true. For instance, if the units corresponding to antecedents 2, 3, 4, and 5 are all active, then Z will be active.

This example illustrates the need for rewriting disjunctive collections of rules to eliminate rules with more than one antecedent. The following theorem proves the necessity of this rewriting.

Theorem 4: *There is no way to correctly encode a set of disjunctive rules each of which has more than one antecedent into a neural network that uses a single unit to encode the consequent.*

Proof:

This proof will proceed by assuming that the theorem is false and showing a contradiction. The following terms are used:

v	a consequent of a set of disjunctive rules.
v_i	a rule for v
N_i	the number of antecedents in rule v_i
θ	the bias on the unit corresponding to v
ω_i^j	the weight on the link corresponding to the jth antecedent of rule v_i

The proof makes three nonrestrictive assumptions: (1) that there are no negative antecedents to rules (this assumption can easily be enforced through the addition of a unit that encodes *not*), (2) that there exists two rules v_1 and v_2 such that no combination of the antecedents of these two rules forms some other rule v_i and (3) that all units have an activation of either zero or one (this assumption is made only for simplicity).

The network encoding this set of rules must satisfy, for each rule i:

$$-\theta < \sum_{j=1}^{N_i} \omega_i^j \tag{C.19}$$

Assume the opposite of the theorem, namely that the only combination of link weights which exceed θ contain at least one of the rules v as a subset. Under these assumptions, the following inequalities are true:

$$-2\theta < \sum_{i=1}^{2}\sum_{j=1}^{N_i} \omega_i^j \tag{C.20}$$

$$-\theta > \omega_1^1 + \omega_2^1 \tag{C.21}$$

Equation C.20 simply states that the sum of two rules must be greater than twice $-\theta$ since each rule by itself must activate the unit. Equation C.21 says that the sum of the first antecedents of rules v_1 and v_2 is less than $-\theta$. This follows from the assumption that no unwanted combinations exceed θ. Subtracting $\omega_1^1+\omega_2^1$ from each side of Equation C.20 yields Equation C.22. Note that the term to the left of the inequality in Equation C.22 subtracts from -2θ an number that is less than $-\theta$. Hence, Equation C.23 follows directly from Equation C.22.

$$-2\theta - \omega_1^1 - \omega_2^1 < \sum_{i=1}^{2}\sum_{j=2}^{N_i} \omega_i^j \tag{C.22}$$

$$-\theta < -2\theta - \omega_1^1 - \omega_2^1 < \sum_{i=1}^{2}\sum_{j=2}^{N_i} \omega_i^j \tag{C.23}$$

But, Equation C.23 indicates that there is a combination of link weights of v_1 and v_2 that exceeds $-\theta$. But, by assumption (2) no rules are composed of the antecedents of v_1 and v_2. This contradiction proves that the theorem is true.
□

The following terms are used in the next theorem:

C_a	the minimum activation for a unit to be considered *active*
C_i	the maximum activation for a unit to be considered *inactive*
$\mathcal{F}(x)$	$1/(1+e^{-x})$, the standard logistic activation function
w_p	the weight on links corresponding to positive dependencies
θ	the bias
R	the number of rules encoding the disjunct
r	a number such that $1 \leq r \leq R$

Theorem 5: *Setting:*

1. $w_p = \omega$
2. $\theta = -\frac{\omega}{2}$

results in a unit that accurately encodes a disjunct under the conditions specified below.

Proof: The method for mapping disjunctive sets of rules into a KBANN-net is correct when Inequalities C.24 and C.25 are satisfied. These two equations simply state the condition that a unit should be active when one or more of its inputs are active. It only should be inactive when all of its inputs are inactive.

$$C_a \leq \mathcal{F}[rw_pA + (R-r)w_pI + \theta] \tag{C.24}$$

$$C_i \geq \mathcal{F}[Rw_pI + \theta] \tag{C.25}$$

As in Theorem 3, this proof will proceed by analysis of the extreme cases. By ensuring that Inequalities C.24 and C.25 are satisfied in the extreme cases, the non-extremes are guaranteed.

The extreme for Inequality C.24 occurs when one of the antecedents is true and carries the smallest signal indicative of truth (i.e., C_a) while all of the other antecedents are false and carry the smallest signal indicative of falsehood (i.e., 0). This extreme condition is given by Equation C.26. The parallel extreme condition for falsehood occurs when every antecedent is false and carries a signal of C_i. This extreme condition is given by Equation C.27.

Equations C.28 and C.29 simply restate Equations C.26 and C.27 in a more algebraically tractable form.

$$C_a = \mathcal{F}[w_pC_a + (R-1)w_p0.0 + \theta] \tag{C.26}$$

$$C_i = \mathcal{F}[Rw_pC_i + \theta] \tag{C.27}$$

$$-lg(\frac{1}{C_a} - 1) = w_p C_a + \theta \tag{C.28}$$

$$-lg(\frac{1}{C_i} - 1) = R w_p C_i + \theta \tag{C.29}$$

$$-2lg(\frac{1}{C_a} - 1) = w_p C_a - R w_p C_i \tag{C.30}$$

$$w_p = \frac{-2lg(\frac{1}{C_a} - 1)}{C_a - R(1 - C_a)} \tag{C.31}$$

Following Theorem 1, let $C_i = (1 - C_a)$. So, subtracting Equation C.29 from Equation C.28 yields Equation C.30 (note that $lg(\frac{1}{C_i} - 1) = -lg(\frac{1}{C_a} - 1)$). Equation C.31 results from solving Equation C.30 for w_p. There are two notes about Equation C.31. First, Theorem 3 has the same limitation as that imposed on Theorem 1 by Lemma 1B, as the Equation C.31 is only correct when $R < \frac{C_a}{1-C_a}$ (since $w_p > 0$). Second, this equation suggests setting $w_p \approx 4$ given the fairly-common conditions that $C_a = 0.9$ and $R = 4$. Hence, the empirically derived value of $\omega = 4$ is supported analytically.

Finally, substituting the value for w_p in Equation C.29 gives Equation C.32. Solving for θ yields Equation C.33 (following much the same process as was used to solve for θ in Theorem 1). Hence, when $C_a + RC_i = 1$ the setting proposed by Theorem 3 of $\theta = -0.5\omega$ is exactly correct. As in Theorem 1, the bound on R of $R < \frac{C_a}{C_i}$ restricts $C_a + RC_i$ to the range $[1 \ldots 2]$. Thus, the setting for the bias proposed in theorem 3 can never be far from correct.

$$-lg(\frac{1}{C_i} - 1) = RC_i \frac{-2lg(\frac{1}{C_a} - 1)}{C_a - R(1 - C_a)} + \theta \tag{C.32}$$

$$\frac{-C_a - RC_i}{2} w = \theta \tag{C.33}$$

Therefore, the scheme for mapping disjuncts into KBANN-nets results in networks that accurately encode disjunctive rules under a wide range of conditions.
□

In conclusion, the proofs of Theorems 3 and 5 show that the simple methods for translating rules into neural networks are close to correct. The proofs also exactly specify correct methods for translating rules into networks. In practice, it is unnecessary to translate with complete fidelity because training of the network is easily able to overcome errors resulting from inexact translation.

Appendix D

ADDITIONAL EXPERIMENTAL RESULTS

This appendix contains the results of tests not reported elsewhere in the thesis. In particular, these results complement the comparisons of the algorithms of Section 3.2. Testing methodology is not described here, it follows Section 3.2.

Tables D.1 and D.2 report statistical comparisons of the results of ten-fold cross-validation testing on each dataset. The splice-junction table repeats data appearing in Figure 3.8. However, a similar table could not be built for the promoter dataset because Figure 3.7 uses a method of summarizing the data that reduces eleven tests to a single result. Hence, the results for promoters in Table D.1 are for ten-fold cross-validation testing follows the procedures used in Section 3.2 for testing splice-junctions. Briefly, each datapoint represents the average over eleven cross-validation tests to average out the effects of example presentation order and testing versus training set splits. In addition, the results for standard ANNs and KBANN-nets are average over ten trials using different initial weight sets to smooth the differences in learning that result from small differences in the initial weights of the neural networks.

Table D.1, which reports tests for promoters, shows KBANN to be significantly superior to the six empirical learning systems with which it is compared. The algorithms are compared using the "relative information score" (RIS); an information theoretic measure of learning efficiency suggested by Kononenko and Bratko [Kononenko91]. Figure D.1 presents the leaving-one-out results that appear in Figure 3.7 alongside the results of ten-fold cross-validation.

Large discrepancies between the two measures point to systems which benefited significantly from the error-counting method used for the leaving-one-out study. For instance, the relative information score for Cobweb differs by more than 30 points between ten-fold cross-validation and leaving-one-out. Recall that examples are counted as incorrect in the leaving-one-out study only when the example was misclassified on a majority of the eleven runs. Hence, this method of error counting enhances the results of systems such as Cobweb which frequently make between one and five classification errors but rarely make more than six errors. The effect of the differences in error counting is quite apparent in Tables D.3 - D.5 which show the results for leaving-one-out and ten-fold cross-validation for all of the 106 examples in the promoter training set.

Tables D.3 - D.5 present the results of both ten-fold cross-validation and leaving-one-out for every promoter example. Hence, these tables allow a visual comparison of the results

Table D.1: Statistical analysis of learning for promoter recognition.

System Name	RIS Avg.	RIS Std. Dev.	*t* statistic Cbwb	ID3	Near Neigh.	Ptron	PEBLS	BP
KBANN	90.4	2.95	18.4	17.2	20.2	15.8	5.7	8.8
BackProp	82.3	3.62	16.0	13.8	10.4	5.2	1.1	—
PEBLS	80.8	3.84	12.8	13.3	7.9	3.6	—	—
Perceptron	75.4	2.38	12.9	10.7	6.3	—	—	—
Near.Neigh	65.4	4.64	6.3	4.5	—	—	—	—
ID3	53.7	6.04	1.3	—	—	—	—	—
Cobweb	51.1	6.25	—	—	—	—	—	—

The t statistic is for one-tailed paired-sample comparisons of the systems. When $t > 1.8$ then the system whose name is given by the row is superior with 95% confidence to the system whose name is given by the column.

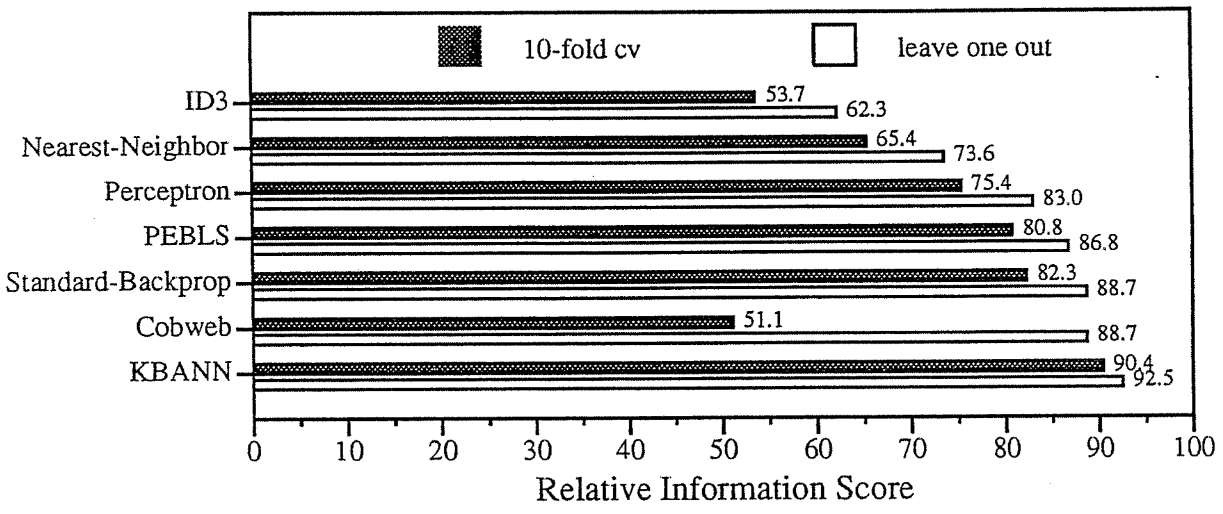


Figure D.1: Promoter recognition scores for both leaving-one-out and ten-fold cross-validation.

under two different testing methods. Numbers in bold in the ten-fold cross-validation columns indicate that the example was classified wrong in the majority of cases. The ten-fold cross-validation column for Perceptron reports decimals because Perceptron often could not make a clear decision. Frequently, it identified an example as being both a promoter and a non-promoter or, it identified en example as being neither. In both of these cases, the error count was incremented by the probability of randomly choosing the correct answer (i.e., 0.5).

An interesting comparison can be made between the results in Tables D.3 - D.5 and Figure D.2 which presents a cluster analysis of the 106 examples. The cluster analysis shows a tight central core of promoters that have names 'RR*'. The seven learning systems detailed in Table D.2 make virtually no errors on these examples. While the non-examples of promoters have no similar central cluster, the examples that are farthest away from the central promoter

Table D.2: Significance of differences in the splice-junction domain.

System Name	RIS Avg.	RIS Std. Dev.	*t* statistic Near Neig.	Cbwb	ID3	Ptron	PEBLS	BP
KBANN	90.2	0.31	74.6	14.8	27.2	22.2	7.2	2.2
BackProp	89.5	1.07	58.6	13.1	13.3	12.5	1.9	—
PEBLS	88.7	0.63	56.1	10.7	18.6	12.3	—	—
KBANN	88.0	0.76	60.5	9.0	13.2	9.5	—	—
Perceptron	83.9	0.84	52.2	3.4	0.2	—	—	—
ID3	83.9	0.78	43.0	2.8	—	—	—	—
Cobweb	81.9	1.86	20.5	—	—	—	—	—
Near.Neigh	69.3	0.82	—	—	—	—	—	—

The *t* statistic is for one-tailed paired-sample comparisons of the systems. When $t > 1.8$ then the system whose name is given by the row is superior with 95% confidence to the system whose name is given by the column.

cluster are consistently correctly classified. To be specific, the 20 examples of non-promoters that cluster farthest from any promoter have about half the average error rate for the full set of examples. The 20 promoters closest to the central promoter cluster have about $\frac{1}{5}$ of the average error rate. On the other hand, the 20 promoters and non-promoters nearest to boundary between promoters and non-promoters have about double the average error rate.

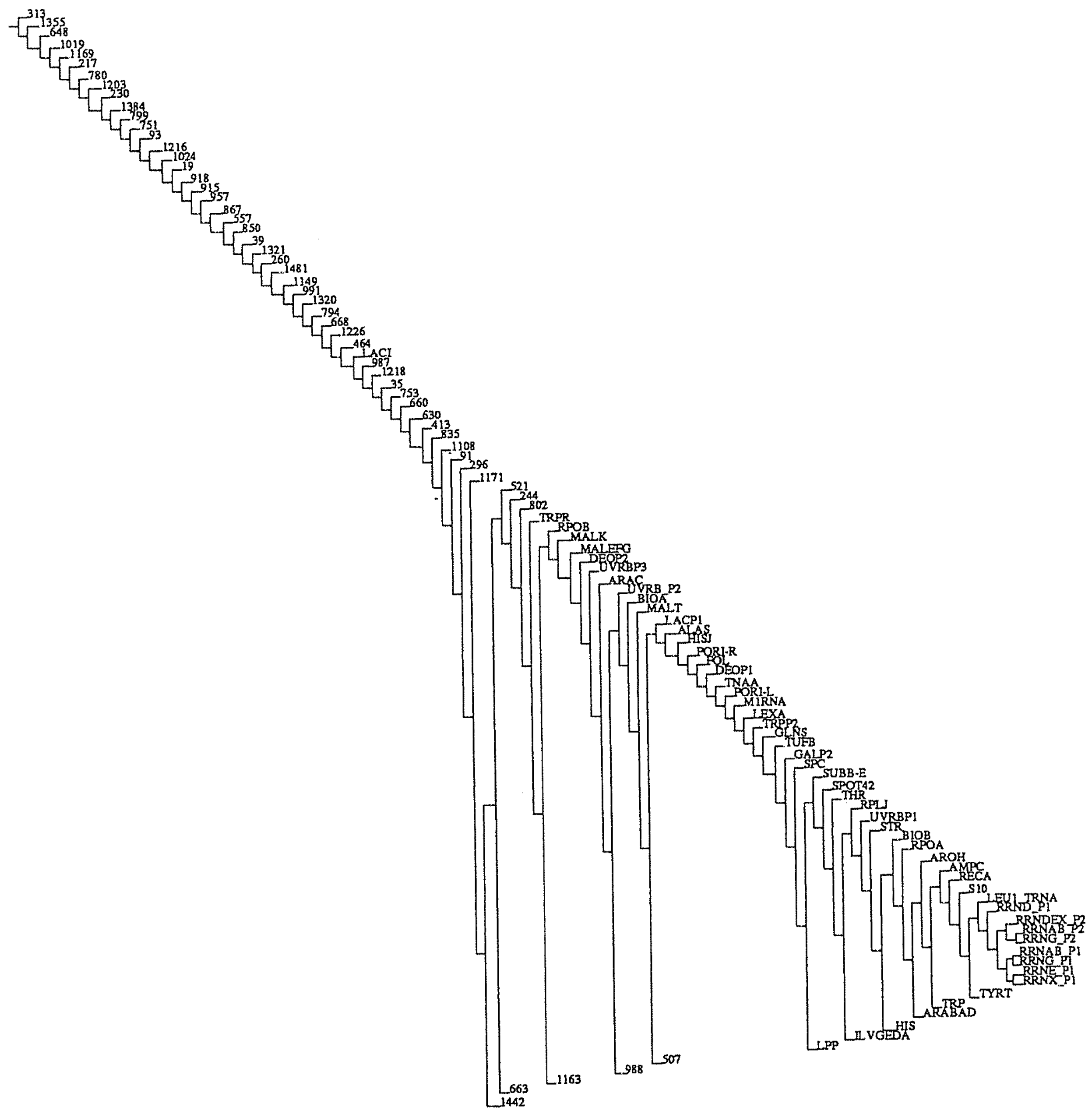


Figure D.2: Cluster analysis of the 106 example promoter set.

Table D.3: Results for every promoter example — part 1

	learning algorithms													
	KBANN		Standard ANN		PEBLS		Nearest Neighbor		ID3		Perceptron		Cobweb	
Example Name	L1	10fold	L1	10fold	L1	10fold	L1	10fold	L1	10fold	L1	10fold	L1	10fold
0019		0		0		0		0		3		0.0		3
0035		1		0		2		4		**8**		2.0		1
0039		2		0		0		0		4		0.5		0
0091		0		2	*	**11**	*	**11**	*	**7**		2.0		0
0093		0		0		0		4		0		0.0		0
0217		0		0		0		2		0		0.0		2
0230		0		0		0		4		4		0.5		0
0244		0		0		0	*	**11**	*	**6**		2.5		1
0260		0		2		0		**6**	*	5		2.0		0
0296	*	**8**		5		0		2		2	*	5.0		1
0313		0		0		0		0	*	1		0.0		0
0413		0		0		1		0		1		0.5		0
0464		0		5	*	**7**	*	**10**	*	**7**		**6.5**		2
0507		4	*	**11**	*	**6**	*	**11**	*	**11**	*	**9.0**	*	**11**
0521		0		0		0	*	**10**		**6**		3.5		2
0557		0		0		0		4		0		0.5		0
0630		0		0		0		0		2		1.0		1
0648		0		0		0		0		0		0.5		0
0660		0		0		0		3	*	0		0.0		1
0663		0		0		0		5		5		2.5		2
0668		2		3	*	**8**		1		3		3.5		1
0751		0		0		0	*	**7**	*	**6**		0.0		1
0753		0		0		2		0	*	3		0.5		1
0780		0		0		0		2	*	0		0.0		0
0794		0		0		0		1		0		1.5		2
0799		0		0		0		0		1		0.0		1
0802		0		3		4		0		4		5.0		0
0835		2		2		1	*	**11**		3		5.0		5
0850		0		0		0		0		5		0.5		2
0867		0		0		0		0		0		0.0		1
0915		0		0		0		0		0		0.0		1
0918		0		0		0		3		1		0.0		1
0957		0		0		0		4	*	**6**		1.0		1
0987		0		0		0		0	*	2		1.5		1
0988		0		3	*	**9**		**11**	*	**9**	*	5.0	*	**7**
0991		0		0		1	*	**11**		0		0.0		0

Notes: 'L1' indicates that the leaving-one-out methodology is used. '*' in a 'L1' column indicates an incorrect classification. Numbers in '10fold' columns indicate the number of incorrect classifications eleven trials using eleven orderings of the examples. Numeric names indicate examples that are non-promoters.

Table D.4: Results for every promoter example — part 2

Example Name	learning algorithms													
	KBANN		Standard ANN		PEBLS		Nearest Neighbor		ID3		Perceptron		Cobweb	
	L1	10fold	L1	10fold	L1	10fold	L1	10fold	L1	10fold	L1	10fold	L1	10fold
1019		0		1		0		0		4		4.0		0
1024		0		0		0		0		1		0.0		0
1108		0	*	**11**		3		3		1	*	**6.0**		5
1149		0		0		2		0		**9**		1.0		3
1163		0		0		2	*	**10**		3		2.5		2
1169		0		0		0		0		0		0.5		0
1171		0		0		0		0		3		0.0		1
1203		0		0		0		0		4		0.5		1
1216		0		0		0		5		1		2.0		1
1218		0		0		0		0	*	**6**		0.0		0
1226		0		0		0		2		0		0.0		0
1320		0		0		0		0		0		0.0		1
1321		0		0		0		0		5		0.0		0
1355		0		0		0		0		0		0.0		0
1384		0		0		0		0		0		0.0		0
1442		0		0		0	*	**11**		0		0.0		2
1481		0		0		0		1		1		0.0		1
alas		0		0		1		0		2		0.5		3
ampc		0		0		0		0		1		1.0		3
arabad		0		0		1		0		0		2.5		2
arac		0		0		0		0		**8**		0.0		1
aroh		0		0		0		0	*	0		0.0		0
bioa		0		0		3		0	*	2		1.0		0
biob		0		0		0		0		0		5.5		1
deop1		0		0		4		0		2		0.0		5
deop2	*	**8**		0		4		0		5		1.0		3
fol		0		0		0		0		3		2.0		3
galp2		0		0		0		0	*	1		0.0		4
glns		0		0		0		0		0		0.0		1
his		0		0		0		0		0		0.0		0
hisj		0		0		0		0		3		0.5		0
ilvgeda		0		0		0		0		1		0.0		0
laci	*	**11**	*	**11**	*	**9**	*	**8**		**11**		3.5	*	**9**
lacp1		0		5		2		1		3	*	3.5	*	**7**
lev1-trna		0		0		0		0	*	**9**		0.0		0
lexa		0		0		0		0		2		0.5		3

Notes: 'L1' indicates that the leaving-one-out methodology is used. '*' in a 'L1' column indicates an incorrect classification. Numbers in '10fold' columns indicate the number of incorrect classifications eleven trials using eleven orderings of the examples. Numeric names indicate examples that are non-promoters.

Table D.5: Results for every promoter example — part 3

Example Name	learning algorithms: KBANN L1	KBANN 10fold	Standard ANN L1	Standard ANN 10fold	PEBLS L1	PEBLS 10fold	Nearest Neighbor L1	Nearest Neighbor 10fold	ID3 L1	ID3 10fold	Perceptron L1	Perceptron 10fold	Cobweb L1	Cobweb 10fold
lpp		0		0		0		0		1		0.0		1
m1rna		0		0		0		0		1		2.0		3
malefg		0		0		0		0		**8**		0.0		6
malk		5		0		4		0		**8**		1.5		2
malt		2	*	**10**		0		0		2	*	**7.0**		4
pori-l		0		0		1		0		0		0.0		2
pori-r		0		1		2		0		**9**	*	5.0		5
reca		0		0		0		0		0		0.0		0
rplj		0		0		0		0		1		0.0		2
rpoa		0		0		0		0		0		0.0		0
rpob		0	*	**11**		0		0		**11**	*	**7.0**	*	**7**
rrnab-p1		0		0		0		0		0		0.0		0
rrnab-p2		0		0		0		0		0		0.0		0
rrnd-p1		0		0		0		0		0		0.0		0
rrndex-p2		0		0		0		0		0		0.0		0
rrne-p1		0		0		0		0		0		0.0		0
rrng-p1		0		0		0		0		0		0.0		0
rrng-p2		0		0		0		0		1		0.0		0
rrnx-p1		0		0		0		0		0		0.5		0
s10		0		0		0		0		0		2.5		1
spc		0		0		0		0		**7**		0.0		1
spot42		0		0		0		0		1		0.0		0
str		0		0		0		0		0		0.0		0
subb-e		0		0		0		0		0		0.0		0
thr		0		0		0		0		0		0.0		0
tnaa		2		0		0		0		2		0.0		5
trp		0		0		0		0		0		0.0		0
trpp2		0		0		0		3		0		0.0		1
trpr	*	**9**	*	**11**		2	*	**9**		0	*	**10.0**	*	**10**
tufb		0		0		0		0		1		1.5		2
tyrt		0		0		0		0		**7**		0.0		0
uvrbp1		0		0		0		0		1		0.0		0
uvrbp2		0		4		1	*	**11**	*	0		0.5		5
uvrbp3		0		2	*	**7**		0		3		**6.0**		4
totals	4	5.1	6	9.4	7	9.1	14	18.4	19	24.5	9	13.0	6	15.9

Notes: 'L1' indicates that the leaving-one-out methodology is used. '*' in a 'L1' column indicates an incorrect classification. Numbers in '10fold' columns indicate the number of incorrect classifications eleven trials using eleven orderings of the examples. Numeric names indicate examples that are non-promoters.

Appendix E

BASE MODEL FOR GEOMETRY LEARNING

Tables E.1 - E.5 present the complete set of rules used by the KBANN-based model of geometry learning to simulate the knowledge of children prior to formal instruction in geometry. That is, these rule reproduce the geometric reasoning of children who have not had formal instruction in geometry.

Table E.6 defines the features used to describe each of the geometric figures used during the training and testing of the model.

Table E.1: Shape-naming rules — part I.

name(triangle, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,medium), pointy(?obj,somewhat), point-direction(?obj,up), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(triangle, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,medium), pointy(?obj,very), point-direction(?obj,up), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(triangle, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,little), pointy(?obj,somewhat), point-direction(?obj,right), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(triangle, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,big), pointy(?obj,somewhat), point-direction(?obj,up), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(triangle, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,medium), pointy(?obj,very), point-direction(?obj,down), shape(?obj,skinny), 2-long-and-2-short-sides(?obj,no).
name(triangle, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,little), pointy(?obj,somewhat), point-direction(?obj,down), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(pentagon, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,medium), pointy(?obj,not), point-direction(?obj,none), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(pentagon, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,large), pointy(?obj,not), point-direction(?obj,none), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(hexagon, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,medium), pointy(?obj,not), point-direction(?obj,none), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(hexagon, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,large), pointy(?obj,not), point-direction(?obj,none), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(quadrilateral, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,no), area(?obj,big), pointy(?obj,not), point-direction(?obj,none), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(quadrilateral, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,no), area(?obj,medium), pointy(?obj,not), point-direction(?obj,none), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).

Table E.2: **Shape-naming rules — part II.**

name(square, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,no), area(?obj,medium), pointy(?obj,not), point-direction(?obj,none), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(square, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,no), area(?obj,big), pointy(?obj,not), point-direction(?obj,none), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(rectangle, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,no), area(?obj,medium), pointy(?obj,not), point-direction(?obj,none), shape(?obj,medium), 2-long-and-2-short-sides(?obj,yes).
name(rectangle, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,no), area(?obj,medium), pointy(?obj,not), point-direction(?obj,none), shape(?obj,skinny), 2-long-and-2-short-sides(?obj,yes).
name(rectangle, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,no), area(?obj,medium), pointy(?obj,not), point-direction(?obj,none), shape(?obj,fat), 2-long-and-2-short-sides(?obj,yes).
name(rhombus, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,medium), pointy(?obj,somewhat), point-direction(?obj,right), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(rhombus, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,little), pointy(?obj,somewhat), point-direction(?obj,up), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(parallelogram, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,medium), pointy(?obj,somewhat), point-direction(?obj,left), shape(?obj,medium), 2-long-and-2-short-sides(?obj,yes).
name(parallelogram, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,big), pointy(?obj,somewhat), point-direction(?obj,right), shape(?obj,fat), 2-long-and-2-short-sides(?obj,yes).
name(parallelogram, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,little), pointy(?obj,very), point-direction(?obj,left), shape(?obj,skinny), 2-long-and-2-short-sides(?obj,yes).
name(trapezoid, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,medium), pointy(?obj,very), point-direction(?obj,right), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(trapezoid, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,medium), pointy(?obj,somewhat), point-direction(?obj,up), shape(?obj,medium), 2-long-and-2-short-sides(?obj,no).
name(trapezoid, ?obj)	:-	tilted(?obj,0), slanty-lines(?obj,yes), area(?obj,big), pointy(?obj,somewhat), point-direction(?obj,left), shape(?obj,fat), 2-long-and-2-short-sides(?obj,no).

Table E.3: Feature-counting rules.

same-name(?obj1, ?obj2)	:-	name(?nme, ?obj1), name(?nme, ?obj2).
same-tilted(?obj1, ?obj2)	:-	tilted(?obj1, ?tlt), tilted(?obj2, ?tlt).
same-slanty-lines(?obj1, ?obj2)	:-	slanty-lines(?obj1, ?slant), slanty-lines(?obj2, ?slany).
same-area(?obj1, ?obj2)	:-	area(?obj1, ?area), (?obj2, ?area).
same-pointy(?obj1, ?obj2)	:-	pointy(?obj1, ?pty), pointy(?obj2, ?pty).
same-point-direction(?obj1, ?obj2)	:-	point-direction(?obj1, ?pd), point-direction(?obj2, ?pd)
same-shape(?obj1, ?obj2)	:-	shape(?obj1, ?shp), shape(?obj2, ?shp).
same-2-2(?obj1, ?obj2)	:-	2-long-and-2-short-sides(?obj1, ?two), 2-long-and-2-short-sides(?obj2, ?two).

Table E.4: Similarity-recognition rules.

Let *8-Antecedents* represent the following set of eight antecedents:
same-2-2(?obj1, ?obj2), same-shape(?obj1, ?obj2),
same-point-direction(?obj1,mkvobj2), same-pointy(?obj1, ?obj2),
same-area(?obj1, ?obj2), same-slanty-lines(?obj1, ?obj2),
same-tilted(?obj1, ?obj2), same-name(?obj1, ?obj2)

similarity(?obj1, ?obj2,very)	:-	n-true-antecedents(7, *8-Antecedents*).
similarity(?obj1, ?obj2,quite)	:-	n-true-antecedents(6, *8-Antecedents*).
similarity(?obj1, ?obj2,mostly)	:-	n-true-antecedents(5, *8-Antecedents*).
similarity(?obj1, ?obj2,fairly)	:-	n-true-antecedents(4, *8-Antecedents*).
similarity(?obj1, ?obj2,sort-of)	:-	n-true-antecedents(3, *8-Antecedents*).
similarity(?obj1, ?obj2,not-very)	:-	n-true-antecedents(2, *8-Antecedents*).

Table E.5: Combining rules.

most-similar-pair(?obj1, ?obj2)	:-	similarity(?obj1, ?obj2,very), not similarity(?obj1, ?obj3,very), not similarity(?obj3, ?obj2,very).
most-similar-pair(?obj1, ?obj2)	:-	similarity(?obj1, ?obj2,quite), not similarity(?obj1, ?obj3,quite), not similarity(?obj3, ?obj2,quite).
most-similar-pair(?obj1, ?obj2)	:-	similarity(?obj1, ?obj2,mostly), not similarity(?obj1, ?obj3,mostly), not similarity(?obj3, ?obj2,mostly).
most-similar-pair(?obj1, ?obj2)	:-	similarity(?obj1, ?obj2,fairly), not similarity(?obj1, ?obj3,fairly), not similarity(?obj3, ?obj2,fairly).
most-similar-pair(?obj1, ?obj2)	:-	similarity(?obj1, ?obj2,sort-of), not similarity(?obj1, ?obj3,sort-of), not similarity(?obj3, ?obj2,sort-of).
most-similar-pair(?obj1, ?obj2)	:-	similarity(?obj1, ?obj2,not-very), not similarity(?obj1, ?obj3,not-very), not similarity(?obj3, ?obj2,not-very).

Table E.6: Features and their possible values.

Feature Name	*Possible Values*
Visual Features	
Tilted	{0 10 20 30 40}
Slanty	{Yes No}
Area	{Little Medium Big}
Shape	{Skinny Fat Medium}
Pointy	{Yes No}
Point Direction	{None Up Down Right Left}
2 long and 2 short sides	{Yes No}
Symbolic Features	
Convex	{Yes No}
Number of Sides	{3 4 5 6 8}
Number of Angles	{3 4 5 6 8}
Number of Right Angles	{0 1 2 3 4}
Number of Pairs of Parallel Sides	{0 1 2 3 4}
Number of Pairs of Equal Opposite Angles	{0 1 2 3 4}
Adajacent Angles Sum to 180	{Yes No}
Number of Pairs of Opposite Sides Equal	{0 1 2 3 4}
Number of Lines of Symmetry	{0 1 2 3 4 5 6 8}
All Sides Equal	{Yes No}
All Angles Equal	{Yes No}
Number of Equal Sides	{0 2 3 4 5 6 8}
Number of Equal Angles	{0 2 3 4 5 6 8}

Bibliography

[Aha91] Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66.

[Ahmad88] Ahmad, S. (1988). A study of scaling and generalization in neural networks. Technical Report CCSR-88-13, University of Illinois, Center for Complex Systems Research.

[Alberts88] Alberts, B. M. (1988). *Mapping and Sequencing the Human Genome.* National Academy Press, Washington, D.C.

[Atlas89] Atlas, L., Cole, R., Connor, J., El-Sharkawi, M., Marks II, R. J., Muthusamy, Y., and Barnard, E. (1989). Performance comparisons between backpropagation networks and classification trees on three real-world applications. In *Advances in Neural Information Processing Systems*, volume 2, pages 622–629, Denver, CO. Morgan Kaufmann.

[Barnard89] Barnard, E. and Cole, R. A. (1989). A neural-net training program based on conjugate-gradient optimization. Technical Report CSE 89-014, Oregon Graduate Institute, Beaverton, OR.

[Bennett88] Bennett, S. (1988). Real world EBL: Learning error tolerant plans in the robotics domain. In *Proceedings of the AAAI Explanation-Based Learning Symposium*, pages 122–126, Stanford, CA.

[Berenji91] Berenji, H. R. (1991). Refinement of approximate reasoning-based controllers by reinforcement learning. In *Proceedings of the Eighth International Machine Learning Workshop*, pages 475–479, Chicago, IL.

[Blum88] Blum, A. and Rivest, R. L. (1988). Training a 3-node neural network is NP-complete. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 9–18, Cambridge, MA.

[Breiman84] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees.* Wadsworth, Belmont, CA.

[Brunak91] Brunak, S., Engelbrecht, J., and Knudsen, S. (1991). Prediction of human mRNA donor and acceptor sites from the DNA sequence. Available in the *neuroprose* archive at archive.cis.ohio-state.edu as brunak.netgene.ps.Z.

[Brunner56] Brunner, J. S., Goodnow, J. J., and Austin, G. A. (1956). *A Study of Thinking.* Wiley, New York.

[Buntine91] Buntine, W. L. and Weigand, A. S. (1991). Bayesian back-propagation.

[Chauvin88] Chauvin, Y. (1988). A back-propagation algorithm with optimal use of hidden units. In *Advances in Neural Information Processing Systems*, volume 1, pages 519–525, Denver, CO. Morgan Kaufmann.

[Cost90] Cost, S. and Salzberg, S. (1990). A weighted nearest neighbor algorith for learning with symbolic features. Technical Report JHU-90/11, Johns Hopkins, Baltimore, MD.

[Craven90] Craven, M. W. (1990). Experiments in optimal brain damage. Unpublished manuscript.

[Craven92] Craven, M. W. and Shavlik, J. W. (1992). Visualizing learning and computation in artificial neural networks. *International Journal on Artificial Intelligence Tools*, 1(2). (Forthcoming).

[Danyluk89] Danyluk, A. P. (1989). Finding new rules for incomplete theories: Explicit biases for induction with contextual information. In *Proceedings of the Sixth International Machine Learning Workshop*, pages 34–36, Ithaca, NY.

[DeJong86] DeJong, G. F. and Mooney, R. F. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1:145–176.

[Dennis91] Dennis, S. and Phillips, S. (1991). Analysis tools for neural networks.

[Diederich88] Diederich, J. (1988). Knowledge-intensive recruitment learning. Technical Report ICSI-TR-88-010, International Computer Science Institute, Berkeley, CA.

[Dietterich86] Dietterich, T. G. (1986). Learning at the knowledge level. *Machine Learning*, 1:287–316.

[Dietterich90] Dietterich, T. G., Hild, H., and Bakiri, G. (1990). A comparative study of ID3 and backpropagation for English text-to-speech mapping. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 24–31, Austin, TX.

[Duda79] Duda, R., Gaschnig, J., and Hart, P. (1979). Model design in the prospector consultant system. In Michie, D., editor, *Expert Systems in the Microelectronic Age*. Edinburgh University Press, Edinburgh, Scotland.

[Fahlman88] Fahlman, S. E. (1988). Faster learning variations on back-propagation: An empirical study. In Touretsky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51. Morgan Kaufmann, San Mateo, CA.

[Fahlman89] Fahlman, S. E. and Lebiere, C. (1989). The cascade-correlation learning architecture. In *Advances in Neural Information Processions Systems*, volume 2, pages 524–532, Denver, CO. Morgan Kaufmann.

[Fisher87] Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172.

[Fisher89] Fisher, D. H. and McKusick, K. B. (1989). An empirical comparison of ID3 and back-propagation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 788–793, Detroit, MI.

[Flann89] Flann, N. S. and Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187–226.

[Fozzard88] Fozzard, R., Bradshaw, G., and Ceci, L. (1988). A connectionist expert system that actually works. In *Advances in Neural Information Processing Systems*, volume 1, pages 248–255, Denver, CO. Morgan Kaufmann.

[Franzini87] Franzini, M. A. (1987). Speech recognition with back propagation. In *IEEE Ninth Annual Conference of the Engineering in Medicine and Biology Society*, pages 1702–1703.

[Fu89] Fu, L. M. (1989). Integration of neural heuristics into knowledge-based inference. *Connection Science*, 1:325–340.

[Fu91] Fu, L. M. (1991). Rule learning by searching on adapted nets. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 590–595, Anaheim, CA.

[Fuys88] Fuys, D., Geddes, D., and Tischler, R. (1988). The van Hiele model of thinking in geometry among adolescents. In *Journal for Research in Mathematics Education, Monograph No. 3.* National Council of Teachers of Mathematics, Reston, VA.

[Gallant88] Gallant, S. I. (1988). Connectionist expert systems. *Communications of the ACM*, 31:152–169.

[Gennari90] Gennari, J. (1990). *An Experimental Study of Concept Formation.* PhD thesis, University of California, Irvine, CA. Avaliable as Technical Report 90-26.

[Goodman83] Goodman, N. (1983). *Fact, Fiction and Forecast.* Harvard University Press, Cambridge, MA.

[Hall88] Hall, R. J. (1988). Learning by failing to explain: Using partial explanations to learn in incomplete or intractable domains. *Machine Learning*, 3:45–77.

[Hanson90] Hanson, S. J. and Burr, D. J. (1990). What connectionist models learn: Learning and representation in connectionist networks. *Behavioral and Brain Sciences*, 13:471–518.

[Hanson88] Hanson, S. J. and Pratt, L. Y. (1988). Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, volume 1, pages 177–185, Denver, CO. Morgan Kaufmann.

[Harley87] Harley, C. B. and Reynolds, R. P. (1987). Analysis of *E. coli* promoter sequences. *Nucleic Acids Research*, 15:2343–2361.

[Hartigan75] Hartigan, J. A. (1975). *Clustering Algorithms.* Wiley, New York.

[Hawley83] Hawley, D. K. and McClure, W. R. (1983). Compilation and analysis of *Escherichia Coli* promoter DNA sequences. *Nucleic Acids Research*, 11:2237–2255.

[Hayashi90] Hayashi, Y. (1990). A neural expert system with automated extraction of fuzzy if-then rules. In *Advances in Neural Information Processing Systems*, volume 3, pages 578–584, Denver, CO. Morgan Kaufmann.

[Hebb49] Hebb, D. O. (1949). *The Organization of Behavior.* Wiley, New York.

[Hinton89] Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40:185–234.

[Hinton91] Hinton, G. E. (1991). Personal communication.

[Hinton86] Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In Rumelhart, D. and McClelland, J., editors, *Parallel Distributed Processing: Explorations in the microstructure of cognition; Vol. 1: Foundations*, pages 77–109. MIT Press, Cambridge, MA.

[Hirsh89] Hirsh, H. (1989). Combining empirical and analytical learning with version spaces. In *Proceedings of the Sixth International Machine Learning Workshop*, pages 29–33, Ithaca, NY.

[Hoehfeld91] Hoehfeld, M. and Fahlman, S. E. (1991). Learning with limited numerical precision using the cascade-correlation architecture. Technical Report CMU-CS-91-130, Carnegie-Mellon, Pittsburgh, PA.

[Holland86a] Holland, J. H. (1986a). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An AI Approach*, volume 2, pages 593–624. Morgan Kaufmann, San Mateo, CA.

[Holland86b] Holland, J. H., Holyoak, R. J., Nisbitt, R. E., and Thagard, P. (1986b). *Induction: Processes of Inference, Learning and Discovery.* MIT Press, Cambridge, MA.

[Holley89] Holley, L. and Karplus, M. (1989). Protein secondary structure prediction with a neural network. *Proceedings of the National Academy of Science*, 56:152–156.

[Hollis90] Hollis, P. W., Harper, J. S., and Paulos, J. J. (1990). The effects of precision constraints in a backpropagation learning network. *Neural Computation*, 2:363–373.

[Holte89] Holte, R. C., Acker, L. E., and Porter, B. W. (1989). Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 813–819, Detroit, MI.

[Honavar88] Honavar, V. and Uhr, L. (1988). A network of neuron-like units that learns to perceive by generation as well as reweighting of its links. In Hinton, G. E., Sejnowski, T. J., and Touretzky, D. S., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 472–484. Morgan Kaufmann, San Mateo, CA.

[Hunter91] Hunter, L. (1991). Classifying for prediction: A multistrategy approach to predicting protein secondary structure. In *Proceedings of the First International Workshop on Multistrategy Learning*, pages 394–402, Harpers Ferry, WV.

[IUB Nomenclature Committee85] IUB Nomenclature Committee (1985). Ambiguity codes. *European Journal of Biochemistry*, 150:1–5.

[Jones89] Jones, M. A. and Story, G. A. (1989). Inheritance reasoning in connectionist networks. In *International Conference on Neural Networks.*

[Judd88] Judd, S. (1988). On the complexity of loading shallow neural networks. *Journal of Complexity*, 4:177–192. Also appears in *Readings in Machine Learning.*

[Katz89] Katz, B. F. (1989). EBL and SBL: A neural network synthesis. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pages 683–689, Ann Arbor, MI.

[Kleene56] Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In Shannon, C. E. and McCarthy, J., editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ.

[Koedinger90] Koedinger, K. R. and Anderson, J. R. (1990). Theoretical and empirical motivations for the design of ANGLE: A new geometry learning environment. In *Proceedings of the AAAI Symposium on Knowledge-Based Environments for Learning and Teaching*, Stanford, CA.

[Kolen90] Kolen, J. F. and Pollack, J. B. (1990). Back-propagation is sensitive to initial conditions. In *Advances in Neural Information Processing Systems*, volume 3, Denver, CO. Morgan Kaufmann.

[Kononenko91] Kononenko, I. and Bratko, I. (1991). Information-based evaluation criterion for classifier's performance. *Machine Learning*, 6:67–80.

[Koton89] Koton, P. (1989). Evaluating case-based problem solving. In *Proceedings of the Case-Based Reasoning Workshop*, pages 173–175, Pensacola, FL.

[Koudelka87] Koudelka, G. B., Harrison, S. C., and Ptashne, M. (1987). Effect of noncontacted bases on the affinity of 434 operator for 434 repressor and Cro. *Nature*, 326:886–888.

[Kruschke88] Kruschke, J. K. (1988). Creating local and distributed bottlenecks in hidden layers of back-propagation networks. In Hinton, G. E., Sejnowski, T. J., and Touretzky, D. S., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 357–370. Morgan Kaufmann, San Mateo, CA.

[Kruschke91] Kruschke, J. K. and Movellan, J. R. (1991). Benefits of gain: Speeded neural learning and minimal hidden layers in back-propagation networks. *IEEE Transactions on Systems, Man and Cybernetics*, 21(1).

[Langley89] Langley, P. (1989). Editorial: Toward a unified science of machine learning. *Machine Learning*, 3:253–259.

[Lapedes89] Lapedes, A., Barnes, C., Burkes, C., Farber, R., and Sirotkin, K. (1989). Application of neural networks and other machine learning algorithms to DNA sequence analysis. In *Computers and DNA, SFI Studies in the Science of Complexity VII*. Addison-Wesley, Reading, MA.

[Le Cun89] Le Cun, Y., Denker, J. S., and Solla, S. A. (1989). Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2, pages 598–605, Denver, CO. Morgan Kaufmann.

[Lebowitz80] Lebowitz, M. (1980). *Generalization and Memory in an Integrated Understanding System*. PhD thesis, Department of Computer Science, Yale University.

[Lebowitz86] Lebowitz, M. (1986). Integrated learning: Controlling explanation. *Cognitive Science*, 10:219–240.

[Lehrer89] Lehrer, R., Knight, W., Love, M., and Sancilio, L. (1989). Software to link action and description in pre-proof geometry. Presented at the Annual Meeting of the American Educational Research Association.

[Litzkow88] Litzkow, M., Livny, M., and Mutka, M. W. (1988). Condor — a hunter of idle workstations. In *Proceedings of the Eighth International Conference on Distributed Computing Systems*.

[Lukashin89] Lukashin, A. V., Anshelevich, V. V., Amirikyan, B. R., Gragerov, A. I., and Frank-Kamenetskii, M. D. (1989). Neural network models of promoter recognition. *Journal of Biomolecular Structure and Dynamics*, 6:1123–1133.

[Maclin91] Maclin, R. and Shavlik, J. W. (1991). Refining domain theories expressed as finite-state automata. In *Proceedings of the Eighth International Machine Learning Workshop*, pages 524–528, Chicago, IL.

[Masuoka90] Masuoka, R., Watanabe, N., Kawamura, A., Owada, Y., and Asakawa, K. (1990). Neurofuzzy system — fuzzy inference using a structured neural network. In *Proceedings of the International Conference on Fuzzy Logic & Neural Networks*, pages 173–177.

[Matheus90] Matheus, C. J. (1990). *Feature Construction: An Analytic Framework and an Application to Decision Trees*. PhD thesis, University of Illinois at Urbana-Champaign.

[McCulloch43] McCulloch, W. S. and Pitts, W. A. (1943). A logical calculus of ideas immanent in nervous activity. *Bulliten of Mathematical Biophysics*, 5:115–133.

[McDermott82] McDermott, J. (1982). R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19.

[McMillan91] McMillan, C., Mozer, M. C., and Smolensky, P. (1991). The connectionist scientist game: Rule extraction and refinement in a neural network. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, Chicago, IL.

[Michalski83] Michalski, R. S. (1983). A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161.

[Miller56] Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97.

[Minsky63] Minsky, M. (1963). Steps towards artificial intelligence. In Feigenbaum, E. A. and Feldman, J., editors, *Computers and Thought.* McGraw-Hill, New York.

[Minsky88] Minsky, M. L. and Papert, S. (1988). *Perceptrons: Expanded Edition.* MIT Press, Cambridge, MA. Original edition published in 1969.

[Mitchell91] Mitchell, T. (1991). Learning to schedule appointments. Personal communication.

[Mitchell82] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18:203–226.

[Mitchell86] Mitchell, T. M., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80.

[Moody88] Moody, J. and Darken, C. (1988). Learning with localized receptive fields. In Hinton, G. E., Sejnowski, T. J., and Touretzky, D. S., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 133–143. Morgan Kaufmann, San Mateo, CA.

[Mooney89] Mooney, R. J. and Ourston, D. (1989). Induction over the unexplained: Integrated learning of concepts with both explainable and conventional aspects. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 5–8, Ithaca, NY.

[Mooney91a] Mooney, R. J. and Ourston, D. (1991a). Personal communication.

[Mooney91b] Mooney, R. J. and Ourston, D. (1991b). Constructive induction in theory refinement. In *Proceedings of the Eighth International Machine Learning Workshop*, pages 178–182, Evanston, IL.

[Mozer88] Mozer, M. C. and Smolensky, P. (1988). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in Neural Information Processing Systems*, volume 1, pages 107–115, Denver, CO. Morgan Kaufmann.

[Murphy85] Murphy, G. L. and Medin, D. L. (1985). The role of theories in conceptual coherence. *Psychological Review*, 91:289–316.

[Murphy91] Murphy, P. M. and Pazzani, M. J. (1991). ID2-of-3: Constructive induction of N-of-M concepts for discriminators in decision trees. In *Proceedings of the Eighth International Machine Learning Workshop*, pages 183–187, Evanston, IL.

[Nessier62] Nessier, U. and Weene, P. (1962). Hierarchies in concept attainment. *Journal of Experimental Psychology*, 64:640–645.

[Ng90] Ng, K. and Lippmann, R. P. (1990). A comparative study of the practical characteristics of neural networks and conventional pattern classifiers. In *Advances in Neural Information Processing Systems*, volume 3, pages 970–976, Denver, CO. Morgan Kaufmann.

[Noordewier91] Noordewier, M. O., Towell, G. G., and Shavlik, J. W. (1991). Training knowledge-based neural networks to recognize genes in DNA sequences. In *Advances in Neural Information Processing Systems*, volume 3, Denver, CO. Morgan Kaufmann.

[Nowlan91] Nowlan, S. J. and Hinton, G. E. (1991). Simplifying neural networks by soft weight-sharing. In *Advances in Neural Information Processing Systems*, volume 4, Denver, CO. Morgan Kaufmann.

[Oliver88] Oliver, W. L. and Schneider, W. (1988). Using rules and task division to augment connectionist learning. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, pages 55–61, Montreal, Canada.

[O'Neill89] O'Neill, M. C. (1989). *Escherichia coli* promoters: I. Consensus as it relates to spacing class, specificity, repeat substructure, and three dimensional organzation. *Journal of Biological Chemistry*, 264:5522–5530.

[O'Neill89] O'Neill, M. C. and Chiafari, F. (1989). *Eserichia Coli* promoters II: A spacing-class dependent promoter search protocol. *Journal of Biological Chemistry*, 264:5531–5534.

[O'Rorke82] O'Rorke, P. (1982). A comparative study of inductive learning systems AQ15 and ID3 using a chess endgame test problem. Technical Report UIUCDCS-F-82-899, University of Illinois, Department of Computer Science, Urbana, IL.

[Ourston90] Ourston, D. and Mooney, R. J. (1990). Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 815–820, Boston, MA.

[Ourston91] Ourston, D. and Mooney, R. J. (1991). Improving shared rules in multiple category domain theories. In *Proceedings of the Eighth International Machine Learning Workshop*, pages 534–538, Evanston, IL.

[Pazzani88] Pazzani, M. J. (1988). *Learning Causal Relationships: An Integration of Empirical and Explanation-Based Learning Methods.* PhD thesis, Computer Science Department, University of California, Los Angles.

[Pazzani89] Pazzani, M. J. and D, S. (1989). The influence of prior theories on the ease of concept acquisition. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pages 844–851, Ann Arbor, MI.

[Pineda87] Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Physics Review Letters*, 59:2229–2232.

[Pratt91] Pratt, L. Y. and Kamm, C. A. (1991). Direct transfer of learned information among neural networks. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 584–589, Anaheim, CA.

[Qian88] Qian, N. and Sejnowski, T. J. (1988). Predicting the secondary structure of globular proteins unsing neural networks models. *Journal of Molecular Biology*, 202:856–884.

[Quinlan86] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

[Quinlan91] Quinlan, J. R. (1991). Improved estimates for the accuracy of small disjuncts. *Machine Learning*, 6:93–98.

[Rajamoney87] Rajamoney, S. A. and DeJong, G. F. (1987). The classification, detection and handling of imperfect theory problems. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 205–207, Milan, Italy.

[Record89] Record, T. (1989). Personal communication.

[Redmond89] Redmond, M. (1989). Combining case-based reasoning, explanation-based learning, and learning from instruction. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 20–22, Ithaca, NY.

[Rendell90] Rendell, L. and Cho, H. (1990). Empirical learning as a function of concept character. *Machine Learing*, 5:267–298.

[Rendell89] Rendell, L. A., Cho, H. H., and Seshu, R. (1989). Improving the design of similarity-based rule-learning systems. *International Journal of Expert Systems*, 2:97–133.

[Rosenblatt62] Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan, New York.

[Rueckl88] Rueckl, J. G., Cave, K. R., and Kosslyn, S. M. (1988). Why are "what" and "where" processed by separate cortical visual systems? A computational investigation. *Journal of Cognitive Neuroscience*, 1(2).

[Rumelhart91] Rumelhart, D. (1991). Personal communication.

[Rumelhart86] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the microstructure of cognition. Volume 1: Foundations*, pages 318–363. MIT Press, Cambridge, MA.

[Saito88] Saito, K. and Nakano, R. (1988). Medical diagnostic expert system based on PDP model. In *Proceedings of IEEE International Conference on Neural Networks*, volume 1, pages 255–262.

[Schank86] Schank, R., Collins, G. C., and Hunter, L. E. (1986). Transcending inductive category formation in learning. *Behavioral and Brain Sciences*, 9:639–686.

[Scott91] Scott, G. M., Shavlik, J. W., and Ray, W. H. (1991). Refining PID controllers using neural networks. In *Advances in Neural Information Processing Systems*, volume 4, Denver, CO. Morgan Kaufmann.

[Sejnowski87] Sejnowski, T. J. and Rosenberg, C. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168.

[Sestito90] Sestito, S. and Dillon, T. (1990). Using multi-layered neural networks for learning symbolic knowledge. In *Proceedings of the 1990 Australian Artificial Intelligence Conference*, Perth, Australia.

[Shavlik90a] Shavlik, J. W. (1990a). Case-based reasoning with noisy case boundaries: An application in molecular biology. Technical Report 988, University of Wisconsin, Madison, WI.

[Shavlik90b] Shavlik, J. W. (1990b). *Extending Explanation-Based Learning by Generalizing Explanation Structures.* Pitman, London.

[Shavlik91] Shavlik, J. W., Mooney, R. J., and Towell, G. G. (1991). Symbolic and neural net learning algorithms: An empirical comparison. *Machine Learning*, 6:111–143.

[Shavlik89] Shavlik, J. W. and Towell, G. G. (1989). An approach to combining explanation-based and neural learning algorithms. *Connection Science*, 1:233–255.

[Shortliffe84] Shortliffe, E. H. and Buchanan, B. G. (1984). A model of inexact reasoning in medicine. In Buchanan, B. G. and Shortliffe, E. H., editors, *Rule-Based Expert Systems*, pages 233–262. Addison-Wesley, Reading, MA.

[Smolensky87] Smolensky, P. (1987). On variable binding and the representation of symbolic structures in connectionist systems. Technical Report CU-CS-355-87, University of Colorado - Boulder.

[Smolensky88] Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11:1–23.

[Sobel87] Sobel, M. A., editor (1987). *Mathematics.* McGraw-Hill, New York.

[Squires91] Squires, C. S. and Shavlik, J. W. (1991). Experimental analysis of aspects of the cascade-correlation learning architecture. Machine Learning Research Group Working Paper 91-1. University of Wisconsin – Madison.

[Stormo90] Stormo, G. D. (1990). Consensus patterns in DNA. In *Methods in Enzymology*, volume 183, pages 211–221. Academic Press, Orlando, FL.

[Tesauro89] Tesauro, G. and Sejnowski, T. J. (1989). A parallel network that learns to play backgammon. *Artificial Intelligence*, 39:357–390.

[Thompson91] Thompson, K., Langley, P., and Iba, W. (1991). Using background knowledge in concept formation. In *Proceedings of the Eighth International Machine Learning Workshop*, pages 554–558, Evanston, IL.

[Touretsky88] Touretsky, D. S. and Hinton, G. E. (1988). A distributed connectionist production system. *Cognitive Science*, 12:423–466.

[Towell91] Towell, G. G. and Shavlik, J. W. (1991). Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In *Advances in Neural Information Processing Systems*, volume 4, Denver, CO. Morgan Kaufmann.

[Towell90] Towell, G. G., Shavlik, J. W., and Noordewier, M. O. (1990). Refinement of approximately correct domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866, Boston, MA.

[Tsoi90] Tsoi, A. C. and Pearson, R. A. (1990). Comparison of three classification techniques, CART, C4.5, and multi-layer perceptrons. In *Advances in Neural Information Processing Systems*, volume 3, pages 963–969, Denver, CO. Morgan Kaufmann.

[van Hiele86] van Hiele, P. M. (1986). *Structure and Insight.* Academic Press, New York.

[van Hiele-Geldof57] van Hiele-Geldof, D. (1957). *De didaktick van de Meetkunde in de eerste klass van het (The Didactics of Geometry in the Lowest Class of Secondary School).* PhD thesis, University of Utretch. English translation by M. Verdonck.

[von Heijne87] von Heijne, G. (1987). *Sequence Analysis in Molecular Biology: Treasure Trove or Trivial Pursuit.* Academic Press, San Deigo, CA.

[Wantanabe69] Wantanabe, S. (1969). *Knowing and Guessing: A Formal and Quantatative Study.* Wiley, New York.

[Waterman86] Waterman, D. A. (1986). *A Guide to Expert Systems.* Addison Wesley, Reading, MA.

[Watson87] Watson, J. D., Hopkins, H. H., Roberts, J. W., Steitz, J. A., and Weiner, A. M. (1987). *The Molecular Biology of the Gene.* Benjamin-Cummings, Menlo Park, CA.

[Wattenmaker87] Wattenmaker, W. D., Nakamura, G. L., and Medin, D. L. (1987). Relationships between similarity-based and explanation-based categorization. In Hilton, D., editor, *Contemporary Science and Natural Explanations: Common Sense Concepts of Causality*, pages 205–41. Harvester Press, Sussex, England.

[Weigand90] Weigand, A. S., Rumelhart, D. E., and Huberman, B. A. (1990). Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing Systems*, volume 3, pages 875–882, Denver, CO. Morgan Kaufmann.

[Weiss89] Weiss, S. M. and Kapouleas, I. (1989). An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 688–693, Detroit, MI.

[Weiss90] Weiss, S. M. and Kulikowski, C. A. (1990). *Computer Systems that Learn.* Morgan Kaufmann, San Mateo, CA.

[Wejchert89] Wejchert, J. and Tesauro, G. (1989). Neural network visualization. In *Advances in Neural Information Processing Systems*, volume 2, pages 465–472, Denver, CO. Morgan Kaufmann.

[Whewell89] Whewell, W. (1989). *Theory of the Scientific Method.* Hackett, Indianapolis. Originally published in 1840.

[Wieland87] Wieland, A. and Leighton, R. (1987). Geometric analysis of neural network capabilities. In *Neural Networks Conference*, San Diego, CA.

[Wisniewski89] Wisniewski, E. J. (1989). Learning from examples: The effect of different conceptual roles. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pages 844–851, Ann Arbor, MI.

[Wisniewski91] Wisniewski, E. J. and Medin, D. L. (1991). Is it a pucket or a purse? Tightly coupled theory and data driven learning. In *Proceedings of the Eighth International Machine Learning Workshop*, pages 564–569, Evanston, IL.

[Zadeh83] Zadeh, L. A. (1983). The role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy Sets and Systems*, 11:199–227.