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-

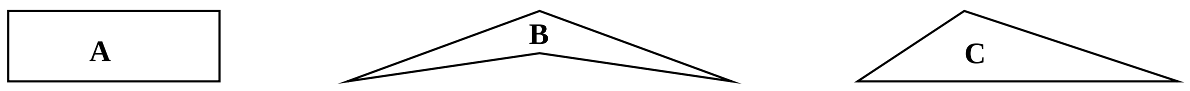

Figure 4.1: Sample triad.

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 (that is 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 (i.e., rule 11 in Table 4.2) 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

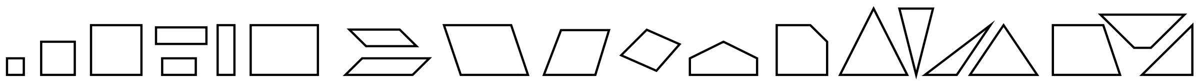

Figure 4.2: Representative textbook shapes.

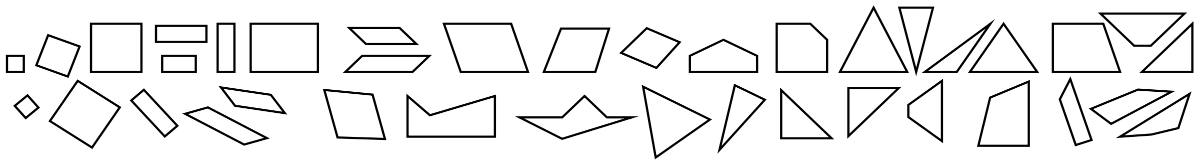

Figure 4.3: Representative shapes encountered using Logo.

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.

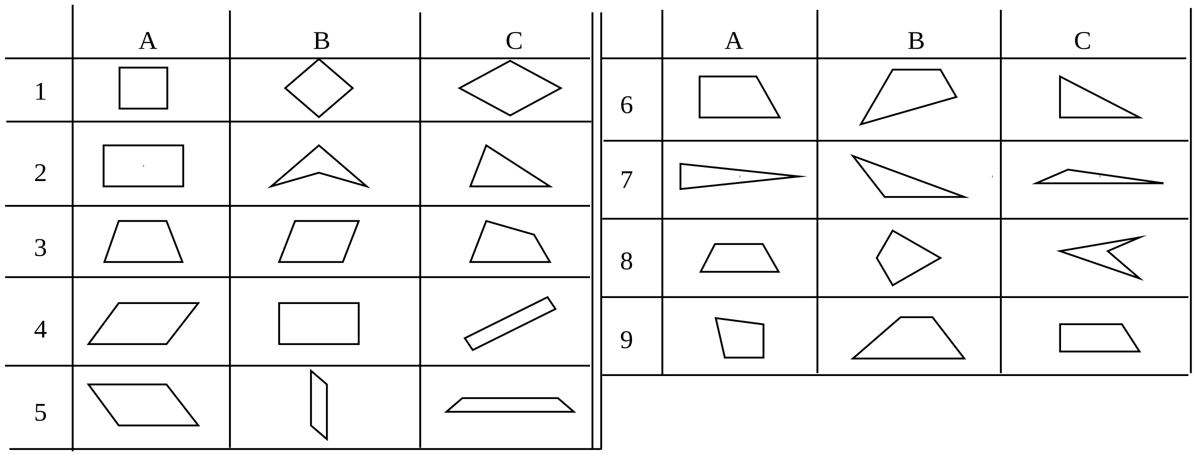

Figure 4.4: Triads used to test learning.

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 training set. *Point direction* and *area* are eliminated because the textbook set makes this

[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.

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. Conversely, 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 is that the provided rule set probably does not identify all of the important features of the input, a task that KBANN has difficulty with. Hence, a symbolic preprocessing phase that identifies important features not used by the initial rules should improve KBANN's learning ability. The first section of this chapter describes how this symbolic preprocessing can be done 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

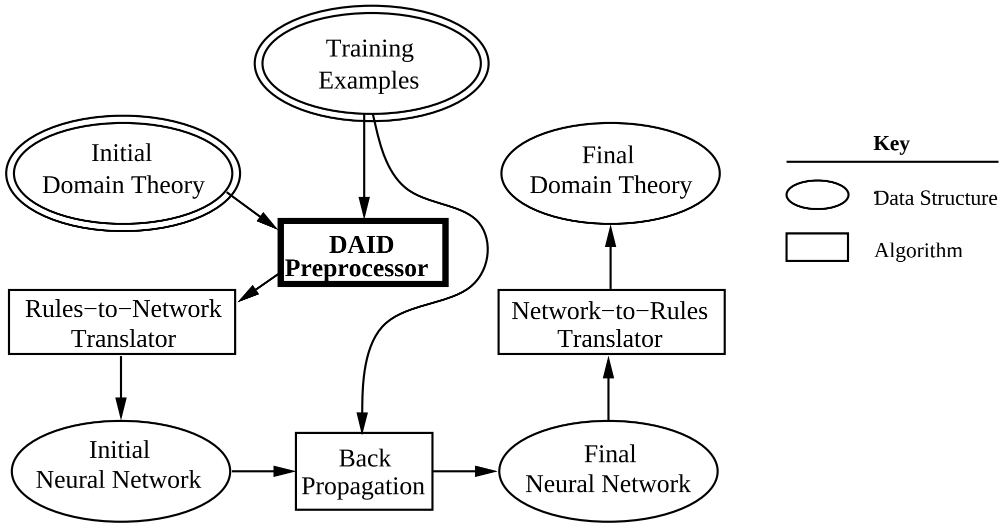

Figure 5.1: The flow of information through KBANN-DAID.

KBANN to unlearn antecedents that are useless than it is for KBANN to learn the importance 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. 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 basic idea of DAID is to assume that all errors occur at the lowest levels of the domain theory.[1] That is, DAID assumes the only rules that need corrections are those corresponding to units in a KBANN-net to which the rules-to-network translator would add low weight connections from the input units. However, DAID does not attempt to correct rules; its goal is only to identify input features whose presence is strongly correlated with incorrect inferences by the lowest-level rules.

Note that DAID's assumption that errors occur solely at the bottom of the rule hierarchy is significantly different for that of backpropagation (and other neural learning methods). These methods assume that error occurs along the whole learning path. Hence, 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. In such networks, backpropagated errors may become smoothly distributed across the network. The result of this smooth distribution of error is that all low level links change in approximately the same way. Hence, the network learns nothing. DAID does not face this problem; its error determination procedure is based upon boolean logic so errors cannot become unintentionally distributed.

Algorithm specification

The main problem that DAID must address is determining the correct conclusions of the lowest rules in a hierarchical rule set. DAID avoids the full force of the "credit-assignment" problem [Minsky63] because its goal is simply to identify *potentially* useful input features. Hence, DAID does not need to uniquely identify the source of an error. 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 5.2.

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 counters for each feature-value pair (i.e., for everything that will become an input unit when the rules are translated into a KBANN-net) for the consequent of each of the lowest-level rules. These counters must cover the following conditions:

[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].

Table 5.1: Assignment of blame by the recursive *BackUpAnswer* procedure.

Antecenends to blame for an incorrect consequent			
		Blame antecedent if	
Type of rule	Consequent should be	dependency is negated	dependency is unnegated
conjunctive	true	antecedent is true	antecedent is false
conjunctive	false	antecedent is false	antecedent is true
disjunctive	true	NA	antecedent is false
disjunctive	false	NA	antecedent is true

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

Hence, each input feature must maintain eight counters for each of the lowest-level consequents. With these counters in place, DAID goes through each of the training examples, running *Back-UpAnswer*, and incrementing the appropriate counters.

BackUpAnswer starts by believing that every consequent is correct. Then, working down from any incorrect final conclusions, it assigns blame to 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 5.1 summarizes how *BackUpAnswer* assigns blame. For instance, if the a consequent of a conjunctive rule is false but should be true, then *BackUpAnswer* assigns blame to every true antecedent with a negated dependency and every false antecedent with an unnegated dependency.

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 numbers used for the initial weights is summarized as the DAID procedure in Table 5.2.

While these numbers are the end result of DAID, they are not an end of themselves. Rather, they are passed to the rules-to-network translator of KBANN where they are used to initialize

Table 5.2: Summary of the DAID algorithm

Daid:

1. Establish eight counters associating each of 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.
   - Determine the correctness 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:

- For each antecedent of the rule being investigated
  - Determine the correctness of the antecedent
  - Recursively call BackUpAnswer if the antecedent is incorrect

Pseudocode for DAID appears in Appendix B.3.

the weights of the low-weighted links that are added to the network (see Section 2.2.1). By using these numbers to initialize weights, the KBANN-net that results for the 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, an 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.

Table 5.3: 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

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 describing 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.

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

The case for adding a negatively weighted link is exactly parallel. The equations for determining the pseudo-correlations are given in Appendix B.3.

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 5.3 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 5.3. 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 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 5.2. (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

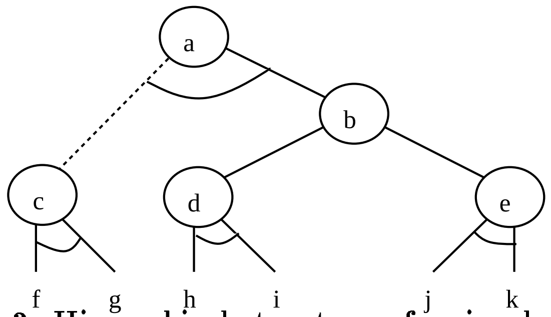

Figure 5.2: Hierarchical structure of a simple rule set.

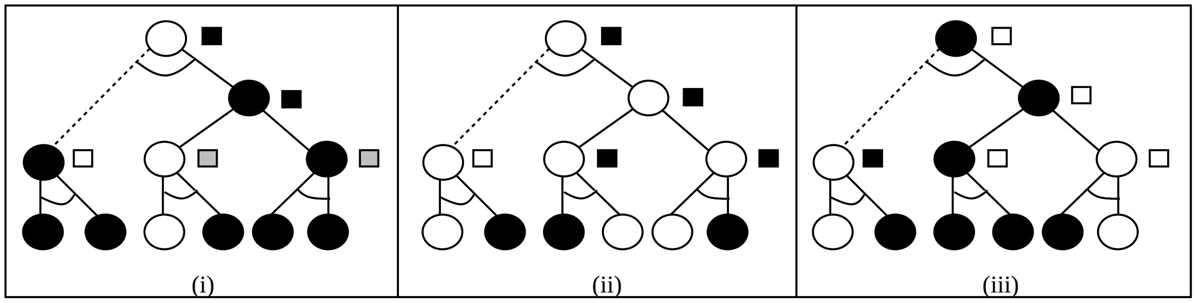

Figure 5.3: The state of the rule set after presenting three examples.

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 5.3, 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.

As shown in Figure 5.4, DAID improves generalization by KBANN-net in the promoter domains and slightly impairs generalization for splice-junctions. The slight impairment in

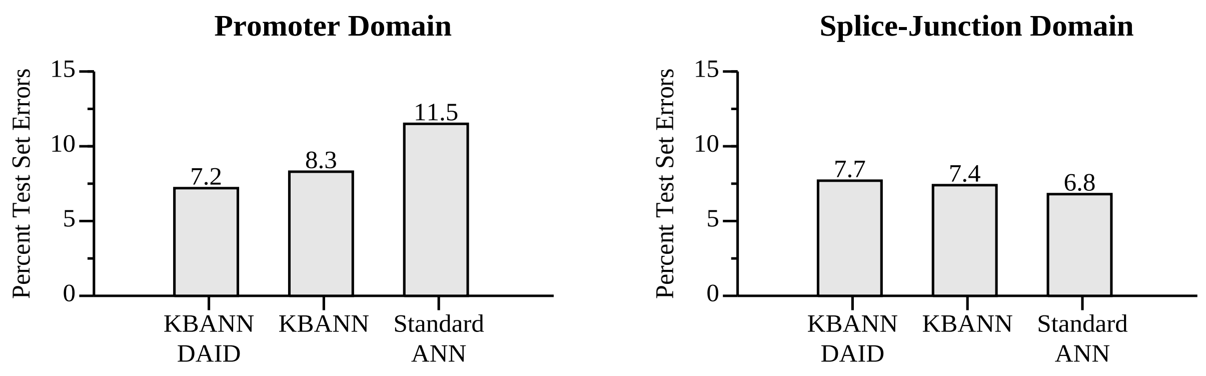

Figure 5.4: Generalization using Daid (measured using 10-fold cross-validation).

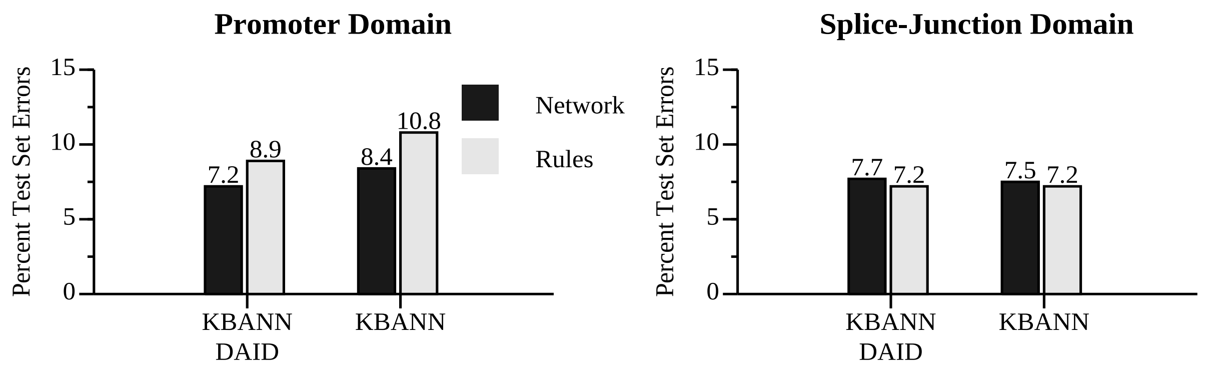

Figure 5.5: Accuracy of extracted rules using Daid (measured using 10-fold cross-validation).

generalization on the splice-junction problem is not statistically significant according to a one-tailed, paired-sample t-test.

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. (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 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 re-

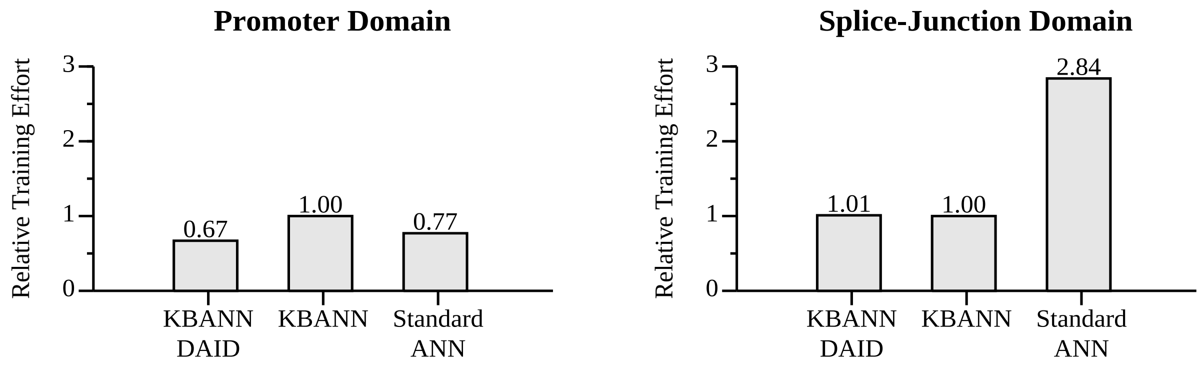

Figure 5.6: Training effort required when using Daid.

quires 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 clearly show that Daid speeds learning on the promoter problem, bringing the effort required to train Kbann-nets down below that of standard Kbann-nets. (This result is statistically significant with greater than 99.5% confidence (t=6.5, d.f. = 10). The difference in learning speed when using Daid on the splice-junction problem is not statistically significant.

5.1.3 Discussion of Kbann-Daid

This section described the Daid preprocessor for Kbann. Daid is motivated by the observation that Kbann appears to be most effective when the networks it creates must unlearn antecedents (as opposed to learning new antecedents). Hence, Daid attempts to identify antecedents, not used in the domain knowledge provided to Kbann, that may be useful in correcting the errors of the domain knowledge.

Results show that Daid is effective in on the promoter data three ways. First, Daid improves the generalization abilities of Kbann-nets. For the splice-junction problem, this improvement results in trained Kbann-nets being superior to standard ANNs. Second, Daid reduces the effort required to train Kbann-nets. While Kbann-nets still require more effort than standard networks the gap in effort is narrowed. Finally, Daid improved the generalization performance of the rules extracted by the NofM algorithm of the network-to-rules translator.

Conversely, on the splice-junction dataset, Daid has little effect. The ineffectiveness of Daid is due to the nature of the splice-junction domain theory. Specifically, as alluded to previously, that domain theory 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.

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 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

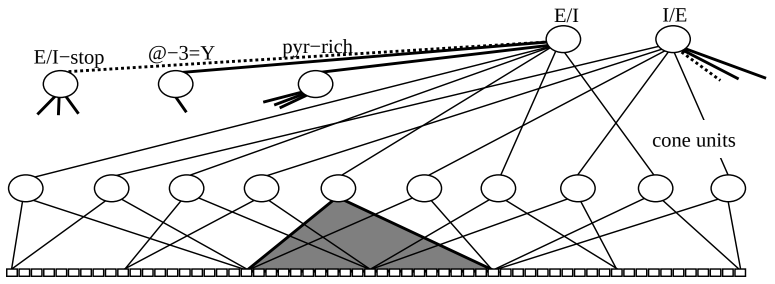

Figure 5.7: Splice-Junction Network with Cone Units

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

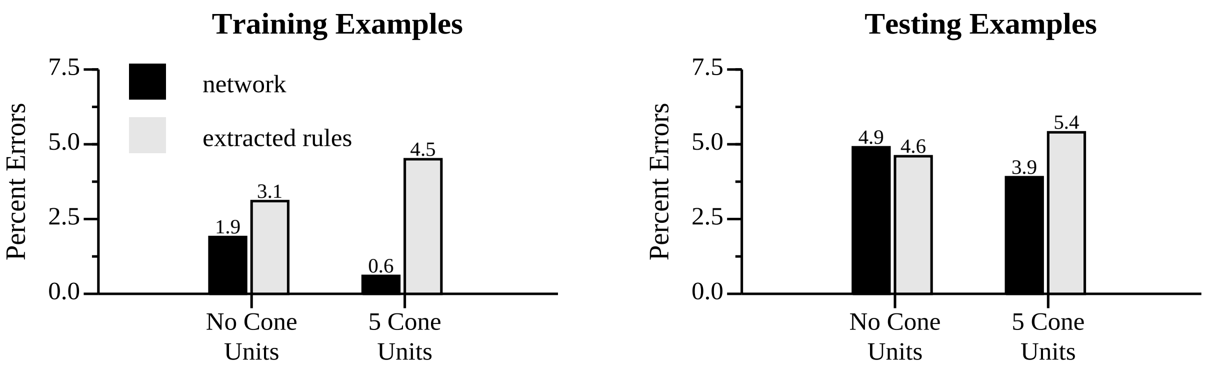

Figure 5.8: Error rates on training and testing txamples.

Table 5.4: 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

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 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 5.4. To simplify comparisons, data for networks that do not include cone units is included in both Figure 5.8 and Table 5.4. 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.

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 5.5 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.

Table 5.5: 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.

Moreover, Table 5.5 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.

Less obvious is that the rules in Table 5.6 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 exact 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, the 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, the backpropagation algorithm has difficulty adjusting weights when the logistic activation function too closely resembles a step function.

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.

Table 5.6: 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. So, nt(@-14 ‘- - - C - - G - -’) would return 1 when matched against the sequence @-14 'AAACAAAAA'.

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 the various approaches to learning from theory and data in the literature 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

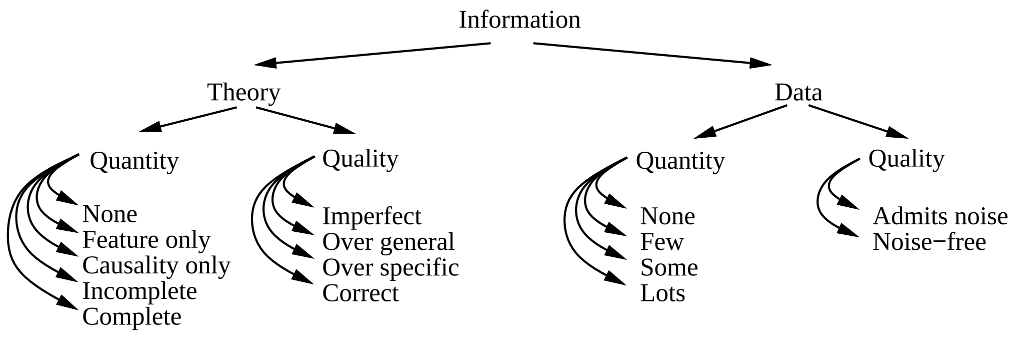

Figure 6.1: Partitioning the space of information available to machine learning algorithms.

"feature correlations are partly supplied by people's theories and that the causal mechanisms 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 sorts of information required by the system. In particular, what sort of theoretical information is required and what sort of examples (data) 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. 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. Thus, 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

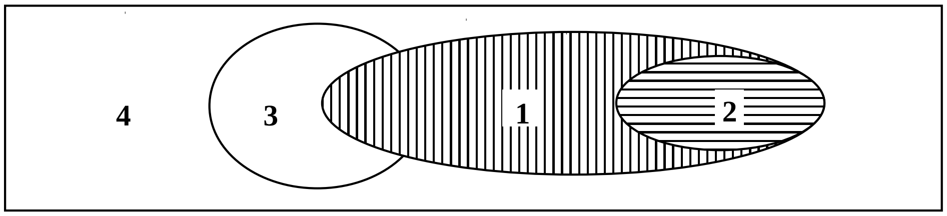

Figure 6.2: Possible classification decisions of a knowledge-based system.

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
[Hall88]	overly-specific	incomplete	noise-free	one or more	special
[Oliver88]	correct	complete	noise-free	some	ANNs
[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
[Hirsh89]	imperfect	zero or more	noise-free	some	Version Space
[Danyluk89]	overly-specific	incomplete	noise free	some	special
[Redmond89]	imperfect	incomplete	noise-free	some	case-based
[Katz89]	imperfect	incomplete	noise-free?	some	ANNs
[Mooney91b]	imperfect	zero or more	noise-free?	some	ID3
[Thompson91]	imperfect	zero or more	noise-free?	some	Cobweb
KBANN	imperfect	zero or more	admits noise	some	ANNs

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.

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 explain which of the weaknesses of empirical and explanation-based learning that each address.

Extended UNIMEM

Extended Unimem [Lebowitz86] was developed to address 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 either explanation-based or 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 empirical component to eliminate unimportant features by giving Occam the ability to consider the meaning of features as well as correlations between examples. Hence, Occam addresses several of the problems of explanation-based and empirical learners. Its use of general causal knowledge makes it insensitive to unimportant features. It is able to build its own 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 the knowledge base 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

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

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 the approaches is their choice of learning algorithms. Whereas KBANN uses a "subsymbolic" learning algorithm, both EITHER and Labyrinth-k use symbolic algorithms. This results in a significant difference in the complexity of the approaches as 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 simple perceptron learning. To train CES using the pocket algorithm, the correct activation of every unit in the network must be known. That is, every hidden and output unit is trained individually. 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. As a result, there have been several approaches to making trained neural networks comprehensible. The simplest 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.

Finally, note that these visualization tools 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 similar 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] report a method 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 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 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 this 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, tight clusters. Finally, tight precision bounds may force link weights into clusters that are not optimally in terms of their relative weights. Hence, the 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 or 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 many 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 are 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 us 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.

Results of 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 to determine 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, 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, the 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 empirical learning system 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 do-

main 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 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 would be 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 most categorization tasks, this assumption is invalid. Rather, the correct assumption is that 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 Equations 7.4 and 7.5). Under 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).

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

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.

Revised weight decay. To improve the interpretability of trained networks, each weight is "decays" towards zero. (This idea was suggested by Hinton [Hinton89]. It is described in detail in Section 3.5.) 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]. As a result, 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 from the initial level.

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 the training with pre-specified Gaussians is initially 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. The do not always pull weights back towards their initial values. Instead, the Gaussians pull weights towards nearby clusters.

Conjugate-gradient optimization

Even using cross-entropy (or the normalized-exponential described above), backpropagation is somewhat ill-suited to training KBANN-nets because all units initially have activations near zero or one. Hence, errors propagated from the hidden units are reduced to very near zero by the term $[o_i * (1 - oi)]$ — 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 above discussion 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. (Note that this topic relates to issues in rule extraction.)

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 three 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 may have 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 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 very tightly around zero.

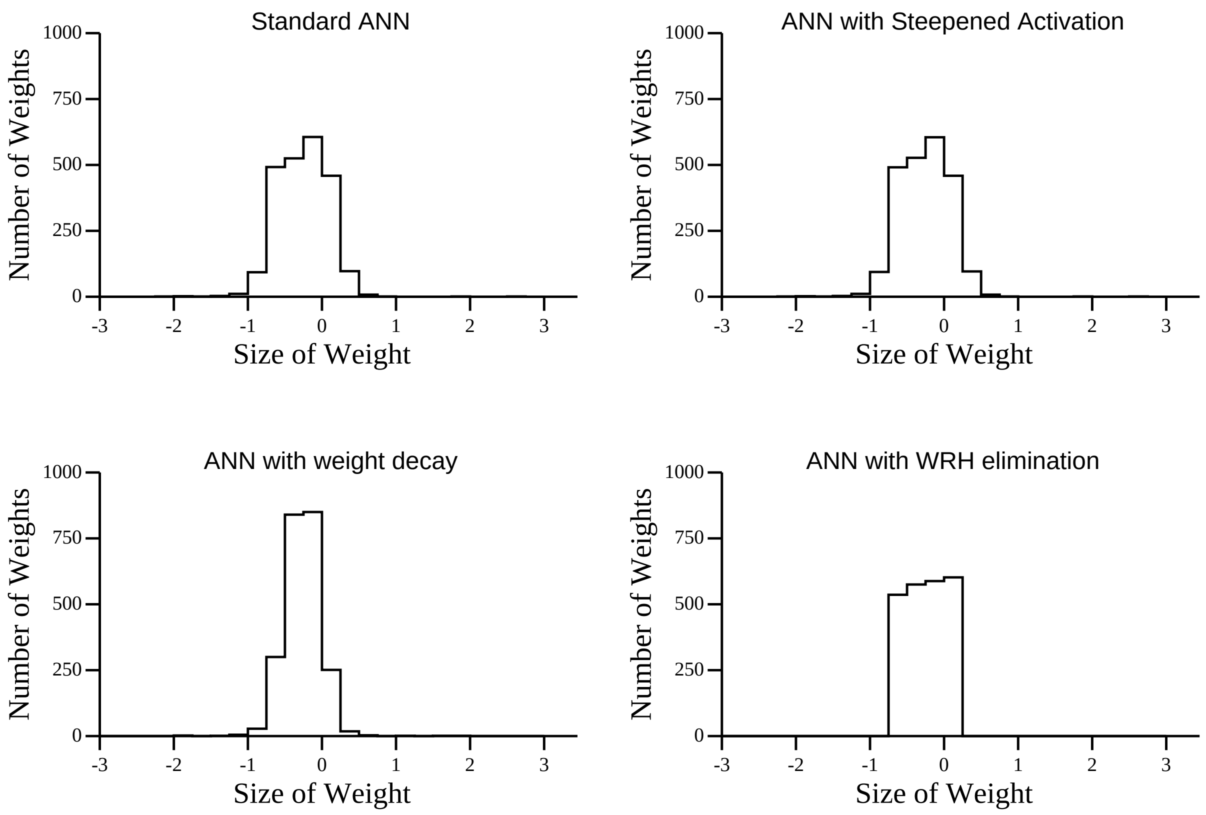

Figure 7.1: Histogram showing distribution of weights in ANNs trained for promoter recognition.

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 suggestion is supposed to push 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]. Their technique expands on the previously-described approach of Weigand, Rumelhart, and Huberman. That approach groups link weights into a cluster near zero and otherwise allows weights to take on arbitrary values. Conversely, the approach of Nowlan and Hinton groups links into a small (< 10) number of clusters and does not allow an otherwise arbitrary spread of link weights.

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 form of the rules produced by this method would be similar to NOFM rules. However, the search paths through weight space taken by this algorithm would be quite different than the paths taken by the NOFM algorithm. 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. One way of implementing this suggestion is to build upon the algorithm for clustering while training of Nowlan and Hinton [Nowlan91].

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 washed out 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 learn 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, as well as two other 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 two small 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. KBANN has also been used by R. Maclin for protein secondary structure prediction [Maclin91] and by G. Scott for process control [Scott91].

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.

While 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 a large first step along a significant research path.