Appears in *Artificial Intelligence*, volume 69 or 70.
Submitted 1/92, Final pre-publication revisions 8/94

# Knowledge-Based Artificial Neural Networks

**Geoffrey G. Towell***
towell@learning.scr.siemens.com
(609) 321-0065

**Jude W. Shavlik**
shavlik@cs.wisc.edu
(608) 262-7784

University of Wisconsin
1210 West Dayton St.
Madison, WI 53706

**Keywords:** machine learning, connectionism, explanation-based learning, hybrid algorithms, theory refinement, computational biology

**Running Head:** Knowledge-Based Artificial Neural Networks

*Current address is: Siemens Corporate Research, 755 College Road East, Princeton, NJ, 08540. Please direct all correspondence to this address.

# Abstract

Hybrid learning methods use theoretical knowledge of a domain and a set of classified examples to develop a method for accurately classifying examples not seen during training. The challenge of hybrid learning systems is to use the information provided by one source of information to offset information missing from the other source. By so doing, a hybrid learning system should learn more effectively than systems that use only one of the information sources. KBANN*(Knowledge-Based Artificial Neural Networks)* is a hybrid learning system built on top of connectionist learning techniques. It maps problem-specific "domain theories", represented in propositional logic, into neural networks and then refines this reformulated knowledge using backpropagation. KBANN is evaluated by extensive empirical tests on two problems from molecular biology. Among other results, these tests show that the networks created by KBANN generalize better than a wide variety of learning systems, as well as several techniques proposed by biologists.

# 1. Introduction

Suppose you are trying to teach someone who has never seen a class of objects to recognize members of that class. One approach is to define the category for your student. That is, state a "domain theory"[1] that describes how to recognize critical facets of class members and how those facets interact. Using this domain theory, your student could distinguish between members and nonmembers of the class. A different approach to teaching someone to recognize a class of objects is to show the person lots of examples. As each example is shown, you would tell your student only whether the example is, or is not, a member of the class. After seeing sufficient examples, your student could classify new examples by comparison to those already seen.

These two methods of teaching roughly characterize two approaches to achieving problem-specific expertise in a computer: *hand-built classifiers* (e.g., expert systems [58]) and *empirical learning* [42, 47]. Hand-built classifiers correspond to teaching by giving a person a domain theory without an extensive set of examples; one could call this *learning by being told*. Conversely, empirical learning corresponds to giving a person lots of examples without any explanation of why the examples are members of a particular class. Unfortunately, for reasons listed in the following section, neither of these approaches to achieving machine expertise is completely satisfactory. They each suffer from flaws that preclude them from being a generally applicable method.

The flaws of each method are, for the most part, complementary (see Sections 2.1–2.2). Hence, a "hybrid" system that effectively combines a hand-built classifier with an empirical learning algorithm might be like a student who is taught using a combination of theoretical information and examples. That student might be able to combine both sources of information to fill gaps in her knowledge which would otherwise exist. Similarly, hybrid learning systems (reviewed in Sections 2.4 and 6) should find synergies that make them more effective than either hand-built classifiers or empirical learning algorithms used in isolation.

KBANN *(Knowledge-Based Artificial Neural Networks)* – the successor to our EBL-ANN algorithm [51] – is such a system. The approach taken by KBANN is outlined in Table 1. Briefly, the idea is to *insert* a set of hand-constructed, symbolic rules (i.e., a hand-built classifier) into a neural network. The network is then *refined* using standard neural learning algorithms and a set of classified training examples. The refined network can then function as a highly-accurate classifier. A final step for KBANN, the extraction of refined, comprehensible rules from the trained neural network, has been the subject of much effort [56] but is beyond the scope of this paper.

Section 3 describes the KBANN algorithm. Empirical tests in Section 5, using the DNA

[1] In machine learning, a *domain theory* [28] is a collection of rules that describes task-specific inferences that can be drawn from the given facts. For classification problems, a domain theory can be used to prove whether or not an object is a member of a particular class.

**TABLE 1** The KBANN approach to learning.

- Given:
  - A list of features used to describe examples
  - An approximately-correct *domain theory* describing the problem to be solved
  - A set of classified training examples
- Do:
  - Translate the domain theory into a neural network
  - Train the kmowledge-based network using the classified examples
  - Use the trained network to classify future examples
  - (Optionally) extract a refined domain theory [56]

sequence-analysis tasks described in Section 4, show that KBANN benefits from its combination of a hand-built classifier and empirical learning. These tests show on the datasets we examine that KBANN generalizes better than methods that learn purely from examples, and other methods which learn from both theory and examples. (Following convention, we assess generalization by testing systems on examples not seen during training.) Further testing reveals that KBANN is able to profitably use domain theories that contain significant amounts of misinformation. Hence, our tests show that, under a broad range of conditions, KBANN yields the hoped-for synergies of a hybrid approach to learning.

# 2. The Need for Hybrid Systems

Before describing KBANN, we further motivate the development of hybrid systems by listing some of the important weaknesses of hand-built classifiers and empirical learning systems. Following these lists is a brief overview of the reasons that hybrid systems are an active area of machine learning research.

## 2.1. Hand-built classifiers

Hand-built classifiers are non-learning systems (except insofar as they are later altered by hand). They simply do what they are told; they do not learn at the knowledge level [9]. Despite their apparent simplicity, such systems pose many problems for those that build them.

- Typically, hand-built classifiers assume that their domain theory is complete and correct. However, for most real-world tasks, completeness and correctness are extremely difficult, if not impossible, to achieve. In fact, in explanation-based learning [28] one of the major issues is dealing with incomplete and incorrect domain theories.

- Domain theories can be intractable to use [28]. To make a domain theory as complete and correct as possible, it may be necessary to write thousands of interacting, possibly recursive, rules. Use of such rule sets may be intolerably slow.
- Domain theories can be difficult to modify [3]. As interactions proliferate in a rule set, it becomes difficult to predict all of the changes resulting from modifying a single rule.

**2.2. Empirical learning**

Empirical learning systems inductively generalize specific examples. Thus, they require little theoretical knowledge about the problem domain; instead they require a large library of examples. Their almost complete ignorance of problem-specific theory means that they do not address important aspects of induction. Some of the most significant problems are:

- An unbounded number of features can be used to describe any object [48]. Hence, the user's choice of features can make a computer and a cookie appear very similar or very different.
- Features relevant to classification are context dependent [48]. For example, the observation that paper money is flammable may be only relevant when a bank is on fire.
- Complex features constructed from the initial features may considerably simplify learning [44]. However, feature construction is a difficult, error-prone, enterprise.
- Even when a large set of examples are available, small sets of exceptions may be either unrepresented or very poorly represented [16]. As a result, uncommon cases may be very difficult to correctly handle.

**2.3. Artificial neural networks**

Artificial neural networks (ANNs), which form the basis of KBANN, are a particular method for empirical learning. ANNs have proven to be equal, or superior, to other empirical learning systems over a wide range of domains, when evaluated in terms of their generalization ability [50, 2]. However, they have a set of problems unique to their style of empirical learning. Among these problems are:

- Training times are lengthy [50].
- The initial parameters of the network can greatly affect how well concepts are learned [1].
- There is not yet a problem-independent way to choose a good network topology, although there has been considerable research in this direction (e.g., [10]).
- After training, neural networks are often very difficult to interpret [56].

**2.4. Hybrid Learning Systems**

There is a significant gap between the knowledge-intensive, learning-by-being-told approach of hand-built classifiers and the virtually knowledge-free approach of empirical learning. Some of this gap is filled by "hybrid" learning methods, which use both hand-constructed rules and classified examples during learning.

Several trends have made the development of such systems an active area in machine learning. Perhaps the most important of these trends is the realization that knowledge-intensive (e.g., [28]) and knowledge-free learning are just two ends of a spectrum along

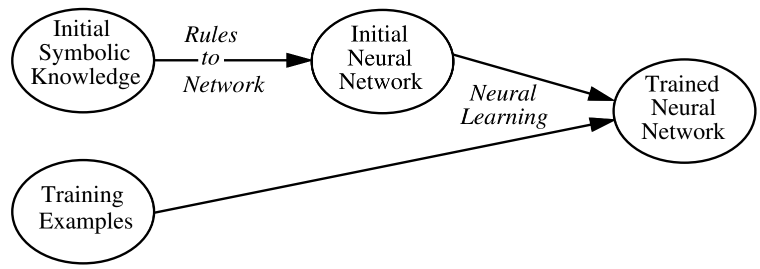



**Figure 1** Flow chart of theory-refinement by KBANN.

which an intelligent system may operate. This realization has lead to recent specialized workshops (e.g., [6, 26]). Staying at one end or the other of the spectrum of possible learning systems simplifies the learning problem by allowing strong assumptions to be made about the nature of what needs to be learned. However, the middle ground is appealing; it offers the possibility that synergistic combinations of theory and data will result in powerful learning systems.

Another trend spurring the development of hybrid systems is the growth of a body of psychological evidence that people rarely, if ever, learn purely from theory or examples [62]. For instance, Murphy and Medin suggest that "feature correlations are partly supplied by people's theories and that the causal mechanisms contained in theories are the means by which correlational structure is represented" [32, page 294]. That is, theory and examples interact closely during human learning. While it is clear that people learn from both theory and examples, the way in which the interaction occurs is yet to be determined. This has been the subject of much research [38, 62] which affects work in machine learning.

Finally, there is a purely practical consideration; hybrid systems have proven effective on several real-world problems [57, 33, 36, 54, 19] (also Section 5).

# 3. KBANN

This section describes the KBANN methodology, which Figure 1 depicts as a pair of algorithms (on the arcs) that form a system for learning from both theory and examples. The first algorithm, labeled "Rules-to-Network", is detailed in Section 3.3. This algorithm *inserts* approximately-correct, symbolic rules into a neural network. Networks created in this step make the same classifications as the rules upon which they are based.

The second algorithm of KBANN, labeled "Neural Learning", *refines* networks using the backpropagation learning algorithm [47]. (Although all of our experiments use back-propagation, any method for supervised weight revision — e.g., conjugate gradient [4] — would work.) While the learning mechanism is essentially standard backprop-agation, the network being trained is not standard. Instead, the first algorithm of KBANN constructs and initializes the network. This has implications for training that

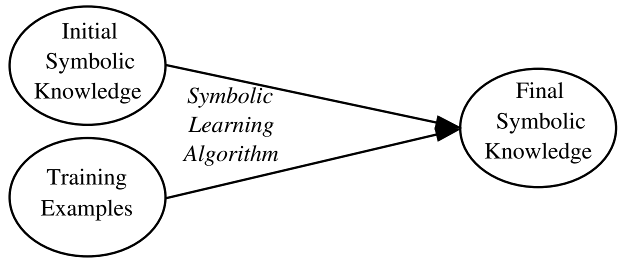



**Figure 2** Flow chart of "all-symbolic" theory-refinement.

are discussed in Section 3.7. At the completion of this step, the trained network can be used as a very accurate classifier.

Before beginning a detailed description of KBANN, consider the difference between Figures 1 and 2. These figures present two alternative architectures for systems that learn from both theory and examples. As described above, Figure 1 shows the architecture of KBANN. By contrast, Figure 2 represents the architecture of EITHER [36] and Labyrinth-k [54], two "all-symbolic" hybrid learning systems to which KBANN is compared in Section 5. Whereas KBANN requires two algorithms, these all-symbolic systems require only a single algorithm because their underlying empirical learning mechanism operates directly upon the rules rather than their re-representation as a neural network. Tests reported in Chapter 5 show that the extra effort entailed by KBANN is well rewarded, as KBANN generalizes better than these all-symbolic systems on our testbeds.

The next subsection presents a brief overview of the type of neural networks we use. Subsequent to this is a high-level overview of KBANN. The following two subsections contain in-depth descriptions of each of KBANN's algorithmic steps.

## 3.1. Neural Networks

The neural networks we use in this paper are all "feedforward" neural networks that are trained using the backpropagation algorithm [47]. Units have a *logistic activation function*, which is defined by Equations 1 and 2. Roughly speaking, when the net incoming activation to a unit exceeds its bias, then the unit has an activation near one. Otherwise, the unit has an activation near zero.

$$NetInput_i = \sum_{j \in \{Connected\_Units\}} Weight_{ji} * Activation_j \tag{1}$$

$$Activation_i = \frac{1}{1 + e^{-(NetInput_i - Bias_i)}} \tag{2}$$

## 3.2. Overview of KBANN

As shown in Figure 1, KBANN consists of two largely independent algorithms: a *rules-to-network translator* and a *refiner* (i.e., a neural learning algorithm). Briefly, the rules-to-networks translation is accomplished by establishing a mapping between a rule set and a neural network. This mapping, specified by Table 2, defines the topology of networks created by KBANN as well as the initial link weights of the network (see Section 3.3).

By defining networks in this way, some of the problems inherent to neural networks and empirical learning are ameliorated. The translation specifies the features that are probably relevant to making a correct decision. This specification of features addresses problems such as spurious correlations, irrelevant features, and the un-

**TABLE 2** Correspondences between knowledge-bases and neural networks.

| *Knowledge Base* | | *Neural Network* |
|---|---|---|
| Final Conclusions | $\Longleftrightarrow$ | Output Units |
| Supporting Facts | $\Longleftrightarrow$ | Input Units |
| Intermediate Conclusions | $\Longleftrightarrow$ | Hidden Units |
| Dependencies | $\Longleftrightarrow$ | Weighted Connections |

boundedness of the set of possible features. Rule translation can specify important "derived" features, thereby simplifying the learning problem [44]. Moreover, these derived features can capture contextual dependencies in an example's description. In addition, the rules can refer to arbitrarily small regions of feature space. Hence, the rules can reduce the need for the empirical portion of a hybrid system to learn about uncommon cases [16]. This procedure also indirectly addresses many problems of hand-built classifiers. For instance, the problem of intractable domain theories is reduced because approximately-correct theories are often quite brief.

The second major step of KBANN is to refine the network using standard neural learning algorithms and a set of classified training examples. At the completion of this step, the trained network can be used as a classifier that is likely to be more accurate than those derived by other machine learning methods. Section 5.1 contains empirical evidence that supports this claim.

### 3.3. Inserting Knowledge into a Neural Network

The first step of KBANN is to translate a set of approximately-correct rules into a knowledge-based neural network (henceforth, a KBANN-net). Rules to be translated into KBANN-nets are expressed as Horn clauses. (See Appendix A for a complete description of the language accepted by KBANN.) There are two constraints on the rule set. First, the rules must be propositional. This constraint results from the use of neural learning algorithms which are, at present, unable to handle predicate calculus variables. Second, the rules must be acyclic. This "no cycles" constraint simplifies the training of the resulting networks. However, it does not represent a fundamental limitation on KBANN, as there exist algorithms based upon backpropagation that can be used to train networks with cycles [40]. Moreover, others have extended KBANN to handle recursive finite-state grammars [23].

In addition to these constraints, the rule sets provided to KBANN are usually hierarchically structured. That is, rules do not commonly map directly from inputs to outputs. Rather, at least some of the rules provide intermediate conclusions that describe useful conjunctions of the input features. These intermediate conclusions may be used by other rules to either determine the final conclusion or other intermediate conclusions. It is the hierarchical structure of a set of rules that creates derived features for use by the example-based learning system. Hence, if the domain knowledge is not hierarchically structured, then the networks created by KBANN will have no derived features that indicate contextual dependencies or other useful conjunctions within example descriptions. Also, the KBANN-net that results from translating a rule set with no intermediate conclusions would have no hidden units. As a result, it would be capable of only Perceptron-like learning [46].

The rules-to-network translator is described in the next three subsections. The first of these subsections provides a detailed description of the translation; the second contains an example of the translation process; and the third contains a pair of intuitive arguments that KBANN's translator is correct (full proofs appear in [55]).

**TABLE 3** The rules-to-networks algorithm of KBANN.

1. Rewrite rules so that disjuncts are expressed as a set of rules that each have only one antecedent.
2. Directly map the rule structure into a neural network.
3. Label units in the KBANN-net according to their “level.”
4. Add hidden units to the network at user-specified levels (*optional)*.
5. Add units for known input features that are not referenced in the rules.
6. Add links not specified by translation between all units in topologically-contiguous levels.
7. Perturb the network by adding near-zero random numbers to all link weights and biases.

**Initial Rules**

```
A :- B, C, D.
A :- D, E, F, G.
```

*Rewriting*

**Final Rules**

```
A   :- A'.
A'  :- B, C, D.

A   :- A''.
A'' :- D, E, F, G.
```

**Figure 3** Rewriting rules to eliminate disjuncts with more than one term, so that the rules may be translated into a network that accurately reproduces their behavior.

## 3.4. The rules-to-network algorithm

Table 3 is an abstract specification of the seven-step rules-to-network translation algorithm. This algorithm initially translates a set of rules into a neural network. It then augments the network so that it is able to learn concepts not provided by the initial rules. In this subsection we describe, in detail, each of the seven steps of this algorithm.

**Step 1, Rewriting.** The first step of the algorithm transforms the set of rules into a format that clarifies its hierarchical structure and makes it possible to directly translate the rules into a neural network. If there is more than one rule for a consequent, then every rule for this consequent with more than one antecedent is rewritten as two rules. (The only form of disjunction allowed by KBANN is multiple rules with the same consequent.) One of these rules has the original consequent and a single, newly-created term as an antecedent. The other rule has the newly-created term as its consequent and the antecedents of the original rule as its antecedents. For instance, Figure 3 shows the transformation of two rules into the format required by the next steps of KBANN. (The need for this rewriting is explained in Section 3.6).

**Step 2, Mapping.** In the second step of the rules-to-network algorithm, KBANN establishes a mapping between a transformed set of rules and a neural network. Using this mapping, shown in Table 2, KBANN creates networks that have a one-to-one correspondence with elements of the rule set. Weights on all links specified by the rule set, and the biases on units corresponding to consequents are set so that the network responds in exactly the same manner as the rules upon which it is based. (See Section 3.6 for an explanation of the precise settings).

At the completion of this step, the KBANN-net has the information from the set of rules concerning relevant input and derived features. However, there is no guarantee that the set of rules refers to all of the relevant features or provides a significant collection of derived features. Hence the next four steps augment the KBANN-net with additional links, inputs units, and (possibly) hidden units.

**Step 3, Numbering.** In this step, KBANN numbers units in the KBANN-nets by their "level." This number is not useful in itself, but is a necessary precursor to several of the following steps. KBANN defines the level of each unit to be the length of the *longest* path to an *input* unit.[2]

**Step 4, Adding hidden units.** This step adds hidden units to KBANN-nets, thereby giving KBANN-nets the ability to learn derived features not specified in the initial rule set but suggested by the expert. This step is optional because the initial rules often provide a vocabulary sufficient to obviate the need for adding hidden units. Hence, hidden units are only added upon specific instructions from a user. This instruction must specify the number and distribution among the levels established in the previous step of the added units.

The addition of hidden units to KBANN-nets is a subject that has been only partially explored. Methods of unit addition are described and evaluated elsewhere (e.g., [55, 35]).

**Step 5, Adding input units.** In this step, KBANN augments KBANN-nets with input features not referred to by the rule set but which a domain expert believes are relevant. This addition is necessary because a set of rules that is not perfectly correct may not identify every input feature required for correctly learning a concept.

**Step 6, Adding links.** In this step, the algorithm adds links with weight zero to the network using the numbering of units established in step 4. Links are added to connect each unit numbered $n-1$ to each unit numbered $n$. Adding links in this way, in conjunction with the numbering technique described above, is slightly better than several other methods for adding links that we have explored [55].

**Step 7, Perturbing.** The final step in the network-to-rules translation is to perturb all the weights in the network by adding a small random number to each weight. This perturbation is too small to have an effect on the KBANN-net's computations prior to training. However, it is sufficient to avoid problems caused by symmetry [47].

### 3.5. Sample rules-to-network translation

Figure 4 shows a step-by-step translation of a simple set of rules into a KBANN-net. Panel *a* shows a set of rules in PROLOG-like notation. Panel *b* is the same set of rules after they have been rewritten in step 1 of the translation algorithm. The only rules affected by rewriting are two which together form a disjunctive definition of the consequent B.

[2]This numbering technique implicitly assumes that every chain of reasoning is complete; that is, every intermediate conclusion is a part of a directed path from one or more inputs to one or more outputs. However, there is no requirement that every chain will be complete. For incomplete chains, we attach the unconnected antecedents directly to every input unit and unconnected consequents directly to every output unit (with low-weight links).

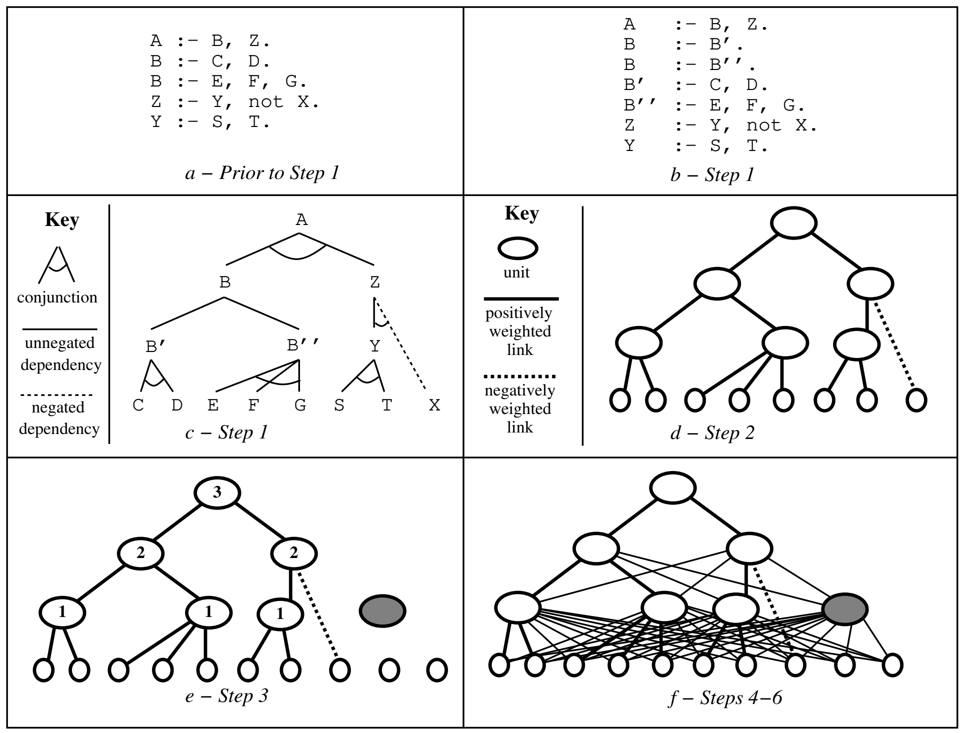



**Figure 4** Sample rules-to-network translation.

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

The next step of the translation algorithm (step 2 in Table 3) is to create a neural network by mapping the hierarchical structure of the rules into a network. As a result, there is little visual difference between the representations of the initial KBANN-net in panel *d* and the hierarchical structure of the rules in panel *c*.

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

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

There is no illustration of the final step of the rules-to-network translation algorithm because the perturbation of link weights results only in minute changes that never qualitatively affect the calculations of the initial network.

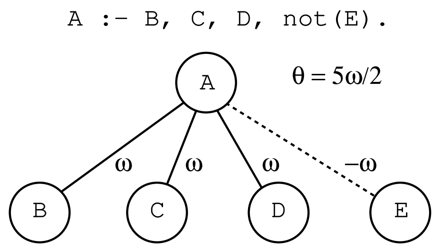



**Figure 5** Translation of a conjunctive rule into a KBANN-net.

### 3.6. Translation of rules into KBANN-nets

This section describes how KBANN translates rules containing the logical connectives AND, OR, and NOT into a KBANN-net. Recall that individual rules are assumed to be conjunctive, nonrecursive, and variable-free; disjuncts are encoded (without loss of generality) as multiple rules as described above. The discussion in this section assumes that features are only binary-valued. (The restriction to binary-valued features is made here only for clarity of explanation. See Appendix A for a specification of the language that KBANN can accept.)

The rules-to-network translator sets weights on the links and the biases of units so that units have significant activation (i.e., activation near one) only when the corresponding deduction can be made using the domain knowledge.[3] Likewise, when the deduction cannot be made using the knowledge base, then the corresponding unit should be inactive (i.e., have activation near zero).

KBANN translates conjunctive rules into a neural network by setting weights on all links corresponding to positive (i.e., unnegated) antecedents to $\omega$, weights on all links corresponding to negated antecedents to $-\omega$, and the bias on the unit corresponding to the rule's consequent to $(P-\frac{1}{2})\omega$. $P$ is the number of positive antecedents on a rule. KBANN commonly uses a setting of $\omega = 4$, a value we empirically found to work well. For example, Figure 5 shows a network that encodes: `A:- B, C, D, not(E).`

Intuitively, this translation method is reasonable. The weighted input from the links minus the bias can only exceed zero when none of the negated antecedents are true and all of the positive antecedents are true. This is the only case in which Equation 2, the logistic activation function, will result in a value near 1.0. For instance in Figure 5, if each of B, C and D have activations approximately equal to 1.0 and E has an activation approximately equal to 0.0 then the net incoming activation to A will be about $3\omega$ which is greater than $\frac{5}{2}\omega$. More generally, units encoding conjunctive rules will only be significantly active when all positively-weighted links carry a signal near one and all negatively-weighted links carry a signal near zero.

KBANN handles disjunctive rules, with two exceptions, in the same way it handles conjunctive rules. The first exception is that KBANN rewrites disjuncts as multiple rules with the same consequent in step 1 of the rules-to-network algorithm. The rewriting is necessary to prevent combinations of antecedents from activating a unit in situations where the corresponding consequent could not be deduced. For example, assume there exists a consequent $\Upsilon$ that can be proven by two rules, $R_1$ and $R_2$. Further assume, that there are seven unnegated antecedents (labeled to 1, ..., 7) to $\Upsilon$ and that [1 2 3] are antecedents for $R_1$ while [4 5 6 7] are antecedents for $R_2$. If the

[3] The translation of rules into neural structures has been described by McCulloch and Pitts [25] and Kleene [20] for units with threshold functions. Thus, the original contribution of this work is the idea of training networks that have been so constructed, rather than simply the construction of these networks.

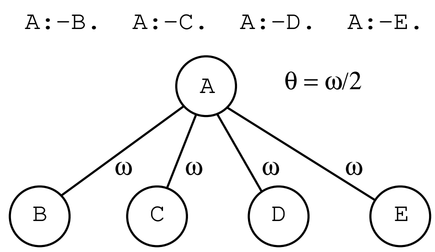



**Figure 6** Translation of disjunctive rules into a KBANN-net.

antecedents of $R_1$ and $R_2$ are all connected to $\Upsilon$ such that either [1 2 3] or [4 5 6 7] can activate $\Upsilon$, then there is no way to set the bias of $\Upsilon$ such that unwanted combinations (e.g., [1 3 4 7]) cannot also activate $\Upsilon$. KBANN avoids this problem by rewriting $R_1$ and $R_2$ as independent rules.

The second difference, from the conjunctive case, in the translation of disjunctive rules is that the bias in the unit encoding the consequent is always set to $\frac{\omega}{2}$. As in the conjunctive case, KBANN sets link weights equal to $\omega$. For example, Figure 6 shows the network that results from the translation of four disjunctive rules into a KBANN-net. Intuitively, this is a reasonable strategy; the incoming activation overcomes the bias when one or more of the antecedents is true.

### 3.7. Refining KBANN-nets

KBANN refines its networks using backpropagation [47], a standard neural learning method (other methods for the refinement of weights given classified examples could be used [59]). Unfortunately, KBANN-nets create problems for backpropagation because they start with confident answers (i.e., the output units have activation near zero or one), regardless of the correctness of those answers. This causes problems because under the standard formulation of backpropagation, when answers are confident, little change is made to the network regardless of the correctness of the answer. Rumelhart *et al.* argue this is a desirable property for standard neural networks because it means that they tend to be noise resistant [47, page 329]. However, when the outputs of networks are completely incorrect, this property makes it very difficult to correct the aspects of the networks that cause the errors.

In the general context of neural networks, several solutions that require only minor adjustments to backpropagation have been proposed for this problem [15]. (Other, more significant changes to backpropagation also address the problem [59].) The results for KBANN-nets described in this article use the *cross-entropy* error function – Eq. 4 – suggested by Hinton [15] rather than the standard error function – Eq. 3. The cross-entropy function interprets the training signal and network outputs as conditional probabilities and attempts to minimize the difference between these probabilities. In these equations, $a_i$ is the activation of output unit i, $d_i$ is the desired activation for unit i, and $n$ is the number of output units.

$$Error = \frac{1}{2}\sum_{i=1}^{n}(d_i - a_i)^2 \tag{3}$$

$$Error = -\sum_{i=1}^{n}[(1 - d_i) * log_2(1 - a_i) + d_i * log_2(a_i)] \tag{4}$$

In our experience, the cross-entropy error function has yielded improvements in both training time and generalization for KBANN-nets.

In addition to changing the definition of error, we experimented with augmenting the error function with a term that penalizes the network for making changes to the original domain theory by adding to Equation 4 the regularization term in Equation 5.

$$\mathit{regularizer} = \lambda \sum_{i \in \omega} \frac{(\omega_i - \omega_{init_i})^2}{1 + (\omega_i - \omega_{init_i})^2} \tag{5}$$

The $\lambda$ term in Equation 5 controls the tradeoff between the ability of the network to learn the training set and distance from the initial rules. The principal effect of the addition of this regularization term is to increase the interpretability of trained KBANN-nets by encouraging networks to leave their original weights unchanged. As interpreting trained networks is beyond the scope of this paper, none of the experiments reported here use Equation 5. However, it has proven beneficial to other work [35].

# 4. Experimental Testbeds

This section describes the two real-world datasets from the domain of molecular biology, in particular DNA sequence analysis, that we use as testbeds. (For the purposes of this article, it is sufficient to view DNA as a linear sequence of characters drawn from {`A, G, C, T`}. Biologists refer to these as nucleotides.) Molecular biology, and especially DNA sequence analysis, is an area well-suited for computational analysis. Because of the 'Human Genome Project' there is a large and growing database of sequenced DNA that has been extensively studied. Thus, there is a large body of training examples to work with. In addition, the Human Genome Project promises to sequence DNA faster than it can be studied in biological laboratories. As a result, there is a need for computer-based analysis that is able to augment the efforts of biologists.

We have previously used both datasets described below to demonstrate the usefulness of the KBANN algorithm [33, 57]. Following a description of the notation used in this paper, there are brief descriptions of the promoter and splice-junction datasets. (See [55] for detailed descriptions.)

## 4.1. Notation

The two biological datasets described below use a special notation for specifying locations in a DNA sequence. The idea is to number locations with respect to a fixed, biologically-meaningful, reference point. Negative numbers indicate sites preceding the reference point while positive numbers indicate sites following the reference point. Figure 7 illustrates this numbering scheme.

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

In addition to this notation for specifying locations in a DNA sequence, Table 4 specifies

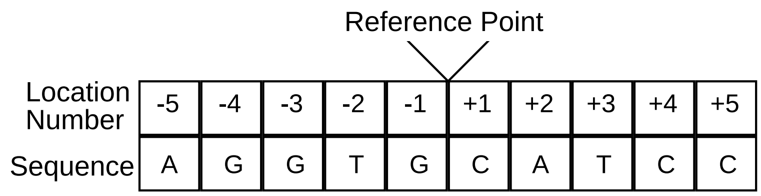



**Figure 7** Location numbering in DNA sequences.

**TABLE 4** Single letter codes for expressing uncertain DNA sequence locations.

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

a standard notation for referring to all possible combinations of 'nucleotides' using a single letter [17]. They are compatible with the codes used by the EMBL, GenBank, and PIR data libraries, three major collections of data for molecular biology.

### 4.2. Promoter recognition

The first testbed used in this article is prokaryotic *promoter recognition*. (Prokaryotes are single-celled organisms that do not have a nucleus — e.g., *E. coli*.) Promoters are short DNA sequences that precede the beginnings of genes. Thus, an algorithmic method of promoter recognition would make it possible to better identify the starting locations of genes in long, uncharacterized sequences of DNA. Promoters can be detected in "wet" biological experiments; they are locations at which a protein named *RNA polymerase* binds to the DNA sequence.

The input features for promoter recognition are a sequence of 57 consecutive DNA nucleotides. Following biological convention, the reference point for promoter recognition is the site at which gene transcription begins (if the example is a promoter). The reference point is located seven nucleotides from the right of the 57-long sequence. Thus, positive examples contain the first seven nucleotides of the transcribed gene.

Table 5 contains the initial rule set used in the promoter recognition task. For example, according to these rules there are two sites at which the DNA sequence must bind to *RNA polymerase* – the *minus 10* and *minus 35* regions. (These regions are named for their distance from the reference point.) These rules were derived from the biological literature [34] by M. Noordewier [57]. The rule set in Table 5 is translated by KBANN into a neural network with the topology shown in Figure 8. Recall that KBANN adds additional low-weighted links (not shown) so that if additional sequence information is relevant, the algorithm can capture that information during training.

The training examples consist of 53 sample promoters and 53 nonpromoter sequences. Prior to training, the rules in Table 5 do not classify any of the 106 examples as promoters. Thus, the rules are useless as a classifier. Nevertheless, they do capture a significant amount of information about promoters.

The promoter recognition problem, which we previously introduced [57], has become

**TABLE 5** Initial rules for promoter recognition. ('*' matches to anything at the sequence location.)

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

minus-35 :- @-37 'CTTGAC'.             minus-10 :- @-13  'TA*A*T'.
minus-35 :- @-36  'TTGACA'.            minus-10 :- @-12   'TA***T'.
minus-35 :- @-36  'TTG*CA'.            minus-10 :- @-14 'TATAAT'.
minus-35 :- @-36  'TTGAC'.             minus-10 :- @-13  'TATAAT'.

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

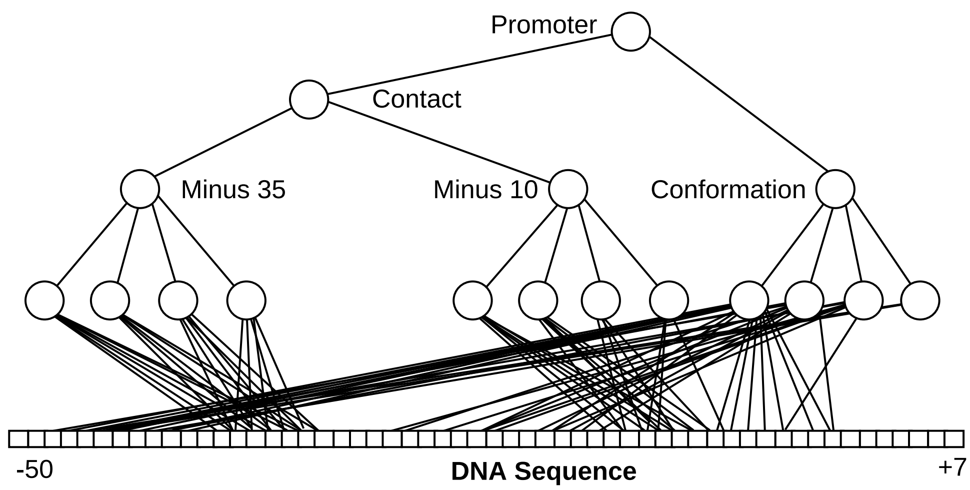



**Figure 8** The initial KBANN-net for promoter recognition (low-weighted links are not shown). Each box at the bottom of the figure represents one sequence location which is encoded by four input units.

one of the standard problems for theory-revision systems. It has been used to test inductive theory-refinement systems (e.g., LABYRINTH [54] and EITHER [36]) and inductive logic-programming systems (e.g., FOCL and GRENDEL [7]). It has also been used as a test problem for exemplar-based learning systems (e.g., [8]). However, to our knowledge no system has exceeded the performance we report in this article.

## 4.3. Splice-junction determination

The second testbed is eukaryotic *splice-junction determination*. (Unlike prokaryotic cells, eukaryotic cells have a nucleus.) Splice junctions are points on a DNA sequence at which the cell removes superfluous DNA during the process of protein creation. The problem posed in this dataset is as follows: given a sequence of DNA, recognize the boundaries between *exons* (the parts of the DNA sequence retained after splicing) and *introns* (the intervening parts of the DNA sequence that are spliced out). This problem consists of two subtasks: recognizing exon/intron boundaries (referred to in this paper as "E/I" sites — biologists call these sites "donors"), and recognizing intron/exon boundaries ("I/E" sites — "acceptors").

**TABLE 6** Initial rules for splice-junction determination.

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

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

I/E :- pyramidine-rich, @-3 'YAGG', not(I/E-stop).
pyramidine-rich :- 6 of (@-15 'YYYYYYYYYY').
For i from (-30 to +30 skipping 0)
           {@<i> 'Y' :- @<i> 'C'.  @<i> 'Y' :- @<i> 'T'.}

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

See Table 4 for meanings of letters other than A, G, T, C. The construct "`For i ...`" creates 120 rules that define a disjunct at each location in the input. Consequents with an antecedent of the form `'n of (...)'` are satisfied if at least *n* of the parenthesized antecedents are true.

This dataset contains 3190 examples, of which approximately 25% are *I/E*, 25% are *E/I* and the remaining 50% are *neither*. (Due to processing constraints, most of our tests involving this dataset use a randomly-selected set of 1000 examples.) Each example consists of a DNA sequence, which is 60 nucleotides long, categorized according to the type of boundary at the center of the sequence; the center of the sequence is the reference location used for numbering nucleotides.

In addition to the examples, information about splice-junctions includes the set of 23 rules in Table 6. (This count does not include the rules defined by the iterative construct `''For i from ...''` which define the meaning of `'Y'`.) M. Noordewier [33] derived this set of rules from the biological literature [60]. Using these rules, networks are configured with output units for the categories *I/E* and *E/I*. The third category (*neither*) is considered true when neither *I/E* nor *E/I* exceeds a threshold. (All tests reported in this work use 0.5 as the threshold.) These rules classify 61% of the examples correctly. As with promoter recognition, the success rate of the initial splice-junction rules is due largely to their tendency to "just say no"; the rules correctly classify only 40% of the I/E and 3% of the E/I examples. Figure 9 depicts an abstracted version of a KBANN-net constructed from these rules.

# 5. Empirical Tests of KBANN

This section details three sets of empirical tests that explore *how well* KBANN works as opposed to Section 3, which describes *how* KBANN works. In this empirical study we address the following questions:

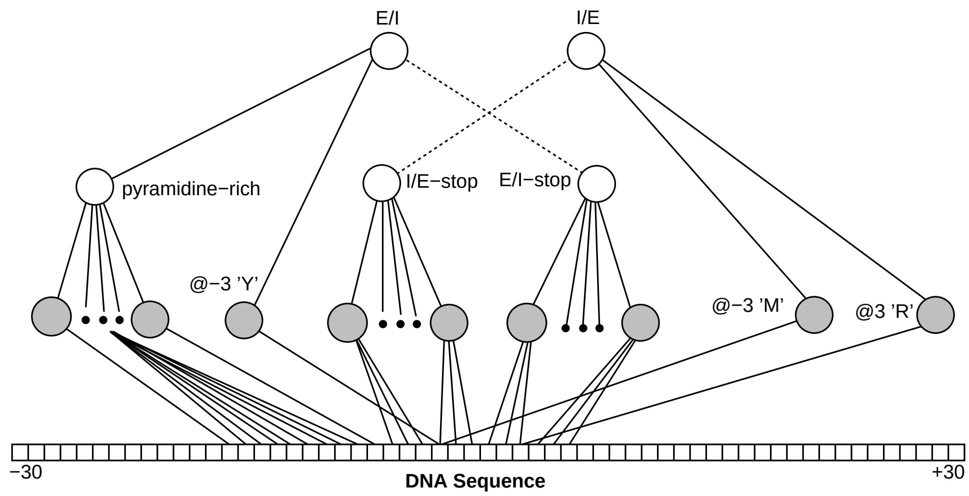



**Figure 9** The initial splice-junction KBANN-net. Each box at the bottom of the figure represents one sequence location which is encoded by four input units. Shaded units represent definitional features. By choice, they are "fixed" in that neither the weights of incoming links nor the bias may change during training.

- How does KBANN compare with:
  - purely inductive learners?
  - other hybrid systems?
  - solutions proposed in the biological literature?
- How much of KBANN's performance is dependent upon:
  - the identification of informative input features?
  - the identification of relevant derived features?
- How does KBANN's relative performance depend upon:
  - the number of training examples?
  - the quality of the domain theory?

Briefly, tests in Section 5.1, show that trained KBANN-nets are very effective classifiers by comparison to every other method we have tried. The next set of tests, also in Section 5.1, show that the relative advantage of KBANN over standard backpropagation (KBANN's underlying empirical algorithm) varies inversely with the number of training examples. Following this, tests in Section 5.5 show that it is the combination of feature identification and derived feature construction that are responsible for KBANN's relative effectiveness. Finally, in Section 5.9, lesion studies show that our approach is robust to errors in the initial domain theory.

### 5.1. KBANN versus other learning systems

Tests in this section explore the hypothesis that KBANN is an effective and efficient learning algorithm, in terms of its ability to generalize to examples not seen during training. We compare KBANN, using Section 4's splice-junction and promoter testbeds, to six empirical learning systems and two systems which learn from both theory and data.

## 5.2. KBANN versus empirical learners

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

### Seven-way comparison of algorithms

In this section, KBANN is compared to six empirical learning algorithms: standard backpropagation [47], ID3 [42], "nearest neighbor", PEBLS [8], Perceptron [46], and Cobweb [11][4]. For standard backpropagation, we use networks with a single, completely-connected layer of hidden units in which weights are initialized to randomly selected values with an absolute value of less than 0.5. (Networks for promoters and splice-junctions respectively have 23 and 24 hidden units.) This topology generalizes as well as, or better than, all others we tried during a coarse search of topology space. The results of these tests show that, in almost every case, KBANN is statistically-significantly superior to these strictly-empirical learners.

**Method.** Following Weiss and Kulikowski's suggestion [61] we evaluate the systems using cross-validation[5]. For the 106-example promoter dataset we use leaving-one-out cross-validation. "Leaving-one-out" is an extreme form of cross-validation in which each example is successively left out of the training set. So, for promoter recognition leaving-one-out requires 106 training passes in which the training set has 105 examples and the testing set has one example. Leaving-one-out becomes prohibitively expensive as the number of available examples grows. Weiss and Kulikowski [61] suggest, as the size of the training set grows, that 10-fold cross-validation yields generalization results that are equivalent to leaving one out. Hence, we use 10-fold cross-validation for the 1000-example splice-junction determination dataset.

In N-fold cross-validation the initial ordering of examples affects the split into training and testing examples. As a result, testing on the splice-junction problem uses eleven different example orderings. The error rates we report are simple averages of the eleven trials. Training and testing set membership is not affected by the initial ordering of examples in leaving-one-out testing as the test set always consists of a single example.

Every algorithm other than ID3 and nearest neighbor is sensitive to the order of example presentation[6]. Hence, the tests of algorithms other than ID3 and nearest neighbor, use eleven different orderings of the 106 promoter examples. Rather than

[4]These tests all use implementations written at Wisconsin that are based on the descriptions in the cited references. The lone exception is Cobweb, for which we used Gennari's CLASSIT code [14].

[5]Note that we use the term "cross-validation" in the sense used by Weiss and Kulikowski [61] to indicate a testing methodology in which the set of examples are permuted and divided into $N$ sets. One division is used for testing, the remaining $N-1$ divisions are used for training. The testing division is *never* seen by the learning algorithm during its training. This procedure is repeated $N$ times so that every partition is used once for testing. This is a different definition for cross-validation than is often found in the neural networks literature, in which a "cross-validation" set is used during training to prevent overfitting of the training set.

[6]Backpropagation is not sensitive to presentation order when weights are updated only after the presentation of all of the training examples. In all of our tests, weights are updated after every example.

simply recording the average number of errors over the eleven trials, statistics are maintained for each example. Examples are counted as incorrect only if they are incorrectly classified on the majority of the eleven trials. This error-scoring method captures the best result of each algorithm; it always results in a decrease in the number of errors. For example, Cobweb is quite sensitive to presentation order. It misses many examples two or three times, but misses few examples more than six times. Hence, the simple average error rate for Cobweb[7] is more than double than the level reported in Figure 10. Conversely, Perceptron is relatively insensitive to presentation order. Hence, this definition of incorrectness has little effect on Perceptron; its error rate drops by less than one example. The effect on other algorithms is between these extremes, reducing the error rate by from one to two examples[8].

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

Finally, the initial randomization of weights in a neural network can affect learning. To compensate for this, an extra level of experimentation is required for both standard backpropagation and KBANN. Specifically, each of the eleven example permutations of each dataset is tested using ten different initial states. Hence, each test requires 110 separate runs of N-fold cross-validation.[9] The average of the output unit activations from the ten trained networks determines the classification. So, if the output unit of a promoter recognition network is 0.60 on nine trials and 0.01 on the tenth trial, then the classification of the network is based upon the relationship between 0.55 and the threshold. For all tests, the threshold was set to 0.5.

We train networks until one of three following stopping criteria is satisfied:

1. On 99% of the training examples, the activation of every output unit is within 0.25 of correct.
2. Every training example is presented to the network 100 times. (I.e., the network has been trained for 100 *epochs*.)
3. The network is classifying at least 90% of the training examples correctly but has not improved it ability to classify the training examples for five epochs. (This implements the *patience* stopping criterion [10].)

In general, networks trained for promoter recognition stopped training on the first criterion – they are able to perfectly learn the training data. On the other hand,

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

[8] Due to processing time considerations, all tests of the promoter domain after Figure 11 use simple average error rates and 10-fold cross-validation.

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

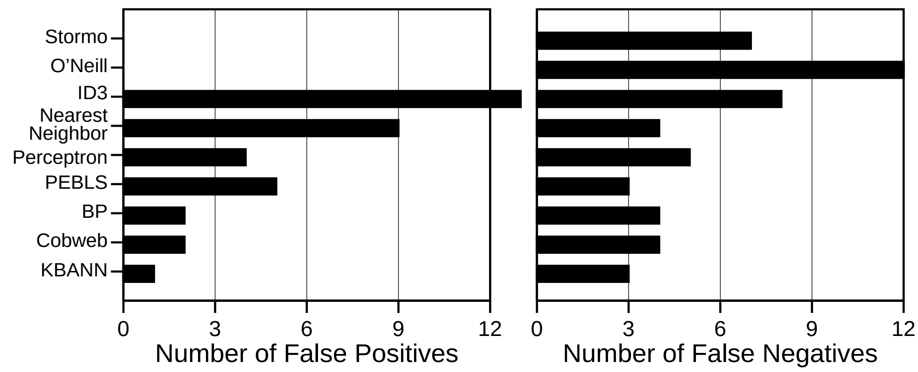



**Figure 10** Test-set performance on the promoter recognition task assessed using leaving-one-out cross-validation.

KBANN-nets for splice-junction determination often terminate on the second or third criterion because it is difficult for them to perfectly learn the training data.

These termination criteria are stringent enough so that it is possible that the networks could overfit the training set, and thereby impair generalization. Hence, in a separate experiment, we empirically tested for overfitting by assessing generalization after each epoch of training. Networks were trained with this periodic generalization assessment until they reached one of the three above criteria. If the networks had overfit the training data, then we would have observed a decline in generalization at the end of the training period. Although generalization often peaked before the end of training, we never observed a significant or consistent decline. Because we were unable to detect overfitting, we did not pursue any techniques for its prevention. For example, we did not prune the decision trees created by ID3 [27]. Nor did we use a "tuning [cross-validation] set" [63] to decide when to stop training our neural networks.

**Results and discussion.** Figures 10 and 11 present the comparisons of KBANN to the six empirical-learning algorithms listed above[10]. In addition to these empirical algorithms, Figure 10 contains the accuracies of two methods suggested by biologists for promoter recognition. One of these methods was suggested by Stormo [53]. This method learns through the presentation of a set of examples of some category. It counts how many times each nucleotide appears at each position. These counts are used to build a scoring matrix, which is then applied to new sequences; those that score above some threshold (we used a threshold of zero) are predicted to be members of the class being learned.

The other method suggested by a biologist is a non-learning, hand-refined, technique for differentiating between promoters and non-promoters described by O'Neill [34]. He analyzed a set of promoters and produced a collection of filters to be used for promoter recognition. If a sequence properly matches the filters, it is classified as a promoter. We directly implemented this approach. (Note that although we derived the promoter domain theory from the conclusions of O'Neill's paper; it is not identical to the promoter-finding technique he proposed.)

KBANN generalizes better than every empirical learning system tested on both of the biological domains. In most cases, differences are statistically significant with 99.5% confidence based on a one-tailed *t*-test. The only exceptions to this generalization

[10]Cost and Salzburg [8] report making only four errors on the promoter testbed. However, we were unable to reproduce their results with our implementation of PEBLS.

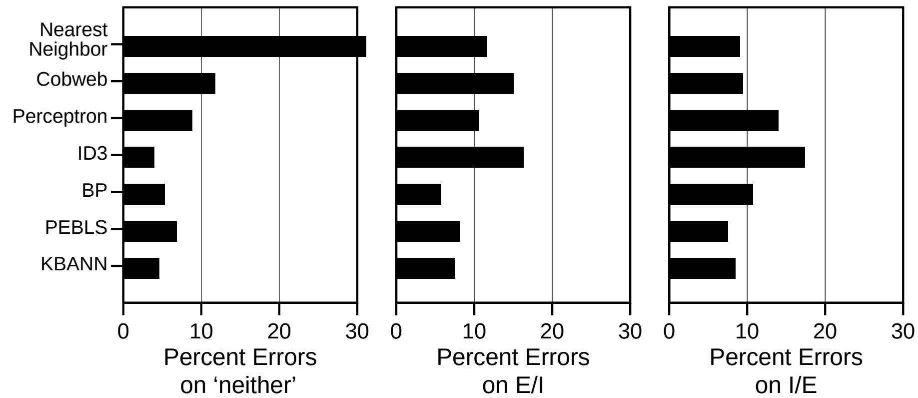



**Figure 11** Test-set performance, assessed using 10-fold cross-validation, on the splice-junction determination task with 1000 training examples.

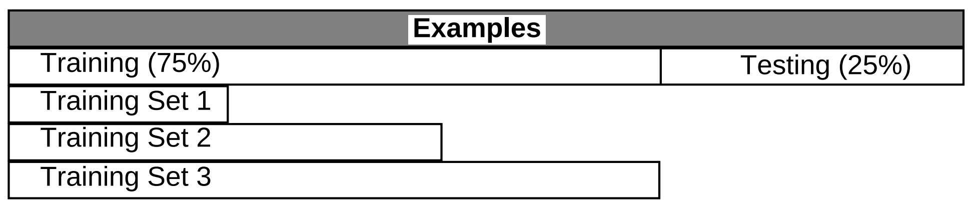



**Figure 12** The partitioning of a set of examples to form a learning curve.

are that KBANN is only marginally better than standard backpropagation and PEBLS on the splice-junction problem. We attribute the comparatively poor performance of KBANN on this problem to the sparseness of the initial theory of splice junctions (see Table 6 or Figure 9).

### Learning from small sets of examples

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

**Method.** These tests compare KBANN only to standard backpropagation, as the above tests show backpropagation to be the most effective of the systems for learning from examples only. “Learning curves” are built by splitting the set of examples into two subsets, one containing approximately 25% of the examples and the other containing the remaining examples. The 25% set is put aside for testing. Then, the 75% set is partitioned into sets of increasing size, such that smaller sets are always subsets of larger sets. Networks are trained using subsets of the 75% set and tested using the 25% set. Figure 12 illustrates the method of example partitioning we use to form learning curves.

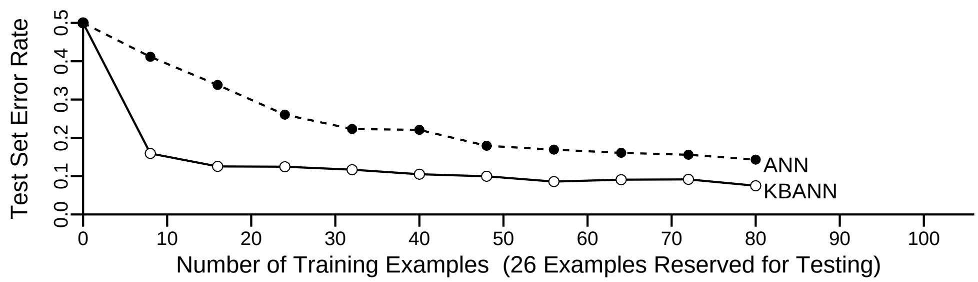



**Figure 13** Learning curves for the promoter domain with 106 examples. For KBANN, zero training examples represents the accuracy of the initial theory. For ANN, zero training examples represents random guessing.

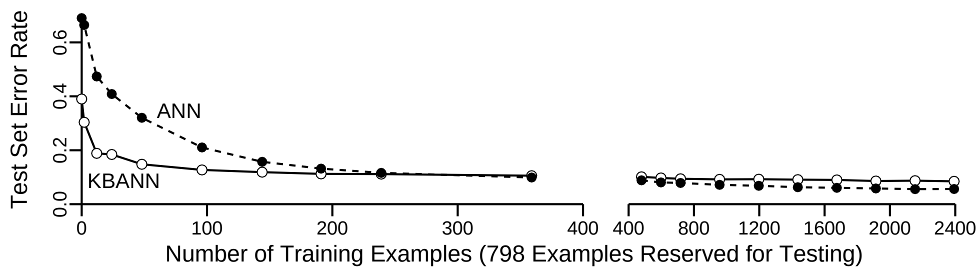



**Figure 14** Learning curves for the splice-junction domain with 3190 examples. For KBANN, zero training examples represents the accuracy of the initial theory. For ANN, zero training examples represents random guessing.

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

**Results and discussion.** The results, presented in Figures 13 and 14, verify the hypothesis that KBANN is more efficient than standard backpropagation in terms of its ability to extract accurate generalizations from a small set of training examples. For both the promoter and splice-junction datasets, the test set error rate on small sets of training data is about 20 percentage points less for KBANN than for standard backpropagation. The difference steadily declines on both datasets as the number of training examples increases. On the promoter set (Figure 13), KBANN has an advantage of about four percentage points with respect to standard backpropagation after training using the largest training set. On the other hand, KBANN has a two percentage point disadvantage with respect to backpropagation on the splice junction set after training on the largest available set of training examples.[11]

A different way of analyzing these learning curves is in terms of how quickly error rates reach the levels attainable using every training example. On the promoter dataset, KBANN-nets require only about 15 training examples to achieve an error rate within five percentage points of that achieved with the full set of 80 training examples. On the other hand, standard backpropagation requires more than 50 training examples to

[11] At 900 training examples, in Figure 14, KBANN is at a 1.2 percentage point disadvantage to standard ANNs. This contrasts with results reported in the seven-way comparison that show KBANN slightly superior to standard ANNs when trained with 900 examples. This dichotomy of results reflects a difference in error counting. Figure 14 uses a simple average whereas the Figure 11 uses the more complex method described earlier in this section.

approach the levels it achieves with 80 training examples. The results on the splice-junction problem are similar. KBANN requires roughly one-third the number of training examples required by backpropagation to reach error rates on testing examples that are close to the levels asymptotically attained by each system.

**Discussion of the comparison of KBANN to empirical learning algorithms**
The results reported in Figures 10 and 11 describe only the asymptotic performance of each system. On the basis of these comparisons, KBANN is the most effective learning algorithm. However, many problems are characterized by a paucity of data. Hence, the performance of systems when there is little data available is at least as important as the performance when a large collection of examples is available. Thus, the learning curves in Figures 13 and 14, which compare the performance of backpropagation to that of KBANN, are at least as important as the results in Figures 10 and 11.

### 5.3. KBANN versus theory and data learners

Tests in this section compare KBANN to two systems that learn from both theory and data. The first system is EITHER [36]. This system uses ID3 to make minimal corrections to domain theories. The second system is Labyrinth-k [54], an extension of Labyrinth that allows the insertion of domain knowledge. Labyrinth is, in turn, based on Cobweb [11]. (See Section 6.1 for a more detailed description of these algorithms.) Like KBANN, these systems are able to use initial knowledge that is both incorrect and incomplete as the basis for learning.

**Method.** Results for both Labyrinth-k and EITHER were supplied by the authors of the respective systems, and appear in published sources [36, 54]. Fortunately, the systems were tested using virtually identical procedures. The testing of KBANN closely follows the method used by these systems.

The systems were tested on only the promoter dataset. To our knowledge, neither Labyrinth-k nor EITHER have been tested on the splice-junction problem. EITHER cannot currently be applied to the splice-junction problem because it is unable to handle negated antecedents. The methodology for constructing these learning curves is very similar to that of the prior section. Specifically, the initial step was to partition the promoter dataset into a test set of 26 examples and a training set of 80 examples. We further partitioned the training set into sets of 10, 20, . . . , 80 examples in which smaller sets are subsets of larger sets. These subsets were used for training. To smooth statistical fluctuations, we repeated this procedure five times.

**Results.** Figure 15 plots the predictive accuracy of each of the three hybrid learning systems. As seen in this figure, KBANN is consistently at least 5 percentage points better than both of the other systems. Aside from a blip in the initial performance of Labyrinth-k, EITHER consistently has the poorest performance on this problem.

### 5.4. Discussion of the empirical comparisons

The results of the tests in this section suggest that KBANN generalizes better than both (a) systems that learn only from examples (i.e., empirical learners) and (b) systems that, like KBANN, learn from both theory and data. KBANN's relative weakness on the splice-junction dataset can be attributed to two factors. First, the splice-junction domain theory is sparse. It allows for little more than Perceptron-like learning. Hence, one might expect that a richer domain theory would improve the performance of KBANN-

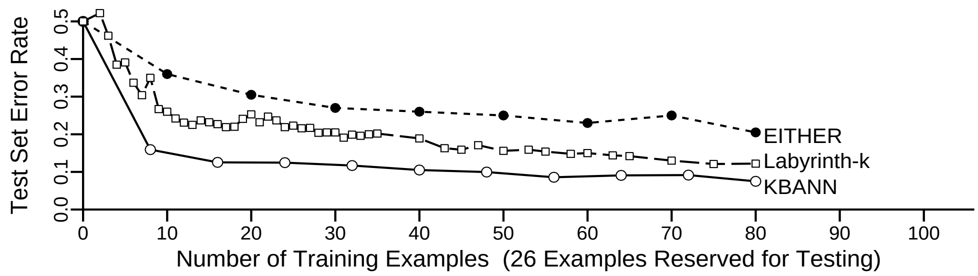



**Figure 15** Accuracy of three hybrid learning systems for promoter recognition.

nets. This hypothesis is yet to be tested, as there is no better domain theory for splice junctions.

A second hypothesis to explain the relatively poor performance of KBANN on the splice-junction problem is that standard backpropagation better utilizes the large dataset available in the splice-junction domain. An alternate view of this hypothesis is that the domain theory prevents the network from learning a solution which differs significantly from the solution proposed by the theory. This hypothesis suggests that standard backpropagation should outperform KBANN only when large amounts of training data are available. The learning curve in Figure 14 supports this hypothesis.

## 5.5. Sources of KBANN's Strength

Taking a *very* narrow view, the results in the prior section indicate that, given just the right amount of training data, KBANN generalizes better than systems that learn from theory and data (Figure 15) and systems that learn from data alone (Figures 10 and 11). This section further explores these results. In particular, the first subsection tests (and rejects) the hypothesis that the difference between KBANN and other hybrid learning systems is due only to differences in the learning algorithms underlying each system. The following subsection investigates two hypotheses that attempt to explain why KBANN improves upon standard backpropagation, its underlying empirical learning algorithm.

## 5.6. Comparing KBANN to other theory & data learners

One hypothesis to account for KBANN's advantage with respect to EITHER and Labyrinth-k (on the datasets we tested) is that the difference is due wholly to KBANN's underlying learning algorithm. In other words, KBANN outperforms both EITHER and Labyrinth-k because backpropagation generalizes better than both ID3 and Labyrinth on our molecular-biology datasets. Data presented earlier (Figures 10 and 11) support this contention; these figures show that backpropagation outperforms both ID3 and Cobweb[12] on both the promoter and splice-junction datasets. Figure 16 compares backpropagation, ID3, and Cobweb using the same experimental methodology as Figure 15. This figure lends further credence to this hypothesis concerning the relative ability of backpropagation. In addition, a number of empirical studies [50, 2] suggest that, over a wide range of conditions, backpropagation is as good or better than other empirical learning systems at classifying examples not seen during training. Therefore, it is reasonable to conclude that some of the power of KBANN is due simply to its underlying empirical learning algorithm.

Yet, Figure 17 indicates that there is more to KBANN than simply backpropagation. This figure plots the improvement of each hybrid system over its underlying empirical algorithm as a percentage of the total possible improvement. It shows that KBANN

[12] Labyrinth slightly outperforms Cobweb [54].

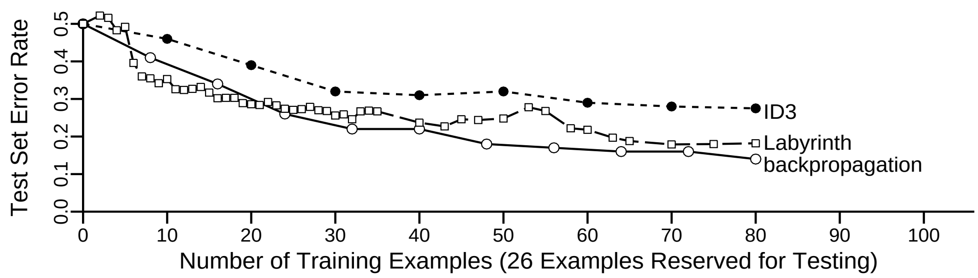
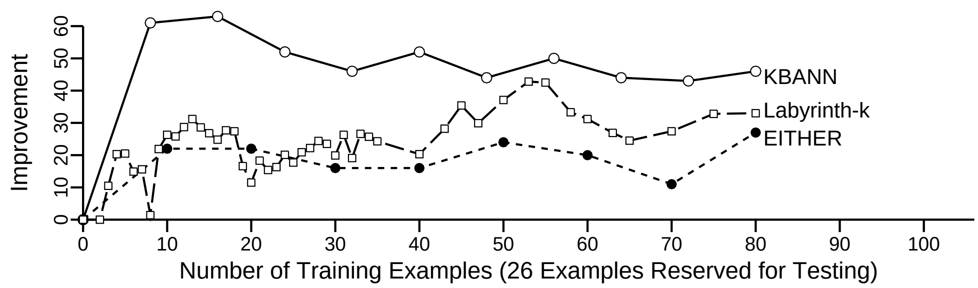



**Figure 16** Accuracy of empirical algorithms underlying hybrid learning systems on the promoter data.




$$\textit{improvement} \equiv 100 * \frac{<\textit{error rate of underlying algorithm}> - <\textit{error rate of hybrid algorithm}>}{<\textit{error rate of underlying algorithm}>}$$

**Figure 17** Improvement of hybrid learning systems over their underlying empirical algorithms.

proportionally improves more on its underlying learning algorithm than both EITHER and Labyrinth-k. In other words, KBANN makes more effective use of the domain knowledge than the other two systems. This result refutes the above hypothesis that KBANN's effectiveness is due *wholly* to the relative superiority of its underlying learning algorithm.

### 5.7. Comparing KBANN to standard backpropagation

The results just presented indicate that KBANN's superiority to other hybrid systems is due partly, but not wholly, to its use of backpropagation. However, the results do not address why KBANN improves upon backpropagation. This is discussed below.

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

1. *Structure is responsible for* KBANN*'s strength.* That is, the topology of a KBANN-net is better suited to learning in a particular domain than a standard ANN (i.e., an ANN with a single layer of hidden units and complete connectivity between layers).
2. *Initial weights are responsible for* KBANN*'s strength.* That is, the initial weights of a KBANN-net select critical features of the domain, thereby simplifying learning and reducing problems resulting from spurious correlations in the training examples.

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

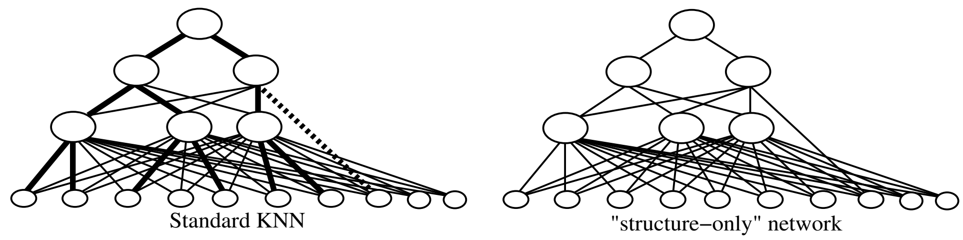



**Figure 18** Standard KBANN-nets versus structure-only networks.

**Structure is responsible for KBANN's strength**
If the first hypothesis is correct, then a network that has the structure given by the rules, but initially random weights, should be as effective at predicting unseen examples as a standard KBANN-net.

**Method.** We test this hypothesis by creating networks that have the exact structure of a KBANN-net, but with all link weights and biases randomly distributed around zero. The learning abilities of these "structure-only" networks can then be compared to that of standard KBANN-nets. Figure 18 graphically depicts the difference between standard and structured networks.

This experiment uses the following procedure to tightly control the results.

```
Create a KBANN-NET.
Duplicate the network.
In the duplicate network, reset all link weights set by the domain
   knowledge to randomly selected values within 0.5 of zero.
In the duplicate network, reset all biases to  randomly selected
   values within 0.5 of zero.
Test the pair of networks using eleven repetitions of ten-fold
   cross-validation. (Use the same example orderings as those used
   in previous experiments.)
```

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

**Results.** Figure 19 contains the results of these tests. It reports two sets of data: generalization performance and training effort (measured in terms of number of epochs required for training). The results clearly refute the hypothesis that the strength of KBANN arises solely from the determination of network topology. Not only is the structured network worse at generalizing to promoter testing examples, it is slower to train than standard KBANN-nets on both domains. The differences are statistically significant with 99.5% confidence according to a one-tailed, paired-sample *t*-test.

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

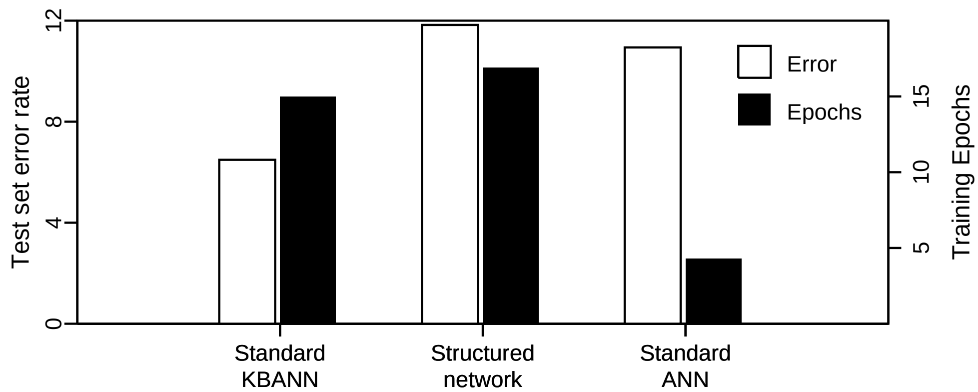



**Figure 19** Learning by standard KBANN-nets and structure-only nets.

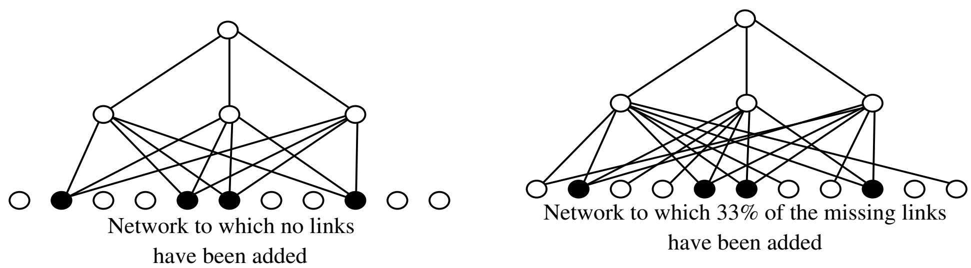



**Figure 20** The creation of networks to test the hypothesis that initial weights are a significant contributor to KBANN. (Darkened input units indicate features identified as important by the domain theory.)

### Initial weights are responsible for KBANN's strength

This section tests the second hypothesis, that the strength of KBANN results from the weights on links in KBANN-nets identifying relevant features of the environment. We were unable to devise a direct test of this hypothesis. Hence, we reformulated this as follows: If feature identification is the source of KBANN's strength, then an ANN that is preferentially connected to the features identified by the domain theory should be as effective at generalization as a KBANN-net.

**Method.** To test this reformulated hypothesis, we used standard ANNs except that the single layer of hidden units was initially connected to only those input units explicitly mentioned by the domain knowledge (as illustrated in Figure 18). In the promoter domain, this creates a network with about $\frac{1}{4}$ of the number of links in a fully-connected network.

To this network, we added the missing links between the input and hidden layers in 5% increments (by random selection) until the hidden and input layers were fully connected. At each 5% step of the missing links were added, the network was copied, and the copy was put aside for experimentation. Figure 20 illustrates this link addition procedure when 33% of the links have been added.

To smooth fluctuations resulting from the addition of particularly important links, we re-

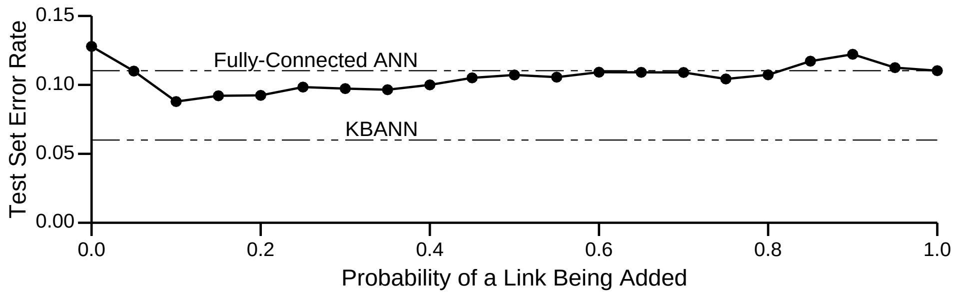



**Figure 21** The classification performance of standard ANNs that initially have links only to those features specified by the promoter domain theory.

peated this procedure three times, thereby creating 60 networks. These 60 networks were trained using ten-fold cross-validation and the standard eleven permutations of the training examples.

**Results.** Figure 21 plots the average error rate of these "missing link" networks. Networks which have only links identified by the initial domain knowledge are worse at generalizing to testing examples than both KBANN-nets and standard ANNs. However, by the time 5% of the missing links have been added, the missing-link networks outperform standard ANNs. Missing-link networks remain superior to standard ANNs until 80% of the missing links have been added. From 80% to 95% of the links added, the missing-link networks are slightly inferior to fully-connected networks. The inversion between 80% and 95% appear to be anomalous; it is not statistically significant. The performance of missing-link networks peaks when 15% of the missing links have been added. This best-case performance is significantly inferior to that of KBANN (with 99.5% confidence using a one-tailed *t*-test). Thus, these results indicate that some, but not all of the improvement of KBANN over standard backpropagation, results from its identification of important input features.

### 5.8. Discussion of the sources of KBANN's strength

These tests indicate that alone neither structure nor weight (focusing) account for the superiority of KBANN-nets over standard ANNs. Therefore, the third hypothesis, that it is a combination the structure and the focusing weights that give KBANN its advantage over backpropagation, is likely true.

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

### 5.9. Noise in domain theories

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

There has been little other work on the addition of noise to domain theories and the existing work does not completely investigate the problem space. For example, Richards tested the effect of the number of errors in a domain theory on his FORTE system [45]. Pazzani *et al.* [37] investigated the effect of several different types of domain-theory problems by introducing random faults into theories in much the style we describe below. Because neither of these papers was extant at the time of our experiments, the types of noise described here, and the methods used to approximate that noise, represent our independent attempt to identify and test important types of "domain-theory noise."

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

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

- *missing rules:* Rules required for the correct operation of the domain theory are missing. The effect of missing rules which are used either to predict an output class or as unnegated antecedents of other rules is to render some chains of reasoning impossible. As a result, a rule set that is missing some rules underpredicts the positive class.
- *added rules:* Rules not required for correct operation of the domain theory are present. The effect of added rules is to make possible proofs that should not occur. Hence, rule sets with extra rules overpredict the positive class.

Antecedent-level noise is independent of whether a rule set is missing rules or has unnecessary rules. Rather, antecedent-level noise looks at the problems that result from the improper statement of the antecedents of individual rules. This impropriety can occur in any of three forms:

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

**Method.** Tests for antecedent-level noise start with the initial domain theory, then probabilistically add noise (with a uniform distribution) to the theory. As with all other experiments involving increasing quantities, we add noise incrementally. That is, to a domain theory with no noise, the smallest noise fraction is added. Then, the slightly noisy rule set has further noise added to it so that the total noise is equal to the second smallest fraction. We repeat this procedure four times for each of the three kinds of antecedent noise.

We add antecedent-level noise using the following procedure:

```
For each antecedent AA of every rule:
  Probabilistically decide whether or not to add noise
  If the decision is yes then add noise using one of the
   following procedures:
   ADD: Randomly select an antecedent to be added from
       among the input features, and any consequents
       'below' - where 'below' is defined by the labeling
       procedure of Section 3.3.1 - the consequent of the
       rule being considered.  Add the selected antecedent
       to the rule of which the scanned antecedent is a member.
   DELETE: Simply delete AA from the rule of which it is a member.
   INVERT: Negate AA. If AA is already negated, delete the negation.
```

We use a similar incremental procedure to probabilistically add rule-level noise to the domain theory. We expand domain theories by creating new rules that are simple conjunctions of randomly-selected input features. The number of antecedents to each putative rule is a random number in the range [2...6].[13] We add the newly-created rules to the domain theory only to existing disjuncts in the domain theory. The reason for this restriction is to allow only minimal changes to the structure of the rule set. Two other kind of additions are possible but not used. First, rules could be added to the conditions of a conjunct. This would require adding antecedent-level noise in addition to rule-level noise. Second, rules could be added that create new ways to prove a consequent that is not disjunctive. This sort of addition requires significant alterations to the hierarchical structure of the rules.

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

**Results and discussion.** Figure 22 presents the results of increasingly adding antecedent-level noise to the promoter domain theory. As might be expected, inverting antecedents has the largest effect of the approaches to inserting antecedent-level noise. After the addition of 30% of this type of noise to the domain theory, the resulting KBANN-net generalizes worse than a standard ANN. On the other hand, irrelevant antecedents have little effect on KBANN-nets, with 50% noise (that is for every two antecedents specified as important by the original domain theory, one spurious antecedent was added) the performance of KBANN-nets is still superior to that of a standard ANN. Not appearing in Figure 22 are tests in which noise of all three types was added to the domain theory. Not surprisingly, the resulting KBANN-nets perform at about the average of the three types of noise individually.

Results were much the same for rule-level noise, as Figure 23 indicates. In this case, up to 10% of the initial rules can be deleted while leaving KBANN-nets superior to ANNs. The addition of randomly created rules has little effect on the accuracy

[13]We use the range [2...6] because it contains the number of antecedents in most of the rules in the promoter and splice-junction domain theories.

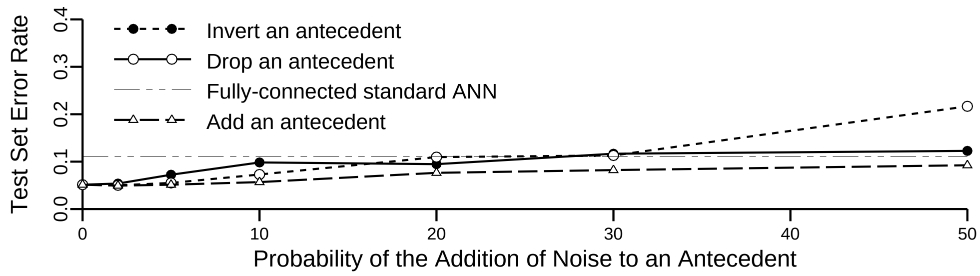



**Figure 22** The effect of antecedent-level noise on classification performance of KBANN-nets in the promoter domain.

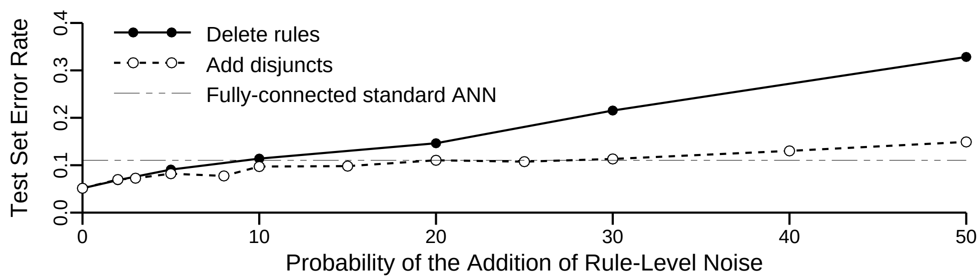



**Figure 23** The effect of rule-level noise on classification performance of KBANN-nets in the promoter domain.

of KBANN-nets. The inability of added rules to influence the correctness of KBANN-nets is not surprising. The added rules have an average of four, randomly-chosen, antecedents. Hence, these rules can be expected to match only one in every 256 patterns (this assumes a DNA sequence analysis domain where each feature has four possible values). As a result, the chances are good that the rules never match the training or testing patterns. Had the introduced rules matched the training patterns, the effect would have been similar to adding a single highly-weighted link to the input unit. Since Figure 22 indicates that KBANN-nets are relatively insensitive to this type of noise, the effect of the irrelevant rules can be expected to be minimal.

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

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

## 5.10. General Discussion of KBANN's Effectiveness

The results of the experiments presented in this section suggest a fairly concise definition of the abilities of KBANN. Namely, KBANN is more effective at generalizing to examples not seen during training than many empirical learners and two systems that learn from both theory and data. This superiority has at least two causes. First, KBANN is built upon a more effective empirical learning system than other systems that learn from both theory and data. Second, KBANN gets more out of the combination of theory and data than other systems.

A separate set of tests indicates that reasonably-accurate domain theories are sufficient to allow KBANN to create a network that generalizes well. Specifically, the results in Section 5.9 show that KBANN is most effective at learning from domain theories that are overly specific. This result is serendipitous; both the promoter and splice-junction domain theories are overly specific. Both domain theories we used for testing were extracted directly from the biological literature by a person who was unfamiliar with this property of KBANN.[14] Hence, the success of KBANN on these domains does not reflect an intentional bias in its favor. As tests on small, artificial domains show qualitatively similar results [51, 35, 56], we are optimistic that future tests will continue to show the success of the KBANN algorithm. Indeed, work by us and our colleagues on another biological domain (protein secondary-structure prediction) [23] and a process control problem [49] have shown the generality of the approach.

Finally, our tests show that KBANN's value is due to both the identification of informative input features and useful derived features (thereby establishing a good network topology). Neither focusing attention nor topology selection suffice alone.

# 6. Other Approaches to Hybrid Learning

We stated earlier that there has been much recent work in the field of hybrid learning (e.g., [6, 26, 30]). Rather than trying to review the whole field, in this section we will closely compare our approach to several representative approaches. The initial systems compared to KBANN are "all-symbolic" approaches. EITHER [36] and Labyrinth-k [54] will be used as examples of this approach. Following the all-symbolic approaches, we compare KBANN to inductive logic programming, using FOCL [37] as a representative of this field. We lastly compare KBANN to the work of Fu [13] and Katz [18] which we use as representatives of systems that, like KBANN, base their hybrid systems on neural networks.

## 6.1. All-symbolic approaches

EITHER [36] and Labyrinth-k [54] take a similar approach to forming a hybrid system, one that is in many ways similar to KBANN. Like KBANN, each of these systems uses a previously developed empirical learning algorithm to modify a user-supplied propositional domain theory. However, because these two systems use symbolic learning algorithms, they avoid the elaborate translation mechanism required by KBANN. As a result, the underlying empirical learner directly modifies the initial knowledge rather than a rerepresentation of that knowledge. Hence, after learning is complete, the modified domain theory is directly observable.

Labyrinth-k uses the initial knowledge to create a structure that Cobweb[15] [11], 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.

[14] At the time the theories were extracted, we too were unaware of this property.

[15] Labyrinth-k actually uses an extension of Cobweb that allows Cobweb to use "structured" domains [54].

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 [42], 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.

Both systems allow the initial knowledge to have arbitrary types of errors. This is a significant step forward form earlier systems (e.g., [22, 12]) which required that the domain theory have only certain classes of mistakes. Moreover, the systems are able to create knowledge (in the form of rules) when supplied with no initial information. In addition, both systems use the initial knowledge to focus learning on relevant features, thereby simplifying the learning problem. Hence, they can be expected to require fewer examples and be less affected by spurious correlations than standard empirical learning systems.

### 6.2. Systems based on inductive logic programming

There has been a recent proliferation of work in "Inductive Logic Programming" (e.g., [30]) often with the same goal as KBANN; namely, to refine an existing domain theory using classified examples. In addition, like KBANN, inductive logic programming has been applied to several problems in computational biology [31, 19]. Inductive logic programming systems are closely related to EITHER and Labyrinth-k except that they are not limited to propositional rules. As an example of these systems, we use FOCL [37].

FOCL is an extension of FOIL [43] that allows the inductive learning mechanism of FOIL to work on user-supplied background knowledge. Hence, FOCL is able to add and delete relations from existing rules as well as add entirely new rules to the existing domain theory. Using an ID3-like information gain heuristic, FOCL is able to correct arbitrary types of errors in the provided domain theory when given a set of positive and negative examples of the problem.

Tests on artificial domains show that FOCL, like KBANN, is tolerant of a variety of errors in the initial domain theory and the training examples [39]. However, FOCL is unable to directly handle the promoter domain theory because that theory classifies every training example in the same way. After a slight modification, FOCL is able to handle this problem, but its generalization is inferior to that of KBANN [7]. (Using 80 examples for training, FOCL has an error rate of 14.2% [7]. With the same number of examples, KBANN's error rate is 7.5%.)

### 6.3. Systems based on Neural Networks

Many hybrid systems have been developed recently that use neural networks as their basis. The focus of these systems has varied from reducing the number of examples required for robot learning [29], to probabilistic logics [24], to the propagation of Mycin-like certainty factors [21], to the refinement of systems for fuzzy logic [5]. Rather than trying to review these diverse system, we focus on two systems developed at about the same time as KBANN which took slightly different paths.

Fu [13] 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, his system is unable to use developments in the methods of training neural networks, (e.g., [59]).

Katz [18] describes a system which, much like KBANN, uses a knowledge base that has been mapped (apparently by hand) into a neural network. The focus of Katz's work 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 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.

### 6.4. Trends in hybrid systems research

There has been a pronounced trend in the development of hybrid systems. Specifically, constraints on the quality and form of information have been eased. Early systems such as UNIMEM [22] required correct theoretical information. While subsequent systems eased the correctness requirement, they introduced new constraints such as strict over-generality of the domain theory [12]. More recent systems such as KBANN allow arbitrary imperfections in theories. On the data side, 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 the need for more examples

In addition, a recent trend is the new hybrid systems tend to be built on top of an established empirical learning system. This is the case for Labyrinth-k, EITHER, FOCL and KBANN. Earlier systems tended to have specially-designed empirical components which were able to take advantage of the strong constraints on theory and data that these systems made.

# 7. Limitations and Future Work

While the empirical results in the previous section show that KBANN is an effective learning system, there is much unfinished about KBANN. Some of the more significant limitations of KBANN are described below, along with proposals for their elimination. (See [55] for additional open issues.)

- *The concepts* KBANN *learns are incomprehensible to humans.* Due to its neural representation, it is difficult to identify the basis for KBANN's classification of future examples. Moreover, trained neural networks cannot explain their decisions, so it is difficult to convince experts of the reliability of the system's conclusions. Also, it is difficult to transfer what KBANN has learned to the solution of related problems. (See [41] for a possible transfer method.) This problem can be partially addressed by more sophisticated training techniques and error functions. For instance, in addition to penalizing networks for incorrect answers, penalize the squared distance between the expert-assigned and trained weights [35]. Doing so should help to maintain the interpretability of the trained networks. Also, a network-to-rules translator that we have developed [55] directly addresses these problems.

- KBANN*'s rule syntax is limited.* KBANN can not handle rules with cycles or variables. A research area is the expansion of KBANN to admit certain type of cyclic rules [23]. However, it is unlikely that in the near future KBANN will be able to handle quantified variables in its rules, as there are currently no appropriate techniques for binding variables within neural networks. Inductive logic programming systems such as FOIL [43], FOCL [37] and GOLEM [19] allow relational rule sets; hence, they avoid the restrictions inherent to KBANN.

- *There is no mechanism for handling uncertainty in rules.* This point cuts in two ways. First, KBANN has no mechanism through for assigning different weights to antecedents. (See [24] for an approach.) Second, KBANN has no mechanism for expressing the partial truth of consequents. Mycin-like certainty factors [52] could be integrated into the rules-to-network translator [13, 21]. These factors would adjust link weights in the resulting network. Hence, links would have weights that depend both upon a global link weight and a certainty factor. This change in the definition of link weights would require modification of the method for setting the bias. However, little else in the rules-to-network translator would need to be changed.

- *Neural learning in* KBANN *ignores the symbolic meaning of the initial network.* As a result, training the networks created by KBANN may be unnecessarily difficult. A tighter integration of symbolic and neural learning may result in reduced training times and improved generalization. Preliminary experiments with an algorithm designed to more tightly integrate the symbolic and neural elements of KBANN support this hypothesis [55].

- *There is no mechanism for changing the topology of the network.* In addition to missing antecedents, rules are often also missing from a domain theory. As it is described here, the only way for KBANN to handle this problem is to somehow alter an existing rule so that it also represents the missing one. One the effects of such combination is to make the network very difficult to interpret. Such combination may also impair generalization because they force otherwise correct rules to change. One solution is to add a mechanism which proposes new rules and appropriately modifies the topology of the network [35].

In addition, almost all of the tests reported in Section 5 should be repeated with an artificial domain for which the theory and relevant features are known with certainty. An artificial problem would allow tightly controlled experiments that are not possible in the real-world problems studied in this paper. For instance, an artificial problem would allow a closely controlled experiment on the effects of irrelevant and missing antecedents. Yet, even without the tests made possible by an artificial problem, the results in Section 5 show KBANN to be a robust learning algorithm.

# 8. Final Summary

This article describes and empirically analyzes the KBANN system, a hybrid method for learning using both domain knowledge and classified training examples. KBANN uses a combination of "subsymbolic" and "symbolic" learning to create an algorithm that learns from examples and rules, one which generalizes better and is more efficient (in terms of the number of training examples required) than algorithms that make use of only one of these sources of information. KBANN operates by translating a propositional rule set into a neural network. In so doing, the topology of the network is set so that the network initially reproduces the behavior of the rules. The network is then trained using a set of classified examples and standard neural learning techniques. Empirical testing shows that a network so created and trained is effective at generalizing to examples not seen during training.

The principle empirical results showing the effectiveness of KBANN appear in Section 5. In that section, KBANN is shown to generalize better than six empirical learning algorithms, as well as two other hybrid algorithms. Moreover, other results presented in this paper show that the method is effective under a wide variety of conditions, ranging from few training examples to low-quality domain knowledge. Also, the results show that KBANN is effective at learning in two real-world domains. These DNA sequence-analysis 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 ideas underlying KBANN have also been used for protein folding [23] and process control [49].

While much work remain to be done on KBANN, as well as the more general topic of learning from both theory and data, the KBANN system represents a first step along a significant research path. In conclusion, this paper has shown that it is possible to profitably combine symbolic and connectionist approaches to artificial intelligence.

## Acknowledgements

This research is partially supported by Office of Naval Research Grant N00014-90-J-1941, National Science Foundation Grant IRI-9002413, and Department of Energy Grant DE-FG02-91ER61129.

For creating the two datasets we investigated, we especially thank Michiel Noordewier. In addition we would like to thank David Aha and Patrick Murphy at UC - Irvine for maintaining a database of machine learning problems. Both of the datasets we describe in this paper are available there.

Finally, we want to thank Richard Maclin, Charles Squires, Mark Craven, Derek Zahn, David Opitz, Sebastian Thrun, R. Bharat Rao, Lorien Pratt and two anonymous reviewers for their insightful commentary and, in some cases, programming assistance.

# References

[1] S. Ahmad, A Study of Scaling and Generalization in Neural Networks, Technical Report CCSR-88-13, University of Illinois, Center for Complex Systems Research (1988).

[2] L. Atlas, R. Cole, Y. Muthusamy, A. Lippman, J. Connor, D. Park, M. El-Sharkawi, and R. J. Marks, A peerformance comparison of trained multi-layer perceptrons and trained classification trees, *Proceedings of IEEE* **78** (1990) 1614–1619.

[3] J. Bachant and J. McDermott, R1 Revisited: Four Years in the Trenches, *AI Magazine* **5** (1984) 21–32.

[4] E. Barnard and R. A. Cole, A Neural-Net Training Program based on Conjugate-Gradient Optimization, Technical Report CSE 89-014, Oregon Graduate Institute, Beaverton, OR (1989).

[5] H. R. Berenji, Refinement of Approximate Reasoning-Based Controllers by Reinforcement Learning, in: *Proceedings of the Eighth International Machine Learning Workshop*, Evanston, IL (1991), 475–479.

[6] L. A. Birnbaum and G. C. Collins, eds, *Proceedings of the Eighth International Machine Learning Workshop*, Morgan Kaufmann, Evanston, IL (1991).

[7] W. W. Cohen, Compiling Prior Knowledge into an Explicit Bias, in: *Proceedings of the Ninth International Machine Learning Workshop*, Aberdeen, Scotland (1992), 102–110.

[8] S. Cost and S. Salzberg, A Weighted Nearest Neighbor Algorithm for Learning With Symbolic Features, *Machine Learning* **10** (1993) 57–78.

[9] T. G. Dietterich, Learning at the Knowledge Level, *Machine Learning* **1** (1986) 287–316.

[10] S. E. Fahlman and C. Lebiere, The Cascade-Correlation Learning Architecture, in: *Advances in Neural Information Processing Systems*, volume 2, Denver, CO (1989), 524–532.

[11] D. H. Fisher, Knowledge Acquisition via Incremental Conceptual Clustering, *Machine Learning* **2** (1987) 139–172.

[12] N. S. Flann and T. G. Dietterich, A Study of Explanation-Based Methods for Inductive Learning, *Machine Learning* **4** (1989) 187–226.

[13] L. M. Fu, Integration of Neural Heuristics into Knowledge-Based Inference, *Connection Science* **1** (1989) 325–340.

[14] J. Gennari, P. Langley, and D. Fisher, Models of incremental concept formation, *Artificial Intelligence* **40** (1989) 11–61.

[15] G. E. Hinton, Connectionist Learning Procedures, *Artificial Intelligence* **40** (1989) 185–234.

[16] R. C. Holte, L. E. Acker, and B. W. Porter, Concept Learning and the Problem of Small Disjuncts, in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI (1989), 813–819.

[17] IUB Nomenclature Committee, Ambiguity Codes, *European Journal of Biochemistry* **150** (1985) 1–5.

[18] B. F. Katz, EBL and SBL: A Neural Network Synthesis, in: *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Ann Arbor, MI (1989), 683–689.

[19] R. D. King, S. Muggleton, R. A. Lewis, and J. E. Sternberg, Drug Design by Machine Learning: The use of Inductive Logic Programming to Model the Structure—Activity Relationships of Trimethoprim Analogues Binding to Dihydrofolate Reductase, *Proceeding of the National Academy of Sciences, USA* **89** (1992) 11322–11326.

[20] S. C. Kleene, Representation of Events in Nerve Nets and Finite Automata, in: *Automata Studies*, C. E. Shannon and J. McCarthy (Eds.), 3–41, Princeton University Press, Princeton, NJ (1956).

[21] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky, Backpropagation Learning in Expert Networks, *IEEE Transactions on Neural Networks* **3** (1992) 62–72.

[22] M. Lebowitz, Integrated Learning: Controlling Explanation, *Cognitive Science* **10** (1986) 219–240.

[23] R. Maclin and J. W. Shavlik, Refining Algorithms with Knowledge-Based Neural Networks: Improving th Chou-Fasman Algorithm for Protein Folding, *Machine Learning* **11** (1993) 195–213.

[24] J. J. Mahoney and R. J. Mooney, Combining connectionist and symbolic learning to refine certainty-factor rule-bases, *Connection Science* **5** (1993) 339–364.

[25] W. S. McCulloch and W. A. Pitts, A Logical Calculus of Ideas Immanent in Nervous Activity, *Bulletin of Mathematical Biophysics* **5** (1943) 115–133.

[26] R. S. Michalski and G. Tecuci, eds, *Proceedings of the First International Workshop on Multistrategy Learning*, George Mason University, Harpers Ferry, W. Va. (1991).

[27] J. Mingers, An Empirical Comparison of Pruning Methods for Decision Tree Induction, *Machine Learning* **4** (1989) 227–243.

[28] T. M. Mitchell, R. Keller, and S. Kedar-Cabelli, Explanation-Based Generalization: A Unifying View, *Machine Learning* **1** (1986) 47–80.

[29] T. M. Mitchell and S. B. Thrun, Explanation-Based Neural Network Learning for Robot Control, in: *Advances in Neural Information Processing Systems*, volume 5, Denver, CO (1992), 287–294.

[30] S. Muggleton, ed, *Inductive Logic Programming*, Academic Press, San Diego, CA (1992).

[31] S. Muggleton, R. D. King, and J. E. Sternberg, Protein Secondary Structure Prediction Using Logic-Based Machine Learning, *Protein Engineering* **5** (1992) 647–657.

[32] G. L. Murphy and D. L. Medin, The Role of Theories in Conceptual Coherence, *Psychological Review* **91** (1985) 289–316.

[33] M. O. Noordewier, G. G. Towell, and J. W. Shavlik, Training Knowledge-Based Neural Networks to Recognize Genes in DNA Sequences, in: *Advances in Neural Information Processing Systems*, volume 3, Denver, CO (1991), 530–536.

[34] M. C. O'Neill, Escherichia Coli Promoters: I. Consensus as it relates to spacing class, specificity, repeat substructure, and three dimensional orgainzation, *Journal of Biological Chemistry* **264** (1989) 5522–5530.

[35] D. W. Opitz and J. W. Shavlik, Heuristically Expanding Knowledge-Based Neural Networks, in: *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France (1993).

[36] D. Ourston and R. J. Mooney, Theory refinement combining analytical and empirical methods, *Artificial Intelligence* **66** (1994) 273–310.

[37] M. Pazzani and D. Kibler, The Utility of Knowledge in Inductive Learning, *Machine Learning* **9** (1992) 57–94.

[38] M. J. Pazzani, The influence of prior knowledge on concept acquisition- Experimental and computational results, *Journal of Experimental Psychology- Learning, Memory & Cognition* **17** (1991) 416–432.

[39] M. J. Pazzani, C. A. Brunk, and G. Silverstein, A Knowledge-Intensive Approach to Learning Relational Concepts, in: *Inductive Logic Programming*, S. Muggleton (Eds.), 373–394, Academic Press (1992).

[40] F. J. Pineda, Generalization of Back-Propagation to Recurrent Neural Networks, *Physics Review Letters* **59** (1987) 2229–2232.

[41] L. Y. Pratt, J. Mostow, and C. A. Kamm, Direct Transfer of Learned Information Among Neural Networks, in: *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA (1991), 584–589.

[42] J. R. Quinlan, Induction of Decision Trees, *Machine Learning* **1** (1986) 81–106.

[43] J. R. Quinlan, Learning Logical Definitions from Relations, *Machine Learning* **5** (1990) 239–266.

[44] L. Rendell and H. Cho, Empirical Learning as a Function of Concept Character, *Machine Learning* **5** (1990) 267–298.

[45] B. L. Richards and R. J. Mooney, Refinement of first-order horn-clause domain theories, *Machine Learning* (to appear).

[46] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan, New York (1962).

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning Internal Representations by Error Propagation, in: *Parallel Distributed Processing: Explorations in the microstructure of cognition. Volume 1: Foundations*, D. E. Rumelhart and J. L. McClelland (Eds.), 318–363, MIT Press, Cambridge, MA (1986).

[48] R. Schank, G. C. Collins, and L. E. Hunter, Transcending Inductive Category Formation in Learning, *Behavioral and Brain Sciences* **9** (1986) 639–686.

[49] G. M. Scott, J. W. Shavlik, and W. H. Ray, Refining PID Controllers using Neural Networks, *Neural Computation* **4** (1992) 746–757.

[50] J. W. Shavlik, R. J. Mooney, and G. G. Towell, Symbolic and Neural Net Learning Algorithms: An Empirical Comparison, *Machine Learning* **6** (1991) 111–143.

[51] J. W. Shavlik and G. G. Towell, An Approach to Combining Explanation-Based and Neural Learning Algorithms, *Connection Science* **1** (1989) 233–255.

[52] E. H. Shortliffe and B. G. Buchanan, A Model of Inexact Reasoning in Medicine, in: *Rule-Based Expert Systems*, B. G. Buchanan and E. H. Shortliffe (Eds.), 233–262, Addison-Wesley, Reading, MA (1984).

[53] G. D. Stormo, Consensus Patterns in DNA, in: *Methods in Enzymology*, volume 183, 211–221, Academic Press, Orlando, FL (1990).

[54] K. Thompson, P. Langley, and W. Iba, Using Background Knowledge in Concept Formation, in: *Proceedings of the Eighth International Machine Learning Workshop*, Evanston, IL (1991), 554–558.

[55] G. G. Towell, *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction*, PhD thesis, Computer Sciences Department, University of Wisconsin, Madison, WI (1991).

[56] G. G. Towell and J. W. Shavlik, Extracting Refined Rules from Knowledge-Based Neural Networks, *Machine Learning* **13** (1993) 71–101.

[57] G. G. Towell, J. W. Shavlik, and M. O. Noordewier, Refinement of Approximately Correct Domain Theories by Knowledge-Based Neural Networks, in: *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA (1990), 861–866.

[58] D. A. Waterman, *A Guide to Expert Systems*, Addison Wesley, Reading, MA (1986).

[59] R. L. Watrous, Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization, in: *Proceedings of the First International Conference on Neural Networks*, volume II (1987), 619–627.

[60] J. D. Watson, H. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner, *The Molecular Biology of the Gene*, Benjamin-Cummings, Menlo Park, CA (1987).

[61] S. M. Weiss and C. A. Kulikowski, *Computer Systems that Learn*, Morgan Kaufmann, San Mateo, CA (1990).

[62] E. J. Wisniewski and D. L. Medin, Is it a Pocket or a Purse? Tightly Coupled Theory and Data Driven Learning, in: *Proceedings of the Eighth International Machine Learning Workshop*, Evanston, IL (1991), 564–569.

[63] D. H. Wolpert, On Overfitting Avoidance as Bias, Technical Report LA-UR-90-3460, Santa Fe Institute, Santa Fe, NM (1992).

# A. KBANN's Language

The rules-to-network translator (Section 3.3) requires from a user two sets of information to describe each problem. The first set of information is a specification of the features that are used to describe the examples. The second set of information is a set of rules that encode the knowledge of the problem which is supplied to the system (i.e., a domain theory). The following two sections define the syntax of each of these information types, and how they are handled by KBANN's rules-to-network translator.

## A.1. Information about Features

KBANN is able to handle examples described using the following five types of features that are commonly used in machine learning:

| | |
|---|---|
| *nominal* | from among a list (e.g., red, green, blue) |
| *Boolean* | values must be either true or false |
| *hierarchical* | values exist in an ISA hierarchy |
| *linear* | values are numeric and continuous |
| *ordered* | values are non-numeric but have a total ordering |

The subsections below contain, for each feature, a brief definition and a description of how it is encoded by KBANN.[16] Missing values are handled in a consistent manner across the feature types. Namely, when the value of a feature is not known, all of the units used to encode it are given an equal activation. In addition, the sum of the activations of the units is set so that the total activation of the units is equal to the total activation of the units had the value been known. In this way, the network cannot learn to distinguish missing values merely on the basis of total activation. Empirical tests [50] have shown that this method of encoding missing values results in better generalization than other methods (such as setting all activations to zero.)

## A.2. Nominal

This is the simplest type of feature; all the possible values of the feature are named. KBANN translates the feature `color` with the values *(red blue green)* into three binary units: *color-is-red*, *color-is-blue*, and *color-is-green*.

## A.3. Binary

*Binary-valued* features are a special subclass of nominal features that have values of only two values (e.g., True, False). To reduce the number of input units, binary-valued features are given only a single input unit. When there is no information available about a binary feature, its activation is 0.5. Note that this is a departure from the "consistent total activation" approach to handling missing variables.

[16] While we have implemented all of these feature types, our testing has focused on nominal and binary features.

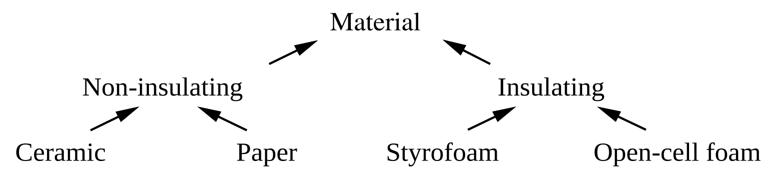



**Figure 24** A hierarchy of cup materials.

## A.4. Hierarchical

*Hierarchical features* are features defined in the context of an *ISA* hierarchy. For instance, Figure 24 illustrates a hierarchy of materials that might be used in making a cup.

Hierarchical features act much like nominal features in that exactly one of the base-level units can be active. When no information is available concerning a hierarchical feature, then the activation of units at each level in the hierarchy is the reciprocal of the number of units in that level. Hence, in the above example, `Styrofoam`, `open-cell-foam`, `paper`, and `ceramic` would all have activations of $\frac{1}{4}$, `insulating` and `non-insul` would have activations of $\frac{1}{2}$, and `material` would have an activation of 1.

## A.5. Linear

*Linear features* have numeric values. Normally these values are continuous, but any feature with numeric values may be described using an linear feature. For linear features, KBANN requires that the user define a set of subranges within the range of the feature. Each of these subranges is translated into an input unit that is activated according to Equations 6 and 7.

$$z_i = \frac{1}{1 + exp(\frac{abs(v - [upB - lowB]/2)}{upB - lowB})} \tag{6}$$

$$a_i = \frac{(z_i)^2}{\sum_j (z_j)^2} \tag{7}$$

In these equations:

| | |
|---|---|
| *upB* | is the top of the subrange |
| *lowB* | is the bottom of the subrange |
| $v$ | is the exact value of the feature |
| $z_i$ | is an intermediate stage in the calculation of the activation |
| $a_i$ | is the activation of the unit |
| $i,j$ | are indices ranging over the units used to encode a feature |

As with nominal features, when no information is available about a linear feature, each unit take a value of $\frac{1}{N}$, where $N$ is the number of units used to encode the feature.

## A.6. Ordered

*Ordered features* are a special type of nominal features for which the values are totally ordered. For example, `size` could be represented using the totally ordered set { `tiny, small, medium, large, huge` }.

Like nominal features, ordered features are handled by creating one input unit for each possible value. However, ordered features cannot be treated like simple nominal features because the boundaries between subsequent values in the ordering are typically indistinct. Therefore, when an object is described as having `medium` size, it might be equally correctly be described as being either `large` or `small`. To account for the lack of precision inherent in ordered features, all values of an ordered feature are activated according to their distance from the given value using the formula given in Equation 8.

$$activation = \frac{1}{2}^{distance\_from\_given\_value} \tag{8}$$

## A.7. Information about Rules

This section describes the syntax of rules that KBANN is currently able to translate into a neural network and specifies how KBANN translates them. Rules for use by KBANN must be in the form of propositional Horn clauses. In addition, disjuncts may only be expressed as multiple rules with the same consequent. Finally, the rules may not contain cycles.

## A.8. Rule Types

KBANN recognizes two distinct types of rules. The first type is "definitional." Definitional rules are rules that are not to be changed during the training phase. (For instance, a definitional rule might be used for "blots" in backgammon.) As a result, the consequents of fixed rules are connected only to those antecedents mentioned in the rule and the weights on the connections to the antecedents are not allowed to change during training.

The second type of rule is "nondefinitional" or "variable." Variable rules are standard rules; they are subject to change during training. Therefore, consequents are connected to antecedents other than those mentioned in the rule, and weights on the links entering into the consequent are changed during neural learning.

## A.9. Special antecedents

In addition to simply listing positive antecedents to form a simple conjunctive rule, KBANN allows the following special predicates in the antecedents of its rules.

| *Predicate* | *Description* | *Example* |
|---|---|---|
| **Not** | Negate an antecedent | Figure 25a |
| **Greater-than** | Only for "linear" and "ordered" features | Figure 25b |
| **Less-than** | Only for "linear" and "ordered" features | Figure 25b |
| **N-true** | Allows compact specification of concepts of the form "if *N* of the following *M* antecedents are true then . . . " | Figure 25c |

The specification of antecedents is recursive. Hence, any of these types of antecedents many be nested within each other.

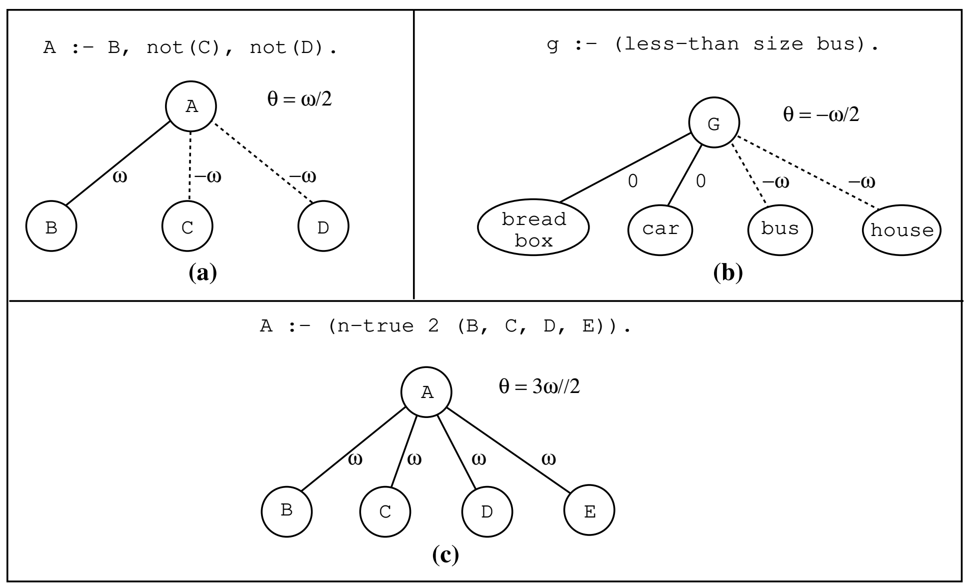



**Figure 25** Encoding rules in a neural network.