Chapter 3

EMPIRICAL TESTS OF KBANN

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

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

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

3.1 Experimental Datasets

Before going into the experiments, this section describes the two real-world datasets from the domain of molecular biology used for testing KBANN. These datasets are used throughout this

chapter and Chapter 5 to test aspects of KBANN. Both datasets have been previously used to demonstrate the usefulness of the KBANN algorithm [Noordewier91, Towell90].

3.1.1 Molecular Biology 101

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

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

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

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

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

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

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

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

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

3.1.2 Notation

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

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

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

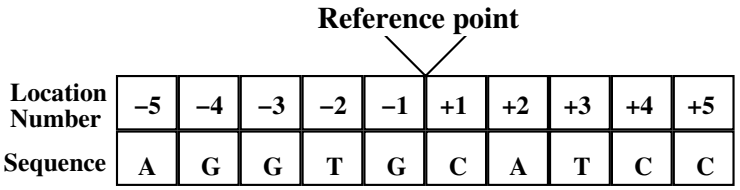


Figure 3.1: The numbering of nucleotide locations in DNA sequences.

Table 3.1: Ambiguity codes for DNA nucleotides.

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

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

3.1.3 Promoter recognition

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

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

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

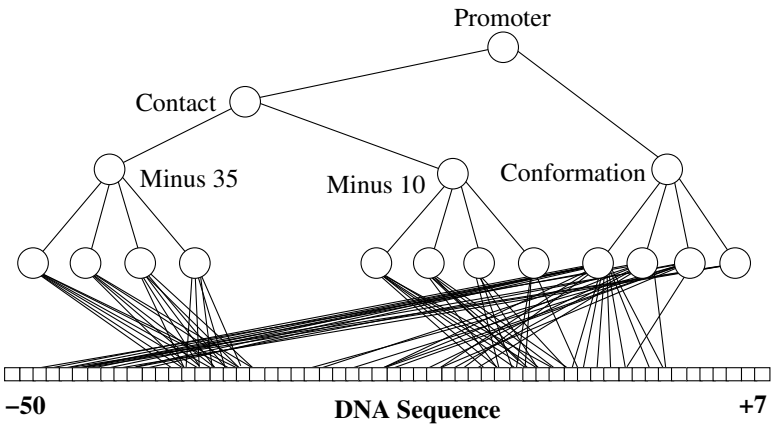


Figure 3.2: The initial KBANN-net for promoter recognition.

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

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

Table 3.2: Rules for promoter-recognition.

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

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

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

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

The rule set in Table 3.2 is translated by KBANN into a neural network with the topology shown in Figure 3.2. Recall that KBANN adds additional low-weighted links (not shown) so

that if additional sequence information is relevant, the algorithm can capture that information during training.

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

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

3.1.4 Splice-junction determination

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

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

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

The dataset used to approach this problem was extracted from the biological literature by M. Noordewier. The full set of examples contains 3190 examples, of which approximately 25% are *I/E* , 25% are *E/I* and the remaining 50% are *neither*. Each example consists of a 60 nucleotide-long DNA sequence categorized according to the type of boundary at the center of the sequence; the center of the sequence is the reference location used for numbering nucleotides. The examples were obtained by taking the documented "split" genes from all primate gene entries in Genbank release 64.1 that are described as complete. This procedure

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

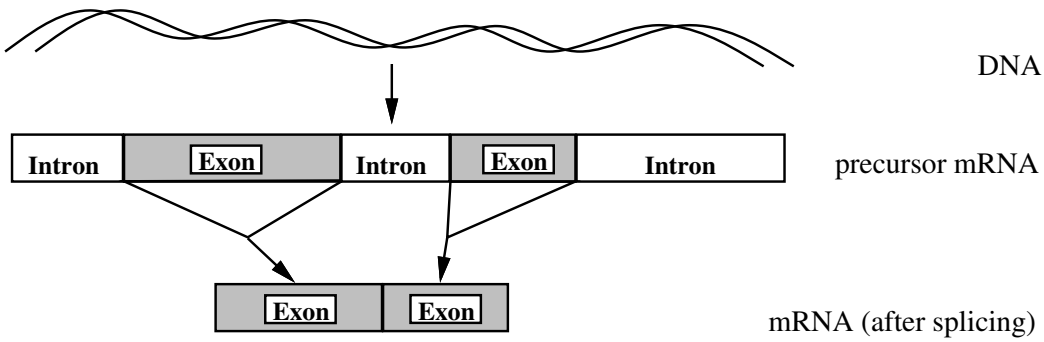


Figure 3.3: The organization of genes in higher organisms.

exon	intron	exon
... M A G	G T R A G T ... Y_6 – Y A G	G K ...

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

Figure 3.4: "Cannonical" splice-junction.

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

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

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

The initial set of rules classifies 61% of the examples correctly. As for promoter recognition, the success rate of the splice junction rules initial rules is due largely to their tendency to "just say no"; the rules correctly classify only 40% of the I/E and 3% of the E/I examples. Figure 3.5

Table 3.3: Initial rules for splice-junction determination.

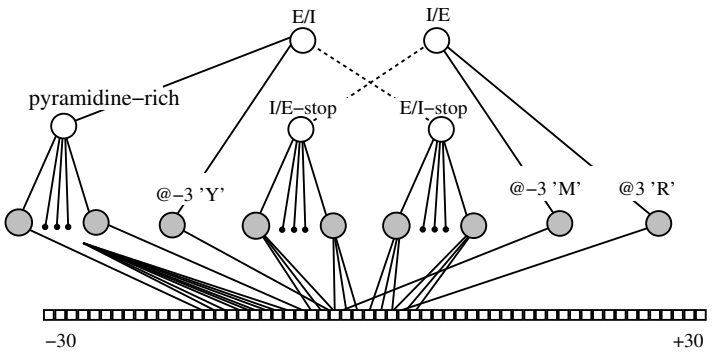
```
E/I :- @-3 'MAGGTRAGT', not(E/I-stop).

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

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

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

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



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

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

depicts an abstracted version of the network that KBANN generates given these rules. Finally, note that this initial set of rules creates a KBANN-net that has only three trainable hidden units. Moreover, these three units are not shared among the two output units. Rather, the *I/E* output units has only a single trainable hidden unit. Hence the *I/E* part of the network is capable no more than perceptron-style learning [Rosenblatt62].

3.2 KBANN versus other Learning Systems

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

3.2.1 KBANN and empirical learners

This section compares the generalization ability of KBANN to systems that learn strictly from training examples. The comparisons are run in three ways. First, KBANN is compared to six algorithms for its ability to extract information from the training examples. Second, KBANN is compared backpropagation — the most effective of the six empirical algorithms — for its ability to learn from small sets of examples. Third, the learning and generalization of KBANN and backpropagation are compared after each training epoch.[4]

Seven-way comparison of algorithms

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

Standard backpropagation [Rumelhart86] is used as a comparison algorithm for two reasons. First, several empirical studies have shown it to be at least as effective at generalizing to examples not seen during training as symbolic learning algorithms [Shavlik91, Fisher89, Weiss89, Atlas89]. Second, standard backpropagation is the empirical algorithm upon which KBANN is based. Hence, comparisons between KBANN and standard backpropagation provide an indication of the utility of the information which the rules-to-networks translator inserts into the networks it creates. Following Shavlik *et al.* [Shavlik91] the number of hidden units in standard ANNs is set to 10% of the number of input units.[5] Hidden units are arranged in a single layer that is fully-connected with both the input and output layers. There are no direct connections between inputs and outputs.[6] All weights are randomly initialized to numbers

[4] Every training example is presented once in a single epoch.

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

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

within 0.5 of zero.

ID3 [Quinlan86] is a symbolic empirical learning algorithm. It uses training data to construct a "decision tree".[7] At each step, a new node is added to the decision tree by partitioning the training examples based on their value for the single, statistically most-informative, feature. ID3 is included in this comparison set because it is a widely-used algorithm that has been tested on a number of large datasets. Also, experimental comparisons with other symbolic learning systems show that ID3 generally performs as well or better than other systems [O'Rorke82, Rendell89]. In addition, ID3 is the underlying empirical algorithm for EITHER, a hybrid system compared to KBANN in Section 3.2.2.

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

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

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

Cobweb [Fisher87], the final algorithm investigated, is an incremental system for *conceptual clustering*. It operates by building a classification tree in which each node is a probabilistic concept that represents an object class. The incorporation of an object is a process of descending the tree and updating counts (statistics are maintained for each feature and each node in

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

the tree) along the appropriate path. When the bottom of the tree is reached, an object is incorporated into the tree by either: (a) simple inclusion into an existing cluster, (b) creation of new clusters by splitting an existing cluster, or (c) the merger several existing clusters into a single cluster. The particular implementation of Cobweb, supplied by J. Gennari, is specialized for classification tasks by the addition of mechanisms for manipulating the class of each example as a special kind of feature. This algorithm is tested herein as it is the basis of Labyrinth [Thompson91], a hybrid learning method whose performance is compared to KBANN in Section 3.2.2 and which is described fully in Chapter 6.

Method. Following Weiss and Kulikowski's suggestion [Weiss90] the systems were evaluated using ten-fold cross-validation for splice-junction determination and leaving-one-out testing for promoter recognition.[8] In N-fold cross-validation, the set of examples is partitioned into N sets of equal size. A system is trained using $N-1$ of the sets and tested using the remaining set. This procedure is repeated N times so that each set is used as the testing set once. In 10-fold cross-validation the initial ordering of examples affects the split of training and testing examples. As a result, each system is tested on the splice-junction problem using eleven different example orderings.[9] The error rates reported are simple averages of the eleven trials.

"Leaving-one-out" is an extreme form of N-fold cross-validation in which N is equal to the number of examples in the set. So, for promoter recognition leaving-one-out requires 106 training passes in which the training set has 105 examples and the testing set has one example. Example order has no affect on leaving-one-out testing as the test set always consists of a single example. However, every algorithm other than ID3 and nearest neighbor is sensitive to the order of example presentation. Hence, each algorithm, other than ID3 and nearest neighbor, was tested using eleven different orderings of the 106 promoter examples. Rather than simply recording the average number of errors over the eleven trials, statistics were maintained for each example. Only those examples that were incorrectly classified by the majority of the eleven trials were regarded as incorrect. This error scoring method was used in an attempt to capture the best result of each algorithm; it always results in a decrease in the number of errors. For example, Cobweb is quite sensitive to presentation order. It misses many examples two or three times, but misses few examples more than six times. Hence, the simple average error rate for Cobweb is more than double than the level reported in Figure 3.7.[10] Conversely,

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

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

[10] 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

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.[11]

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

The initial randomized configuration of a neural network can affect learning. To compensate for this, an extra level of testing is required for both standard backpropagation and KBANN. Specifically, each of the eleven example permutations of each dataset is tested using ten different initial states. Hence, each test requires 110 separate runs of n-fold cross-validation.[12] This test design is illustrated in Figure 3.2.1. The classification derived from the ten trials varying the starting state is based on the average of the output unit activations. So, if the output unit of a promoter recognition network is 0.52 on 9 trials and 0.01 on the tenth trial, then the classification of the network is based upon the relationship between 0.46 and the threshold.[13]

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

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

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

3. The network is classifying at least 90% of the training examples correctly but has not improved it ability to classify the training examples for five epochs. (This implements the *patience* stopping criteria suggested by Fahlman and Lebiere [Fahlman89] and which Squires [Squires91] showed is a useful method of preventing overfitting of the training data.)

In general, networks trained for promoter recognition stopped training on the first criterion

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.

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

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

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

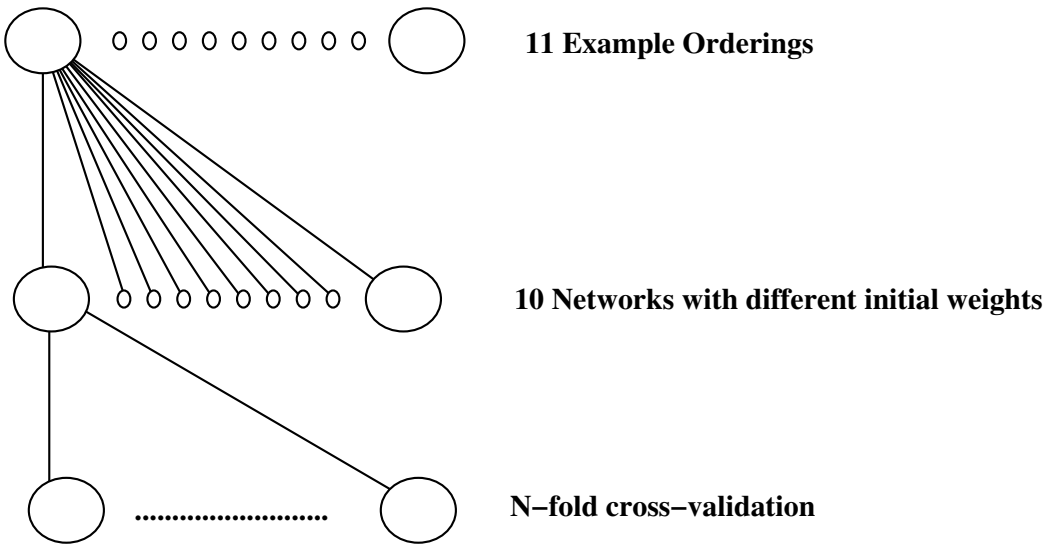


Figure 3.6: The design of experiments to test the generalization accuracy of learning systems based on neural networks.

– they are able to perfectly learn the training data. On the other hand, KBANN-nets for splice-junction determination often terminate on the second or third criterion because they are unable to perfectly learn the training data.

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

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

The advantage of this measure is that it accurately reflects learning over domains in which the distribution of examples is skewed or in which there is more than one decision to be made. For instance, consider a domain in which 90% of the examples fall into a single class. Merely

Table 3.4: Calculation of the Relative Information Score

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

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

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

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

where:

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

guessing the majority class results in a 10% error rate but an RIS of only 26. (When a two-category problem has an equal number of examples in each category, always guessing one category gives an RIS of 0.0.) Hence, RIS is used in this section to compare the effectiveness of each of the learning systems. Unfortunately, RIS requires considerably more detailed record keeping than simple error rates. Therefore, all subsequent tests in this thesis report simple error rates.

Results and discussion. Figures 3.7 and 3.8 present the comparisons of KBANN to the six empirical-learning algorithms described above. In addition to the above described empirical algorithms, Figure 3.7 contains the accuracies of two methods suggested by biologists for promoter recognition. The first method, labeled "Stormo", is a method which learns through the presentation of positive-only examples [Stormo90]. The second method, labeled "O'Neill" [O'Neill89], is a non-learning, hand-refined, technique for differentiating between promoters and non-promoters.[14]

[14]Stormo's technique for constructing consensus sequences is a learning algorithm. His approach collects a set of aligned examples of some category and counts how many times each nucleotide appears at each position. This information is used to build a scoring matrix, which is then applied to new sequences; those that score above some threshold (we used a threshold of zero) are predicted to be members of the class being learned.

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

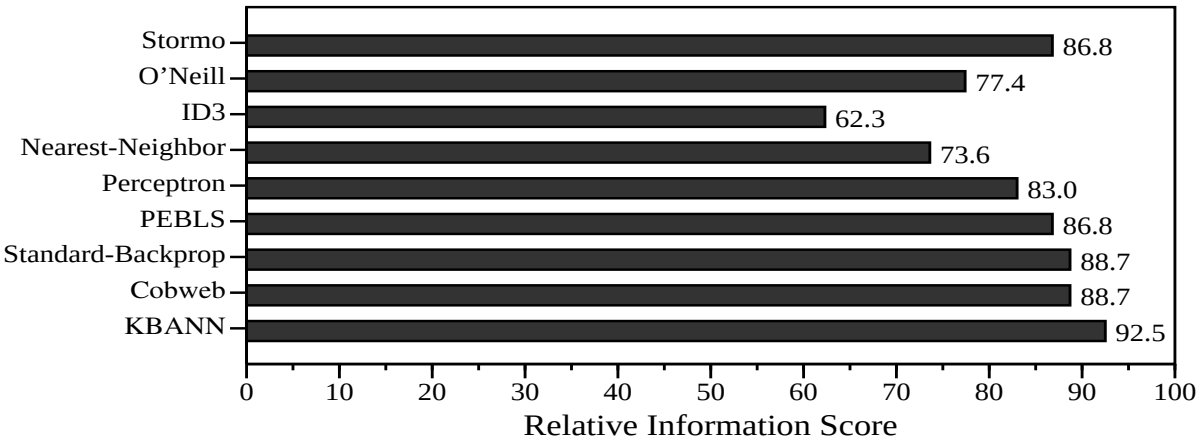


Figure 3.7: Performance on the promoter recognition task.

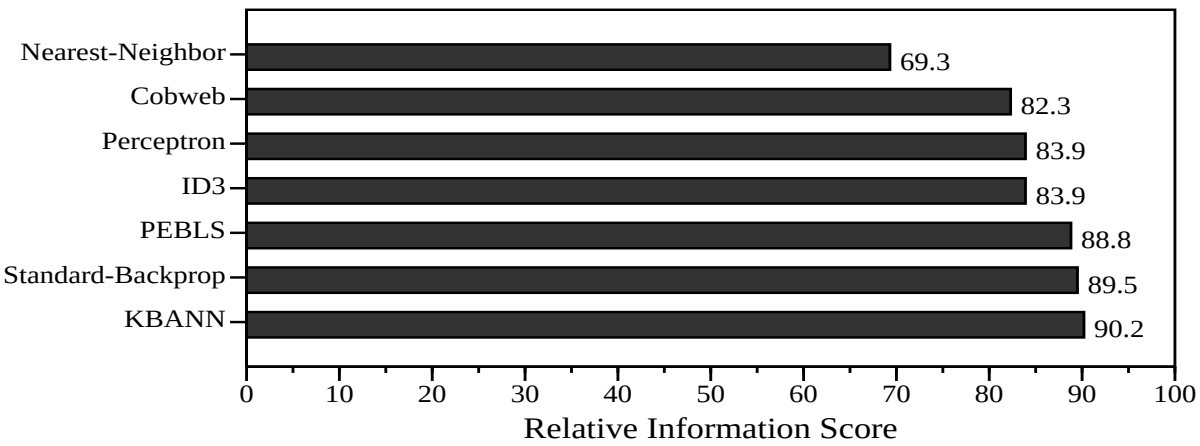


Figure 3.8: Performance on the splice-junction determination task with 1000 training examples.

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

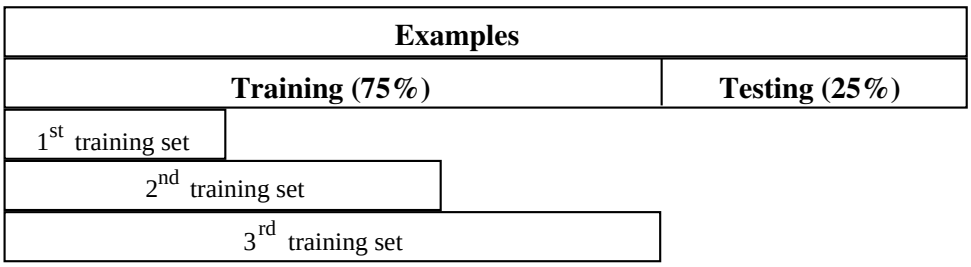


Figure 3.9: The partitioning of a set of examples to form a learning curve.

Learning from small sets of examples

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

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

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

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

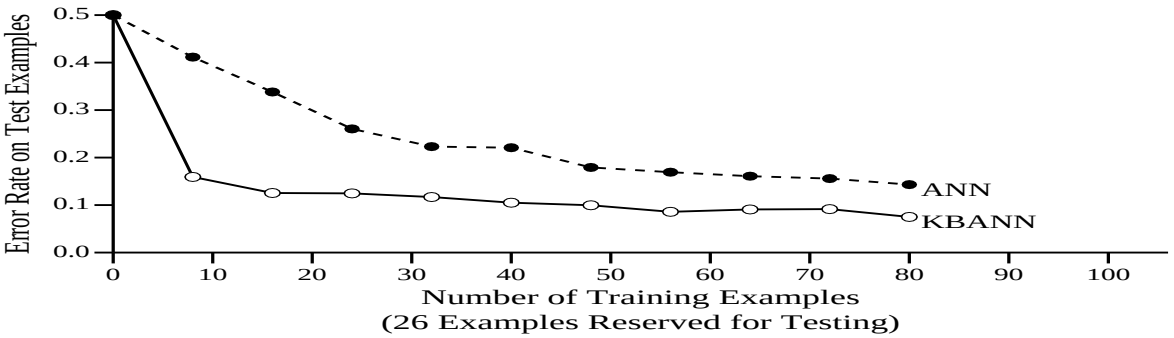


Figure 3.10: Learning curves for the promoter domain with 106 examples.

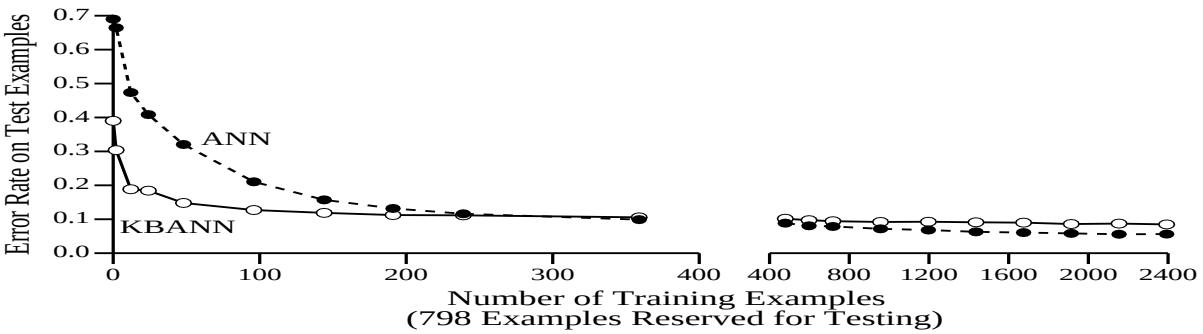


Figure 3.11: Learning curves for the splice-junction domain with 3190 examples.

declines on both datasets. On the promoter set, (Figure 3.10) KBANN is at an advantage of about 4% after training using the largest training set. On the other hand, KBANN is at a 2% disadvantage to backpropagation on the splice junction set after training on the largest available set of training examples.[15]

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

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

Learning and generalization as a function of training effort

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

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

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

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

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

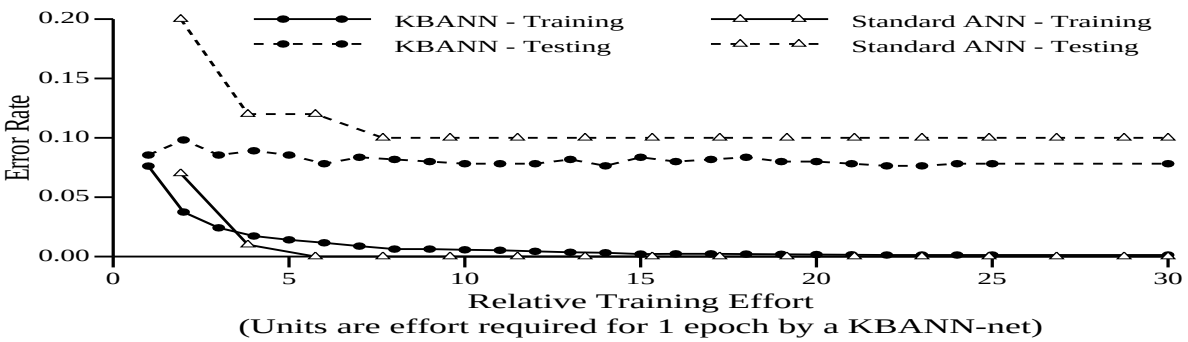


Figure 3.12: Learning as a function of training effort on the promoter domain.

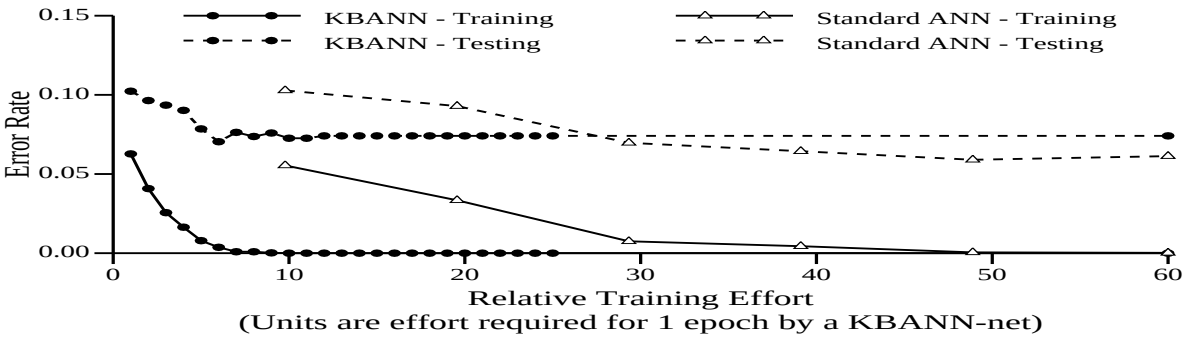


Figure 3.13: Learning as a function of training effort on the splice-junction domain with 1000 examples.

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

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

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

Discussion of the comparison of KBANN to empirical learning algorithms

The results reported in Figures 3.7 and 3.8 describe only the asymptotic performance of each system. On the basis of these comparisons, KBANN is the most effective learning algorithm. 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 3.10 and 3.11 that compare the performance of backpropagation to that of KBANN are at least as important as the results in Figures 3.7 and 3.8.

Both Figures 3.10 and 3.11 clearly show the advantage of KBANN-nets when there are few training examples available. Therefore, it is reasonable to conclude this section with a paraphrase of an old axiom, namely "A domain theory is worth 100 examples."

3.2.2 KBANN and theory & data learners

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

Testing methodology

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

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

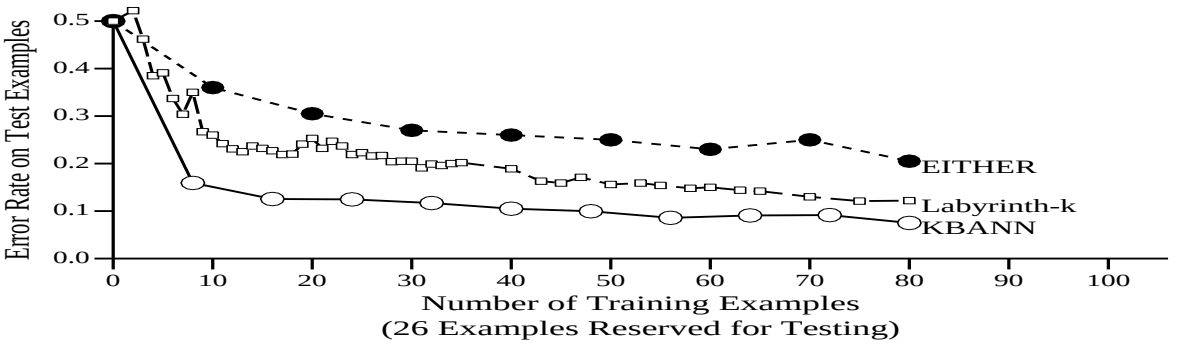


Figure 3.14: Accuracy of Theory/Data systems for Promoter Recognition.

Results

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

3.2.3 Discussion of the empirical comparisons

The results of the tests in this section indicate that KBANN is generally superior to both (a) systems that learn only from examples (i.e., empirical learners) and (b) systems that, like KBANN, learn from both theory and data. KBANN's relative weakness on the splice-junction dataset has been attributed to two factors. First, the splice-junction domain theory is relatively poor. It allows for little more than perceptron-like learning. Hence, a richer domain theory might be expected to improve the performance of KBANN-nets. This hypothesis cannot currently be tested as there is no better domain theory for splice junctions.

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

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

3.3 Sources of KBANN's Strength

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

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

One hypothesis to account for KBANN's superiority to EITHER and Labyrinth-k is that the difference is due wholly to the superiority of KBANN's underlying learning algorithm. In other words, KBANN is superior to both EITHER and Labyrinth-k because backpropagation is superior to both ID3 and Labyrinth. Data presented in Figures 3.7 and 3.8 support this contention; these figures show backpropagation to be superior to both ID3 and Cobweb[16] on both the promoter and splice-junction datasets. Figure 3.15, which compares backpropagation, ID3 and Cobweb using the same format as Figure 3.14, lends further credence to this hypothesis. In addition, a number of empirical studies suggest that, over a wide range of conditions, backpropagation is as good or better than other empirical learning systems at classifying examples not seen during training [Shavlik91, Fisher89, Weiss89, Atlas89]. Therefore, it is reasonable to conclude that some of the power of KBANN is due simply to the superiority of its underlying empirical learning algorithm.

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

3.3.2 Why is KBANN superior to standard backpropagation?

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

[16] Labyrinth slightly outperforms Cobweb [Thompson91]

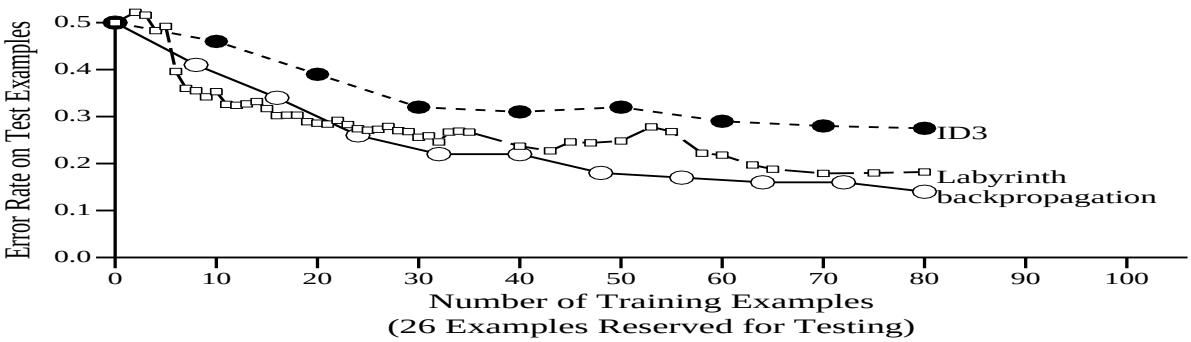
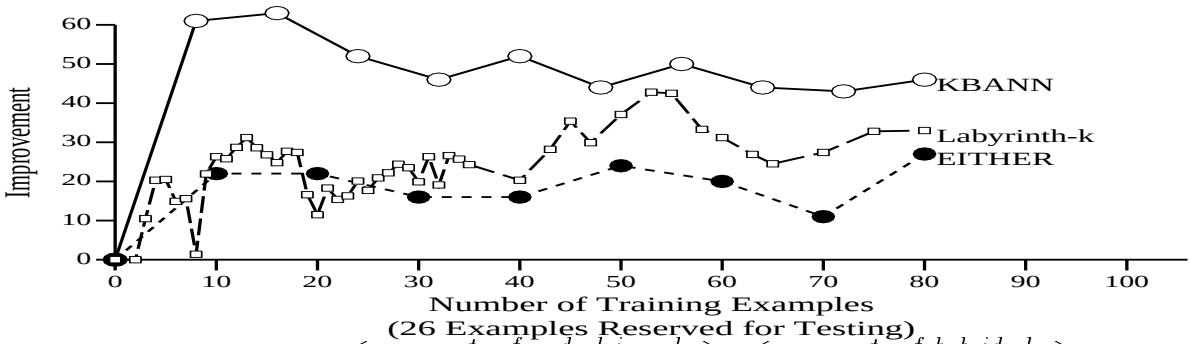


Figure 3.15: Accuracy of empirical algorithms underlying Theory/Data systems on the promoter data.



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

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

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

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

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

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

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

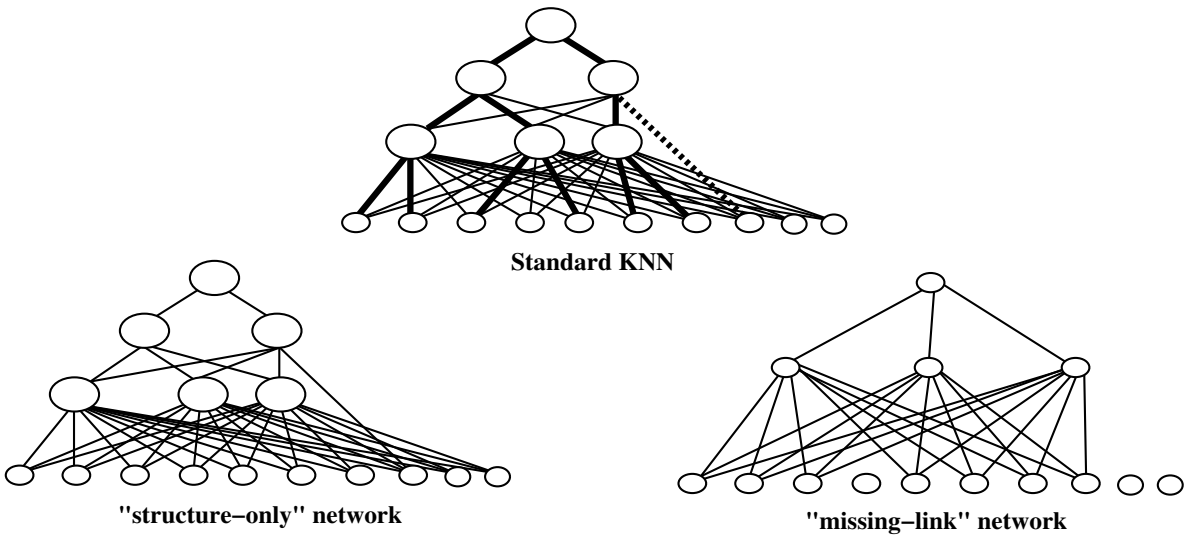


Figure 3.17: Standard, structured, and weighted networks.

Structure is responsible for KBANN's strength

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

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

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

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

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

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

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

is the structured network worse at generalizing to promoter testing examples, it is slower to train than standard KBANN-nets on both domains. The differences are statistically significant using a one-tailed, paired sample t-test with 99.5% confidence.[17]

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

Initial weights are responsible for KBANN's strength

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

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

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

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

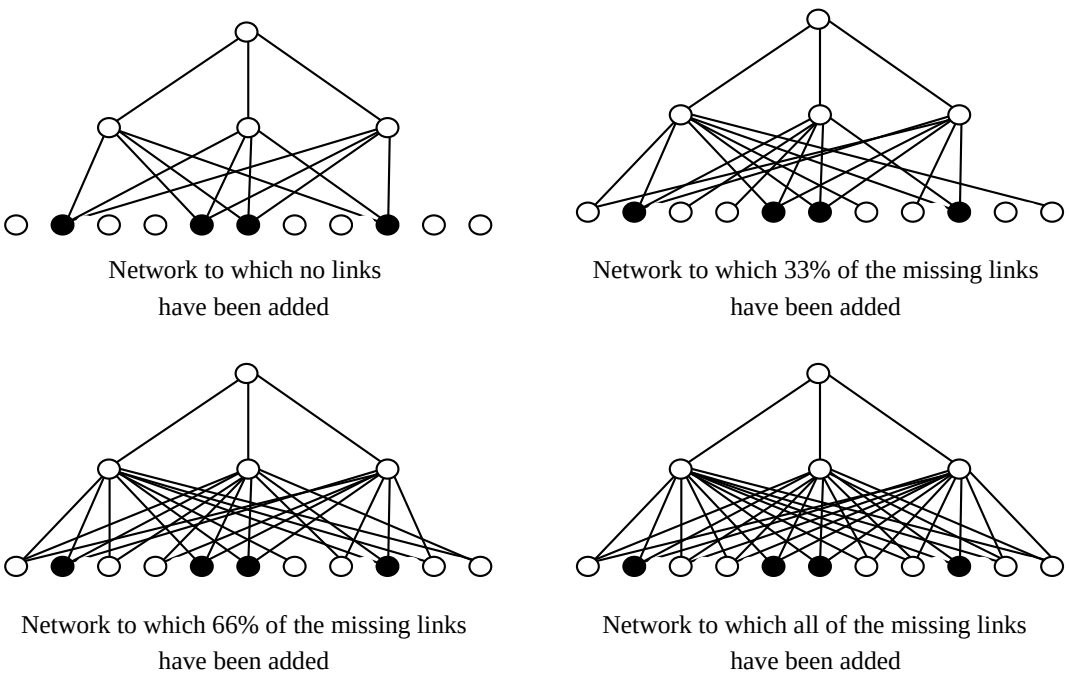


Figure 3.18: The creation of networks to test the hypothesis that initial weights are a significant contributor to KBANN.

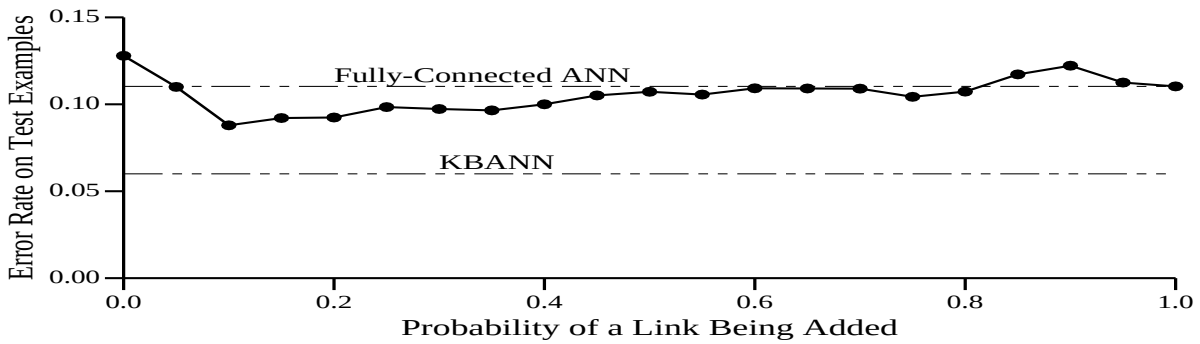


Figure 3.19: The classification performance of standard ANNs that initially have links to only those features specified by the promoter domain theory.

method of link addition is in the same spirit as the learning curves reported previously (e.g., Figure 3.14).

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

Results. Figure 3.19 plots the average error rate of these "missing link" networks. Networks which have only links identified by the initial domain knowledge are worse at generalizing to testing examples than both KBANN-nets and standard ANNs. However, by the time 5% of the missing links have been added, the missing-link networks are superior to standard ANNs. Missing-link networks remain superior to standard ANNs until 80% of the missing links have been added.

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

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

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

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

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

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

3.3.3 Discussion of the sources of KBANN's strength

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

While this hypothesis was confirmed by the results, its confirmation does not tell the whole story as further tests show that KBANN gains more from its theory/data hybrid than do other hybrid systems. Hence, a subsequent suite of tests investigate the reasons why KBANN is able to improve so greatly upon its underlying algorithm. These tests reveal that KBANN's strength

is based on its ability to focus empirical learning on the relevant features *and* the creation of useful derived features; neither alone is sufficient.

3.4 When is KBANN superior to backpropagation?

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

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

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

3.4.1 Noise in domain theories

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

Domain theory noise can be split into two categories: rule-level noise and antecedent-level noise. Each of these categories has several subcategories. The discussion of each type of noise below assumes, for the sake of clarity of explanation, a two-category (i.e., positive and negative) domain and that the rules use negation-by-failure.[18]

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

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

- *added rules* Rules not required for correct operation of the domain theory are present. The effect of added rules is to make possible proofs that should not occur. (Assuming

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

that the extra rules are incorrect, not merely redundant.) Hence, rule sets with extra rules overpredict the positive class.

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

- *extra antecedents* Extra antecedents are those not required in the correct rule. Their effect is to overconstrain the application of a rule. Hence, extra antecedents result in a rule set underpredicting the positive class.

- *missing antecedents* Missing antecedents are antecedents that should be part of a rule, but are not. The effect of missing antecedents is to underconstrain the application of a rule. As a result, missing antecedents, much like added rules, result in overpredicting the positive class.

- *inverted antecedents* Inverted antecedents should appear in a rule, but in the opposite way. That is, a negated antecedent should actually be unnegated (or vice versa). The effect of an inverted antecedent is to make the rule apply in exactly the wrong situation. This may result in both labeling positive examples as negative or the converse.

Method

Tests for antecedent-level noise start with the initial domain theory then probabilistically add noise to it (the probability is based on the total number of antecedents in the domain theory).[19] As with all other experiments involving increasing quantities, noise is added incrementally. That is, to a domain theory with no noise, the smallest noise fraction is added. Then, the slightly noisy rule set has further noise added to it so that the total noise is equal to the second smallest fraction. This procedure is repeated four times for each of the three kinds of antecedent noise.

Antecedent-level noise is added by scanning each antecedent of each rule. As each antecedent is scanned, the noise-addition procedure probabilistically determines whether or not to add noise. If the decision is to add noise, and the kind of noise to be added is either deletion or inversion, then the appropriate action was taken on the antecedent being scanned (the scanned antecedent is either inverted or dropped). If antecedents are to be added, then an antecedent is randomly selected from among the input features as well as any consequent "below" the consequent of the rule being considered[20]. The selected antecedent is then added

[19] All distributions are uniform.

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

to the rule of which the scanned antecedent is a member.

A similar incremental procedure was used to add rule-level noise to the domain theory. Again, noise is added probabilistically (where the probability is based upon the number of rules in the initial domain theory). Details about the insertion of rule-level noise are given in the next two paragraphs.

Rules are added by creating new rules that are simple conjunctions of randomly-selected input features. The number of antecedents to each added rule is a random number in the range [2...6].[21] Rules so created are added to the domain theory only to existing disjuncts in the domain theory. The reason for this restriction is that such additions require minimal changes to the structure of the rule set. Two other kind of additions are possible but not used. First, rules could be added to the conditions of a conjunct. This sort of addition requires adding antecedent-level noise in addition to rule-level noise. Second, rules could be added that create new ways to prove a consequent that is not disjunctive. This sort of addition requires significant alterations to the hierarchical structure of the rules.

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

Results and discussion

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

Results were much the same for rule-level noise. In this case, up to 10% of the initial rules can be deleted while leaving KBANN-nets superior to ANNs. The addition of randomly created rules has little effect on the accuracy of KBANN-nets. The inability of added rules to influence the correctness of KBANN-nets is not surprising. The added rules have an average of four, randomly-chosen, antecedents. Hence, these rules can be expected to match only one in every 256 patterns.[22] As a result, the chances are good that the rules never match the training or

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

[22] The number 256 assumes a DNA sequence analysis domain where each feature has four possible values.

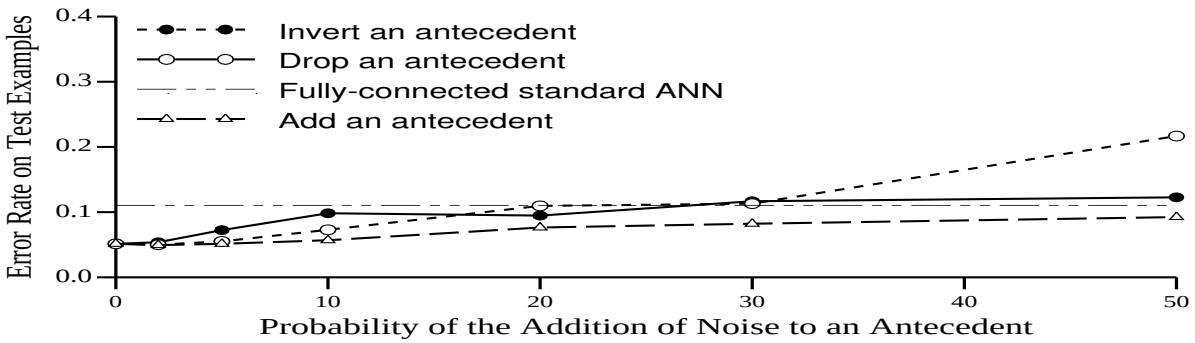


Figure 3.20: The effect of antecedent-level noise on classification performance of KBANN-nets in the promoter domain.

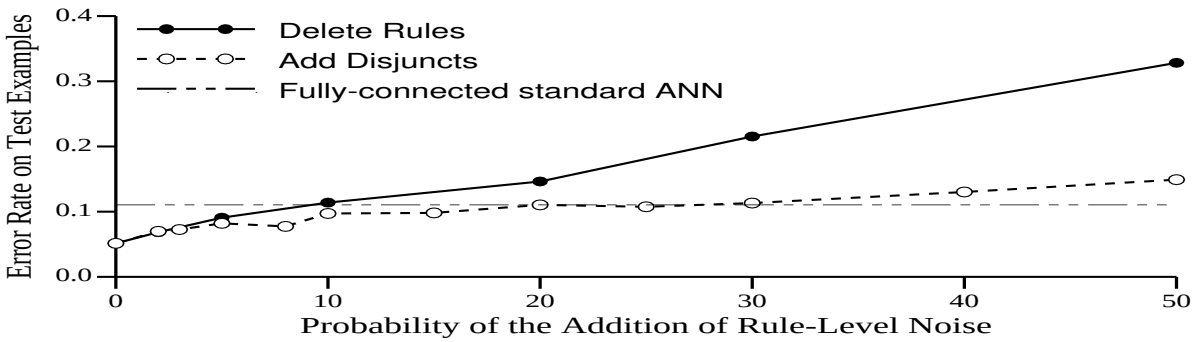


Figure 3.21: The effect of rule-level noise on classification performance of KBANN-nets in the promoter domain.

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. As Figure 3.20 indicates that KBANN-nets are relatively insensitive to this type of noise, the effect of the irrelevant rules can be expected to be minimal.

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

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

Hence given four antecedents looking a four different features, the number of possible combinations is 4^4. Also, these odds assume that DNA is randomly assembled from its four-character alphabet. However, DNA is non-random

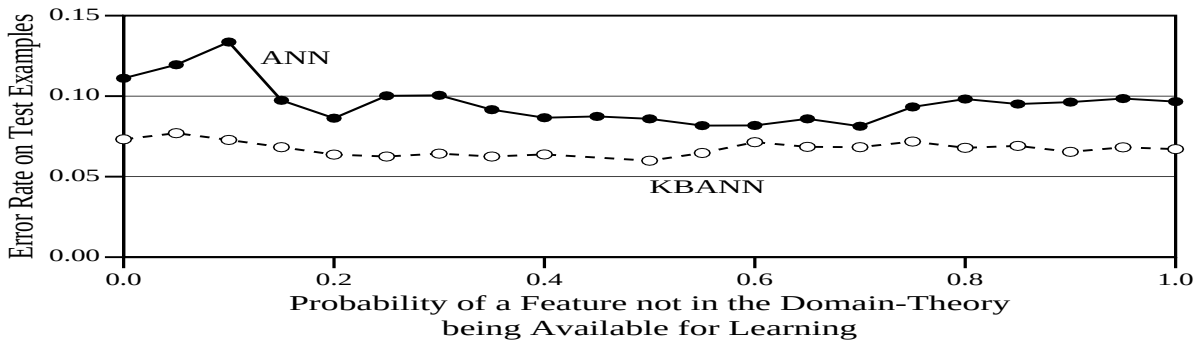


Figure 3.22: The effect of the presence of irrelevant features on classification performance in the promoter domain.

3.4.2 Irrelevant features

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

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

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

Results and discussion

The results in Figure 3.22 clearly demonstrate that KBANN-nets are less sensitive to the presence of irrelevant features than are ANNs. After an initial decrease in accuracy, KBANN-nets slowly improve with the addition of features so that the most accurate KBANN-nets are con-

nected to about 50% of the input features. The accuracy of these networks is statistically indistinguishable from those with access to every feature. The most accurate ANNs are those connected to approximately 60% of the features not mentioned in the domain theory. The difference in accuracy between ANNs connected to 60% of the missing features and ANNs connected to every feature is statistically significant with 99% confidence.

3.5 Experiments in the Extraction of Rules

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

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

A hypothesis advanced when SUBSET was introduced is that there is a tradeoff between the number of rules extracted by SUBSET and the accuracy of those rules. Figure 3.23 shows this tradeoff between size and accuracy of the extracted set of rules on the promoter domain. The figure shows that, as expected, the number of extracted rules increases with β_p.[23] Also as expected, the accuracy of the extracted rules increases with the size of the rule set. Significantly, the accuracy of the rules does not steadily approach the accuracy of the network. Instead, the rules improve rapidly until reaching a level about 2% worse than the network. After this point, the rate of improvement in accuracy slows considerably. Thus, increasing β_p beyond 70 results in very little gain in accuracy but a large gain in the number of rules. Although it is not shown, the asymptotic behavior of the extracted rules also occurs on the splice-junction dataset. All subsequent experiments use $\beta_p = 50$ and $\beta_n = 5$, values which are chosen as a good tradeoff between size and accuracy. At this level, SUBSET discovers an average of 300 rules in the promoter domain.[24]

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

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

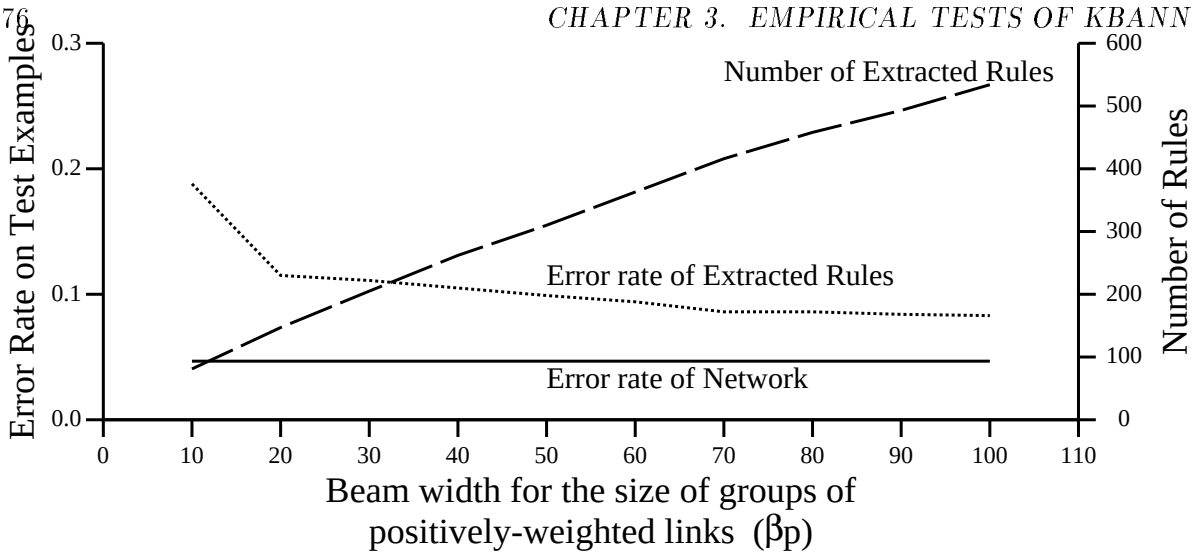


Figure 3.23: The tradeoff between number of rules and accuracy in the promoter domain using the SUBSET algorithm.

3.5.1 Testing methodology

Testing methodology largely follows that of Section 3.2. Briefly, following Weiss and Kulikowski [Weiss90], networks are trained using repeated 10-fold cross-validation for assessing the quality of learning in both domains. (Note that this is a change in methodology; earlier in this chapter (Section 3.2) promoter recognition was tested using "leaving-one-out".) This change results from processing time considerations.) Networks are trained until 99% of the training examples are correctly classified or every example has been presented 100 times. Following Hinton's [Hinton89] suggestion for improved network interpretability, all weights are subject to gentle "decay" during training.[25] Finally, as in the previous chapter, networks are trained using the *cross-entropy* error function [Hinton89]. This methodology is used, except when noted, in all experiments described in this section.

3.5.2 Rule quality

The issue of overriding importance to this work is the quality of the extracted rules. As a measure, *quality* is at least two dimensional. First, the rules must accurately categorize examples that were not seen during training. If the rules lose the advantage in accuracy that KBANN-nets provide over most symbolic learning algorithms, then there is little value in rule extraction. It would be simpler to use an "all symbolic" method (e.g., EITHER [Ourston90]) to develop rules,

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

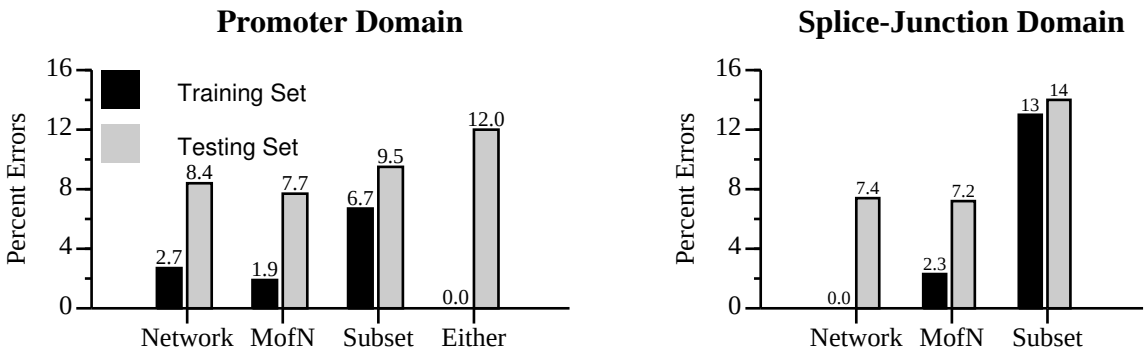


Figure 3.24: Error rates of extracted rules.

while using KBANN-nets only to develop highly-accurate, but not understandable, classifiers. Second, the extracted rules must capture the information contained in the KBANN-net. This is necessary if the extracted rules are to be useful for determining that a trained KBANN-net is likely to behave correctly in the future. On each of these measures, the results in this section show that the NOFM method is superior to the SUBSET method. More significantly, the NOFM method extracts rules that are superior to those of all-symbolic systems, such as EITHER, and are at least equivalent to the networks from which they came at classifying testing examples.

Accuracy

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

Recall the initial rule sets for promoter recognition and splice-junction determination correctly categorized 50% and 61%, respectively, of the examples. Hence, each of the systems plotted in Figure 3.24 improves upon the initial rules. Comparing only the systems that produce refined rules, the NOFM method is the clear winner. On the training examples, the error rate for rules extracted by NOFM is slightly worse than EITHER, but superior to the rules

extracted using SUBSET. On the testing examples the NOFM rules are much more accurate than both EITHER and SUBSET. (One-tailed, paired-sample t-tests indicate that for both domains the NOFM rules are superior to the SUBSET rules with 99.5% confidence. Statistical comparisons to EITHER are not made due to lack of the appropriate data for EITHER.)

Perhaps the most significant result in this section is that, on the testing examples, the error rate of the NOFM rules is superior to that of the networks from which the rules are extracted. (One-tailed, paired-sample t-tests indicate that the difference is statistically significant with 99.5% confidence for the promoter domain (t=5.5, d.f.=9) and 90% confidence for the splice-junction domain (t=1.6, d.f.=9).) Conversely, the error rate of the SUBSET rules on testing examples is statistically worse than the networks in both problem domains. Discussion at the end of this section analyses reasons why NOFM rules are superior to the networks from which they are extracted.

Fidelity

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

3.5.3 Comprehensibility

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

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

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

Global comprehensibility

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

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

Taking the initial rule sets in each domain as a standard for interpretability, the rules refined by EITHER are likely to be easy to understand. (Recall, however, that these rules are not particularly accurate.) Rules extracted by NOFM are also likely to be easily understood; they have virtually the same number of rules and antecedents as the initial rule sets. On the other hand, the rules SUBSET extracts are much less likely to be comprehensible. (See Tables 3.8 and 3.9 for examples of rules extracted by NOFM and SUBSET, respectively.)

Antecedents per rule. A second important global statistic is the number of antecedents per rule. If this value is large, individual rules are unlikely to be understandable [Brunner56].

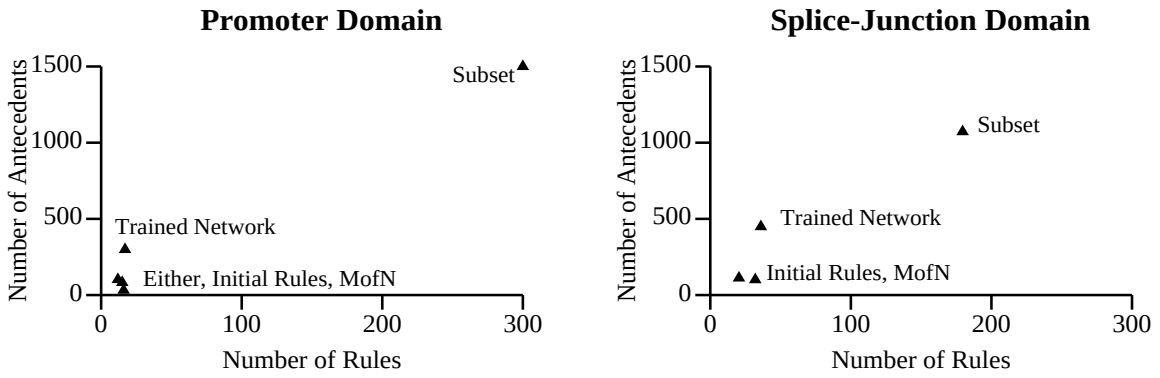


Figure 3.25: Comprehensibility of extracted rules.

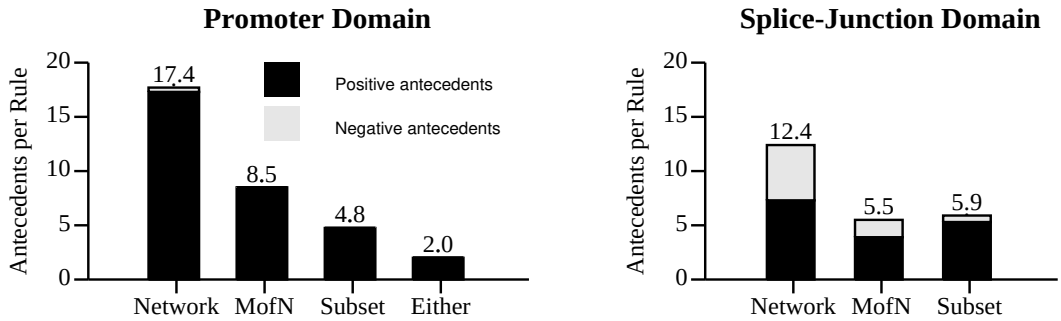


Figure 3.26: Average number of antecedents per rule.

Furthermore, negated antecedents add to the difficulty of evaluating rules [Nessier62]. However, the effects of negated antecedents are difficult to quantify. Weighted antecedents, such as those appearing in NofM rules and which implicitly appear in trained networks, cloud the picture further. Thus, Figure 3.26 takes a very simple approach to judging rules on the basis of antecedents. Each bar consists of a stack that represents the simple sum of the negative and positive antecedents to an average rule.

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

Consistency. Finally, the "consistency" of rules is important because if the extracted rules vary considerably between training sessions, it is difficult to ascribe any significance to a particular rule set. There are several ways to determine the consistency of two or more sets of rules. Arguably the best way, and certainly the most thorough, is to test each rule set with every possible input and compare the actions of each set. However, this approach is impossible

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

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

for it could require testing up to 10^{34} combinations for the promoter problem and about 10^{36} combinations for the splice-junction problem.[26]

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

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

Individual comprehensibility

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

Tables 3.8 and 3.9 present the rules NofM and Subset extract from the same network for promoter recognition. Because rule sets extracted by Subset can be very large, for these tables β_p and β_n (beam widths used during the search of rule space) are respectively set to five

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

Table 3.8: Promoter rules NofM extracts.

```
Promoter :- Minus35, Minus10.

Minus-35                                    Minus-10 :- 2 of @-14 '---CA---T' and
 :-10 < 4.0 * nt(@-37 '--TTGAT-') +                    not 1 of @-14 '---RB---S'.
        1.5 * nt(@-37 '----TCC-') +         Minus-10
        0.5 * nt(@-37 '---MC---') -          :-10 < 3.0 * nt(@-14 '--TAT--T-') +
        1.5 * nt(@-37 '--GGAGG-').                    1.8 * nt(@-14 '-----GA--') +
Minus-35                                              0.7 * nt(@-14 '----GAT--') -
 :-10 < 5.0 * nt(@-37 '--T-G--A') +                   0.7 * nt(@-14 '--GKCCCS-').
        3.1 * nt(@-37 '---GT---') +         Minus-10
        1.9 * nt(@-37 '----C-CT') +          :-10 < 3.8 * nt(@-14 '--TA-A-T-') +
        1.5 * nt(@-37 '---C--A-') -                   3.0 * nt(@-14 '--G--C---') +
        1.5 * nt(@-37 '------GC') -                   1.0 * nt(@-14 '---T---A-') -
        1.9 * nt(@-37 '--CAW---') -                   1.0 * nt(@-14 '--CS-G-S-') -
        3.1 * nt(@-37 '--A----C').                    3.0 * nt(@-14 '--A--T---').
Minus-35 :- @-37 '-C-TGAC-'.                Minus-10 :-  @-14 '-TAWA-T--'.
Minus-35 :- @-37 '--TTD-CA'.
```

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

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

and one. These values are too small to deliver reasonably accurate rule sets; they each make about 25% more errors than the networks from which they are extracted.

While the rules extracted by NofM in Table 3.8 are somewhat murky, they are vastly more comprehensible than the network of 3000 links from which then came. Moreover, the rules in this table can be rewritten in a form very similar to one used in the biological community [Stormo90], namely weight matrices.

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

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

Table 3.9: Promoter rules extracted by SUBSET.

```
Promoter :- Minus35, Minus10.

Minus35  :- Minus35b, Minus35d.
Minus35  :- Minus35a, Minus35b.
Minus35  :- Minus35a, Minus35d.

Minus35  :- @-37 '-T-T--A'.
Minus35  :- @-37 '-TT---A'.
Minus35  :- @-37 '---G-CA'.
Minus35  :- @-37 '--T--C-'.
Minus35  :- @-37 '-T---CA'.

Minus35a :- @-37 'CT--A--'.
Minus35a :- @-37 'C--G-C-'.
Minus35a :- @-37 'C-T--C-'.
Minus35a :- @-37 'CT---C-'.
Minus35a :- @-37 'C---AC-'.
Minus35b :- @-37 '---GA--'.
Minus35b :- @-37 '--T--CA'.
Minus35b :- @-37 '-T---C-'.
Minus35b :- @-37 '---G-CA'.
Minus35b :- @-37 '----ACA'.
Minus35d :- @-37 '-TT--C-'.
Minus35d :- @-37 '-T-G-C-'.
Minus35d :- @-37 '--T-AC-'.
Minus35d :- @-37 '---GAC-'.
Minus35d :- @-37 '-T--AC-'

Minus10  :- @-14 '-ATA----'.
Minus10  :- @-14 '-T--A-A-'.
Minus10  :- @-14 '--A-T-T-'.
Minus10  :- @-14 '-TA-----'.
Minus10  :- @-14 '-T------'.
Minus10  :- @-14 '---A---T'.
Minus10  :- @-14 '--T----T'.
Minus10  :- @-14 '--TA----'.
Minus10  :- @-14 'T-T-A---'.
Minus10  :- @-14 '-AT--T--'.
Minus10  :- @-14 '--TAA---'.
Minus10  :- @-14 '--TA-T--'.
```

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

prevents the rule from ever being satisfied.

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

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

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

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

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

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

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

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

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

3.5.4 Discussion of rule extraction

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

Two hypotheses explain the superiority of NofM over rule-refinement method that directly modify rules. First, the re-representation of a rule set as a neural network allows for more fine-grained refinement. When cast as a neural network, rules can be modified in very small steps. Hence, it may be possible to more closely fit the target concept than when taking the

Table 3.10: The splice-junction rules NofM extracts

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

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

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

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

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

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

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

large steps required by direct rule-refinement. Second, DNA sequence-analysis problems have aspects that can be perspicuously captured by N-of-M style rules. For instance, in the promoter problem, there are several potential sites at which hydrogen bonds can form between DNA and a protein; if enough of these bonds form, promoter activity can occur. As neither Either nor Labyrinth (nor the Subset method) can easily express N-of-M rules, these algorithms may be at a disadvantage on the molecular biology test domains.

The superiority of the NofM rules over the networks from which they are extracted occurs

Table 3.11: The splice-junction rules SUBSET extracts

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

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

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

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

I/Estop8 :- @3 'TAG'.

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

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

because the rule-extraction process reduces overfitting of the training examples. The principle evidence in support of this hypothesis is that the difference in ability to correctly categorize testing and training examples is smaller for NOFM rules than for trained KBANN-nets. In other words, the rules that the NOFM method extracts are only slightly better at classifying training examples than at classifying testing examples. (While the differences between training and testing set performance are smaller, they are statistically significant with 99.5% confidence

using a one-tailed, paired-sample, t-test.) The rules SUBSET extracts also have this property, but they are worse on the training set than both NOFM rules and trained KBANN-nets on the testing set.

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

3.6 General Discussion of KBANN's Effectiveness

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

A separate set of tests indicate that reasonably-accurate domain theories are sufficient to allow KBANN to create a network that is quite effective at generalizing to examples not seen during training. Specifically, the results show that KBANN is most effective at learning from domain theories that have too many rules with too many antecedents (i.e., domain theories that are overly specific). At the very least, this result is interesting for its serendipity as both the promoter and splice-junction domain theories are overly specific. However, both domain theories were extracted directly from the biological literature by a person who was unfamiliar with this property of KBANN.[27] Hence, some of the effectiveness of KBANN by comparison to EITHER and Labyrinth may be due to the promoter domain theory better fitting the biases of KBANN than those systems.

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

Almost all of the tests in this chapter should be repeated with an artificial domain in

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

which a classification theory and the relevant features are known with certainty. An artificial problem would allow very tightly controlled experiments that are not possible in the real-world problems studied in this chapter. For instance, an artificial problem would allow an analysis of rule extraction methods to look beyond classification accuracy, asking instead how well the extracted rules match the true theory. Similarly, tests of sensitivity to irrelevant features would benefit from the absolute knowledge about which features are important.

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