DIRECTED PROPAGATION OF TRAINING SIGNALS THROUGH KNOWLEDGE-BASED NEURAL NETWORKS

by

Geoffrey G. Towell and Jude W. Shavlik

Computer Sciences Technical Report #989

December 1990

Submitted (12/90) to IJCAI-91: Twelfth International Joint Conference on Artificial Intelligence

Directed Propagation of Training Signals through Knowledge-Based Neural Networks*

Geoffrey G. Towell
towell@cs.wisc.edu
(608) 262-6613

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

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

Keywords: Machine Learning, Connectionism

Abstract

The previously-described KBANN system is a method for integrating existing knowledge into neural networks by defining the network topology and setting initial link weights. Standard neural learning techniques can then be used to train such networks, thereby refining the information upon which the network is based. However, standard neural learning techniques ignore a significant amount of information contained in the networks created by KBANN. This paper describes an algorithm for training such networks that uses the previously-ignored information. Empirical evidence shows that this method improves not only learning speed, but also the ability of networks to generalize correctly to examples not seen during training.

* This research was partially supported by Office of Naval Research Grant N00014-90-J-1941 and National Science Foundation Grant IRI-9002413.

1 Introduction

Since the early 1980s, machine learning researchers have recognized that a learning system should make effective use of what is known about the task at hand. Explanation-based learning researchers [2, 5] have studied some aspects of the role that this knowledge, commonly called a *domain theory*, can play in the learning process. However, standard approaches to explanation-based learning cannot handle flawed knowledge. Thus, questions such as what can be done in the presence of flawed knowledge, and should initial knowledge be subject to refinement, are not addressed.

This paper describes a major expansion of KBANN [9, 10], a method of using neural networks for inductive theory refinement that addresses issues left open by early explanation-based learning researchers. Moreover, in its use of neural networks, KBANN explores the benefits of, and problems caused by, integrating existing knowledge into artificial neural networks (ANNs). This paper describes, and proposes a solution for, one of the problems resulting from knowledge integration.

KBANN creates knowledge-based neural networks (KNNs)[1] by establishing a mapping between domain theories composed of hierarchical sets of roughlu-correct, non-recursive, propositional rules and feedforward neural networks. This mapping is used to define the topology of KNNs as well as their initial link weights. By defining KNNs in this way, problems such as the choice of an initial network topology and the sensitivity of the network to its initial conditions are either eliminated or significantly reduced. Furthermore, unlike ANNs, KNNs have their attention focused upon features believed to be relevant. Thus, they are less susceptible to spurious correlations in the training data than ANNs.

After their creations, KNNs are trained using backpropagation [8], and a set of correctly classified examples, to eliminate errors in the theory upon which the KNN is based. Unfortunately, backpropagation implicitly assumes that the network being trained is randomly weighted [8, pg. 330]. Because KNNs violate this assumption, backpropagation can be less than an effective training method. In addition, backpropagation ignores the information underlying KNNs, further reducing its effectiveness for training KNNs.

This paper describes, in the next three sections, the DAID (*Directed Activation Identification and Descent*) algorithm that uses, during learning, the knowledge underlying KNNs. Empirical tests of KBANN using DAID presented in the subsequent section show that its use reduces the amount of training required by KNNs, and improves the generalization of trained KNNs. The concluding sections describe the relationship of KBANN-DAID to other work and the direction of future work on the algorithm.

2 Algorithm for KBANN-DAID

Table 1 contains a pseudocode abstraction of the KBANN-DAID algorithm. Briefly, the algorithm operates by presenting each example to a newly created KNN. If the KNN misclassifies the example, then the desired activation for that example is propagated to the parts of the network responsible for the errors (Section 3). Rather than making changes based on a sin-

[1] For this paper, ANN refers to any artificial neural network that is not knowledge based.

Table 1: The KBANN-DAID algorithm

1. Translate the roughly correct set of rules into an KNN
2. For each example in the training set
   2.1. Classify using the KNN
   2.2. If the classification is incorrect then propagate desired activation through the network (Section 3)
   2.3. Collect suggestions resulting from the example
      2.3.1. When the desired activation reaches the input units call the suggestions *strong* (Section 4.1)
      2.3.2. Call all other suggestions *weak* (Section 4.2)
3. With collected suggestions (Section 4.3)
   3.1. Find the most common receiving unit (call it ϕ)
      3.2. While there exists strong suggestions to ϕ
      3.2.1. Find the most mentioned link among the specific suggestions
      3.2.2. Make the suggested change
      3.2.3. Remove suggestions to ϕ from examples corrected by the change
4. With remaining weak suggestions, make small changes to the link weights
5. Train the KNN using backpropagation

gle example, suggestions for change from every misclassified example are collected. After all the training examples have been presented, changes are made that reflect the most common suggestions (Section 4). While the changes are very likely good, they are not guaranteed to be correct nor are they guaranteed to be complete. Thus, the final step of KBANN-DAID is to train the KNN using backpropagation over the whole set of training examples.

3 Descent of Desired Activation

The directed descent of desired activations through KNNs takes advantage of the mapping between newly created KNNs and rules upon which KNNs are based. Specifically, units in newly-created KNNs are active (i.e., have activation near one) when their corresponding consequent can be deduced. Conversely, units are inactive (i.e., have activation near zero) when their corresponding consequent cannot be deduced. This interpretation of unit activations can be used, under some circumstances, to precisely identify the cause of errors at output units and to recursively propagate the desired activation through KNNs. The rest of this section describes when DAID can and cannot identify the source of error.

3.1 Unit corresponds to a conjunct

Figure 1A represents a small KNN. Given that the bias[2] of unit A in this figure is as shown, units B, D and F are *mandatory* (i.e., B, D and F must be active) for A to be active. Thus,

[2]The bias of a unit is the minimum amount of activation that a unit must receive to be active.

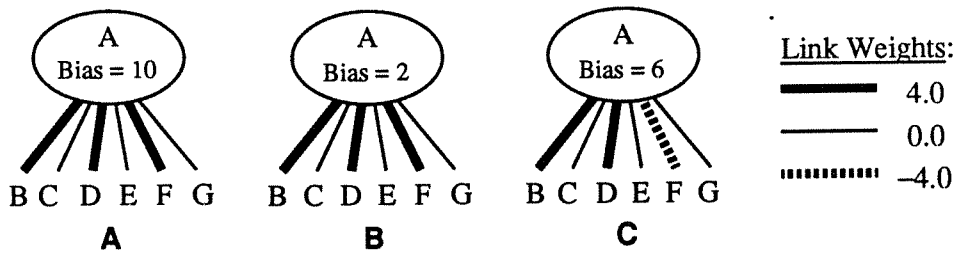

Figure 1: KNNs illustrating conjuncts, disjuncts and prohibitory links.

this KNN encodes the rule: `A:-B∧D∧F.`

When an output unit that corresponds to the consequent of a conjunctive rule (such as Figure 1A's unit A) is inactive and the desired activation of that unit is 1, then the DAID algorithm can be applied. To illustrate, suppose that unit A is inactive and that the desired activation of A is 1. Further suppose that B and F are inactive while D is active. Since B, D and F are mandatory for A, they must all be active. Only D meets this requirement. Hence, B and F can be both be blamed for the incorrect activation of A. The desired activation of B and F is known to be 1, so B and F can be treated as if they were output units and have the algorithm applied to them recursively.

Suppose instead that the training signal to Figure 1A's unit A is 0, but the unit is active. Again, for A to be active, B, D, and F must all be active; making any of these units inactive would correct A's activation. Hence, it is not possible to uniquely identify the subset of units B, D and F which should have their activations reduced. Therefore, DAID can not operate directly in this situation.

3.2 Unit corresponds to a disjunct

Figure 1B encodes the rules: `A:-B. A:-D. A:-F.`[3] The only difference between Figure 1A, which encodes a conjunctive rule, and Figure 1B, which encodes disjunctive rules, is the bias on A. This change is sufficient to allow Figure 1B's unit A to be active when one or more of B, D and F are active.

Let Figure 1B's unit A be inactive and have a desired activation of 1. Unit A can only be inactive when B, D and F are all inactive. So, raising the activation of any one of B, D or F is sufficient to make A active. However, there is no reasoned basis for choosing to raise the activation of any, or all, of these units. Thus, DAID cannot operate as the units to which the desired activation should be sent cannot be uniquely identified.

In the complementary situation, when the desired activation is 0 and unit A is active, DAID can be applied. In this case, all of B, D and F that are active can be blamed for the incorrect activation of A. Thus, the desired activation can be propagated back to those units which are active.

[3]This set of rules represents the full range of possibilities for disjuncts as rules with "internal" disjunctions are translated into a set of conjunctive rules and a single disjunctive rule with all clauses of length 1. So, `A:-(B∧C) or (D∧E) or F.` would be translated to: `A:-`A_1 `or` A_2 `or F.` A_1`:-B∧C.` A_2`:-D∧E.`

3.3 Prohibitory links

Prohibitory links are links with a large negative weight that carry signals preventing the activation of a unit. For example, Figure 1C encodes the rule: A:-B∧D∧¬F. Prohibitory links are treated by DAID exactly as links from mandatory units except that the desired activation inverts across the link. For instance, if the desired activation of Figure 1C's unit A is 1, then the desired activation of unit F (at the opposite end of a prohibitory link), would be 0. This inverted desired activation is used to determine whether or not F is to be blamed for the incorrect activation of A and is used for further propagation of the desired activation if it is indicated.

4 Link weight changes using DAID

Desired activations are propagated backward through the network by DAID until either: (1), the desired activation reaches an input unit, or (2) DAID cannot further propagate the desired activation. In both of these cases DAID can identify changes to the KNN that might be relevant. The following subsections describe the suggestions for change from each of these cases. The final part of this section describes how the suggestions are brought together and actual link weight changes are made.

4.1 Desired activations reach input units

The suggestions for changes to link weights when desired activations reach the input units are labeled ADD and DROP to reflect the action that they advocate (described below). Suggestions are dependent upon the type of rule that the unit sending the desired activation encodes and whether that unit must be generalized or specialized. Hence, this subsection is partitioned like Section 3.

4.1.1 Desired activation sent by a conjunctive unit

When the unit from which the desired activation is propagated encodes a conjunct, and the desired activation of that unit is 1, then the unit must be generalized. One method of generalization, referred to as *antecedent deletion*, is based upon the idea that the inactive mandatory units (i.e., the units causing the incorrect activation of the unit) are not important. Using Figure 2A as an example, this method suggests reducing to zero the weight on the link to the inactive mandatory input unit and decreasing the bias by an equal amount. This change doubles the number of situations in which the hidden unit is active.

A different generalization method, called *antecedent addition*, is based upon the idea that the mandatory unit responsible for the incorrect activation is important, but not mandatory. Again using Figure 2A as an example, this method would suggest increasing the weight on the link to active, non-mandatory input unit while leaving all other weights unchanged. This change increases the number of situations in which a unit is active by N where N is the original number of mandatory antecedents.

When a conjunct unit is active and the desired activation is 0, then the number of situations in which the unit can be active must be reduced. The simplest way to to this is to increase the weight on a link to an inactive input unit raise the bias of the the conjunct

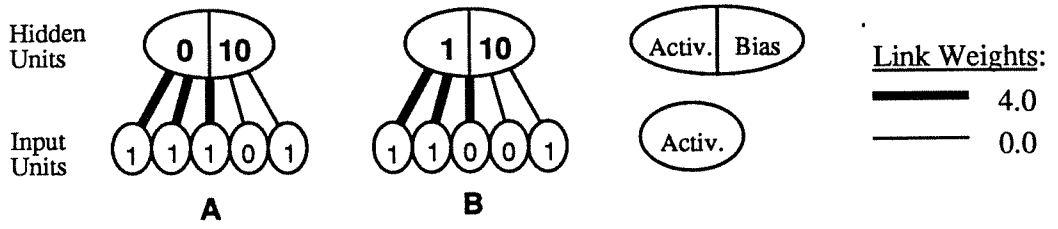

Figure 2: KNNs to illustrate suggestions for link weight change.

unit. For example, in Figure 2B the suggestion would be to increase the weight on the link to non-mandatory, inactive input unit to 4.0 while increasing the bias of the hidden unit to 14. This change reduces the number of situations in which the unit may be active by N.

4.1.2 Prohibitory links and disjunctive units

When a prohibitory unit is found to make either a conjunct unit overly specific or a disjunct unit overly general, a change similar to antecedent deletion is suggested by DAID. Specifically, the weight on the link should be reduced in magnitude to near zero while the bias on the unit is left unchanged. Antecedent addition is not appropriate in this case as it takes advantage of KBANN's arbitrary choice to give prohibitory links a weight that is the negative of the weight on mandatory links. Prohibitory antecedents could as easily be given a much larger negative weight by KBANN.

DAID's suggested change in the case of an overly-general disjunct is the same method of antecedent deletion used to handle prohibitory links. Overly-specific disjuncts are generalized by antecedent addition.

4.2 Desired activations fail to reach input units

DAID cannot propagate errors to the output units when it is unable to identify specific units to blame for the incorrect activation of the unit being processed. These situations are handled by treating each of the units among which it is impossible to place specific blame as as if they were the only unit blamed for the error. Making this assumption allows DAID to continue with its normal recursive propagation of the desired activation. However, a change in the activation of any one of the units to which the desired activation was sent would result in the needed change. Therefore, indications about which link weight changes resulting from this procedure can only be taken as weak suggestions. As a result, these suggestions are called ADD? and DROP?.

4.3 Integration of suggestions in DAID

Changes to the KNN must be selected from among the collected suggestions becuase they often overlap in that multiple changes are suggested for links sending activation to a single unit and multiple changes are suggested to correct a single error. For each receiving unit, overlapping suggestions are resolved by grouping suggestions by the affected link. The link with the most ADD or DROP suggestions is then modified. When several links have the

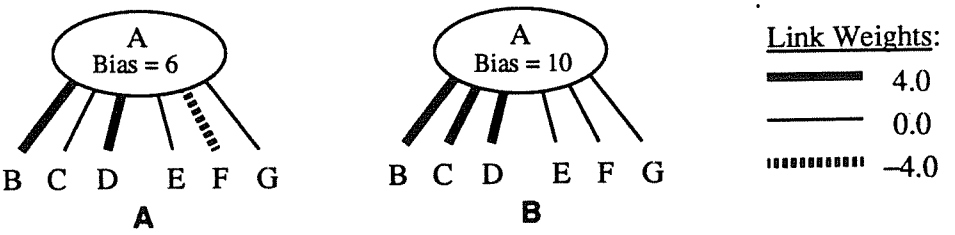

Figure 3: Changes made to a unit using DAID.

same number of suggestions, the tie is resolved by choosing the link with the most ADD? and DROP? suggestions. If the tie persists, the tied links are all appropriately modified. Following this change, all suggestions that result from examples addressed by the change just made are removed from the list of suggestions. This process repeats until no ADD or DROP suggestions remain.

For example, the following is a hypothetical list of change suggestions collected over all misclassified training examples for the hidden unit in Figure 3A. Each suggestion has four parts: the example that caused the suggestion, the unit to which the link connects, the unit from which the link originates and the action to be taken.

(ex1 A B DROP)	(ex1 A C ADD)	(ex1 A E ADD)	(ex3 A B DROP)
(ex3 A C ADD)	(ex5 A F DROP)	(ex7 A B DROP)	(ex7 A C ADD)
(ex7 A G ADD)	(ex9 A C ADD?)	(ex9 A G ADD?)	(ex10 A E ADD?)

Grouping specific (i.e., ADD and DROP) suggestions by frequency of occurrence results in a tie between (1) deleting the link from B to A, and (2) adding a link from C to A. The tie is resolved in favor of the added link due to the weak suggestion of ex9. Making this change resolves the errors of: ex1, ex3, ex7 and ex9. All suggestions related to these examples are removed from consideration leaving only the suggestions related to ex5 and ex10. As the suggestion related to ex5 is specific, that suggestion is followed and the link from F to A, deleted. After that link is deleted, all errors resulting in specific suggestions are resolved, so DAID takes no immediate additional actions on the links into unit A. Following these changes, the links weights and biases for unit A would be as in Figure 3B.

As a final step DAID collects the ADD? and DROP? of all examples for which no change has been made that addresses the identified error. For all such suggestions, DAID makes small changes to link weights in the direction indicated by the suggestion.

5 Empirical Tests of KBANN-DAID

To demonstrate the efficacy of KBANN-DAID, this section presents a series of tests that compare the algorithm to KBANN without DAID and to standard ANNs. These tests support previous results that KBANN is superior to standard ANNs [10] and indicate that KBANN-DAID is superior to both.

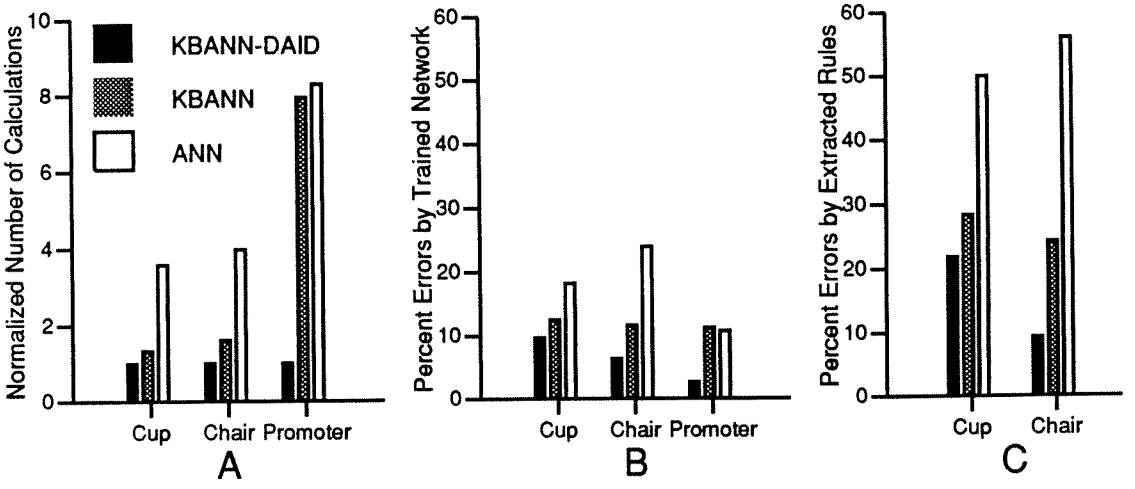

Figure 4: Results of tests comparing learning rate and generalization ability.

normally use $\epsilon = 0.25$; N is the number of training examples.) During testing, examples are assigned the classification of the class to which they are closer. The cross-entropy function suggested by Hinton [4] is used as the measure of error for all training, as it has been found to speed learning in networks created by KBANN [11]. The learning rate is set to 0.04 and momentum to 0.9, values empirically found to work well given this training approach and the cross-entropy error measure.

Initial networks for KBANN-DAID and standard KBANN are produced by KBANN[10]. The initial ANNs are completely connected networks with a single layer of hidden units. For cups, there are two hidden units in the ANN. For promoter recognition ANNs have 16 hidden units, the same number of hidden units as the KNN.

Figures 4A-C, respectively, plot the relative computational effort required to learn the training set, the error rate of trained networks on testing sets and the error rate of rules extracted from trained networks on testing sets.[6] The training set for both cup recognition consisted of four positive and four negative examples of the concept selected randomly from the example population.[7] The testing set consists of 50 positive and 50 negative examples of the category, also randomly drawn from the population. For promoters, the training set consists of 86 examples randomly selected from the 106 examples. The testing set contained the 20 examples not used during training.

For each training and testing set, networks were trained and tested 10 times to smooth differences resulting from presentation order during training and variations in the initial weights of the networks. The same training and testing examples were used to test each of

[6]The "chair" dataset, not described due to space limitations, is a toy problem similar to cup recognition [9]. Training and testing of chairs follows that for cups.

[7]Examples are randomly drawn by randomly setting the value of each feature of the environment. The example created in this manner is then classified using the ideal rule set and grouped with positive or negative examples accordingly.

Table 6: Statistical analysis of experimental results.

	Learning Method	Learning Rate	Errors by Network	Errors by Extracted Rules
Cup	KBANN-DAID vs KBANN	2.71 (0.01)	2.81 (0.01)	1.23 (—)
	KBANN-DAID vs. ANN	12.66 (0.005)	4.69 (0.005)	14.22 (0.005)
	KBANN vs. ANN	13.29 (0.005)	4.35 (0.005)	12.87 (0.005)
Chair	KBANN-DAID vs KBANN	1.80 (—)	5.93 (0.005)	4.80 (0.005)
	KBANN-DAID vs. ANN	10.21 (0.005)	15.22 (0.005)	8.91 (0.005)
	KBANN vs. ANN	10.54 (0.005)	9.65 (0.005)	6.01 (0.005)
Promoter	KBANN-DAID vs KBANN	16.17 (0.005)	7.21 (0.005)	—
	KBANN-DAID vs. ANN	35.04 (0.005)	7.74 (0.005)	—
	KBANN vs. ANN	0.74 (—)	-0.84 (—)	—

the three learning systems. Figure 4 is based upon 20 repetitions of this procedure. Thus, each bar represents the average of 200 learning trials.

Figure 4C plots the ability of symbolic rules automatically extracted from trained networks to correctly classify testing examples. Extraction uses a technique described in [11]. Unfortunately, this technique currently does not work over large networks such as those for promoter recognition. Hence, this data does not appear in either Figure 4C or Table 6.

Table 6 presents a statistical analysis of the data in Figure 4 using a paired sample t-test. Each entry in the table contains of two numbers, the t statistic and the probability that the difference is insignificant. Negative values of the t statistic indicate that the average of the second value is lower than the average of the first.

6 Discussion

This section analyses the results presented previously. In addition, the final subsection attempts to identify the differences between KBANN-DAID and backpropagation that account for the success of the algorithm this paper describes.

6.1 Rate of learning

Based only upon the amount of computation required for training, KBANN-DAID is clearly superior to both standard KNNs and ANNs. It required about one-eighth the computation of the other learning methods to correctly categorize the promoter training examples, and never learned more slowly than the other methods.

6.2 Generalization

KNNs trained using KBANN-DAID generalize significantly better than either standard KNNs or ANNs. (Unreported results indicate that KBANN-DAID generalizes significantly better on promoter recognition than KBANN without DAID using networks described in previous work [10].) The improvement in generalization seems to be caused by DAID moving the KNN to a point in its weight space from which few changes are required. As a result, there is little

Table 2: Features of the cup environment.

Features	Values	Features	Values	Features	Values
Has Handle	yes, no	Handle on Top	yes, no	Handle on Side	yes, no
Bottom is Flat	yes, no	Has Concavity	yes, no	Concavity Up	yes, no
Light	yes, no	Fragile	yes, no	Expensive	yes, no
Made of	Ceramic, Paper, Styrofoam				

Table 3: A roughly-correct rule set for recognizing cups.

Cup	:-	Stable, Liftable, Open-Vessel.
Stable	:-	Flat-Bottom.
Liftable	:-	Graspable, Light.
Graspable	:-	Has-Handle.
Open-vessel	:-	Has-Concavity, Concavity-Points-Up.

Table 4: Changes to initial cup rules to create the ideal.

ADD	Graspable	:-	Handle-on-Side.
ADD	Graspable	:-	Made-of-Styrofoam, (NOT Handle-on-Top).
DEL	Graspable	:-	Has-Handle.

5.1 Datasets

The principle dataset is a set of examples and a roughly-correct set of rules for the real-world problem of DNA *promoter* recognition. This dataset has previously been used to test integrated learning systems such as KBANN [10] and EITHER [6]. The other dataset, cup recognition, is one of two toy problems previously used to test KBANN [9]. Results of tests this dataset will be used to support and illustrate conclusions derived from tests on the promoter dataset.

5.1.1 Cup recognition

The Cup dataset consists of three sets of information. First, a description of the information available about objects. This information, summarized in Table 2, contains the features used to describe objects and the values that those features may have. Second, a roughly-correct set of rules, in Table 3, that describes what it means to be a cup in terms of the features in Table 2. Third, an "ideal" rule set (i.e., a set of rules that correctly label items drawn from the general population). The ideal set is never seen by the learning system; it serves as a reference point. That is, the learning problem is to adjust the initial rules so that they mimic the ideal. The ideal rule set is defined in Table 4 in terms of the changes to the initial set of rules required to make the ideal set.

5.1.2 Promoter recognition

This dataset consists of a set of 106 training examples evenly divided between positive and negative instances and a rule set extracted from the biological literature.[4] The rule set

[4]Promoters are a short piece of DNA that encourage the expression of a gene. The dataset is described more fully in [10].

Table 5: A roughly-correct rule set for recognizing promoters.

promoter	:- contact, conformation.	
contact	:- minus_35, minus_10.	
minus_35	:- @-37 "cttgac".	minus_35 :- @-36 "ttgxca".
minus_35	:- @-36 "ttgaca".	minus_35 :- @-36 "ttgac".
minus_10	:- @-14 "tataat".	minus_10 :- @-13 "taxaxt".
minus_10	:- @-13 "tataat".	minus_10 :- @-12 "taxxxt".
conformation	:- @-45 "aaxxa".	
conformation	:- @-45 "axxxa", @-4 "t", @-28 "txxxtxaaxxtx".	
conformation	:- @-49 "axxxxt", @-1 "a", @-27 "txxxxaxxtxtg".	
conformation	:- @-47 "caaxttxac", @-22 "gxxxtxc", @-8 "g[illegible]xcc".	

(see Table 5) for promoters is more complex than for cups; it contains more rules and is disjunctive. There is no ideal for promoter recognition because it is a real-world problem with no general solution.

In addition to the rules of Table 5, there is an unstated rule that regions of the DNA fragment not appearing in the rules are insignificant. Thus, KNNs based upon the promoter-recognition rules will be connected to only those locations that the rules indicate are significant. This unstated rule reduces by an order of magnitude the number of connections in KNNs for promoter recognition. Note that using this implicit rule changes the promoter recognition problem from that reported previously [10].

5.2 Tests of KBANN-DAID

Two common measures with which one can evaluate a training strategy are: (1) the amount of training required to learn the training set, and (2) the ability of the learner to generalize from a set of training examples to the general population. For ANNs a good method for evaluating the first criterion is the number of calculations that are made until no training examples are misclassified. The second criterion is normally approximated by measuring the error rate on a distinct test set after the training set has been learned. These measures will be used to compare KBANN-DAID, KBANN without DAID, and standard ANNs in the following section.

5.3 Results

All results presented in this section use the same training method during backpropagation. Specifically, examples are presented using a "pick and replace" procedure that is biased towards picking previously misclassified examples.[5] Links weights are changed after every example presentation. Networks are considered to have learned the training set when the outputs are within ϵ of the desired level for $3 * N$ consecutive training examples. (We

[5]The probability of selecting a particular example is directly related to $e_i / \sum_{j=1}^{n} e_j$ where e_i is the difference betweeb the actual and desired output for an example.

Table 7: Misclassified examples over multiple runs on a single training set.

	DAID	Standard KNN	ANN
0	91	668 987 TNAA 835	668 987 91 521 UVRB-P2 835
1	91	987 835	668 987 91 753 464 UVRB-P2
2	91	668 1163	668 987 TNAA 91 521 753 UVRB-P2 835
3	91	MALK 835	668 987 521 UVRB-P2
4	91	TNAA	668 91 1203 753 557 464 UVRB-P2 835
5	91	1163 91	987 753 464 UVRB-P2
6	91	TNAA 835	987 ARAC MALK 464 UVRB-P2
7	91	987	668 1163 91 521 753 557 464
8	91	668 835	668 987 91 753 464 UVRB-P2
9	91	MALK 835	668 987 753 MALK 464 UVRB-P2

opportunity for learning spurious correlations that detract from generalization. Two pieces of evidence support this hypothesis.

First, KNNs trained using KBANN-DAID are more consistent in their classification of testing examples than both standard KNNs and ANNs. For instance, Table 7 lists the examples incorrectly classified by each system over a set of 10 trials for a single training and testing set on the promoter recognition problem.[8] KBANN-DAID networks consistently got the same example incorrect. By contrast, standard KNNs averaged two incorrect classifications, but made mistakes on seven different examples. Even more inconsistent were ANNs; misclassifying 14 different examples while averaging six mistakes.

Second, link weights in KNNs trained using KBANN-DAID are very consistent. For example, consider a network that is the average of the 10 networks trained using the same set of examples. For KBANN-DAID, the average network always made errors made by the majority of the averaged networks. By contrast, average ANNs frequently performed little better than chance and average standard KNNs occasionally misclassified examples that were correctly by the majority of the averaged KNNs. This consistency of the average network when KBANN-DAID is used for training indicates that networks trained using this algorithm are less sensitive to initial conditions and training example presentation order than networks trained using other methods. Also, consistency is an important consideration for the future work of automating interpretation of KNN; mentioned briefly with Figure 4C, this is a development area for the algorithm.

6.3 KBANN-DAID and Backpropagation

The principle difference between KBANN-DAID and backpropagation is the locus of change in the network. Backpropagation changes weights throughout the network with a slight bias towards changes near the output units. For ANNs this bias is perfectly acceptable. However, for KNNs this bias may tend to hinder learning. For example, on the promoter recognition

[8]Names like "MALK" and "TNAA" indicate promoters while numbers like "91" and "668" indicate nonpromoters.

task, there are good biological reasons to believe that regions near position -35 (i.e., 35 nucleotides before the start of the gene) and -10 are significant. So changing the weights to any of these rules is undesirable. However, the important nucleotides in those regions are less certain. Thus, changes to KNNs for promoter recognition should be concentrated at the links between hidden units and inputs units rather than higher up in the network. This is exactly the effect of KBANN-DAID.

William Whewell's philosophical work on the "consilience" of inductions [12] supports the idea that changes to KNNs should be concentrated near the inputs. Specifically, Whewell suggested that the most certain conclusions are those which are built on top of other conclusions. For KNNs this implies that changes should initially be at the input levels and only when those changes fail to make the required corrections should changes at levels higher up in the network be considered. Therefore, Whewell's theory and an analysis of the promoter domain both indicate that an algorithm similar to KBANN-DAID is desirable for training KNNs.

7 Related Work

DAID was inspired by, but differs significantly from, Hall's work on *learning by failing to explain* (LFE) [3]. Hall's approach works under the assumption that the rules provided to the learning system are correct but incomplete. By contrast, KBANN-DAID allows rules to be both incorrect and incomplete. LFE proposes rules by connecting parts of "bottom-up" and "top-down" explanations. Hence, unlike KBANN-DAID, LFE may propose rules at any level in the explanation. Furthermore, the rules proposed by LFE are guaranteed correct while the changes proposed by DAID have no such guarantee. Finally, LFE and other theory refinement systems (e.g., [7, 1]) make an assumption there is a single source of error in a given explanation. KBANN-DAID does not require this assumption.

Like, KBANN-DAID, Ourston and Mooney's EITHER [6] does not require the single fault assumption. EITHER has an advantage over KBANN-DAID in that the revisions are made directly to the rules rather than to an alternate expression of the rules. Thus, after the changes have been made, they can be read directly. However, tests on the promoter recognition problem indicate that EITHER is less accurate than the system described in this paper.

8 Future Work

The principle area for future work on KBANN-DAID is to achieve a closer integration of the algorithm and backpropagation. Currently DAID acts like a preprocessor for KNNs because it requires the clarity of interpretation available in the initial set of rules. After running either backpropagation or DAID on the network, the rules have been obscured sufficiently to preclude DAID's reuse.

Work on integration will focus on two areas. First, the further development of techniques that recognize situations in which DAID can work. These techniques could allow the system to decide for each example whether or not the clarity required by DAID exists. Second, techniques that allow DAID to work in a broader set of situations. For instance, DAID cannot currently handle predicates that require N of M antecedents to be true.

A second area of development is to alter the process of creating KNNs. This could involve using DAID to add links during training. Currently, links with near-zero weights are added to KNNs so that intermediate and final conclusions have access to information not specified in the initial, partially-incorrect, rules. DAID merely identifies the added links that might have their weight increased. The algorithm might be adapted to identify places where links should be added. This ability would eliminate the need to initially add near-zero weight links, thereby significantly reducing the number of links in KNNs. Reducing the number of links may improve generalization of KNNs, by preventing KNNs from overfitting the training data, and reduce computation required for learning.

Finally, we are working on methods to automatically extract symbolic rules from trained KNNs. Extraction is important as it would close the information passing loop between KNNs and their users. Currently we use a brute force technique based upon link weights and biases in trained networks. However, this technique bogs down over large networks. We are developing methods that use the initial knowledge in KNNs to constrain the search required of our current interpretation algorithm.

9 Conclusions

The KBANN system for theory refinement uses neural learning techniques to modify sets of roughly-correct rules by transforming the rules into KNNs [10]. However, backpropagation does not use all of the information available to it when operating over KNNs. This observation motivated the creation of the DAID algorithm that uses the knowledge upon which KNNs are based and a set of training examples to make a good set of initial changes to KNNs. Empirical tests showed this method to be effective on several different levels. First, it reduces the amount of training that KNNs require to learn a set of training examples. Second, it improves the generalization ability of KNNs. Finally, it enhances the consistency of learning, making future work on the automatic interpretation of trained KNNs possible.

Beyond its empirically-proven learning ability, the KBANN-DAID has philosophical support in the work of William Whewell. He suggests that theories built upon other theories are more reliable than the theories upon which they are built. Applying this theory to KNNs suggests that the primary place that changes to KNNs are required is at the connection to input units. This is precisely the place where DAID changes the network. Thus, there are both empirical and theoretical reasons to believe that methods like DAID are useful for training KNNs.

References

[1] A. P. Danyluk. Finding new rules for incomplete theories: Explicit biases for induction with contextual information. In *Proceedings of the Sixth International Machine Learning Workshop*, pages 34–36, Ithaca, NY, 1989.

[2] G. F. DeJong and R. F. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1:145–176, 1986.

[3] R. J. Hall. Learning by failing to explain: Using partial explanations to learn in incomplete or intractable domains. *Machine Learning*, 3:45–77, 1988.

[4] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[5] T. M. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80, 1986.

[6] D. Ourston and R. J. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 815–820, Boston, MA, Aug 1990.

[7] M. J. Pazzani. Detecting and correcting errors of omission after explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 713–718, Detroit, MI, Aug 1989.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: explorations in the microstructure of cognition. Volume 1: Foundations.* MIT Press, Cambridge, MA, 1986.

[9] J. W. Shavlik and G. G. Towell. An approach to combining explanation-based and neural learning algorithms. *Connection Science*, 1:233–255, 1989.

[10] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of approximately correct domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990.

[11] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. The operation of knowledge based neurtal networks. Technical report, University of Wisconsin, 1991.

[12] W. Whewell. *Theory of the Scientific Method.* Hackett, Indianapolis, 1989. Originally published in 1840.