Appears in *Proceedings of the Tenth National Conference on Artificial Intelligence*

Using Symbolic Learning to Improve Knowledge-Based Neural Networks*

Geoffrey G. Towell† **Jude W. Shavlik**
towell@learning.siemens.com shavlik@cs.wisc.edu
University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

Abstract

The previously-reported KBANN system integrates 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 are reputed to have difficulty training networks with multiple layers of hidden units; KBANN commonly creates such networks. In addition, standard neural learning techniques ignore some of the information contained in the networks created by KBANN. This paper describes a symbolic inductive learning algorithm for training such networks that uses this previously-ignored information and which helps to address the problems of training "deep" networks. Empirical evidence shows that this method improves not only learning speed, but also the ability of networks to generalize correctly to testing examples.

Introduction

KBANN is a "hybrid" learning system; it combines rule-based reasoning with neural learning to create a system that is superior to either of its parts. Using both theory and data to learn categorization tasks, KBANN has been shown to be more effective at classifying examples not seen during training than a wide variety of machine learning algorithms (Towell *et al.*, 1990; Noordewier *et al.*, 1991; Towell, 1991).

However, recent experiments (briefly described on the next page) point to weaknesses in the algorithm. In addition, neural learning techniques are commonly thought to be relatively weak at training networks that have several layers of hidden units. Unfortunately, the networks created by KBANN (KBANN-nets) frequently have this "deep network" property. Hence, algorithms such as backpropagation (Rumelhart *et al.*, 1986) may not be well suited to training KBANN-nets.

To address both this problem with the training of KBANN-nets and KBANN's empirically discovered weaknesses, this paper introduces the DAID *(Desired Antecedent IDentification)* algorithm. Following a description of DAID, we present results which empirically verify that DAID achieves both of its goals.

The KBANN Algorithm

KBANN, illustrated in Figure 1, is an approach to combining rule-based reasoning with neural learning. The principle part of KBANN is the rules-to-network translation algorithm, which transforms a knowledge base of domain-specific inference rules (that define what is initially known about a topic) into a neural network. In so doing, the algorithm defines the topology and connection weights of the networks it creates. Detailed explanations of this rules-to-network translation appear in (Towell *et al.*, 1990; Towell, 1991).

As an example of the KBANN rules-to-network translation method, consider the small rule set in Figure 2a that defines membership in category A. Figure 2b represents the hierarchical structure of these rules: solid and dotted lines represent necessary and prohibitory dependencies, respectively. Figure 2c represents the KBANN-net that results from the translation of this do-

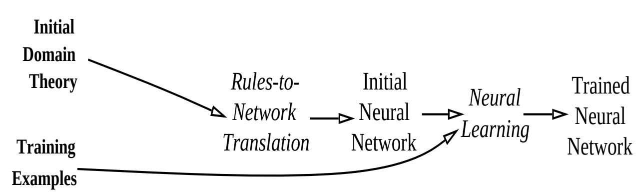

Figure 1: The flow of information through KBANN.

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

†Currently at: Siemens Corporate Research, 755 College Road East, Princeton, NJ 08540.

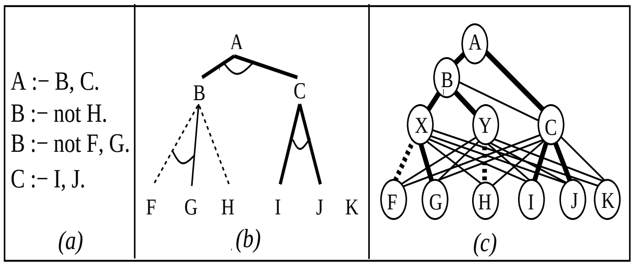

Figure 2: Translation of a domain theory into a KBANN-net.

main knowledge into a neural network. Units X and Y in Figure 2c are introduced into the KBANN-net to handle the disjunction in the rule set (Towell *et al.*, 1990). Otherwise, each unit in the KBANN-net corresponds to a consequent or an antecedent in the domain knowledge. The thick lines in Figure 2c represent heavily-weighted links in the KBANN-net that correspond to dependencies in the domain knowledge. Weights and biases in the network are set so that the network's response to inputs is exactly the same as the domain knowledge. The thin lines represent links with near zero weight that are added to the network to allow refinement of the domain knowledge. *More intelligent initialization of the weights on these thin lines is the focus of this paper.*

This example illustrates the two principal benefits of using KBANN. First, the algorithm indicates the features that are believed to be important to an example's classification. Second, it specifies important derived features, thereby guiding the choice of the number and connectivity of hidden units.

Initial Tests of KBANN

The tests described in this section investigate the effects of *domain-theory noise* on KBANN. The results of these tests motivated the development of DAID.

These tests, as well as those later in this paper, use real-world problems from molecular biology. The *promoter recognition* problem set consists of 106 training examples split evenly between two classes (Towell *et al.*, 1990). The *splice-junction determination* problem has 3190 examples in three classes (Noordewier *et al.*, 1991). Each dataset also has a partially correct domain theory.

Earlier tests showing the success of KBANN did not question whether KBANN is robust to domain-theory noise. The tests presented here look at two types of domain-theory noise: *deleted* antecedents and *added* antecedents. Details of the method used for modifying existing rules by adding and deleting antecedents, as well as studies of other types of domain-theory noise, are given in (Towell, 1991).

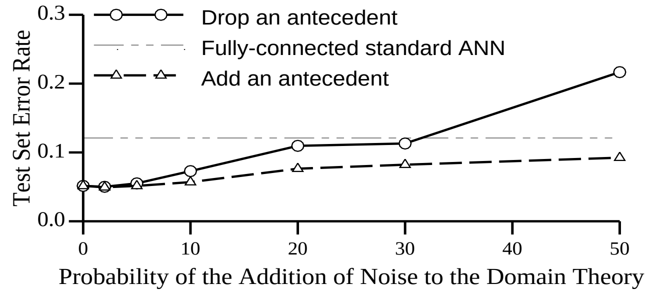

Figure 3: Effect of antecedent-level noise on the classification accuracy in the promoter domain.

Figure 3 presents the results of adding noise to the promoter domain theory. All results represent an average over three different additions of noise. Eleven randomized runs of ten-fold cross-validation (Weiss and Kulikowski, 1990) are used to test generalization. Not surprisingly, this figure shows that test set error rate increases directly with the amount of noise. More interesting is that the figure shows the effect of deleting antecedents is consistently larger than the effect of adding antecedents.

Clearly, irrelevant antecedents have little effect on KBANN-nets; with 50% noise[1] the performance of KBANN-nets is still superior to that of a fully-connected standard ANN (i.e., an Artificial Neural Network with one layer of hidden units that are fully connected to both the input and output units). Conversely, dropping only 30% of the original antecedents degrades the performance of KBANN-nets to below that of standard ANNs.

Symbolic Induction on Domain Theories

The experiments in the previous section indicate that it is easier for KBANN to *discard* antecedents that are useless than to *add* antecedents initially believed to be irrelevant. Hence, a method that tells KBANN about potentially useful features not mentioned in the domain theory might be expected to improve KBANN's learning abilities.

The DAID algorithm, described and tested in the remainder of this paper, empirically tests this hypothesis. It "symbolically" looks through the training examples to identify antecedents that may help eliminate errors in the provided rules. *This is* DAID*'s sole goal.* The algorithm accomplishes this end by estimating correlations between inputs and corrected intermediate conclusions. In so doing, DAID suggests more antecedents than are

[1] 50% noise means that for every two antecedents specified as important by the original domain theory, one spurious antecedent was added.

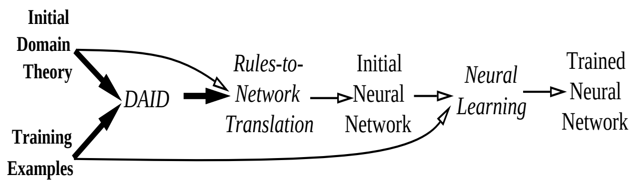

Figure 4: Flow of information using KBANN-DAID.

useful; it relies upon KBANN's strength at rejecting useless antecedents.

DAID adds an algorithmic step to KBANN, as the addition of the thicker arrows in Figure 4 illustrates. Briefly, DAID uses the initial domain theory and the training examples to supply information to the rules-to-network translator that is not available in the domain theory alone. As a result, the output of the rules-to-network translator is not simply a recoding of the domain theory. Instead, the initial KBANN-net is slightly, but significantly, shifted from the state it would have assumed without DAID.

Overview of the DAID algorithm

The assumption underlying DAID is that errors occur primarily at the lowest levels of the domain theory.[2] That is, DAID assumes the only rules that need correction are those whose antecedents are all features of the environment. DAID's assumption that errors occur solely at the bottom of the rule hierarchy is significantly different from that of backpropagation (and other neural learning methods). These methods assume that error occurs along the whole learning path. As a result, it can be difficult for backpropagation to correct a KBANN-net that is only incorrect at the links connecting input to hidden units. Thus, one of the ways in which DAID provides a benefit to KBANN is through its different learning bias.

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

[2]This idea has a firm philosophical foundation in the work of William Whewell. His theory of the *consilience* of inductions suggests that the most uncertain rules are those which appear at the lowest levels of a rule hierarchy and that the most certain rules are those at the top of the hierarchy (Whewell, 1989).

Algorithm specification

To determine correlations between inputs and intermediate conclusions, DAID must first determine the "correct" truth-value of each intermediate conclusion. Doing this perfectly would require DAID to address the full force of the "credit-assignment" problem (Minsky, 1963). However, DAID need not be perfect in its determinations because its goal is simply to identify *potentially* useful input features. Therefore, the procedure used by DAID to track errors through a rule hierarchy simply assumes that every rule which can possibly be blamed for an error is to blame. DAID further assumes that all the antecedents of a consequent are correct if the consequent itself is correct. These two ideas are encoded in the recursive procedure BACKUPANSWER that is outlined in Table 1, and which we describe first.

BACKUPANSWER works down from any incorrect final conclusions, assigning blame for incorrect conclusions to any antecedents whose change could lead to the consequent being correct. When BACKUPANSWER identifies these "changeable" antecedents, it recursively descends across the rule dependency. As a result, BACKUPANSWER visits every intermediate conclusion

---

Table 1: Summary of the DAID algorithm.

DAID:

`GOAL: Find input features relevant to the corrected low-level conclusions.`

1. Establish eight counters (see text) associating each feature-value pair with each of the lowest-level rules.
2. Cycle through each of the training examples and do the following:
   - Compute the truth value of each rule in the original domain theory.
   - Use BACKUPANSWER to estimate the correct value of each lowest-level consequent.
   - Increment the appropriate counters.
3. Compute correlations between each feature-value pair and each of the lowest-level consequents.
4. Suggest link weights according to correlations.

BACKUPANSWER:

`GOAL: Determine the ''correct'' value of each intermediate conclusion.`

1. Initially assume that all antecedents are correct
2. For each antecedent of the rule being investigated:
   - Determine the correctness of the antecedent
   - Recursively call BACKUPANSWER if the antecedent is incorrect

---

which can be blamed for an incorrect final conclusion.

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

- does the feature have this value?
- is the consequent true for this et of features?
- does the value of the consequent agree with BACKUPANSWER's expected value?

Hence, each of the lowest-level consequents must maintain eight counters for each input feature (see Table 1). DAID goes through each of the training examples, running BACKUPANSWER, and incrementing the appropriate counters.

After the example have been presented, the counters are combined into the following four correlations[3]: Notice that these correlations look only at the relation-

1. between conseq(incorrect|false) and feat(present), i.e., a consequent being false and disagreeing with BACKUPANSWER and a feature being present,
2. between conseq(correct|false) and feat(present),
3. between conseq(incorrect|true) and feat(present),
4. between conseq(correct|true) and feat(present).

ship between features that are present and the state of intermediate conclusions. This focus on features that are present stems from the DAID's emphasis on the addition of antecedents. Also, in our formulation of neural networks, inputs are in the range $[0 \ldots 1]$ so a feature that is not present has a value of zero. Hence, the absence of a feature cannot add information to the network.

DAID makes link weight suggestions according to the relationship between the situations that are aided by the addition of a highly-weighted features and those that are hurt by the addition of a highly-weighted feature. For instance, correlation 1 between a feature and a consequent has a large positive value when adding that feature to the rule would correct a large portion of the occasions in which the consequent is incorrectly false. In other words, correlation 1 is sensitive to those situations in which adding a positively-weighted antecedent would correct the consequent. On the other hand, correlation 2 has a large positive value when adding a positively-weighted antecedent would make a correct consequent incorrect. Hence, when suggesting that a feature be given a large positive weight it is important to consider both the value of correlation 1 and the difference between correlations 1 and 2. Similar reasoning suggests that the best features to add with a large negative weight are those with the largest differences between correlations 3 and 4. Actual link weight suggestions are a function of the difference between the relevant correlations and the initial weight of the links corresponding to antecedents in the domain theory (i.e., the thick lines in Figure 2).[4]

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

Example of the algorithm As an example of the DAID algorithm, consider the rule set whose hierarchical structure is depicted in Figure 5. In this figure, solid lines represent unnegated dependencies while dashed lines represent negated dependencies. Arcs connecting dependencies indicate conjuncts.

Figure 6 depicts the state of the rules after the presentation of each of the three examples. In this figure, circles at the intersections of lines represent the values computed for each rule – filled circles represent "true" while empty represent "false." The square to the right of each circle represents the desired truth value of each consequent as calculated by the BACKUPANSWER procedure in Table 1. (Lightly-shaded squares indicate that the consequent may be either true or false and be considered correct. Recall that in BACKUPANSWER, once the value of a consequent has been determined to be correct, then all of its dependencies are considered to be correct.)

Consider, for example, Figure 6*i* which depicts the rule structure following example *i*. In this case the final consequent *a* is incorrect. On its first step backward, BACKUPANSWER determines that *c* has an incorrect truth value while the truth value of *b* is cor-

[3]The correlations can be computed directly from the counters because the values being compared are always either 0 or 1.

[4]Our use of correlations to initialize link weights is reminiscent of Fahlman and Lebiere's (1989) cascade correlation. However, our approach differs from cascade correlation in that the weights given by the correlations are subject to change during training. In cascade correlation, the correlation-based weights are frozen.

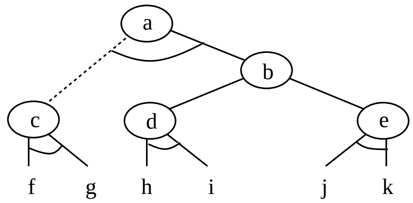

Figure 5: Hierarchical structure of a simple rule set.

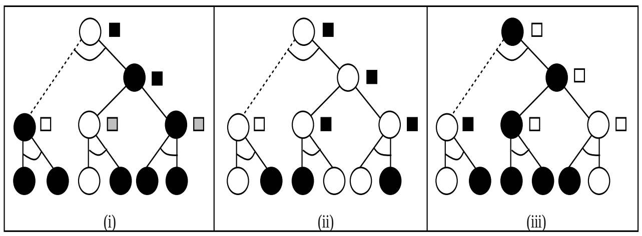

Figure 6: The state of the rule set after presenting three examples.

rect. (Desired truth values invert across negative dependencies.) Because b is correct, all its supporting antecedents are considered correct regardless of their truth values. Hence, both d and e are correct.

After seeing the three examples in Figure 6, DAID would recommend that the initial weights from f to d and e remain near 0 while the initial weight from f to c be set to a large negative number. However, the suggestion of a large negative weight from f to c would be ignored by the rules-to-network translator of KBANN because the domain theory specifies a dependency in that location.

Tests of KBANN-DAID

Results in this section demonstrate the effectiveness of the KBANN-DAID combination along two lines: (1) generalization, (2) effort required to learn. Following the methodology of the results reported earlier, these results represent an average of eleven ten-fold cross-validation runs.

Figure 7 shows that DAID improves generalization by KBANN-nets in the promoter domain by almost two percentage points. The improvement is significant with 99.5% confidence according to a one-tailed t-test. DAID only slightly improves generalization for splice-junctions.

Also important is the computational effort required to learn the training data. If DAID makes learning easier for KBANN-nets, then it might be expected to appear in the training effort as well as the correctness reported above. Figure 8 plots the speed of learning on both splice-junctions and promoters. (DAID requires about the number of arithmetic operations as in a single backpropagation training epoch.) Learning speed is measured in terms of the number of arithmetic operations required to learn the training set.

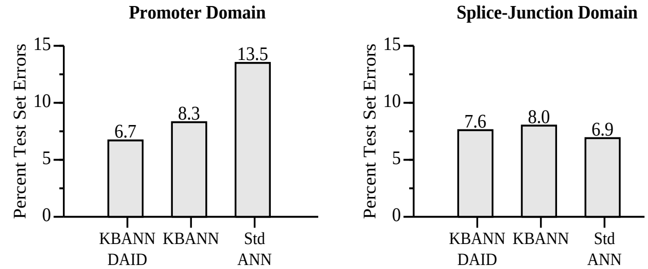

Figure 7: Generalization using DAID. (Estimated using 10-fold cross-validation).

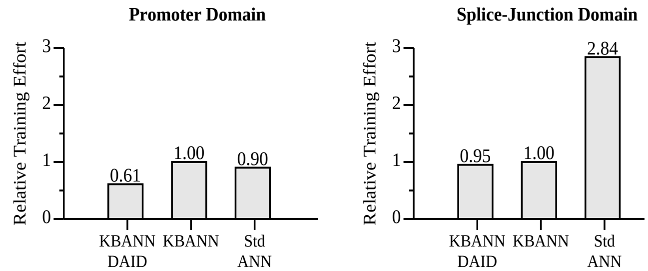

Figure 8: Training effort required when using DAID. (Effort is normalized to basic KBANN.)

The results show that DAID dramatically speeds learning on the promoter problem. (This result is statistically significant with greater than 99.5% confidence.) DAID also speeds learning on the splice-junction problem. However, the difference is not statistically significant.

In summary, these results show that DAID is effective on the promoter data along both of the desired dimensions. DAID significantly improves both the generalization abilities, and learning speed of KBANN-nets. Conversely, on the splice-junction dataset, DAID has little effect. The difference in the effect of DAID on the two problems is almost certainly due to the nature of the respective domain theories. Specifically, the domain theory for splice-junction determination provides for little more than Perceptron-like learning (Rosenblatt, 1962), as it has few modifiable hidden units. (Defining "depth" as the number of layers of modifiable links, the depth of the splice-junction domain theory is one for one of the output units and two for the other.) Hence, the learning bias that DAID contributes to KBANN – to changes at the lowest level of the domain theory – is not significant. On the other hand, the promoter domain theory has a depth of three. This is deep enough that there is a significant difference in the learning biases of DAID and backpropagation.

Future Work

We are actively pursuing two paths with respect to KBANN-DAID. The simpler of the paths is the investigation of alternate methods of estimating the appropriate link weights. One approach replaces correlations with ID3's (Quinlan, 1986) information gain metric to select the most useful features for each low-level antecedent. This approach is quite similar to some aspects of EITHER (Ourston and Mooney, 1990). Another method we are investigating tracks the specific errors addressed by each of the input features. Rather than collecting error statistics across sets of examples, link weights are assigned to the features to specifically correct all of the initial errors.

The more challenging area of our work is to achieve a closer integration of DAID with backpropagation. Currently DAID can be applied only prior to backpropagation because it assumes that the inputs to each rule can be expressed using Boolean logic. After running either backpropagation or DAID, this assumption is invalid. As a result, DAID's (re)use is precluded. It may be possible to develop techniques that recognize situations in which DAID can work. Such techniques could allow the system to decide for each example whether or not the clarity required by DAID exists. Hence, DAID would not be restricted to application prior to neural learning.

Conclusions

This paper describes the DAID preprocessor for KBANN. DAID is motivated by two observations. First, neural learning techniques have troubles with training "deep" networks because error signals can become diffused. Second, empirical studies indicate that KBANN is most effective when its networks must learn to ignore antecedents (as opposed to learning new antecedents). Hence, DAID attempts to identify antecedents, not used in the domain knowledge provided to KBANN, that may be useful in correcting the errors of the domain knowledge. In so doing, DAID aids neural learning techniques by lessening errors in the areas that these techniques have difficulty correcting.

DAID is a successful example of a class of algorithms that are not viable in their own right. Rather, the members of this class are symbiotes to larger learning algorithms which help the larger algorithm overcome its known deficiencies, Hence, DAID is designed to reduce KBANN's problem with learning new features. Empirical tests show that DAID is successful at this task when the solution to the problem requires deep chains of reasoning rather than single-step solutions.

DAID's success on deep structures and insignificance on shallow structures is not surprising, given the learning bias of DAID and standard backpropagation. Specifically, DAID is biased towards learning at the bottom of reasoning chains whereas backpropagation is, if anything, bias towards learning at the top of chains. In a shallow structure like that of the splice-junction domain, there is no difference in these biases. Hence, DAID has little effect. However, in deep structures, DAID differs considerably from backpropagation. It is this difference in learning bias that results in the gains in both generalization and speed that DAID provides in the promoter domain.

References

Fahlman, S. E. & Lebiere, C. 1989. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver, CO. Morgan Kaufmann.

Minsky, M. 1963. Steps towards artificial intelligence. In Feigenbaum, E. A. & Feldman, J., editors, *Computers and Thought*. McGraw-Hill, New York.

Noordewier, M. O.; Towell, G. G.; & Shavlik, J. W. 1991. Training knowledge-based neural networks to recognize genes in DNA sequences. In *Advances in Neural Information Processing Systems*, volume 3, Denver, CO. Morgan Kaufmann.

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

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning*, 1:81–106.

Rosenblatt, F. 1962. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, New York.

Rumelhart, D. E.; Hinton, G. E.; & Williams, R. J. 1986. Learning internal representations by error propagation. In Rumelhart, D. E. & McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the microstructure of cognition. Volume 1: Foundations*, pages 318–363. MIT Press, Cambridge, MA.

Towell, G. G.; Shavlik, J. W.; & Noordewier, M. O. 1990. Refinement of approximately correct domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866, Boston, MA.

Towell, G. G. 1991. *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction.* PhD thesis, University of Wisconsin, Madison, WI.

Weiss, S. M. & Kulikowski, C. A. 1990. *Computer Systems that Learn.* Morgan Kaufmann, San Mateo, CA.

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