Appears in the Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)

Heuristically Expanding Knowledge-Based Neural Networks*

David W. Opitz and Jude W. Shavlik
Computer Sciences Department
University of Wisconsin – Madison
Madison, WI 53706, U.S.A.
opitz@cs.wisc.edu

Abstract

Knowledge-based neural networks are networks whose topology is determined by mapping the dependencies of a domain-specific rulebase into a neural network. However, existing network training methods lack the ability to add new rules to the (reformulated) rulebases. Thus, on domain theories that are lacking rules, generalization is poor, and training can corrupt the original rules, even those that were initially correct. We present TopGen, an extension to the KBANN algorithm, that heuristically searches for possible expansions of a knowledge-based neural network, guided by the domain theory, the network, and the training data. It does this by dynamically adding hidden nodes to the neural representation of the domain theory, in a manner analogous to adding rules and conjuncts to the symbolic rulebase. Experiments indicate that our method is able to heuristically find effective places to add nodes to the knowledge-base network and verify that new nodes must be added in an intelligent manner.

1 Introduction

The task of *theory refinement* is to improve a set of approximately-correct domain-specific inference rules (called a *domain theory*) using new data [Ginsberg, 1990; Ourston and Mooney, 1990; Towell *et al.*, 1990]. The initial theory about a given task can involve such information as textbook knowledge or approximate rules of thumb obtained from an expert. A learning system should make repairs that minimize the changes to the initial domain theory, while making it consistent with the data. We present a connectionist approach to theory refinement, particularly focusing on the task of expanding *impoverished* domain theories.

Our system builds on the KBANN system [Towell, 1992]. KBANN translates the initial theory into a neural network, thereby determining the network's topology and initial weights. KBANN has been shown to be more effective at classifying previously-unseen examples than a wide variety of machine learning algorithms [Towell *et al.*, 1990; Towell, 1992]. A large part of the reason for KBANN's superiority over other symbolic systems has been attributed to both its underlying learning algorithm (i.e., backpropagation) and its effective use of domain-specific knowledge [Towell, 1992].

However, KBANN suffers from the fact that, since it does not alter the initial network's topology, it can only add and subtract antecedents of existing rules. Thus it is unable to add new symbolic rules to an impoverished rule set. Towell and Shavlik [1992] have shown that, while KBANN is reasonably insensitive to extra rules in a domain theory, its ability to generalize[1] degrades significantly as rules are removed from a domain theory. In addition, with sparse domain theories, KBANN needs to significantly alter the original rules in order to account for the training data. *Hence, our goal is to expand, during the training phase, knowledge-based neural networks – networks whose topology is determined as a result of the direct mapping of the dependencies of a domain theory – so that they are able to learn the training examples without needlessly corrupting their initial rules.*

The TopGen (Topology Generator) algorithm, the subject of this paper, heuristically searches through the space of possible expansions of a knowledge-based network, guided by the symbolic domain theory, the network, and the training data. It does this by adding hidden nodes to the neural representation of the domain theory. TopGen uses beam search, rather than a faster hill-climbing algorithm, because CPU cycles are becoming increasingly plentiful and cheap. It therefore seems wise to search more of the hypothesis space to find a good network topology. Finding such a topology allows better generalization, provides the network with the ability to learn without overly corrupting the initial set of rules, and increases the interpretability of the network so that efficient rules may be extracted. This paper presents evidence for these claims.

TopGen differs from other network-growing algorithms [Fahlman and Lebiere, 1989; Frean, 1990] in that it is designed for knowledge-based networks. TopGen uses a symbolic interpretation of the trained network to help

*This work was partially supported by DOE Grant DE-FG02-91ER61129, NSF Grant IRI-9002413, and ONR Grant N00014-90-J-1941.

[1] We use *generalization* to mean classification accuracy on examples not seen during training.

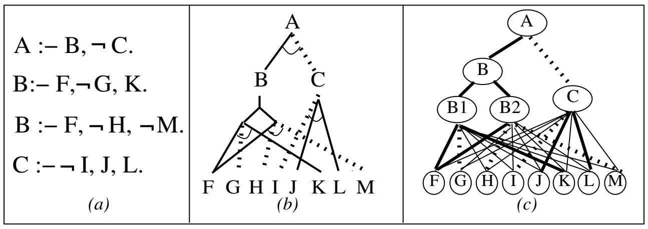

Figure 1: Translating rules into a neural network.

decide where the primary errors are in the network. Units are added in a matter analogous to adding rules and conjuncts to the symbolic rulebase. Adding hidden nodes in this fashion synergistically combines the strengths of refining the rules symbolically with the strengths of refining them with backpropagation.

The rest of the paper is organized as follows. In the next section, we give a brief overview of KBANN. We present the details of the TopGen algorithm in Section 3. This is followed by an example of how TopGen works. In Section 5 we present results from four real-world Human Genome domains and controlled studies on an artificial domain. This is followed by discussion of these results, as well as future and related work.

2 The KBANN Algorithm

KBANN is an approach for combining rule-based reasoning with neural networks. The relevant part of KBANN for this paper is the rules-to-network translation algorithm, which translates a set of propositional rules, representing what is initially known about the topic, into a neural network. This translation defines the network's topology and connection weights. Details of this translation process appear in [Towell *et al.*, 1990; Towell, 1992].

An example of this process is shown in Figure 1. Figure 1a shows a Prolog-like rule set that defines membership in category A. Figure 1b represents the hierarchical structure of these rules, with solid lines representing necessary dependencies and dotted lines representing prohibitory dependencies. Figure 1c represents the resulting network created from this translation. Units B1 and B2 in Figure 1c are introduced to handle the disjunction in the rule set. Thick lines in Figure 1c are heavily-weighted links corresponding to dependencies in the domain theory, while thin lines represent lightly-weighted links added to allow refinement of the knowledge base during backpropagation training. Biases are set so that nodes representing disjuncts have an output near 1 only when at least one of their heavily-weighted children is correct, while nodes representing conjuncts must have all of their heavily-weighted children active. Otherwise activations are near 0.

KBANN refines the network links with training examples. This training refines the antecedents of existing rules; however, KBANN does not have the capability of inducing new rules. For example, KBANN is unable to add a new rule for inferring B. *Being able to introduce such new rules is the focus of this paper.*

3 The TopGen Algorithm

TopGen heuristically searches through the space of possible ways of adding nodes to the network, trying to find the network that best refines the initial domain theory (as measured using "tuning sets"[2]). Briefly, TopGen looks for nodes in the network with high error rates, and then adds new nodes to these parts of the network.

Table 1 summarizes the beam-search-based TopGen algorithm. TopGen uses two tuning sets, one to evaluate the different network topologies, and one to help decide where new nodes should be added (we also use the latter tuning set to decide when to stop training individual networks). TopGen uses KBANN's rule-to-network translation algorithm to define an initial guess for the network's topology. TopGen trains this network using backpropagation [Rumelhart *et al.*, 1986] and places it on a search queue. In each cycle, TopGen takes the best network from the search queue (as measured by `tuning-set-2`), decides possible ways to add new nodes, trains these new networks, and places them on the search queue. This process repeats until reaching either (a) a `tuning-set-2` accuracy of 100% or (b) a previously-set time limit.

3.1 *Where* Nodes Are Added

TopGen must first find nodes in the network with high error rates. It does this by scoring each node using examples from `tuning-set-1`. By using examples from this tuning set, TopGen adds nodes on the basis of where the network fails to generalize, not where it fails to memorize the training set. TopGen makes the empirically-verified assumption that almost all of the nodes in a trained knowledge-based network are either fully active or inactive. By making this assumption, each non-input node in a TopGen-net can be treated as a step function (or a Boolean rule) so that errors have an all-or-nothing aspect to them. This concentrates topology refinement on misclassified examples, not on erroneous portions of *each* example. Towell [1992], as well as self-inspection of our networks, has shown this to be a valid assumption.

TopGen keeps two counters for each node, one for false negatives and one for false positives[3], defined with respect to each individual node's output. TopGen increments counters by recording how often changing the "Boolean" value of a node's output leads to a misclassified example being properly classified. That is, if a node is active for an erroneous example, and changing its output to be inactive results in correct classification, then TopGen increments the node's false-positives counter. TopGen increments a node's false-negatives counter in a similar fashion. By checking for single points of failure, TopGen looks for rules that are near misses. TopGen adds nodes where counter values are highest, while breaking ties by preferring nodes farthest from the output node.

We also tried other approaches for blaming nodes for error, but they did not work as well on our testbeds. One

[2] Data is first split into two disjoint sets: training and testing sets. The training set is further split into a set of training instances and two sets of tuning instances.

[3] An example is considered a false negative if it is incorrectly classified as a negative example, while a false positive is one incorrectly classified as a positive example.

Table 1: The TopGen Algorithm

TopGen:

GOAL: Search for the best network describing the domain theory and training examples.

1. Set aside a testing set. Break the remaining examples into a training set and two tuning sets (`tuning-set-1` and `tuning-set-2`).
2. Place the trained network, produced by KBANN, on the search queue.
3. Until *stopping criteria* met:
   (a) Remove the best network, according to `tuning-set-2`, from the search queue.
   (b) Use **ScoreEachNode** to determine the N best ways to expand the topology.
   (c) Create N new networks, train and put on the search queue.
   (d) Prune search queue to length M.
4. Output the best network seen so far according to `tuning-set-2`.

ScoreEachNode:

GOAL: Use the errors in `tuning-set-1` to suggest good ways to add new nodes.

1. Set each node's `correctable-false-negative` and `correctable-false-positive` counters to 0. Assume each node is a threshold unit.
2. For each misclassified example in `tuning-set-1`, consider each node and determine if modifying its output will correctly classify the example, incrementing the counters when appropriate.
3. Use the counters to order possible node corrections. High `correctable-false-negative` counts suggest adding a disjunct while high `correctable-false-positive` counts suggest adding a conjunct.

such method is to propagate errors back by starting at the final conclusion and recursively considering a rule's antecedent to be incorrect if both its consequent is incorrect and the antecedent does not match its "target." We consider an antecedent to have the same target as its consequent, unless negated. While this method works for symbolic rules, TopGen suffers under this method because its antecedents are weighted. Because of this, we also tried using the backpropagated error to blame nodes; however, backpropagation error can become diffused when networks have many layers, such as the ones often created by TopGen. Since these methods are just heuristics to help guide the search of where to add new nodes, TopGen is able to backtrack from "bad" choices.

3.2 *How* Nodes Are Added

Once we know where to add new nodes, we need to know *how* to add these nodes. TopGen makes the assumption that when training one of its networks, the meaning of a node does not shift significantly. Making this assumption allows us to alter the network in a fashion similar to

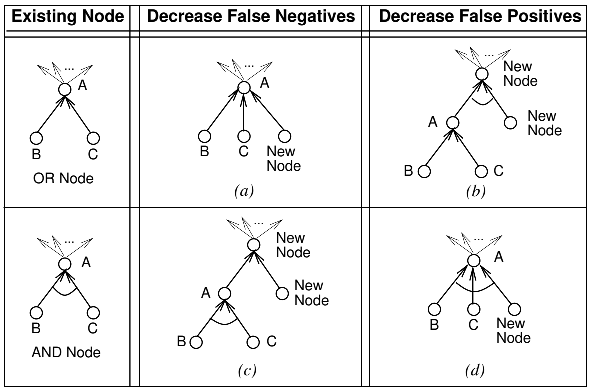

Figure 2: Possible ways TopGen adds new nodes to a knowledge-based neural network. Arcs indicate AND nodes.

refining symbolic rules. Towell [1992] showed that making a similar assumption about KBANN-nets was valid.

Figure 2 shows the possible ways TopGen adds nodes to a TopGen-net. In a symbolic rulebase that uses negation-by-failure, we can decrease false negatives by either dropping antecedents from existing rules or adding new rules to the rulebase. Since KBANN is effective at removing antecedents from existing rules, TopGen adds nodes, intended to decrease false negatives, in a fashion that is analogous to adding a new rule to the rulebase. If the existing node is an OR node, TopGen adds a new node, fully-connected to the inputs, as its child (see Figure 2a). If the existing node is an AND node, TopGen creates a new OR node that is the parent of the original AND node and another new node that TopGen fully-connects to the inputs (see Figure 2c); TopGen moves the outgoing links of the original node (A in Figure 2c) to become the outgoing links of the new OR nodes.

To decrease false positives in a symbolic rulebase, we can either add antecedents to existing rules or remove rules from the rulebase. While KBANN can effectively remove rules [Towell, 1992], it is less effective at adding antecedents to rules and is unable to invent (constructively induce) new terms as antecedents. Figures 2b,d show the ways (analogous to Figures 2a,c explained above) of adding constructively-induced antecedents. By allowing these additions, TopGen is able to add rules whose consequents were previously undefined to the rulebase.

TopGen handles nodes that are neither AND nor OR nodes by deciding if such a node is closer to an AND node or an OR node (by looking at the node's bias and incoming weights). TopGen classifies previously-added nodes in such a manner, when deciding how to add more nodes to them at a later time.

3.3 Additional Algorithmic Details

After new nodes are added, TopGen must train the network. While we want the new weights to account for most of the error, we also want the old weights to change if necessary. That is, we want the older weights to retain what they have previously learned, while at the same time move in accordance with the change in error caused by adding the new node. In order to address this issue, TopGen multiplies the learning rates of existing weights

1–1	1–2	1–3	1–4	1–5
2–1	2–2	2–3	2–4	2–5
3–1	3–2	EMPTY 3–3	3–4	3–5
4–1	4–2	4–3	4–4	4–5

Figure 3: Portion of the chess board considered by domain.

by a constant amount ($\leq$1) every time new nodes are added, producing an exponential decay of learning rates.

To help address the trade-off between changing the domain theory and disregarding the misclassified training examples as noise, TopGen uses a variant of weight decay [Hinton, 1986]. Weights that are part of the original domain theory decay toward their initial value, while other weights decay toward zero. Thus, we add to the usual cost function, a term that measures the distance of each weight from its initial value:

$$Cost = \sum_{k \in T} (target_k - output_k)^2 + \lambda \sum_{i \in W} \frac{(\omega_i - \omega_{init_i})^2}{1 + (\omega_i - \omega_{init_i})^2}$$

The first term sums over all training examples T, while the second term sums over all weights W. The tradeoff between performance and distance from initial values is weighted by λ.

4 Example of TopGen

Assume that Figure 1a's domain theory is missing the following rule from the target concept:

$$B \text{ :- } \neg F,\ G,\ H.$$

Although we trained the KBANN-net shown in Figure 1c with all possible examples, it was unable to improve. TopGen begins with this KBANN-net, then proceeds by using misclassified examples from `tuning-set-1` to find potentially useful places to add nodes. KBANN misclassifies the following positive example of category A.

$$\neg F \wedge G \wedge H \wedge \neg I \wedge \neg J \wedge \neg K \wedge L \wedge M$$

While node C (from Figure 1c) is correctly false in this example, node B is incorrectly false. B is false since both $B1$ and $B2$ are false. If B had been true, the networks would have correctly classified this example (since C is correct), so TopGen increments the `correctable-false-negative` counter for B. TopGen also increments the counters of $B1$, $B2$, and A, using similar arguments.

Since nodes A, B, $B1$, and $B2$ will all have high `correctable-false-negative` counts, TopGen considers adding OR nodes to nodes A, $B1$, and $B2$, as done in Figure 2c, and also considers adding another disjunct to node B, analogous to Figure 2a. Any of these additions allows the network to learn the target concept.

5 Experimental Results

We tested TopGen on five domains: an artificial chess-related domain, and four real-world Human Genome problems. While real-world domains are clearly useful in exploring the utility of an algorithm, they are difficult to use in closely-controlled studies that examine different aspects of an algorithm. An artificial domain allows us to determine the relationship between the theory provided to the learning system and the correct domain theory.

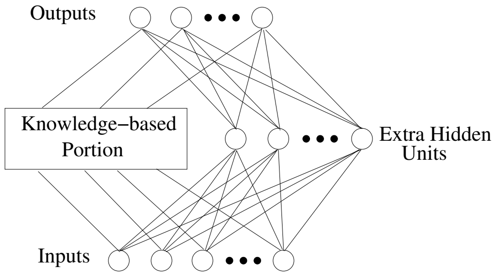

Figure 4: Topology of networks used by Strawman.

5.1 A Chess-Related Domain

The first domain, derived from the game of chess, defines board configurations where moving a king one space forward is legal. Figure 3 shows the subset of the chess board this domain considers. We want to move the king from position 4-3 to position 3-3. Possible pieces include a queen, a rook, a bishop, and a knight for both sides.

To help test the efficiency of TopGen's multi-level approach of adding hidden nodes, we compare its performance with a simple approach (referred to as *Strawman* hereafter) that adds one layer of fully-connected hidden nodes "off to the side" of the KBANN-net. Figure 4 shows the topology of such a network. The topology of the original KBANN-net remains intact, while we add extra hidden nodes in a fully-connected fashion between the inputs nodes and the output nodes. If a domain theory is impoverished, it is reasonable to think that simply adding nodes in this fashion would increase performance. Strawman trains 21 different networks (using weight decay), ranging from 0 to 20 extra hidden nodes and, like TopGen, uses a tuning set to choose the best network. (In the experiments presented, TopGen never tested networks with more than 20 new nodes.)

Our initial experiment addresses the generalization ability of TopGen when rules are deleted from a correct domain theory. Figure 5 shows the test set error when we randomly delete rules from the chess domain theory. The results are averages of five runs of five-fold cross-validation. The top horizontal line results from a fully-connected, single-layer feed-forward neural network. For each fold, we trained various networks containing up to 100 hidden nodes and used a tuning set to choose the best network. The next curve down, the top diagonal curve, is the test set error of the initial, corrupted domain theory given to Strawman, KBANN, and TopGen. The next two curves, produced by KBANN and Strawman, cut the test set error of the initial domain theory almost in half.[4] Strawman produced almost no improvement over KBANN. Finally, TopGen, the bottom curve, had a significant increase in accuracy, having an

[4] Running KBANN without weight decay produced almost no improvement over the initial domain theory.

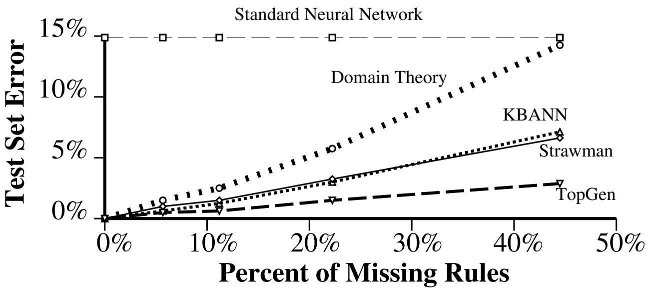

Figure 5: Test set error on the chess problem.

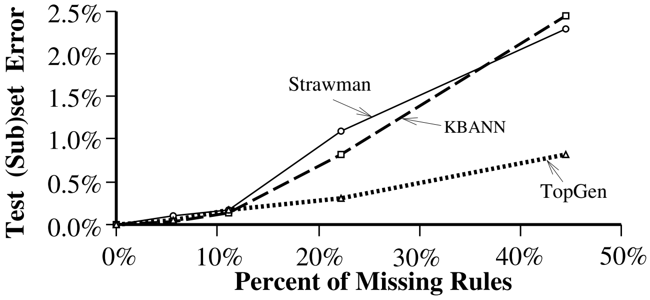

Figure 6: Error on the subset of the test set that the initial domain theory correctly classifies.

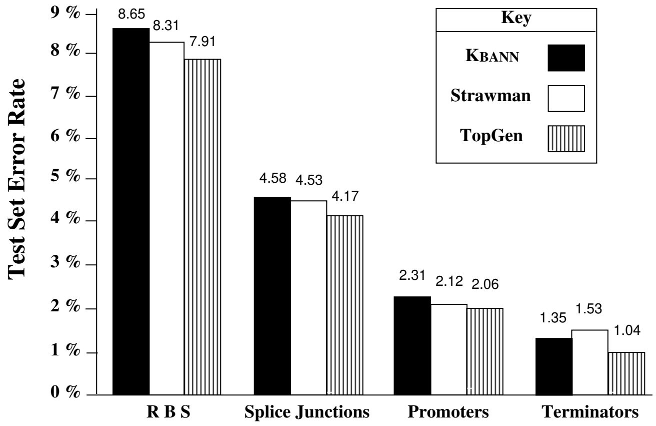

Figure 7: Error rates on four Human Genome problems.

Table 2: Total number of nodes added (on average).

Domain	TopGen	Strawman
RBS	8.2	8.0
Splice Junction	4.0	5.2
Promoters	4.4	5.0
Terminators	9.4	1.2

error rate of about half that of either KBANN or Strawman. As a point of comparison, when 45% of the rules were deleted, TopGen added 10.9 nodes, while Strawman added 5.2 nodes on average, and two-sample one-tailed t-tests indicate that TopGen differs from both KBANN and Strawman at the 99.5% confidence level.

As stated earlier, it is important to correctly classify the examples while deviating from the initial domain theory as little as possible. Since this prior knowledge may be in a different form than what the learning algorithm uses, we are concerned with semantic distance, rather than syntactic distance, when measuring this deviation. We can estimate semantic distance by testing TopGen using only those examples in the test set that the original domain theory classifies correctly. Error on these examples indicates how much the learning algorithm has corrupted *correct* portions of the domain theory.

Figure 6 shows accuracy on the portion of the test set where the original domain theory is correct. When the initial domain theory has few missing rules (less than 20%), neither KBANN nor Strawman overly corrupt this domain theory in order to compensate for the missing rules. However, as more rules are deleted, both KBANN and Strawman corrupt their domain theory more than TopGen. In fact, when 45% of the rules are missing, TopGen has less than half the error rate on originally-correct examples as both KBANN and Strawman.

5.2 Four Human Genome Domains

We also ran TopGen on four problems from the Human Genome Project. Each of these problems aid in locating genes in DNA sequences. The first domain, *promoter recognition*, contains 234 positive examples, 4,921 negative examples, and 17 rules. (Note that this data set and domain theory are a larger version of the one that appears in Towell [1992] and Towell *et al.* [1990]). The second domain, *splice-junction determination*, contains 3,190 examples distributed among three classes, and 23 rules. The third domain, *transcription termination sites*, contains 142 positive examples, 5,178 negative examples, and 60 rules. Finally, the last domain, *ribosome binding sites*, contains 366 positive examples, 1,511 negative examples, and 17 rules. See Craven and Shavlik [1993] for a detailed description of these tasks. (We would like to thank Michiel Noordewier for creating these domains.)

Our experiment addresses the test set accuracy of TopGen on these domains. The results in Figure 7 show that TopGen generalizes better than does either KBANN or Strawman. These results are averages of five runs of five-fold cross-validation. Two-sample one-tailed t-tests indicate TopGen differs from both KBANN and Strawman at the 97.5% confidence level on all four domains, except with Strawman on the promoter domain. Table 2 shows that TopGen and Strawman added about the same number of nodes on all domains, except the terminator data set. On this data set, adding nodes off to the side of the KBANN-net, in the style of Strawman, usually decreases accuracy. Therefore, when Strawman picked a network other than the KBANN network, its generalization usually decreased. Even with Strawman's difficulty on this domain, TopGen was still able to effectively add nodes to increase performance.

6 Discussion and Future Work

Towell [1992] has shown that KBANN is superior to a wide variety of machine learning algorithms on the promoters and splice-junctions domains, including purely symbolic approaches to theory refinement. Yet, even though an expert (M. Noordewier) believed all four Human Genome domain theories are large enough for

KBANN to adequately learn the concepts, TopGen was able to effectively add new nodes to the corresponding network. The effectiveness of adding nodes in a manner similar to reducing error in a symbolic rulebase is verified with comparisons to a naive approach to adding nodes. If a KBANN-net, resulting from an impoverished domain theory, suffered only in terms of capacity, then adding nodes between the input and output nodes would have been just as effective as TopGen's approach to adding nodes. This difference is particularly pronounced on the terminator data set. TopGen has a longer run-time than KBANN; however, we believe this is a wise investment, since computers are becoming faster. We hope to obtain even better results as we increase the time limit.

Future work includes using a rule-extraction algorithm to measure the interpretability of a refined TopGen-net. We hypothesize that TopGen builds networks that are more interpretable than naive approaches of adding nodes, such as the approach taken by Strawman. Trained KBANN networks are interpretable because (a) the meaning of its nodes does not significantly shift during training and (b) almost all the nodes are either fully active or inactive [Towell, 1992]. TopGen adds nodes in a fashion that does not violate these two assumptions.

Other future work includes testing new ways of adding nodes. Nodes are currently added so that they are fully connected to all input nodes. Other possible approaches include: adding them to only a portion of the inputs, adding them to nodes having high correlations with the error, or adding them to the next "layer" of nodes.

7 Related Work

We described TopGen's relationship to both network-growing algorithms and the KBANN system earlier in this paper. Another extension to KBANN is the DAID algorithm [Towell and Shavlik, 1992]. DAID helps train a KBANN-net by trying to locate low-level *links* with errors, while TopGen expands a KBANN-net by searching for *nodes* with errors. Propositional theory-refinement systems, such as EITHER [Ourston and Mooney, 1990] and RTLS [Ginsberg, 1990], are also related to TopGen. These systems differ from TopGen, in that their approaches are purely symbolic. Even though TopGen adds nodes in a manner analogous to how a symbolic system adds antecedents and rules, its underlying learning algorithm is "connectionist." EITHER, for example, uses ID3 for its induction component.

8 Conclusion

Although KBANN has previously been shown to be an effective theory-refinement algorithm, it suffers because it is unable to add nodes during training. When domain theories are sparse, KBANN's generalization degrades significantly and it must overly corrupt the original rules to account for the training examples. Our algorithm, TopGen, heuristically searches through the space of possible expansions of the original network, guided by the symbolic domain theory, the network, and the training data. It does this by adding hidden nodes to the neural representation of the domain theory, in a manner analogous to adding rules and conjuncts to a symbolic rulebase.

Experiments indicate that our method is able to heuristically find effective places to add nodes to the knowledge bases of four real-world problems and an artificial chess domain. Experiments also verified that nodes must be added in an intelligent manner. Our algorithm showed statistically-significant improvements over KBANN in all five domains, and over a strawman approach in four domains. Hence, our new algorithm is successful in overcoming KBANN's limitation of not being able to dynamically add new nodes. In doing so, our system promises to increase KBANN's ability to generalize and learn a concept without needlessly corrupting the initial rules, while at the same time, increasing the comprehensibility of rules extracted from a trained network. Thus, our system further increases the applicability of neural learning to problems having a substantial body of preexisting knowledge.

References

[Craven and Shavlik, 1993] M. Craven and J. Shavlik. Machine learning approaches to gene recognition. Machine Learning Research Group Working Paper 93-1, Univ. of Wisconsin – Madison, 1993.

[Fahlman and Lebiere, 1989] S. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Adv. in Neural Info. Processing Systems 2*, pages 524–532, Denver, CO, 1989. Morgan Kaufmann.

[Frean, 1990] M. Frean. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209, 1990.

[Ginsberg, 1990] A. Ginsberg. Theory reduction, theory revision, and retranslation. *Proc. of the 8th Nat. Conf. on Artificial Intelligence*, pages 777–782, 1990.

[Hinton, 1986] G. Hinton. Learning distributed representations of concepts. In *Proc. of the 8th Annual Conf. of the Cognitive Science Soc.*, pages 1–12, 1986.

[Ourston and Mooney, 1990] D. Ourston and R. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *Proc. of the 8th Nat. Conf. on Artificial Intelligence*, pages 815–820, 1990.

[Rumelhart *et al.*, 1986] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing, Volume 1*. MIT Press, Cambridge, MA, 1986.

[Towell and Shavlik, 1992] G. Towell and J. Shavlik. Using symbolic learning to improve knowledge-based neural networks. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence*, pages 177–182, 1992.

[Towell *et al.*, 1990] G. Towell, J. Shavlik, and M. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proc. of the 8th Nat. Conf. on Artificial Intelligence*, pages 861–866, Boston, MA, 1990.

[Towell, 1992] G. Towell. *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction.* PhD thesis, Univ. of Wisconsin, Madison, WI, 1992.