Appears in *Computational Learning Theory and Natural Learning Systems*, Vol. III,
T. Petsche, S. Hanson and J. Shavlik, editors, (pp. 3-19), MIT Press, 1995.

Using Heuristic Search to Expand Knowledge-Based Neural Networks

David W. Opitz and **Jude W. Shavlik**

opitz@cs.wisc.edu
(608) 262-6613

1210 W. Dayton St.
Computer Sciences Department
University of Wisconsin – Madison
Madison, WI 53706

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. TopGen 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 rule base. Experiments indicate that our method is able to heuristically find effective places to add nodes to the knowledge bases of four real-world problems, as well as an artificial chess domain. The experiments also verify that new nodes must be added in an intelligent manner. Our algorithm showed statistically-significant improvements over Kbann in all five domains.

1 Introduction

The task of *theory refinement* is to improve an imperfect "domain theory" using new data (Ginsberg, 1990; Ourston & Mooney, 1990; Towell et al., 1990). The initial theory can involve such information as textbook knowledge or rules of thumb obtained from an expert or learning algorithm. 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.

Kbann (Towell, 1992) is a connectionist theory-refinement system that translates a set of approximately-correct, domain-specific inference rules (called a *domain theory*) into a neural network, thereby determining the network's topology. It then applies backpropagation to refine these reformulated rules. Kbann has been shown to generalize to previously-unseen examples better than many other inductive learning algorithms (Towell et al., 1990; Towell, 1992; Towell & Shavlik, in press). Kbann's superiority over other symbolic systems has been mostly attributed to both its underlying learning algorithm (i.e., backpropagation) and its effective use of domain-specific knowledge (Towell, 1992; Towell & Shavlik, in press).

However, Kbann suffers from the fact that, since it does not alter the initial network's topology, it essentially only adds and subtracts antecedents of existing rules. Thus it is unable to add new symbolic rules to an impoverished rule set. Towell (1992) has shown that, while Kbann is reasonably insensitive to extra rules in a domain theory, its ability to generalize (i.e., correctly classify examples not seen during training) 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. While it is clearly important to classify the examples as accurately as possible, changes to the initial domain theory should be kept to a minimum because the domain theory presumably contains useful information, even if it is not completely correct. Also, large changes to the domain theory greatly complicated rule extraction following training (Towell & Shavlik, 1993). *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 heuristic 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 & Lebiere, 1989; Frean, 1990; Mezard & Nadal, 1989) in that it is designed for knowledge-based networks. TopGen uses a symbolic interpretation of the trained network to help locate the primary errors in the

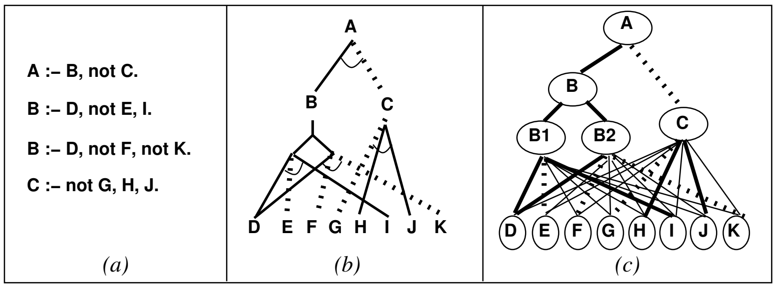

Figure 1: Translation of a knowledge base into a neural network.

network. Units are added in a matter analogous to adding rules and conjuncts to the symbolic rule base. 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 of this paper, 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. We present our conclusions in the last section.

2 The KBANN Algorithm

KBANN translates a set of propositional rules, representing what is initially known about a domain, into a neural network. This translation algorithm defines the topology and the initial connection weights of the network it creates.

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. KBANN creates nodes $B1$ and $B2$ in Figure 1c to handle the two rules deriving B in the rule set. The thin lines in Figure 1c are low-weighted links that KBANN added to allow refinement of these rules during backpropagation training. Biases are set so that nodes representing disjuncts have an output near 1 only when at least one of their high-weighted antecedents is correct (i.e., near 1 for positive links and near 0 for negative links), while nodes representing conjuncts must have all of their high-weighted antecedents correct. Otherwise activations are near 0. Refer to (Towell et al., 1990; Towell, 1992; Towell & Shavlik, in press) for more details.

KBANN refines the network links using training examples. This training alters 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 "validation sets"). 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 validation sets, one to evaluate the different network topologies, and one to help decide where new nodes should be added (we also use the second validation 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. This network is trained using backpropagation (Rumelhart et al., 1986) and is placed on an OPEN list. In each cycle, TopGen takes the best network from the OPEN list (as measured by `validation-set-2`), decides possible ways to add new nodes, trains these new networks, and places them on the OPEN list. This process is repeated until reaching either (a) a `validation-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 (which corresponds to a rule in the symbolic domain theory) using examples from `validation-set-1`. By using examples from this validation 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 unit in a TopGen network can be treated as a step function (or a Boolean rule) so that errors have an all-or-nothing aspect, thus concentrating 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, defined with respect to each individual node's output, not the final output. 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. 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 for the example, then the node's false-positives counter is incremented. 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 such method is to propagate errors back by starting at the final conclusion and recursively considering an antecedent of a rule to be incorrect if both its consequent is incorrect and the antecedent does not match its "target." We approximate targets by starting with the output node, and recursively considering a node to have the

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 validation sets (`validation-set-1` and `validation-set-2`).
2. Train, using backpropagation, the initial network produced by KBANN's rules-to-network translation and put on *OPEN* list.
3. Until *stopping criterion* reached:
   (a) Remove best network, according to `validation-set-2`, from *OPEN* list.
   (b) Use **ScoreEachNode** to determine *N* best places to expand topology.
   (c) Create *N* new networks, train and put on *OPEN* list.
   (d) Prune *OPEN* list to length *M*.
4. Output the best network seen so far according to `validation-set-2`.

ScoreEachNode:

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

1. Score each node in the network as follows:
   (a) Set each node's `correctable-false-negative` and `correctable-false-positive` counters to 0. Assume each node is a threshold unit.
   (b) For each misclassified example in `validation-set-1`, cycle through each node and determine if modifying the output of that node will correctly classify the example, incrementing the counters when appropriate.
2. 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.

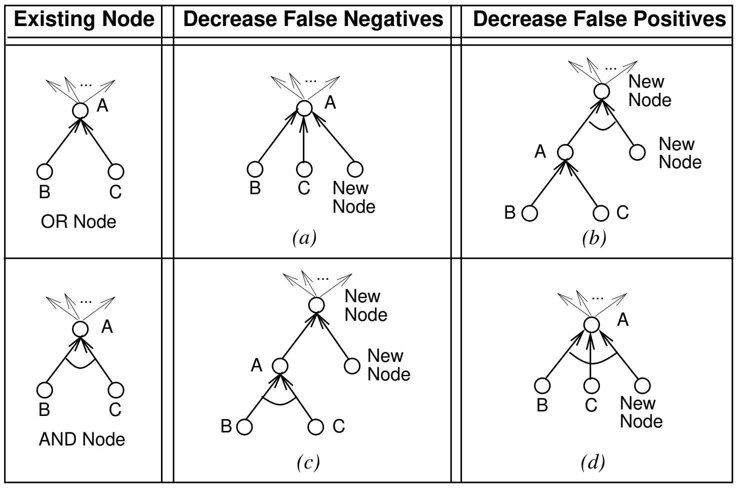

Figure 2: Possible ways to add new nodes to a knowledge-based neural network. Arcs indicate AND nodes.

same target as its parent, if the weight connecting them is positive, or the opposite target, if this weight is negative. While this method works for symbolic rules, TopGen suffers under this method because its antecedents are weighted. Antecedents with small-weighted links are counted as much as antecedents with large-weighted links. Because of this, we also tried using the backpropagated error to blame nodes, however backpropagated error becomes too diffuse in networks having many layers, such as the ones often created by TopGen. It is important to note that these methods are just heuristics to help guide the search of where to add new nodes, thus TopGen is able to backtrack if a "bad" choice is made.

3.2 *How* Nodes Are Added

Once we estimate where we should 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 refining symbolic rules. Towell (Towell & Shavlik, 1993; Towell, 1992) showed that making a similar assumption about KBANN networks was valid.

Figure 2 shows the possible ways TopGen adds nodes to a TopGen network. In a symbolic rule base 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 for decreasing 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 as its child (see Figure 2a), and fully-connects this new node to the input nodes. 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 (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.

In a symbolic rule base, we can decrease false positives by either adding antecedents to existing rules or removing rules from the rule base. 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 by a constant amount ($\leq$1) every time new nodes are added, producing an exponential decay of learning rates.

We also do not want to change the domain theory unless there is considerable evidence that it is incorrect. That is, there is a trade-off between changing the domain theory and disregarding the misclassified training examples as noise. To help address this, 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.

Our weight decay term, then, decays weights as a function of their distance from their initial value and is a slight variant of the term proposed by Rumelhart in 1987 (Weigand et al., 1990). The idea of our weight decay is to add, to the usual cost function, a term that measures the distance of each weight from its initial value:

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

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

4 Example of TopGen

Assume that Figure 1a's domain theory should have also included the following rule:

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

Although we trained the KBANN network shown in Figure 1c with all possible examples, it was unable to learn the correct concept.

TopGen begins by training the network in Figure 1c, obtaining no improvement to the original rule base. It then proceeds by taking misclassified examples from `validation-set-1` to find places where adding nodes could be beneficial. The following example of category A is incorrectly classified by the domain theory:

$$notF \wedge G \wedge H \wedge notI \wedge notJ \wedge notK \wedge L \wedge M$$

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, this example would have been correctly classified (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.

Nodes A, B, $B1$, and $B2$ will all have high `correctable-false-negative` counters after all the examples are processed. Given these high counts, TopGen considers adding OR nodes to nodes A, $B1$, and $B2$, as done in Figure 2c, and also considers adding another disjunct, analogous to Figure 2a, to node B. Any one of these for corrections allows the network to learn the target concept. Since TopGen breaks ties by preferring nodes farthest from the output node, it prefers $B1$ or $B2$.

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. Knowing this relationship allows us to better understand the effectiveness of the learning algorithm.

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 (i.e., the king would not be in check). Figure 3 shows the subset of the chess board considered by this domain. The king wants to move from position 4-3 to position 3-3. Possible pieces include a queen, a rook, a bishop, and a knight for both sides.

In order to investigate how well TopGen completes impoverished domain theories, we ran experiments where we perturbed the correct domain theory in various ways, and gave these incorrect domain theories to TopGen. One could perturb a domain theory by either: (a) adding an antecedent to a rule, (b) deleting an antecedent from a rule, (c) adding a rule, or (d) deleting a rule. Since Towell (1992) previously showed that KBANN effectively corrects perturbations (a-c), we only ran experiments where rules are deleted from the correct domain theory (i.e., the domain theory given to the learning algorithms is impoverished).

To help test the efficiency of TopGen's approach of choosing where to add hidden nodes, we compare its performance with a simple approach (referred to as *Strawman* from here after) that adds one layer of fully-connected hidden nodes "off to the side" of the KBANN network.

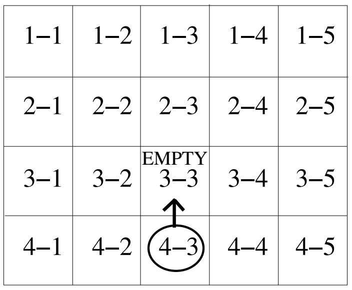

Figure 3: Portion of the chess board covered by the domain theory.

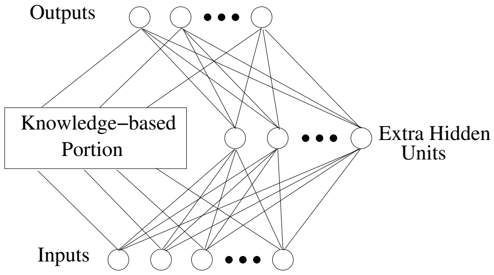

Figure 4: Topology of networks used by Strawman.

Figure 4 shows the topology of such a network. The topology of the original KBANN network 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 validation set to choose the best network. (In the experiments presented in this paper, 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 validation set to choose the best network. The line is horizontal because the neural network does not use any of the domain theories. 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; Strawman produced almost no improvement over KBANN. Finally, TopGen, the bottom curve, had a significant increase in accuracy, having an 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 on the average, while the average of the best Strawman networks added 5.2 nodes. One-tailed, two-sample t-tests indicate that the difference between TopGen and KBANN (t=31.71, d.f.=8)

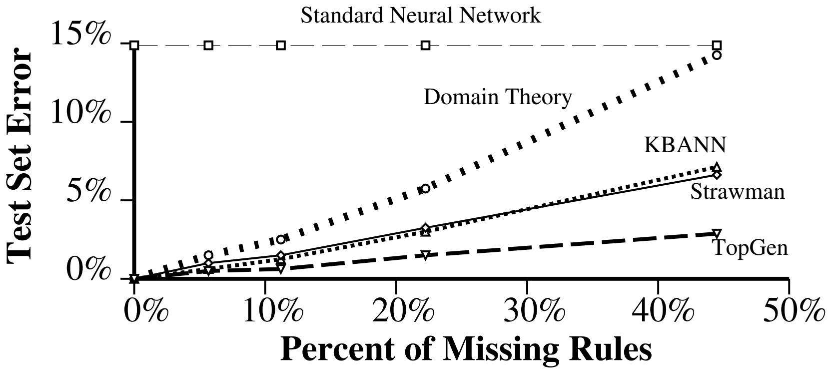

Figure 5: Test set error on the chess problem.

and the difference between TopGen and Strawman (t=27.41, d.f.=8) are significant at the 99.5% confidence level (when 45% of the rules were deleted).

As stated earlier, it is important to correctly classify the examples while deviating from the initial domain theory as little as possible. Because the domain theory may have been inductively generated from past experiences, we are concerned with semantic distance, rather than syntactic distance, when deciding how far a learning algorithm has deviated from the initial domain theory. Also, syntactic distance is difficult to measure, especially if the learning algorithm generates rules in a different form than the initial domain theory. However, we can estimate semantic distance by 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 15%), neither TopGen, KBANN, nor Strawman overly corrupt this domain theory in order to compensate for these missing rules. However, as more rules are deleted, both KBANN and Strawman corrupt their domain theory more than TopGen does. For example, 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 thank Michiel Noordewier for creating these domains.)

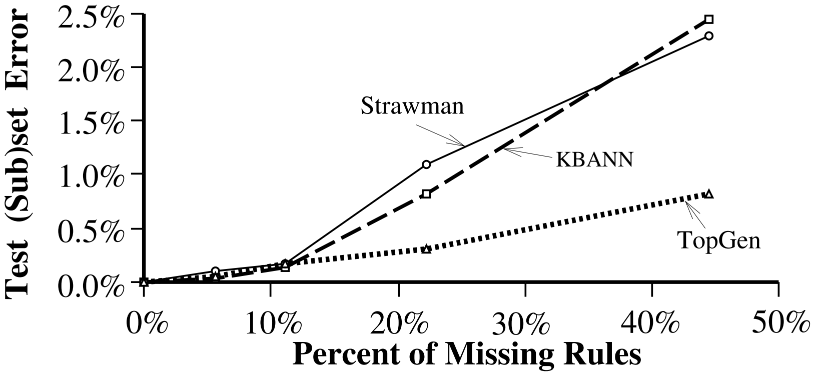

Figure 6: Error on the subset of the test set where the initial domain theory is correct.

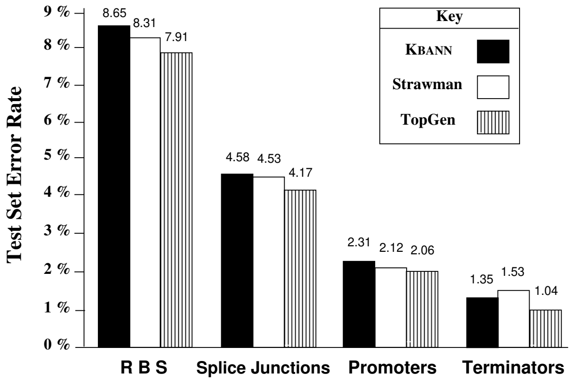

Figure 7: Error rates on four Human Genome problems.

Our experiment addresses the test-set accuracy of TopGen on these domains. The results in Figure 7 show that TopGen generalizes better than does both KBANN and 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 network, 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 generalizes better than many machine learning algorithms on the promoters and splice-junctions domains, including purely symbolic approaches

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

to theory refinement. Yet, even though a domain expert (M. Noordewier) believed the four Human Genome domain theories were large enough for KBANN to adequately learn the concepts, TopGen is 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 rule base, is verified with comparisons to a naive approach to adding nodes. If a KBANN network, 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. The difference between TopGen and this naive approach 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 computer cycles are becoming cheaper. A future plan is to implement a parallel version of TopGen. In doing so, we hope to increase the number of networks considered, from 30 networks (the current number for results presented in this paper) to several hundred networks, and in the process, obtain even better results.

Future work includes using a rule-extraction algorithm (Towell & Shavlik, 1993; Sestito & Dillon, 1990; Fu, 1991) to measure the interpretability of a refined TopGen network. 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 & Shavlik, 1993). Not only does TopGen add nodes in a symbolic fashion, it adds them in a fashion that does not violate these two assumptions.

Other future work includes extensively testing other approaches for localizing error. Even though this is only a heuristic to help guide the search, a good heuristic will allow more efficient search of the hypothesis space. Methods of using the back-propagated error as well as symbolic techniques for determining error have been tested, but did not improve performance, for reasons explained earlier. A future research direction includes trying variants of these techniques.

A final research direction includes testing new ways of adding nodes to the network. Nodes are currently added so that they are fully connected to all the input nodes. Other possible approaches include: adding them to only a portion of the inputs, adding them to nodes that have a high correlation with the error, or adding them to the next "layer" of nodes.

7 Related Work

The most obvious related work is the KBANN system (Towell, 1992), described in detail earlier in this paper. The DAID algorithm (Towell & Shavlik, 1992), an extension to KBANN, uses the domain knowledge to help train the KBANN network. Because KBANN is more effective at dropping antecedents than adding them, DAID tries to find potentially-useful inputs features not mentioned in the domain theory. DAID backs-up errors to the lowest level of the domain theory, computes correlations with the features, and increases the weight value of potentially useful features. In summary, DAID, tries to locate low-level *links* with errors, while TopGen searches for *nodes* with errors.

Additional related work includes theory-refinement systems. Systems such as EITHER (Ourston & Mooney, 1990) and RTLS (Ginsberg, 1990) are propositional in nature. 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. While Towell (1992) showed that KBANN was superior to EITHER on a promoter problem, TopGen outperformed KBANN. Systems such as FOCL (Pazzani et al., 1991) and FORTE (Richards & Mooney, 1991) revise first-order theories. One drawback to these types of systems is that, due to their computational demands, the problems currently used by these systems are quite simple. Another drawback is that many such systems are unable to create new predicates. This is because current predicate-invention methods are computationally expensive and often require an oracle.

A final area related to TopGen is network growing algorithms (Fahlman & Lebiere, 1989; Frean, 1990; Mezard & Nadal, 1989). The most obvious difference between TopGen and these algorithms is that TopGen uses domain knowledge and symbolic rule-refinement techniques to help determine the network's topology. A second difference is that these other algorithms restructure their network based solely on training set error. Also, TopGen uses beam search, rather than hill climbing when determining where to add nodes.

8 Conclusion

Although KBANN has previously been shown to be an effective theory-refinement algorithm, it suffers because it is unable to add new nodes (rules) during training. KBANN suffers when domain theories are sparse because (a) generalization degrades significantly and (b) the original rules are significantly altered in order to account for the training data. 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 the symbolic rule base.

Experiments indicate that our method is able to heuristically find effective places to add nodes to the knowledge bases of four real-world problems, as well as an artificial chess domain. 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.

9 Acknowledgement

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

References

Fahlman, S. E. & Lebiere, C. (1989). The cascade-correlation learning architecture. In Touretzky, D., editor, *Advances in Neural Information Processing Systems (volume 2)*, (pp. 524–532), San Mateo, CA. Morgan Kaufmann.

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

Fu, L. M. (1991). Rule learning by searching on adapted nets. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, (pp. 590–595), Anaheim, CA.

Ginsberg, A. (1990). Theory reduction, theory revision, and retranslation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, (pp. 777–782), Boston, MA.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, (pp. 1–12), Amherst, MA.

Mezard, M. & Nadal, J.-P. (1989). Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A*, 22:2191–2204.

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*, (pp. 815–820), Boston, MA.

Pazzani, M. J., Brunk, C. A., & Silverstein, B. (1991). A knowledge-intensive approach to relational concept learning. In *Proceedings of the Eighth International Machine Learning Workshop*, (pp. 432–436), Evanston, IL.

Richards, B. L. & Mooney, R. J. (1991). First-order theory revision. In *Proceedings of the Eighth International Machine Learning Workshop*, (pp. 447–451), Evanston, IL.

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*. MIT Press, Cambridge, MA.

Sestito, S. & Dillon, T. (1990). Using multi-layered neural networks for learning symbolic knowledge. In *Proceedings of the 1990 Australian Artificial Intelligence Conference*, Perth, Australia.

Towell, G. & Shavlik, J. (1992). Using symbolic learning to improve knowledge-based neural networks. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, (pp. 177–182), San Jose, CA.

Towell, G. & Shavlik, J. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101.

Towell, G. & Shavlik, J. (in press). Knowledge-based artificial neural networks. *Artificial Intelligence*.

Towell, G. G. (1992). *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction*. PhD thesis, University of Wisconsin, Madison, WI.

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*, (pp. 861–866), Boston, MA.

Weigand, A. S., Rumelhart, D. E., & Huberman, B. A. (1990). Generalization by weight-elimination with application to forecasting. In Lippmann, R., Moody, J., & Touretzky, D., editors, *Advances in Neural Information Processing Systems (volume 3)*, (pp. 875–882), San Mateo, CA. Morgan Kaufmann.