COMBINING EXPLANATION-BASED AND NEURAL LEARNING: AN ALGORITHM AND EMPIRICAL RESULTS

by

Jude W. Shavlik

Geoffrey G. Towell

Computer Sciences Technical Report #859

June 1989

Combining Explanation-Based and Neural Learning: An Algorithm and Empirical Results

Jude W. Shavlik
Geoffrey G. Towell

Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706
shavlik@cs.wisc.edu
(608) 262-7784

Keywords: explanation-based learning, neural networks, connectionism, symbolic systems, imperfect domain theories, neural network topologies, hybrid machine learning systems

Abstract

Machine learning is an area where both symbolic and neural approaches have been heavily investigated. However, there has been little research into the synergies achievable by combining these two learning paradigms. A hybrid approach that combines the symbolically-oriented explanation-based learning paradigm with the neural back-propagation algorithm is described. Most realistic problems can never be formalized exactly. However, there is much to be gained by utilizing the capability to reason nearly correctly. In the presented EBL-ANN algorithm, a "roughly-correct" explanatory capability leads to the acquisition of a classification rule that is almost correct. The rule is mapped into a neural network, where subsequent refinement improves it. This approach overcomes problems that arise when using imperfect domain theories to build explanations and addresses the problem of choosing a good initial neural network configuration. Empirical results show that the hybrid system more accurately learns concepts than an explanation-based system by itself and that the hybrid also learns them much faster than a neural learning system by itself.

Combining Explanation-Based and Neural Learning: An Algorithm and Empirical Results

1. INTRODUCTION

Explanation-based learning (EBL) and artificial neural networks (ANNs) are two actively pursued approaches to machine learning. These two styles approach the learning problem from very different viewpoints. EBL falls within the traditional symbolically-oriented approach to artificial intelligence. It has shown promise in high-level tasks such as planning [Minton89], natural language processing [Mooney89a], and diagnosis [Hammond88]. Neural (or connectionist) networks are largely numerically-oriented. Large numbers of processing units, each with minimal capability, operate without global control and together produce intelligent behavior. ANNs have proven successful on low-level tasks, such as perception [McClelland81] and signal processing [Ahalt89], as well as on classification and diagnostic tasks [Mooney89b]. Each approach has its strengths and weaknesses. An approach that combines the two styles, building on each's strengths and overcoming each's weaknesses, is described and a successful implementation reported. The approach is not intended to be plausible at the level of neurophysiology. Rather, it investigates whether symbolic and neural learning techniques can be profitably combined to produce more powerful, general-purpose machine learning algorithms.

Explanation-based learning is a recently developed and actively pursued approach to machine learning [DeJong86, Mitchell86]. In this type of learning, the solution to a sample problem is generalized into a form that can be later used to solve conceptually similar problems. The generalization process is driven by the explanation of why the solution worked. Knowledge about the domain allows the explanation to be developed, and then generalized, thereby producing a general concept from the solution to a specific problem. EBL systems can be used to produce general classification rules by analyzing the classification of a specific object (e.g., a rule for recognizing cups [Mitchell86]) or to produce general plans by observing the achievement of a goal in a specific situation (e.g., a plan for building towers [Shavlik90]).

One advantage of EBL systems is that they only require a small number of examples to learn a concept, unlike empirically-based learning methods (e.g., [Michalski83, Mitchell82, Quinlan86, Rumelhart86b]) which require large numbers of examples. Only a few examples are needed because EBL systems possess the ability to explain what is relevant in these examples. Other approaches, including connectionist ones, rely on statistics to determine relevance and hence need many examples. They may be mislead by spurious correlations unless someone initially selects a small set of relevant features and uses them to describe the training examples.

The strength of EBL arises largely from it use of prior knowledge to guide its learning. However, EBL's dependence on prior knowledge is the major source of EBL's weaknesses. The basic algorithms (e.g., [Hirsh87, Kedar-Cabelli87, Mooney86, Shavlik89a]) assume a complete and correct ''domain theory'' [Mitchell86, Rajamoney87]. This domain theory, which is usually a collection of inference rules describing a given domain, is used to build the explanations that are then generalized. If the domain theory is not complete, some examples cannot be explained and, hence, nothing is learned. Incorrect domain theories lead to incorrectly learned results. A final problem is that, even if correct and complete, a domain theory may not be usable because its complexity makes it intractable.

In any real-world domain, a computer system's model can only approximate reality. Needed inference rules may be missing, while many of those possessed will lead to incorrect results in some situations. Furthermore, the complexity of problem solving prohibits any semblance of completeness and tractability. Finally, some domains are inherently uncertain. The actions of other agents, both competitors and cooperators, cannot be fully determined, nor can the physical world be exactly predicted. In these circumstances a domain theory is necessarily approximate [Bennett88]. To be truly useful, EBL systems must be able to work in uncertain, intractable, incompletely-specified, and changing environments, something they currently are incapable of doing.

Artificial neural networks have also generated much interest recently. ANNs consist of a collection of simple, neuron-like processing units. These units are connected by weighted links, where the weights indicate the degree of influence one unit has on another. The activation of a unit is determined by a weighted function of the activations of all connecting units. Computation proceeds by activating *input* units and waiting until the activation of the *output* units stabilizes. During this process, activation may traverse through intermediary, or *hidden*, units.

One area in which ANNs have appeal is that of inductively acquiring concepts from examples. A set of examples, each labelled as belonging to a particular class, is provided to the ANN. Its task is to determine a procedure for classifying future examples. Impressive performance has been achieved on problems such as converting text to speech [Sejnowski87], evaluating moves in backgammon [Tesauro89], and diagnosing diseases [Mooney89b].

Several algorithms have been developed for training an ANN to classify new examples (e.g., [Hinton86, Rosenblatt62, Rumelhart86c]). Training is accomplished by analyzing a set of provided, pre-classified examples to determine how to appropriately modify the connection weights so that these training examples are more accurately classified.

While the performance of ANNs that adjust their connection weights has been impressive, there are some problems with this form of machine learning:

(1) *Training times are lengthy.* Recent experiments [Fisher89, Mooney89b, Shavlik89b, Weiss89] indicate that while ANNs classify new examples slightly better than do symbolic learning algorithms (such as ID3 [Quinlan86]), they require 100 to 1000 times as much training time.

(2) *The initial weights can greatly effect how well the concept is learned.* Because training ANNs usually involves a hill-climbing algorithm, there is the problem of local minima. Occasionally the learning algorithms cannot find any way to adjust weights to locally improve performance. Classification correctness on new examples can depend significantly on the initial set of weights of the ANN. For example, in some experiments [Shavlik89b] involving the conversion of text to speech, correctness on a large dictionary of examples ranged from 1% to 31%. The only difference between the runs was the randomly-chosen initial set of weights. Others have also reported that performance can be highly dependent on the initial set of weights [Ahmad88]. One way to overcome this problem is to train several networks, keeping the one that performs best on the training examples. For example, in [Ahmad88] usually 20 runs are performed. However, this substantially increases the training time.

(3) *There is no known problem-independent way to choose a good network topology.* The performance of an ANN can be greatly effected by the layout and number of hidden units, as well as the specification of which units are connected to which other units. Again, several runs with different topologies can be performed, keeping the best.

(4) *Many training examples are needed.* As is standard in learning systems that inductively acquire concepts from examples, ANNs, unlike EBL systems, need many training examples to acquire a concept [Baum89].

(5) *There is no way to focus attention on significant features.* ANNs have traditionally been trained in environments where the set of features to be considered is carefully selected by the experimenter. Without this pre-selection, the ANN would be forced to consider thousands of features normally dismissed as irrelevant by the human user of the system.

(6) *The learned network is difficult to interpret.* While the meanings of the ANN's input and output units are easily understood because they are fixed by the environment, the meaning of individual hidden units frequently cannot be determined. Hence, it is usually hard to ascertain reasons for the answers provided by the ANN.

The next section describes the hybrid approach that overcomes the weaknesses of both EBL and ANNs. Following that, the algorithm is presented in detail and applied to two sample problems. Related research and some open issues are then described. Common Lisp code that implements the algorithm appears in the appendix.

2. AN INTEGRATED APPROACH

EBL systems can take advantage of existing knowledge and, hence, need few examples to learn a new concept. However, they require a well-formed base with which to build explanations. Incorrect concepts may be learned if a faulty domain theory is used. Conversely, ANNs provide a way to incrementally adjust the representation of a concept on the basis of additional examples. Their weakness is that they require a large number of examples and many training epochs to converge on an acceptable concept, especially if they start from a randomly chosen initial state.

An approach that integrates these two styles of learning may accrue the benefits of each while avoiding each's shortcomings. The hybrid EBL-ANN system described below has been developed with this goal in mind. Basically, the EBL system produces categorization rules rapidly from a small number of examples. These rules are then refined by an ANN, using some additional examples. In this hybrid system, a complete and correct domain theory is not necessary, as the ANN refines learned concepts. Also, fewer training examples are needed than in a pure ANN, as the EBL portion of the system produces an initial network that is close to the correct concept.

In the hybrid EBL-ANN approach, an approximate domain theory is assumed. With this domain theory, the EBL portion of the system can build only roughly-correct explanations. These explanations need not be proofs, rather they need be reasonable arguments as to why the current item belongs to a specific category or why the current plan meets the given goal. That is, the domain theories can be incorrect. For domain theories that are intractable or incomplete, simplying or default assumptions can be made in order to produce these roughly-correct explanations [Chien89, Mitchell86].

Many EBL systems have domain theories of the desired form already, in particular those involving natural language (e.g, [Mooney85]). However, they treat their domain theories as being logically correct as theories representing mathematical calculations (e.g., [Shavlik85]). The hard part is not writing roughly-correct domain theories. The hard part is developing a learning system that properly treats the domain theory as roughly correct.

In the hybrid EBL-ANN system, once the EBL portion of the system produces a rough explanation, the explanation is generalized using a standard explanation-based generalization algorithm. The final, generalized result is then converted into a neural network. The antecedents of the classification rule determine the weights running from the input units to the hidden units, while the consequents determine the weights from the hidden units to the output units. Features that are believed to be important, according to the result produced by the EBL portion of the system, are highly weighted. Features that the rule says should be absent receive large negative weights. Finally, features that appear to be irrelevant receive weights near zero. These apparently irrelevant features must appear in the ANN because the roughly-correct EBL system may have mistakenly thought they were unimportant.

Once the ANN is produced, additional training examples are provided to refine the network, using standard neural network learning techniques. The final result can be some function of the inputs that is not expressible using the symbols and vocabulary in the domain theory. Hence, the hybrid system can increase the expressiveness of the basic EBL system, thereby increasing accuracy. Also, since the approach is naturally incremental, if the concept being learned is slowly varying over time, the ANN can capture these changes if it periodically receives feedback. Using only an EBL system would require rewriting portions of the domain theory whenever a concept changed. Finally, ANNs can store exceptions along with general concepts [McClelland86], thereby supporting another way to correct an EBL domain theory.

The major assumption underlying the EBL-ANN approach is that the space of possible concepts is reasonably ''smooth.'' When this assumption holds, it is likely that starting at a nearly-correct point in the space will faciliate reaching the correct concept. The validity of this assumption in real-world domains must be tested empirically. The assumption about the smoothness of concept space is also made by *case-based reasoning* researchers [Koton89]. In this paradigm, previous solutions are slightly modified ("tweaked") so that they apply to the current problem. The case-based approach has proven successful in many real-world domains.

Ahmad [Ahmad88] reports on an experiment that supports the hypothesis underlying the EBL-ANN hybrid. In this experiment, a neural network is configured with the ''correct'' set of weights. Next, the weights are perturbed. After some retraining, the network performs substantially better than if it had started off with random weights, even when the perturbations are large. This indicates the value of starting off with a set of weights that is close to a correct solution. Similar experimental results are reported in [Sejnowski87].

3. THE EBL-ANN ALGORITHM

Figure 1 contains the hybrid EBL-ANN learning algorithm. A roughly-correct domain theory involving binary features[1] is assumed, as are a collection of training examples and criteria for termination of learning.

In the hybrid algorithm, the EBL portion first uses the domain theory to classify one of the positive examples.[2] An explanation can be viewed as a deductive proof that the training example is a member of the concept being learned. The resulting explanation is then generalized using the EGGS algorithm [Mooney86]. This algorithm assumes that, in the course of solving a problem, a collection of pieces of general knowledge (e.g., inference rules, rewrite rules, or plan schemata) are interconnected, using unification to insure compatibility. The resulting explanation is generalized by determining the most general unifier that allows the general pieces of knowledge to be connected in the same way. This involves replacing the constants in the specific explanation with constrained variables. The result is a new composite knowledge structure that contains the unifications that must hold in order for the knowledge pieces to be combined in the given way.

To visualize the EBL process, assume a tree-structured explanation such as the one in figure 2. In this figure, all conclusions are conjunctive in that all antecedents must be satisfied in order for a node to be true. If the leaf nodes can be consistently satisfied, the root node, which concludes the example is a member of the concept, will also be satisfied. Internal nodes can be ignored. There is no need to again reason about combining the pieces of knowledge together to make the classification. Since a substantial amount of work can be expended constructing the original solution, the new knowledge structure can lead more rapidly to future solutions. A generalized version of the solution is saved and can be efficiently applied to future problems. Also notice that the explanation selects which features are relevant to classification.

Notice, though, that in this standard style of EBL, no *knowledge level learning* [Dietterich86] occurs. That is, only concepts in the deductive closure of a system's existing knowledge can be learned. Basic EBL systems speed up problem solvers but do not change what a system could solve if given enough time. The ANN portion of the hybrid system

---

[1] Being binary is not a restriction, in that multiple-valued features can easily be converted into binary features. For example, assume *color* is a feature that can take on the values *red, blue,* or *yellow.* This can be converted to the following binary features: *color-is-red; color-is-blue; color-is-yellow.*

[2] If the domain theory can also explain why something is *not* a member of the concept being learned, two ANNs can be produced.

GIVEN

1) List of binary features that will be used to describe examples.
2) Roughly-correct domain theory that can explain why an example is a member of the concept being learned.
3) Collection of training examples, classified as to whether or not they are members of the concept being learned.
4) Specification of a desired classification accuracy on the training examples.
5) Bound on the maximum number of training epochs.

DO

1) Use the roughly-correct domain theory to explain one of the positive examples of the concept to be learned.
2) Generalize the explanation using the EGGS algorithm.
3) Map the generalized explanation into an ANN (see text).
4) With the classified examples, train the ANN using the back-propagation algorithm.
5) If the training examples are correctly classified to within the specified accuracy or if the maximum number of epochs is reached, *STOP*. Otherwise randomly permute the list of training examples and go to step 4.

Figure 1. The EBL-ANN Algorithm

addresses the issue of knowledge level learning by refining the rules encapsulated in the ANN.

Since dealing with variables in neural networks is an open research area, the EBL-ANN algorithm currently assumes that domain theories are purely propositional (i.e., they do not contain any variables). This means that finding the most general unifier is trivial. In this case, the primary role of the EBL system is to focus attention on the features most likely relevant to classifying an example. The relevant features are those appearing in the leaf nodes of the explanation.

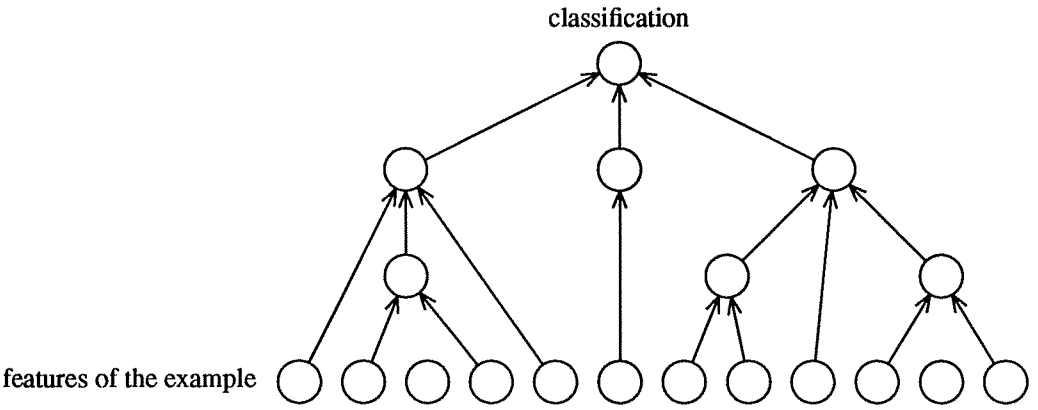

Figure 2. A Sample Explanation for a Classification Task

Following the application of the EGGS algorithm, the generalized explanation is converted into an ANN, as explained below. From this point, training continues following a standard neural network training procedure called *back-propagation* [Rumelhart86c]. Back-propagation adjusts inter-unit weights to reduce the difference between the ANN's output and the desired output. Training continues until enough training examples are properly classified or until a training limit is reached. An example is considered correctly classified if the ANN's output is within 0.25 of the correct output. (The setting of this parameter is discussed further later.)

Translation of the classification rule learned by the EBL component into an ANN network is guided by the structure of the explanation. The core layout of the ANN is isomorphic to the explanation structure, with the correspondences in figure 3 being made. Each conclusion, final or intermediate, in the explanation structure has a corresponding unit in the ANN. The final conclusion becomes the output unit for the network, while intermediate conclusions are the hidden units. Similarly, features used as inputs to the explanation correspond to units in the input layer of the network. Highly weighted links are added to the network in positions corresponding to the connection between antecedents which the explanation structure indicates are necessary for a consequent to be true. Conversely, links with large negative weights are added to the network in positions corresponding to the connection between antecedents which must be false for a consequent to be true. One important question is what values the weights and biases on the ANN's nodes should initially have. These are set initially such that the network

EXPLANATION	ANN
conclusion (root nodes)	output unit
supporting facts (leaf nodes)	input units
intermediate conclusions (internal nodes)	hidden units
dependencies (arcs)	weighted links

Figure 3. Correspondences between Explanations and Neural Networks

will only respond when the generalized explanation would also do so. The bias on each unit is set so that the unit will only be significantly active when: 1) all the units corresponding to antecedents that are necessary are active; 2) none of the units corresponding to antecedents which must be false are active.

To understand how weight and bias setting works, consider a unit with $n+m$ links, n from antecedents that are necessary, and m from antecedents that must be absent. The weights on the n links from the necessary antecedents are set to ω while the weights on the m links from the antecedents which must be absent are set to $-\omega$. The bias for the unit is set to $n\omega - \frac{\omega}{2}$. Hence, for the threshold to be exceeded, all of the necessary antecedents must be present and none of those that must be absent can be present. To prevent symmetries that are hard to break [Rumelhart86c], all weights are adjusted by a random amount between $-\varepsilon$ and $+\varepsilon$. Currently, $\omega=5$ and $\varepsilon=0.1$ are used.

Recall that the generalized explanation will most likely be somewhat incorrect. The neural learning algorithm will refine the learned concept, but it must have the nodes and links that allow it to do so. At a mimimum, this means additional input units must be added and connected to the network for every possible binary feature not mentioned in the explanation. To make the additional connections the level of each unit is established by determining the minimum length path from some input unit to the current unit. The new links, with a randomly-chosen weight

within ε of zero, are made from all units at level *i* to all units at level *i+1*, unless there already is a link between these two units or a loop would be formed. Thus, if a unit has even one connection from an input feature, it will be given low weight connections to *all* input features. The addition of these low-weight connections allows the insertion into the ANN network of features which were thought to be irrelevant by the roughly-correct domain theory.

After the ANN is initialized in a nearly correct configuration, additional training using back-propagation completes the learning process.

4. TWO EXPERIMENTS

The implementation of the EBL-ANN hybrid algorithm has been tested using two simple domain theories. The first involves recognizing cups and the second involves recognizing chairs. Using the same parameter settings, the hybrid system outperforms stand-alone EBL and ANN systems in both domains.

Learning to Recognize Cups

An adaptation of the traditional cup recognition problem [Mitchell86] is used as the first experimental domain. Figure 4 presents a set of Prolog rules which define a roughly-correct theory for the recognition of cups. The first rule can be read "X is a cup if it is stable, liftable and an open vessel." Using these rules something analogous to a coffee mug can be recognized as a cup. However, cups without handles (e.g. styrofoam cups) cannot be recognized. Also, a pail, where drinking is prevented by the handle across the mouth of the cavity [DeJong86], is falsely classified as a cup.

The EBL-ANN algorithm is run using the training set of ten cups and non-cups appearing in figure 5. Notice that styrofoam cups, which cannot be explained by the domain theory, are included (cups 2 and 3), as are pails (cups 5 and 7). Also notice that irrelevant features (e.g., *expensive* and *fragile*) are included.

Figure 6 contains the ANN as initially constructed from the explanation of the first positive training example. For clarity, units are labelled by the name of the concept they represent. Only the links and units mentioned in the explanation are shown, but recall that additional lowly-weighted links are also in the ANN. Figure 7 presents the hybrid system's performance averaged over 10 trials, where each trial involves different random weights.

For comparison, the performance of stand-alone versions of EBL and back-propagation, also averaged over 10 trials, are included. As the EBL component can use only the rules

cup	:-	stable, liftable, open-vessel.
stable	:-	bottom-is-flat.
liftable	:-	graspable, light.
graspable	:-	has-handle.
open-vessel	:-	has-concavity, concavity-points-up.

Figure 4. A Roughly-Correct Domain Theory for Recognizing Cups

	Cups				*Non-Cups*					
Features	1	2	3	4	5	6	7	8	9	10
Has Handle	√			√	√		√		√	
Handle on Top					√		√			
Handle on Side	√			√					√	
Bottom is Flat	√	√	√	√	√	√	√			√
Has Concavity	√	√	√	√	√		√	√	√	√
Concavity Points Up	√	√	√	√	√		√	√		
Light	√	√	√	√	√	√	√		√	
Made of Ceramic	√				√		√	√		
Made of Paper				√					√	
Made of Styrofoam		√	√			√				√
Expensive	√		√				√		√	
Fragile	√	√			√	√		√		√

Figure 5. The Training Examples for Learning to Recognize Cups

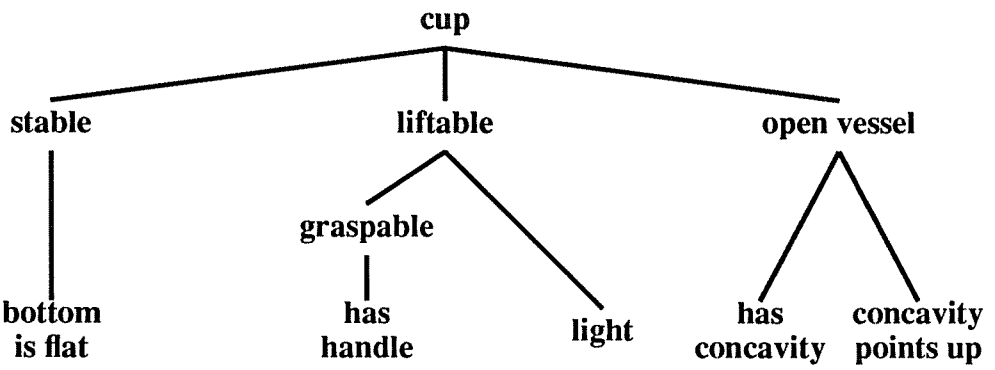

Figure 6. The Initial Network for Recognizing Cups (only highly weighted links shown)

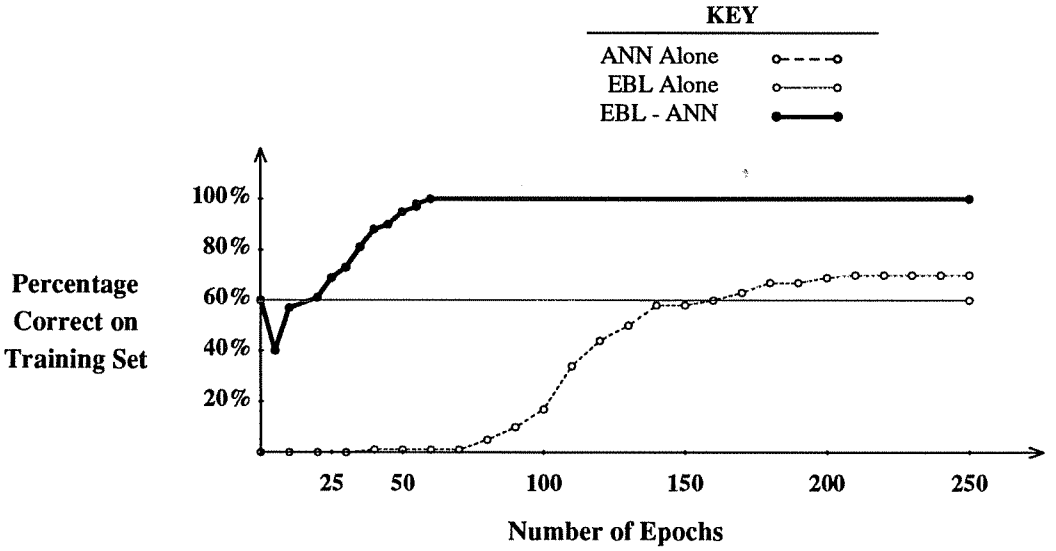

Figure 7. Classification Accuracy for Cups as a Function of Training Amount

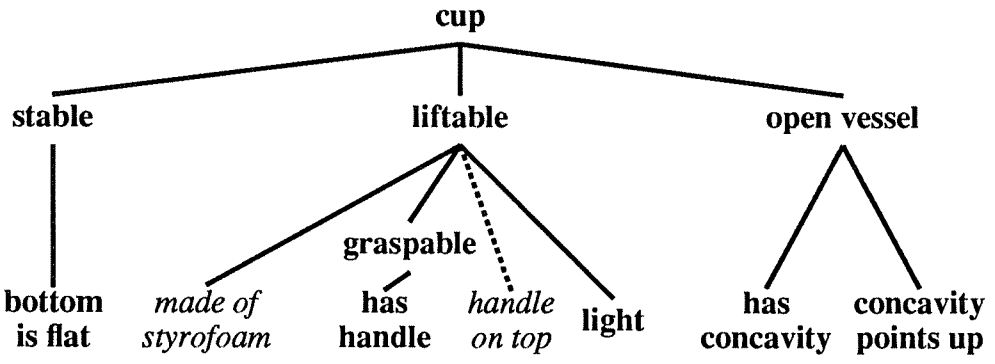

Figure 8. The Final Network for Recognizing Cups (only highly weighted links shown)

appearing in figure 4, and those rules will correctly classify only six of the ten examples, the EBL component by itself can never do better than 60% correct. Thus, in figure 7 the line representing the performance of stand-alone EBL is horizontal at 60% correct. For stand-alone back-propagation, the conventions used in [Mooney89b] are followed: there is one-layer of hidden units; the number of hidden units is set at 10% of the number of input plus output units (with a minimum of two hidden units); each input unit is connected to each hidden unit; each hidden unit is connected to the output unit; there are no direct connections between the input and output units. Having one layer of hidden units and full connectivity between layers is a common topology in neural network experiments.

The performance of the hybrid algorithm is initially the same as stand-alone EBL because the procedure which translates explanations into networks always generates networks which make the same decision as the EBL rule. After a slight period of floundering around, the hybrid quickly achieves 100% correctness. Averaged over the 10 trials, the hybrid system takes 44 epochs and has a standard deviation of 11 epochs. The stand-alone back-propagation algorithm takes an average of 398 epochs and has a standard deviation of 395 epochs. The hybrid has an speedup of over 9.

On three of the ten runs, the stand-alone back-propagation system became stuck at local minima, which accounts for this system not reaching 100% correctness. Notice that having an explanation guide the initial network configuration also promise to reduce the neural network problem of local minima.

The results presented in figure 7 indicate that the hybrid algorithm achieves the goals which motivated its creation. First, it is able to correctly classify the training set using one-fifth the number of training epochs required by stand-alone back-propagation. Second, 100% correctness is achieved, even though an imperfect domain theory is used.

Figure 8 presents a representative final ANN. Although not explicit in the figure, the weights on *made of styrofoam* and *graspable* are such that only one need be active to infer *liftable*. Hence, now a cup can be considered graspable if it is made of styrofoam. The dotted line represents a *negatively* weighted link. Thus, the handle of a cup cannot be on the top. Both false negatives and false positives, with respect to the domain theory, are now correctly classified.

Speedup achieved by using the hybrid heavily depends on the setting of the parameter that determines how close the output must be in order to be considered correct. If this parameter is set too small (e.g., 0.05), many epochs are needed to perform the final refinements. This reduces

the relative advantage of starting with a roughly-correct ANN. The trade-off is that if this parameter is set too large (e.g., 0.5), the final ANN is less likely to appropriately correct the domain theory. Spurious changes to inter-unit connections can do a good enough job to properly classify the training examples, thereby prematurely terminating training. For example, when the *close-enough* parameter is set higher than 0.25, highly-weighted links from *handle on top* to *open vessel* often appear in figure 8. (With larger numbers of training examples, higher settings may be appropriate. A good setting for this parameter is an open research issue.)

Learning to Recognize Chairs

The second example involves learning how to recognize chairs. A roughly-correct set of rules appear in figure 9. Among other things, this theory requires that chairs have four legs, a padded seat, and back support.

The EBL-ANN algorithm is run using the training set of ten chairs and non-chairs appearing in figure 10. All but the first chair are not recognized by the domain theory. For example, chairs 2 and 3 are three-legged. Conversely, non-chair 7, which cannot support weight, is classified as a chair. Hence, the domain theory properly classifies only half of the sample chairs.

Figure 11 contains the ANN as initially constructed from the explanation of the first training example. As before, units are labelled by the name of the concept they represent and only the links and units mentioned in the explanation are shown. The performance of the three

---

```
chair             :-  can-support, comfortable, stable.
can-support       :-  supports-bottom, supports-back.
supports-bottom   :-  horizontal-seat, seat-at-medium-height, medium-sized-seat.
supports-back     :-  vertical-back.
comfortable       :-  smooth-seat, padded-seat.
stable            :-  four-legs, equal-length-legs.
```

Figure 9. A Roughly-Correct Domain Theory for Recognizing Chairs

---

	Chairs					*Non-Chairs*				
Features	1	2	3	4	5	6	7	8	9	10
Vertical Back	√		√		√		√	√	√	√
Smooth Seat	√	√	√	√	√		√	√	√	√
Padded Seat	√		√	√		√	√	√	√	√
Horizontal Seat		√	√	√	√	√	√	√	√	√
Medium Sized Seat	√	√	√	√	√	√	√	√	√	
Seat at Medium Height	√	√	√	√	√	√	√		√	√
Four Legs	√			√	√	√	√		√	√
Three Legs		√	√					√		
Equal Length Legs	√	√	√	√	√	√	√	√		√
Supports Medium Weight	√	√	√	√	√	√			√	√
Has Arms	√		√	√		√		√	√	√
Has Wheels	√	√			√	√	√	√	√	√

Figure 10. The Training Examples for Learning to Recognize Chairs

systems appears in figure 12. The curves are the averages over ten trials, where each trial involves different random weights. The average number of epochs needed by the EBL-ANN hybrid is 89, while for the stand-alone back-propagation system it is 298. As when learning to recognize cups, the hybrid system is more accurate than a pure EBL system and learns faster than a pure ANN system.

As shown in figure 13, back-propagation training corrects the results of the EBL system in several ways. First, the requirements for back support and padded seats are dropped. Chairs 2 and 4 do not have back support, while chairs 2 and 5 do not have padded seats. With the cup domain theory, links are added but not dropped. In EBL terms, this means that in the chair domain some antecedents are added to rules and some dropped, while in the cup domain antecedents are only added. Second, three-legged chairs are recognized (chairs 2 and 3). The weights are such that a chair can have either three or four legs, so long as they are equal length. Third, to be a chair, an object must be able to support a sizable load.

Occasionally the hybrid system does not connect a new antecedent in the proper place. For example, in nine of the ten runs, the ability to support a sizable load is connected to the unit for bottom support. Once, however, instead it connects to the units for both stability and comfort. This problem of the ''reasonableness'' of the changes made by the ANN is further discussed in

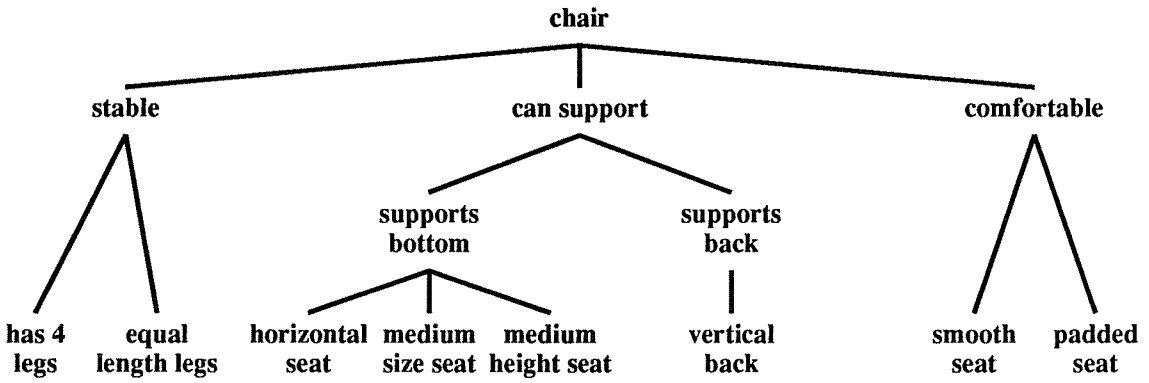

Figure 11. The Initial Network for Recognizing Chairs (only highly weighted links shown)

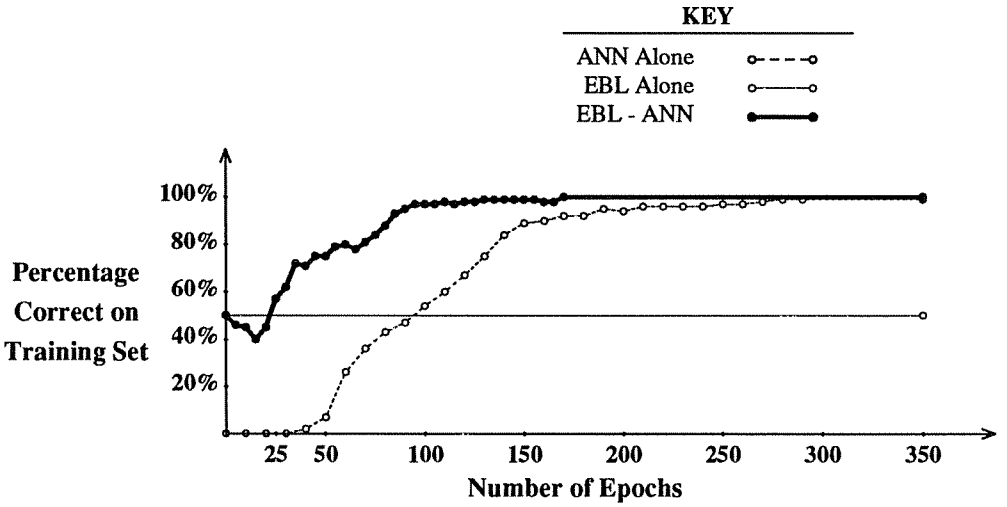

Figure 12. Classification Accuracy for Chairs as a Function of Training Amount

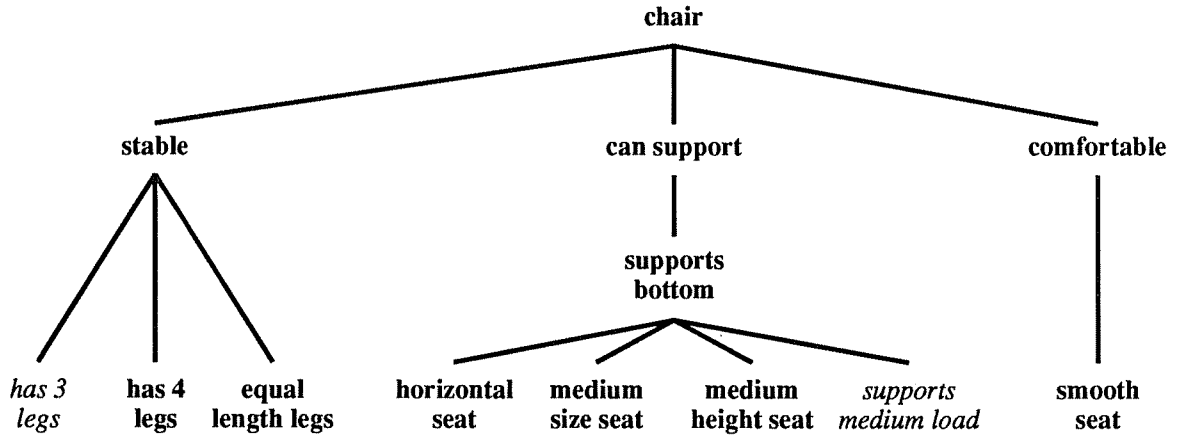

Figure 13. The Final Network for Recognizing Chairs (only highly weighted links shown)

the section on open research issues.

5. RELATED WORK

Three research areas bear directly on this work. One, there is a growing body of work on combining explanation-based and empirical approaches to learning. Two, work on representing symbolic data structures in neural networks is relevant to the issue of mapping from the EBL subsystem into ANNs. Three, there is additional research on using rules to guide neural network learning.

Others have investigated the combination of explanation-based and empirical learning methods(e.g., [Bergadano88, Danyluk87, Dietterich88, Lebowitz86, Pazzani88]). Some systems use empirical learning to provide input to an EBL system. For example, the UNIMEM [Lebowitz86] system analyzes a collection of examples and finds similarities. These similarities are then explained using EBL. Approaches more closely related to the EBL-ANN approach are those that use EBL to produce training examples for an empirical learning system. For instance, the IOE system [Dietterich88] uses a conventional inductive learning algorithm to empirically analyze a collection of explanations, thereby refining the domain theory. However, no one has investigated the possible synergies between an EBL system and neural networks. All of the research to date has focussed on combining systems that perform conventional, ''symbol-based'' computation.

Many researchers have investigated the representation of symbolic data structures in ANNs (e.g., [Rumelhart86a, Shastri88, Thrift89, Whitehead88]). Rumelhart *et al.* [Rumelhart86a] discuss the representation of schemata in neural networks. The results of EBL systems, which learn useful compositions of other concepts, can be viewed as schemata. In case-based reasoning systems, concrete episodes are stored. A neural approach for representing and selecting cases is described in [Thrift89]. EBL systems produce generalized cases and the issue of selecting the appropriate generalization to apply to a new problem is important [Minton88], but is currently ignored in the EBL-ANN system.

Oliver and Schneider [Oliver88] are investigating using rules to decompose problems into subproblems. Their results show that decomposing problems significantly reduces the learning time of neural networks. Unlike the EBL-ANN approach, they do not address the problem of dealing with incorrect rules. Mappings from rules to neural networks have been proposed [Gallant88, Touretsky88]. These approaches, though, do not deal with mapping useful combinations of rules into ANNs, nor do they address the issue of correcting rules.

6. OPEN RESEARCH ISSUES

While the current implementation of the EBL-ANN system has produced promising results, several aspects of it require refinement and extension. In addition, the approach must be tested in several real-world domains, in order to get a better estimate of its value. Along these lines, the post-learning correctness of the hybrid system on *unseen* examples must be ascertained and compared to that of stand-alone ANN systems.

Adding Additional Hidden Units

The current algorithm adds input units in addition to those involved in the explanation, but does not add any hidden units. Incorporating more hidden units may increase the power of the algorithm. Rules of thumb for the amount and connectivity of additional hidden units are needed.

Allowing Variables

Another extension of the initial algorithm involves the use of variables. The current algorithm assumes explanations are *propositional* and do not involve variables. A major strength of EBL algorithms is that they determine the most general constraints on the objects in a problem, a task usually accomplished by reasoning about which variables have to be equated with constants or other variables. Techniques proposed in [Smolensky87, Touretsky88] for variable binding in ANNs are being investigated. These techniques will be extended so that variables initially believed to be identical by the EBL subsystem can become separate variables following further training. Conversely, initially distinct variables may need to become unified.

Allowing Real-Valued and Hierarchical Features

A major shortcoming of the initial implementation is that all features must be *nominal*. That is, the possible values of a feature are described in an unordered set. The algorithm will be extended so that features that are *linear* (e.g., weight as a real number) or *hierarchical* (e.g., shapes where squares are polygons) can be handled. These three are the standard types of features used in inductive learning [Michalski83].

One way of using real-valued features that nicely falls within the EBL-ANN approach is to have the EBL subsystem reason using qualitative values, while having the ANN subsystem use quantitative values. For example, in order to simplify the building of explanations, the EBL system could use *small*, *medium*, and *large* as possible sizes. After the initial ANN is constructed, actual measurements could be used.

Providing Feedback from the ANN Subsystem to the EBL Subsystem

The research undertaken so far investigates having an EBL system provide initial guidance to an ANN. It is also desirable to have the results produced by the ANN affect the EBL portion of the system. Two approaches for doing so are under investigation. The first is to map the refined ANN back into the vocabulary of the EBL system so that the acquired concept can be symbolically analyzed and used by the EBL subsystem to acquire additional concepts. The second is to analyze what has changed in the ANN in order to locate the most uncertain portions of the concept, which can then be focussed upon. These two approaches are elaborated below.

After the ANN refines a rule acquired by the EBL portion of the system, the rule cannot be used by the EBL subsystem to produced additional explanations. The rule cannot be used because it is not in the vocabulary of the EBL subsystem. Intuitively, the hybrid approach nicely matches the phenomenon where after extensive experience with a task, one can no longer articulate how decisions are made. However, techniques are being developed for mapping the ANN back into the EBL subsystem's vocabulary. The re-represented rules can then be used to build new explanations. If additional hidden units appear in the ANN, some undefined symbols may appear in the rules, thereby extending the vocabulary of the EBL subsystem. The techniques could also be used as a way of making the knowledge encoded in an ANN human-comprehensible and to refine the basic rules of the EBL system. For example, after training the domain theory could be re-expressed and presented to the system's user.

By having a symbolic system analyze which weights in the ANN are fluctuating the most and by considering the magnitude of the back-propagated error signals, the most uncertain aspects can be localized. This information can be used in several different ways. For example, it can help produce focussed questions of a human advisor, e. g., "Is feature x important?". Also, if the system designs its own experiments [Carbonell87, Rajamoney89], this information can suggest which experiments will prove informative.

Using EBL Failure to Direct Back-Propagation

An issue previously mentioned is that the EBL-ANN system does not always add new links in the most appropriate place. For example, in different runs, the hybrid system learned that a strong negative weight from *handle on top* to any of the four intermediate conclusions produces the correct classifications of the sample cups and non-cups. This problem is reduced by having a low setting for the parameter that determines when the output is close enough to being correct, as low settings lead to additional back-propagation

training but at the cost of reducing the training time advantage of the hybrid. Methods of localizing back-propagation by examining explanation failure are being investigated in order to reduce the chance of inappropriate connections.

This method would only operate when back-propagation produced a false negative response. Rather than simply propagating error backward through the entire network, the EBL subsystem would be called, and an attempt to explain how the example fits the concept would be made. If the EBL subsystem is unable to complete the explanation, the areas of the explanation which are uncompleted could be identified. Those identified areas would be the only areas of the ANN to receive back-propagated error. For example, in attempting to explain that a styrofoam cup is a cup, both the stable and open-vessel consequents can be fully explained by the provided rules. However, liftable cannot be explained because graspable cannot be explained. Thus, the EBL component would indicate that back-propagation should focus its efforts on determining some other way to suggest that an object is graspable.

Applying the EBL-ANN Hybrid in Planning Domains

Classification tasks provide an attractive test bed in which to initially develop the EBL-ANN hybrid. Inductive learning systems, such as ANNs, have long done well on this type of task. Planning is a more challenging task with which to further develop the approach. There has been much less progress on using inductive learning to produce plans. Largely, this is due to the interdependencies among the steps in a plan. A learner must be able to reason about the causal relationships among the entities in the plan in order to decide what is and what is not relevant. EBL systems are well-suited to learning about plans, as they are capable of explaining the interdependencies among the steps in a plan. However, as previously mentioned, EBL systems are limited by the accuracy and coverage of their domain theories. Applying the EBL-ANN approach to planning problems promises to eliminate this limitation.

Techniques for representing and executing stored plans in neural networks are reported in [Whitehead88]. The applicability of these techniques to the EBl-ANN approach are being investigated.

Acquiring Recursive Concepts

Many, if not most, concepts and plans have a recursive or iterative nature. That is, a concept can involve any number of repeated components or some action actions can be performed as many times as necessary in order to achieve the goal of a plan. For example,

a planner may need to make a number of adjustments in order to complete a task. Sometimes four adjustments may be needed, sometimes six, other times none. The exact number will depend on the details of the current situation - presence of other agents, weather conditions, etc. Nevertheless, underlying this variability is essentially a single plan. It would be foolish for a performance system to have a different plan for each possible number of adjustments.

The recently-developed **BAGGER2** algorithm [Shavlik89a] recognizes the presence of recursive processes in the explanations of specific problems. When any such traces are encountered, a recursive concept or plan is produced. When recursive processes are not detected, the result is the same as that produced by **EGGS**. The **BAGGER2** algorithm greatly increases the generality of acquired concepts and plans, thereby reducing training time and improving the effectiveness of the system that uses the acquired results. This ability to acquire a recursive concept from a single training example is especially important in domains where only a small number of training scenarios are available.

Within neural network research, *recurrent* networks are used to represent recursive processes [Pineda87]. The EBL-ANN hybrid is being extended so that recursive concepts can be acquired. This is being accomplished by using the **BAGGER2** algorithm to produce roughly-correct concepts, which are then mapped into recurrent ANNs and refined.

Integrating Multiple Explanations into a Single ANN

A major issue in EBL is that a single training example may not suffice to fully learn the complete concept. For example, a general plan may involve several different ways to accomplish a subgoal while the training example may only use one of these. One way this problem can be addressed is by merging the results of multiple explanations into a single learned concept. Techniques are being developed for combining multiple explanations into a single ANN. A complicated concept, possibly containing several special cases, will be acquired by generalizing the explanations of several training examples. This means that the individual training examples can be simpler and, thus, more easily explained.

7. CONCLUSION

Using an explanation-based learning system to guide the construction of the initial neural network is an approach that overcomes the shortcomings of both explanation-based and neural learning. In the presented EBL-ANN system, the initial neural network configuration is determined by the generalized explanation of the solution to a specific classification task. Since the construction of the initial network is motivated, many fewer training examples are needed than if the initial network configuration is randomly chosen. The presence of irrelevant features is also much less of a problem because the EBL subsystem focusses attention on the features most likely to be relevant. In addition, this hybrid system overcomes a major shortcoming of explanation-based learning, namely, the problem of incorrect domain theories.

In the EBL-ANN system, a ''roughly-correct'' domain theory leads to the acquisition of a classification rule that is nearly correct. Subsequent refinement by the neural network improves it. Empirical results show that the hybrid system more accurately learns a concept than just the explanation-based system by itself and that the hybrid also learns it much faster than the neural learning system by itself.

Most realistic problems can never be formalized exactly. However, there is much to be gained by utilizing the capability to reason approximately correctly. EBL provides a way to profitably use causal models of the world, while ANNs provide a way to refine roughly-correct concepts. By combining symbolic and neural learning algorithms, the presented research promises to broaden the applicability of machine learning techniques, producing learning algorithms that are not brittle and which can produce concepts whose accuracy improves through experience.

Acknowledgement

This research was partially supported by the University of Wisconsin Graduate School.

Appendix - Common Lisp Version of the EBL-ANN Algorithm

```
;;;; A Common Lisp implementation of the hybrid EBL-ANN system for learning from examples.

;;;; Copyright (c) 1989 by Geoffrey Towell and Jude William Shavlik.
;;;;         This program may be freely copied, used, or modified provided that this
;;;;         copyright notice is included in each copy of this code and parts thereof.

;;;; This code assumes the presence of code to implement back-propagation
;;;; (BP, as described by Rumelhart et. al. in Chapter 8 of the PDP volumes) and EGGS
;;;; (as described by Mooney and Bennett in AAAI 1986).  Also, the existence of an
;;;; "explanation-builder" is assumed.

;;;; Interaction between this code and back-propagation occurs only
;;;; in the function RUN-HYBRID.  This code assumes that
;;;; back-propagation will alter link weights after each training example.

;;;; Interaction between this code and EGGS occurs only in the function
;;;; GENERALIZE.  This function determines the necessary unifications
;;;; in the explanation structure produced by EXPLAIN-AN-EXAMPLE.

;;;-------------------------------------------------------------------------
;;;     Global Variables
;;;-------------------------------------------------------------------------

;; Parameters which effect the operation of the hybrid system ------------------

(defvar *link-weight* 5
  "The initial weight on a link corresponding to a connection between
   an antecedent and a consequent in the explanation structure.")
(defvar *epsilon* 0.1
  "The limit of random variation added to links and biases in order
   to prevent unbreakable symmetries in back-propagation.")
(defvar *close-enough* 0.25
  "How close to correct the output of the ANN must be to be treated as correct.")
(defvar *desired-correctness* 100
  "Terminate learning if this percentage of the examples properly classified.")
(defvar *max-epochs* 1000
  "Terminate ANN training if all training examples are not classified
   correctly within this number of epochs (i.e., cycles through the training examples).")
(defvar *input-features* nil
  "List of binary features used to describe the training examples.")
(defvar *trace-hybrid* t
  "If T then produce useful output to the user.")
```

```
;; Variables to hold information about the ANN ------------------------------

(defstruct unit
  "This structure holds all the local information for units in the ANN."
  name              ; The identifier used in the explanation to identify an
                    ;   antecedent of a consequent.  Used during translation from
                    ;   the explanation structure to the ANN.
  (level -1)        ; 1 + minimum path length from this unit to the input layer.
                    ;   Hence, units at the input layer have level=1.
  (links nil)       ; A list of lists;
                    ;   each sublist consists of a location of a unit in
                    ;   *array-of-units* and the weight of the link between
                    ;   the pair of units.  This information is kept only by the
                    ;   receiving unit.
  (bias 0)          ; The bias of this unit.
  (activation 0)    ; The activation of the unit.
  (error 0))        ; The amount of error accumulated at the unit during back-prop.

(defvar *array-of-units* (make-array 0 :element-type 'unit :adjustable t :fill-pointer 0)
  "A 1-D array holding all of the units in the ANN.  A 1-D array is used because the depth
   of the ANN is not pre-determined.  Also, the array is generally sparse.")

(defvar *units-at-level* nil
  "An association list, the first element is the level,
   succeeding elements are units at that level.  This list is kept sorted
   by level, lowest first.  Level of input layer is 1.")

;;;---------------------------------------------------------------------------
;;;     The main function to run the hybrid EBL-ANN system
;;;---------------------------------------------------------------------------

(defun run-hybrid (training-examples goal &optional (max-epochs *max-epochs*)
                   &aux (epoch-counter 0) location-of-top-unit number-ANN-correct
                        (correctness-target (* (/ *desired-correctness* 100)
                                               (length training-examples))))
  "Executes the hybrid EBL-ANN system.  A goal def'n and training examples are provided."
  ;; Assumes that examples are in a list of lists.  The first element of each
  ;; sublist is from {t,nil} and indicates if the example is positive for the
  ;; concept.  The second element of the sublist is a simple numeric designation for
  ;; the example.  The rest of the sublist contains the features of the example.
  (setf location-of-top-unit (establish-network training-examples goal))
  (if *trace-hybrid* (format t "~&cycle   correct   computed~&"))
  ;; This loop command forces execution to continue until either all examples are
  ;; classified correctly or the maximum number of epochs has been reached.
  ;; Each pass through the loop is equivalent to one epoch.
```

```
  (loop
   (setf number-ANN-correct 0)
   (incf epoch-counter)
   (if *trace-hybrid* (format t "~%~3D" epoch-counter))
   (dolist (ex-to-use (permute training-examples))
           ;; Call back-propagation (BP) on this example with two arguments:
           ;;  the input feature values in a list and the correct output.
           ;; BP must set the activation of the units in *array-of-units*.
           (single-pass-through-BP (rest (rest ex-to-use))
                                   (if (positive-example? ex-to-use) 1 0))
           (let* ((ANN-answer (unit-activation (aref *array-of-units* location-of-top-unit)))
                  (correct-answer (if (positive-example? ex-to-use) 1 0))
                  (error (abs (- correct-answer ANN-answer))))
             (if *trace-hybrid* (format t "~5T~4D~6D     ~6,3f"
                                        (second ex-to-use) correct-answer ANN-answer))
             (if (<= error *close-enough*) (incf number-ANN-correct))))
   (if *trace-hybrid* (format t "~%ANN correctness:~6,2F%"
                              (* (/ number-ANN-correct (length training-examples)) 100)))
   (when (>= number-ANN-correct correctness-target)
     (if *trace-hybrid* (format t "Correctness goal reached."))
     (return))
   (when (>= epoch-counter max-epochs)
     (if *trace-hybrid* (format t "Maximum number of epochs reached."))
     (return))))

;;;-------------------------------------------------------------------------
;;;     Functions that call the EBL component
;;;-------------------------------------------------------------------------

(defun explain-an-example (examples goal &aux rule-proof)
  "Traverse through the list of examples, passing positive examples of the category
   to the explaining component until an explanation is generated."
  (clear-memory goal)  ;clear effects of previous experiments.
  (dolist (example examples)
          (when (positive-example? example)
                (if (setf rule-proof (explain-example-meets-goal example goal)) (return))))
    ;; The function EXPLAIN-EXAMPLE-MEETS-GOAL must be written.
    ;; It uses an EBL "domain theory" to build a "rule proof" demonstrating
    ;; that an example satisfies a goal (e.g., is a member of the concept being learned).
    ;; A <proof> of <goal> is defined recursively as either a
    ;; fact from the database which matches <goal> or a proof based on a rule:
    ;;    (rule-proof <consequent> ((<antecedent> <proof>)(<antecedent> <proof>)...))
    ;; where <consequent> and <antecedents>'s constitute a uniquely variablized rule.
    ;; This form of proof constitutes an "explanation structure" and is used by EGGS to
    ;; generate a generalized explanation.
  (if rule-proof (generalize rule-proof) (format t "~%NO RULE PROOF POSSIBLE~%")))
```

```
;;; Rule proofs are Common Lisp structures:
(defstruct (rule-proof (:type list) :named)
     consequent
     antecedent-proofs)

(defun generalize (rule-proof)
  "Apply the EGGS algorithm to this rule proof."
  ;; The function APPLY-EGGS-TO-RULE-PROOF must be written, also.
  ;; It takes a rule proof and determines the mimimal necessary unifications that
  ;; support all the inter-rule connections in a rule proof.  These unifications are
  ;; then applied to the rule proof, producing a generalized explanation, which is
  ;; returned.  (For propositional domain theories, this function need only return the
  ;; provided rule proof.)
  (apply-EGGS-to-rule-proof rule-proof))

;;;-----------------------------------------------------------------------------
;;;     Functions to translate an explanation into an ANN
;;;-----------------------------------------------------------------------------

(defun establish-network (examples goal)
  "Create a network.  Returns the array location of the top-level consequent."
  (translate-explanation-to-ANN (explain-an-example examples goal))
  (second (first (last *units-at-level*))))

(defun translate-explanation-to-ANN (explanation-to-translate)
  "Translate an explanation structure into an ANN."
  (initialize-network)              ;clear array holding the ANN
  (convert-explanation-to-ANN       ;add ANN units according to
       explanation-to-translate)    ;units in explanation
  (establish-BP-links)              ;set the inter-unit weights
  (add-additional-units)            ;add features not appearing in explanation
  (establish-levels)                ;determine the levels of the units
  (add-additional-links))           ;add additional links between levels

(defun convert-explanation-to-ANN (proof)
  "Recursively descend through the explanation structure, adding a unit to
   the ANN for exery antecedent and consequent in the explanation."
  (cond
   ((null proof) nil)                    ; nothing left on this branch of proof tree
   ((rule-proof-p proof)                 ; at an intermediate or terminal consequent
                                         ;    create a unit if one does not exist
                                         ;    then continue recursive descent.  On
                                         ;    return from descent, catch list of array
                                         ;    locations of the antecedents
```

```
    (let ((unit-position (make-a-unit? (rule-proof-consequent proof)))
          (collect-links nil))
      (dolist (rest-proof (rule-proof-antecedent-proofs proof))
        (if (rule-proof-p (second rest-proof))
            (push (convert-explanation-to-ANN (second rest-proof)) collect-links)
          (push (convert-explanation-to-ANN rest-proof) collect-links)))
      (push collect-links (unit-links (aref *array-of-units* (abs unit-position))))
      unit-position))
   (t                                   ; at a leaf of the tree.  If a unit has not
    (make-a-unit? (first proof)))))     ;    been made previously, make one.

(defun establish-BP-links ()
  "Takes the links which were identified in the translation of the
   explanation structure, and puts them into the format of an ANN."
  (dotimes (i (fill-pointer *array-of-units*))
    (let ((unit (aref *array-of-units* i))
          (new-links nil))
      ;; first set the bias to be N omega - omega/2,
      ;; where N is the number of necessary antecedents
      (setf (unit-bias unit) (vary-number
                              (- (* *link-weight*  ;necessary antecedents have positive number
                                    (count-if #'(lambda (x) (> x 0)) (first (unit-links unit))))
                                 (/ *link-weight* 2.0))))
      ;; add links for each antecedent in the explanation structure
      (dolist (link (first (unit-links unit))) (push (add-a-link link) new-links))
      (setf (unit-links unit) new-links))))

(defun add-additional-units ()
  "Add those input features to the ANN that did not appear in the explanation structure."
  (dolist (input *input-features*) (make-a-unit? (list input))))

(defun establish-levels (&aux units-without-level still-no-level)
  "For each unit in the ANN, determine the minimum path length from some
   input unit to that unit."
  (setf *units-at-level* nil)
  (dotimes (i (fill-pointer *array-of-units*)) (push i units-without-level))
  (loop (if (null units-without-level) (return))
        (setf still-no-level nil)
        (dolist (unit units-without-level)
          (let ((min-link (find-min-link (unit-links (aref *array-of-units* unit)))))
            (if min-link (setf (unit-level (aref *array-of-units* unit)) min-link)
              (push unit still-no-level))))
        (setf units-without-level still-no-level))
```

```
  ;; collect list of units at a given level
  (dotimes (i (fill-pointer *array-of-units*))
           (let* ((level-of-unit (unit-level (aref *array-of-units* i)))
                  (units-at-same-level (assoc level-of-unit *units-at-level*)))
             (if units-at-same-level (nconc units-at-same-level (list i))
               (push (list level-of-unit i) *units-at-level*))))
  (setf *units-at-level* (sort *units-at-level* #'< :key #'first)))

(defun add-additional-links ()
  "Connect each unit to all units whose path length to the input layer is shorter by 1."
  (dotimes (i (fill-pointer *array-of-units*))
    (let* ((unit (aref *array-of-units* i))
           (already-linked-units (mapcar #'first (unit-links unit))))
      (dolist (unit-at-i-1 (rest (assoc (- (unit-level unit) 1) *units-at-level*)))
              (if (not (member unit-at-i-1 already-linked-units))
                  (push (list  unit-at-i-1 (vary-number 0)) (unit-links unit)))))))

;;;-------------------------------------------------------------------------
;;;     Miscellaneous Utility Functions
;;;-------------------------------------------------------------------------

;;; Utilities specific to the translation of explanations into networks --------

(defun find-unit (unit-name)
  "Given the name of a unit, this function either returns the location of the
   unit in the array holding the network, or a location at which the unit can
   be added to the array."
  (or (dotimes (i (fill-pointer *array-of-units*))
                (if (equal (unit-name (aref *array-of-units* i)) unit-name) (return i)))
      (fill-pointer *array-of-units*)))

(defun make-a-unit? (unit-list &aux (name-of-unit (get-name-for-unit unit-list)))
  "Given the name of a unit, determine if a unit of that name already exists.
   If no unit exists, then create one.  Links to negative antecedents are temporarily marked by
   returning the negative of their location in the unit array (this is undone in ADD-A-LINK)."
  (let ((cur-pos (find-unit name-of-unit)))
    (if (= cur-pos (fill-pointer *array-of-units*))
        (vector-push-extend (make-unit :name name-of-unit) *array-of-units*))
    (if (negative-antecedent? unit-list) (- cur-pos) cur-pos)))

(defun get-name-for-unit (list)
  "Given a list which contains an antecedent or a consequent from the
   EBL explanation structure, pull out the name field in the list."
  (if (listp list)
      (if (eq (first list) 'NOT) (second list) (first list))
    list))
```

```
(defun negative-antecedent? (list)
  "Returns T if the antecedent is one which must not be present for the
   consequent to be true."
  (and (listp list) (eq (first list) 'NOT)))

(defun add-a-link (link-to-unit &aux (link-weight (vary-number *link-weight*)))
  "Given a unit location number, returns a list of the form (unit link-weight).
   Note that if the unit location is negative, then the unit corresponds to an
   antecedent which must not be present.  Thus, its link weight is less than 0."
  (if (> link-to-unit 0)
      (list link-to-unit link-weight)
    (list (abs link-to-unit) (- 0 link-weight))))

(defun find-min-link (links &aux (result most-positive-fixnum))
  "Given a list of links from a unit, find the unit with the minimum level to which
   one of the links point.  If the level of any pointed to unit has not yet been
   determined, then return nil."
  (if (null links) 1              ;true only of input units
    (dolist (link links (if (not (= most-positive-fixnum result)) (1+ result) nil))
      (let ((new-level (unit-level (aref *array-of-units* (first link)))))
        (if (and (> new-level 0) (< new-level result)) (setf result new-level))))))

;; Minor utility functions ------------------------------------------

(defun vary-number (&optional (number 0) (epsilon *epsilon*))
  "Returns a real number which is randomly varied by a number within +/- epsilon."
  (+ number (* epsilon (/ (- (random 20000) 10000) 10000)))) ;multiply by number in [-1,1]

(defun clear-memory (goal)
  "Erases any memory of previous learning by the EBL component."
  (setf (get (first goal) 'newrules) nil))

(defun initialize-network ()
  "Empties the network."
  (setf (fill-pointer *array-of-units*) 0))

(defun permute (list)
  "Returns a list with the same elements as the input list, but in a
   different, pseudo-random order."
  (mapcar #'rest (sort (mapcar #'(lambda (x) (cons (random 10000) x)) list)
                       #'> :key #'first)))

(defun positive-example? (example)
  "Returns T if the training example is a positive example of the concept."
  (first example))
```

References

[Ahalt89] S. C. Ahalt, F. D. Garber, I. Jouny and A. K. Krishnamurthy, "Performance of Synthetic Neural Network Classification of Noisy Radar Signals," in *Advances in Neural Information Processing Systems, Volume 1*, D. S. Touretsky (ed.), Morgan-Kaufmann, Los Altos, CA, 1989, pp. 281-288.

[Ahmad88] S. Ahmad, "A Study of Scaling and Generalization in Neural Networks," Technical Report CCSR-88-13, Center for Complex Systems Research, University of Illinois, Urbana, IL, November 1988.

[Baum89] E. B. Baum and D. Haussler, "What Size Net gives Valid Generalization?," *Neural Computation 1*, (1989), pp. 151-160.

[Bennett88] S. Bennett, "Real World EBL: Learning Error Tolerant Plans in the Robotics Domain," *Proceedings of the AAAI Explanation-Based Learning Symposium*, Stanford, CA, March 1988, pp. 122-126.

[Bergadano88] F. Bergadano and A. Giordana, "A Knowledge Intensive Approach to Concept Induction," *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, June 1988, pp. 291-297.

[Carbonell87] J. G. Carbonell and Y. Gil, "Learning by Experimentation," *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, CA, June 1987, pp. 256-266.

[Chien89] S. A. Chien, "Using and Refining Simplifications: Explanation-Based Learning of Plans in Intractable Domains," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Danyluk87] A. P. Danyluk, "The Use of Explanations for Similarity-Based Learning," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 274-276.

[DeJong86] G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning 1*, 2 (1986), pp. 145-176.

[Dietterich86] T. G. Dietterich, "Learning at the Knowledge Level," *Machine Learning 1*, 3 (1986), pp. 287-316.

[Dietterich88] T. Dietterich and N. Flann, "An Inductive Approach to Solving the Imperfect Theory Problem," *Proceedings of the AAAI Explanation-Based Learning Symposium*, March 1988, pp. 42-46.

[Fisher89] D. H. Fisher and K. B. McKusick, "An Empirical Comparison of ID3 and Back-propagation," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Gallant88] S. I. Gallant, "Connectionist Expert Systems," *Communications of the Association for Computing Machinery 31*, 2 (1988), pp. 152-169.

[Hammond88] K. J. Hammond and N. Hurwitz, "Extracting Diagnostic Features from Explanations," *Proceedings of the AAAI Explanation-Based Learning Symposium*, Stanford, CA, March 1988, pp. 31-35.

[Hinton86] G. E. Hinton and T. J. Sejnowski, "Learning and Relearning in Boltzmann Machines," in *Parallel Distributed Processing, Vol. I*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 282-317.

[Hirsh87] H. Hirsh, "Explanation-Based Generalization in a Logic-Programming Environment," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 221-227.

[Kedar-Cabelli87] S. T. Kedar-Cabelli and L. T. McCarty, "Explanation-Based Generalization as Resolution Theorem Proving," *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, CA, June 1987, pp. 383-389.

[Koton89] P. Koton, "Evaluating Case-Based problem Solving," *Proceedings of the Case-Based Reasoning Workshop*, Pensacola, FL, May 1989, pp. 173-175.

[Lebowitz86] M. Lebowitz, "Integrated Learning: Controlling Explanation," *Cognitive Science 10*, 2 (1986), pp. 219-240.

[McClelland81] J. L. McClelland and D. E. Rumelhart, "An Interactive Activation Model of the Effect of Context in Perception: Part 1. An Account of Basic Findings," *Psychological Review 88*, (1981), pp. 375-407.

[McClelland86] J. L. McClelland and D. E. Rumelhart, "A Distributed Model of Human Learning and Memory," in *Parallel Distributed Processing, Vol. II*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 170-215.

[Michalski83] R. S. Michalski, "A Theory and Methodology of Inductive Learning," *Artificial Intelligence 20*, 2 (1983), pp. 111-161.

[Minton88] S. Minton, "Quantitative Results Concerning the Utility of Explanation-Based Learning," *Proceedings of the National Conference on Artificial Intelligence*, St. Paul, MN, August 1988, pp. 564-569.

[Minton89] S. Minton, in *Learning Effective Search Control Knowledge: An Explanation-Based Approach*, Kluwer Academic Publishers, Hingham, MA, 1989.

[Mitchell82] T. M. Mitchell, "Generalization as Search," *Artificial Intelligence 18*, 2 (1982), pp. 203-226.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning 1*, 1 (1986), pp. 47-80.

[Mooney85] R. J. Mooney and G. F. DeJong, "Learning Schemata for Natural Language Processing," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 681-687.

[Mooney86] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551-555.

[Mooney89a] R. J. Mooney, in *A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding*, Pitman, London, *in press* .

[Mooney89b] R. J. Mooney, J. W. Shavlik, G. G. Towell and A. Gove, "An Experimental Comparison of Symbolic and Connectionist Learning Algorithms," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Oliver88] W. L. Oliver and W. Schneider, "Using Rules and Task Division to Augment Connectionist Learning," *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Montreal, Quebec, August 1988, pp. 55-61.

[Pazzani88] M. J. Pazzani, "Integrated Learning with Incorrect and Incomplete Theories," *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, June 1988, pp. 291-297.

[Pineda87] F. J. Pineda, "Generalization of Backpropagation to Recurrent and Higher Order Neural Networks," *Proceedings of the Conference on Neural Information Processing Systems*, Denver, 1987.

[Quinlan86] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning 1*, 1 (1986), pp. 81-106.

[Rajamoney87] S. Rajamoney and G. DeJong, "The Classification, Detection and Handling of Imperfect Theory Problems," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 205-207.

[Rajamoney89] S. A. Rajamoney, "Explanation-Based Theory Revision: An Approach to the Problems of Incomplete and Incorrect Theories," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, January 1989. (Also appears as TR UILU-ENG-88-2264, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Rosenblatt62] F. Rosenblatt, *Principles of Neurodynamics*, Spartan, New York, 1962.

[Rumelhart86a] D. E. Rumelhart, P. Smolensky, J. L. McClelland and G. E. Hinton, "Schemata and Sequential Thought Processes in PDP Models," in *Parallel Distributed Processing: Explorations in the Micro-Structure of Cognition, Vol. II*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 7-57.

[Rumelhart86b] D. E. Rumelhart, G. E. Hinton and J. L. McClelland, "A General Framework for Parallel Distributed Processing," in *Parallel Distributed Processing: Explorations in the Micro-Structure of Cognition*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 46-73.

[Rumelhart86c] D. E. Rumelhart, G. E. Hinton and J. R. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing, Vol. I*, D. E. Rumelhart and J. L. McClelland (ed.), MIT Press, Cambridge, MA, 1986, pp. 318-362.

[Sejnowski87] T. J. Sejnowski and C. R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems 1*, (1987), pp. 145-168.

[Shastri88] L. Shastri, "A Connectionist Approach to Knowledge Representation and Limited Inference," *Cognitive Science 12*, (1988), pp. 331-392.

[Shavlik85] J. W. Shavlik, "Learning about Momentum Conservation," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 667-669.

[Shavlik89a] J. W. Shavlik, "Acquiring Recursive Concepts with Explanation-Based Learning," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Shavlik89b] J. W. Shavlik, R. J. Mooney and G. G. Towell, "Symbolic and Neural Net Learning Algorithms: An Experimental Comparison," Technical Report, Department of Computer Science, University of Wisconsin, Madison, WI, 1989.

[Shavlik90] J. W. Shavlik, *Generalizing the Structure of Explanations in Explanation-Based Learning*, Pitman, London, in press .

[Smolensky87] P. Smolensky, "On Variable Binding and the Representation of Symbolic Structures in Connectionist Systems," Technical Report CU-CS-355-87, Department of Computer Science, University of Colorado - Boulder, 1987.

[Tesauro89] G. Tesauro and T. J. Sejnowski, "A Parallel Network that Leans to Play Backgammon," *Artificial Intelligence 39*, 3 (1989), .

[Thrift89] P. Thrift, "A Neural Network Model for Case-Based Reasoning," *Proceedings of the Second Case-Based Reasoning Workshop*, Pensacola Beach, FL, May 1989, pp. 334-337.

[Touretsky88] D. S. Touretsky and G. F. Hinton, "A Distributed Connectionist Production System," *Cognitive Science 12*, 3 (1988), pp. 423-466.

[Weiss89] S. M. Weiss and I. Kapouleas, "An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Whitehead88] S. D. Whitehead and D. H. Ballard, "Connectionist Designs on Planning," in *Proceedings of the 1988 Connectionist Models Summer School*, G. E. Hinton, T. J. Sejnowski and D. S. Touretzky (ed.), Morgan Kaufmann, San Mateo, CA, 1988, pp. 357-370.