The Truth Will Come to Light: Directions and Challenges in Extracting the Knowledge Embedded Within Trained Artificial Neural Networks

Alan B. Tickle, Robert Andrews, Mostefa Golea, and Joachim Diederich

***Abstract*— To date, the preponderance of techniques for eliciting the knowledge embedded in trained artificial neural networks (ANN's) has focused primarily on extracting rule-based explanations from feedforward ANN's. The ADT taxonomy for categorizing such techniques was proposed in 1995 to provide a basis for the systematic comparison of the different approaches. This paper shows that not only is this taxonomy applicable to a cross section of current techniques for extracting rules from trained feedforward ANN's but also how the taxonomy can be adapted and extended to embrace a broader range of ANN types (e.g., recurrent neural networks) and explanation structures. In addition the paper identifies some of the key research questions in extracting the knowledge embedded within ANN's including the need for the formulation of a consistent theoretical basis for what has been, until recently, a disparate collection of empirical results.**

***Index Terms*— Feedforward neural networks, finite-state automata, fuzzy neural networks, knowledge insertion, recurrent neural networks, rule extraction, rule refinement.**

I. Introduction

SINCE their renaissance in the mid-1980's, artificial neural network (ANN) techniques have been successfully applied across a broad spectrum of problem domains such as pattern recognition and function approximation [36], [37]. However, ANN's remain something of a numerical enigma. In particular they afford an end user little or no insight into either the process by which they have arrived at a given result or, in general, the totality of "*knowledge*" actually embedded therein.

To redress this situation, considerable effort has been recently expended on addressing the general problem of supplementing ANN's with the requisite explanation capability. In particular, a substantial part of this effort has focused on a line of investigation involving the development of techniques for extracting and representing the *modus operandi* of trained feedforward ANN's as sets of symbolic rules. In order to provide a rigorous basis on which systematic comparisons of such ANN rule-extraction techniques could be made, Andrews *et al.* [1] proposed a taxonomy for categorizing the numerous contributions in this area. Subsequently, other authors expanded on this original survey and examined various topical issues in the field of rule-extraction from trained feedforward ANN's including, in particular, factors such as algorithmic complexity which limit what is actually achievable through the rule-extraction process [38]–[40].

In parallel with the development of techniques for extracting rules from trained feedforward ANN's, useful progress has also been made in two other facets of eliciting the knowledge embedded within ANN's. In the first instance, this includes techniques for extracting finite-state machine (FSM) representations from recurrent neural networks (RNN's). A second area in which considerable progress has also been made is that of using an ANN to refine an existing rule base. In this process, the initial knowledge about a given problem domain is inserted into an ANN by programming some of the weights. After training the network on the available datasets, a "refined" set of rules is then extracted. Importantly, the process of knowledge insertion has, in certain cases, been shown to lead to reduced ANN training times and improved generalization [15].

The range and diversity of techniques being developed underscores the fact that extracting the knowledge embedded within trained ANN's is still an active and evolving discipline. Hence the purpose of the ensuing discussion is:

1) to provide a brief critique of the ADT taxonomy for classifying techniques for extracting rules from trained feedforward ANN's;
2) to identify the important lessons which have been learned so far in the area of ANN rule-extraction;
3) to show how the taxonomy can to be modified so that it can be applied to the more general problem of categorizing knowledge-extraction techniques;
4) to assess the efficacy of this revised taxonomy by using it to categorize a small but representative sample of recent contributions in the areas of both rule-extraction and rule-initialization/refinement;
5) to identify and discuss some of the key research questions which will have an important bearing on the direction that future development of the field will take.

Manuscript received February 21, 1997; revised November 24, 1997. This work was supported by the Australian Research Council and the QUT Meritorious Project Grant Scheme to A. B. Tickle and J. Diederich. This paper was originally scheduled as part of a special issue on Neural Networks and Hybrid Intelligent Models, vol. 9, no. 5, September 1998. Guest Editors were C. L. Giles, R. Sun, and J. M. Zurada.

A. B. Tickle, R. Andrews, and J. Diederich are with the Neurocomputing Research Centre, Queensland University of Technology, GPO Brisbane, Queensland 4001, Australia.

M. Golea is with the Department of Systems Engineering, Research School of Information Sciences and Engineering, Australian National University, Canberra, Australian Capital Territory, Australia.

Publisher Item Identifier S 1045-9227(98)06183-9.

II. A Review of the Salient Points of the Taxonomy for Categorizing Techniques for Rule Extraction from Trained Feedforward ANN's

The survey conducted by Andrews *et al.* [1] of techniques for extracting the knowledge embedded within trained ANN's focused exclusively on the (preponderant) subset of techniques for extracting rules from trained feedforward ANN's. This survey introduced a taxonomy for categorizing such techniques which incorporated a total of five primary classification criteria viz.

1) the *expressive power* (or, alternatively, the *rule format*) of the extracted rules;
2) the *quality* of the extracted rules;
3) the *translucency* of the view taken within the rule extraction technique of the underlying ANN units;
4) the algorithmic complexity of the rule extraction/rule refinement technique;
5) the extent to which the underlying ANN incorporates specialized training regimes (i.e., a measure of the *portability* of the rule extraction technique across various ANN architectures).

For the first of these criteria i.e., the "*expressive power (or, alternatively, the rule format) of the extracted rules,*" the authors suggested three groupings of rule formats viz. 1) conventional (Boolean, propositional) symbolic rules; 2) rules based on fuzzy sets and logic; and 3) rules expressed in first-order-logic form, i.e., rules with quantifiers and variables. Building upon previous work by other authors such as Towell and Shavlik [42], Andrews *et al.* [1] suggested a set of four measurements of *rule quality* viz.:

1) rule accuracy (i.e., the extent to which the rule set is able to classify a set of previously unseen examples from the problem domain correctly);
2) rule fidelity (i.e., the extent to which the rule set mimics the behavior of the ANN from which it was extracted);
3) rule consistency (i.e., the extent to which, under differing training sessions, the ANN generates rule sets which produce the same classifications of unseen examples);
4) rule comprehensibility (e.g., measuring the size of the rule set in terms of the number of rules and the number of antecedents per rule).

The third criterion (i.e., the *translucency criterion*) sought to categorize a rule extraction technique based on the granularity of the underlying ANN which was either explicitly or implicitly assumed within the rule extraction technique. As shown in Fig. 1, Andrews *et al.* [1] used three key identifiers i.e., decompositional, eclectic, and pedagogical, to define reference points on the spectrum of such perceived degrees of granularity.

For example, at one extremity of the translucency spectrum are positioned those techniques which view the ANN at its maximum level of granularity i.e., as a "black box." This so-called "pedagogical" group of techniques extract global relationships between the inputs and the outputs of the ANN directly, without analyzing the detailed characteristics of the underlying ANN solution. At the opposite end of the translucency spectrum are the so-called "decompositional" techniques. The characteristic difference about these techniques is that they view the ANN at its minimum (or finest) level of granularity i.e., at the level of individual (hidden and output) units. In particular, in decompositional techniques, rules are first extracted at the individual (hidden and output) unit level within the ANN solution. These subsets of rules are then aggregated to form global relationships. Unfortunately for the classification process, not all techniques map neatly into one or the other extremities of the translucency spectrum. For example, the label "eclectic" was devised to accommodate essentially hybrid techniques (e.g., DEDEC [40]) which analyze the ANN at the individual unit level but which extract rules at the global level.

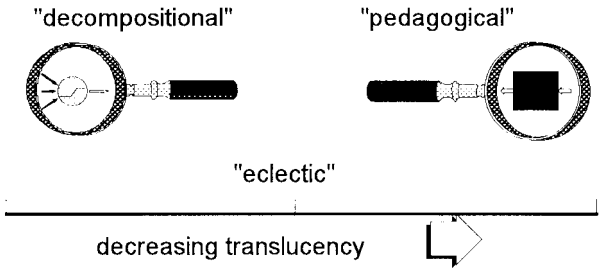

Fig. 1. The "translucency" criterion for categorizing techniques for extracting rules from trained feedforward ANN's.

The fourth of the proposed criteria addressed the key issue of the complexity of the core algorithm(s) which underpin the rule-extraction technique. However, as noted by Andrews *et al.* comparisons in this area are hampered by the fact that a substantial number of authors do not report or comment on this issue.

The final criterion assessed ANN rule-extraction techniques in terms of the extent to which a given technique could be applied across a range of ANN architectures and training regimes. The need for such a criterion was established on the basis that a characteristic feature of techniques for extracting rules from trained ANN's was the preponderance of what might be termed as "*specific purpose*" techniques, i.e., those where the rule extraction technique had been designed specifically to work with a particular ANN architecture.

The taxonomy as summarized above provided a useful means of categorizing and comparing the type of rule-extraction techniques which were available at the time it was proposed. During the intervening period, additional techniques have been proposed to achieve a similar result. In light of this substantive progress, the point has now been reached where it is possible to draw some general conclusions as to what has been achieved to date in ANN rule extraction. This is the subject of the discussion in the ensuing section.

III. Four "Take Home" Messages from ANN Rule Extraction

The field of rule extraction from trained feedforward ANN's now encompasses a rich and diverse range of ideas, mechanisms, and procedures. Hence it is useful at this juncture to make some general observations and comments on what has been achieved to date. In particular, it is possible to glean at

least four main lessons of potential importance to a prospective user of such techniques. These are as follows.

1) There now exists a substantive body of results which confirms both the effectiveness and utility of the overall ANN rule-extraction approach as a tool for rule-learning in a diverse range of problem domains. In particular Fu [12] has shown that the combination of ANN's and rule-extraction techniques outperforms established rule-learning techniques such as C4.5 (Quinlan, [30]) in situations in which there is noise in the datasets and also in situations involving multicategory learning. In addition, both Towell and Shavlik [42], and Omlin and Giles [26] have identified situations in which the extracted knowledge representation (viz. rules in the case of Towell and Shavlik, deterministic finite-state automata [DFA's] in the case of Omlin and Giles) outperformed the ANN from which they were extracted. (Towell and Shavlik [42] suggest that this may be because the rule-extraction process reduces over-fitting of the training examples.)
2) No clear difference has yet emerged between the performance of *specific purpose* rule-extraction techniques and those techniques with more general portability (Andrews *et al.* [2]). Moreover, while the utility of specific purpose techniques is attractive to an end-user, there is an ongoing demand for techniques which can be applied in situations where an ANN solution already exists.
3) Computational complexity of the rule extraction process may be a limiting factor on what is achievable from such techniques. In particular Golea [18] showed that, in many cases, the computational complexity of extracting rules from trained ANN's and the complexity of extracting the rules directly from the data are both NP-hard. Hence the combination of ANN learning and rule-extraction potentially involves significant additional computational cost over direct rule-learning techniques.
4) The possibility exists of significant divergence between the rules that capture the totality of knowledge embedded within the trained ANN and the set of rules that approximate the behavior of the network in classifying the training set (Tickle *et al.* [40]). For example, Tickle [41] reported such a situation in extracting rules from a number of ANN's trained in a problem domain (the edible mushroom problem domain) characterized by a) the availability of only a relatively small amount of data from the domain and b) the fact that many of the attributes appear to be irrelevant in determining the classification of individual cases. Overall however, the lesson from this result is that it simply behooves an end user to validate any rules extracted either by using further test data from the problem domain or by having the rules scrutinized by a domain expert.

IV. Extension of the Taxonomy Beyond ANN Rule Extraction

While the ADT taxonomy was conceived in the context of categorizing ANN rule-extraction techniques, in fact it embodies five quite general notions viz. 1) *rule format*; 2) *quality*; 3) *translucency*; 4) *algorithmic complexity*; and 5) *portability*. Consequently these notions can be applied to other forms of representing the knowledge extracted from ANN's as well as to other ANN architectures and training regimes. However, as will be shown in the ensuing discussion, recent developments necessitate some additional refinement in the labeling of reference points in the translucency spectrum (Fig. 1). For example, an important area of on-going development is in techniques for extracting DFA's from recurrent neural networks. In particular, one such technique [15] analyzes the state space consisting of all recurrent neurons in order to find clusters of hidden neuron activations. Hence the granularity of the underlying ANN which is explicitly assumed within this DFA-extraction technique is at the level of ensembles of neurons rather than individual neurons. Hence the technique is not strictly decompositional because it does not extract rules from individual neurons with subsequent aggregation to form a global relationship. Moreover, the DFA extraction technique does not fit into the eclectic category either because there is no aspect which fits the pedagogical profile. To accommodate such techniques which operate at an intermediate level of granularity, it is proposed to use the label "compositional"—which denotes techniques which extract rules from ensembles of neurons rather than individual neurons.

The purpose of the ensuing discussion is to show how this more general taxonomy can be applied to a variety of new techniques and ideas for ANN knowledge extraction.

V. Validation of the Taxonomy in the Light of New Techniques and Ideas for ANN Knowledge Extraction

The ensuing survey is essentially aimed at fulfilling two roles. The overall aim is to show how the extended ADT taxonomy introduced in the preceding section can be applied to a representative sample of the more recent contributions to the field of extracting knowledge from ANN's. The second aim is to demonstrate the rich diversity of techniques being developed to achieve ANN knowledge extraction.

For the purposes of presentation, the selected techniques have been divided into two main groups. The first group comprises what could be considered as straightforward rule-extraction techniques, while the second group involves those techniques which incorporate aspects of rule initialization and rule refinement.

A. Rule Extraction Techniques

1) The COMBO Algorithm—Krishnan [22]:

Rule Format:	Propositional with Boolean input.
Quality:	Extracted rules show high fidelity, accuracy, and comprehensibility.
Translucency:	Decompositional.
Complexity:	Exponential in the worst case.
Portability:	Generally applicable to feedforward networks with Boolean inputs.

Maire [24] and Krishnan [22] suggest a heuristic based on ordering of weights that limits the search space of decompositional algorithms such that important rules are not overlooked.

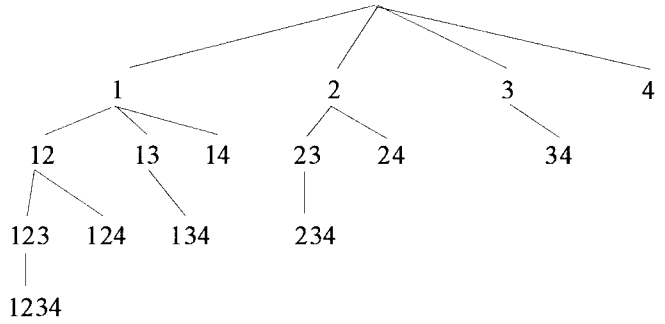

Fig. 2. A combination tree.

The method suggested by Maire [24] is similar to the COMBO algorithm [22]. COMBO is applicable to feedforward networks with Boolean inputs. COMBO first sorts the incoming weights of a particular node in descending magnitude then forms a "combination tree" of the sorted weights. The combination tree for a unit with four (4) incoming weights is shown in Fig. 2. The combination tree is systematically searched to form rules of the form

IF $\sum w_p +$ bias on neuron $>$ threshold on neuron
THEN concept corresponding to the neuron is true

where w_p is the set of weights at the node under consideration of the combination tree.

The premise is that judicious pruning of the combination tree can reduce the search space while at the same time preserving all important rules. Pruning can occur:

1) **at the same level in the tree.** If any combination at any level fails then all other combinations at this level can be pruned away. This is because all the combinations at the same level have the same length, and, because of the ordering of the weights, if a combination at a level fails, then all other combinations at the same level will also fail as their weighted sum will be less than the weighted sum of the combination that failed. Hence they need not be considered in the search for rules.
2) **at deeper levels of the tree.** If a combination at a level succeeds in forming a rule then all combinations in the subtree of which it is a root can be pruned away. Although these combinations will also succeed in forming a rule, these rules will be less general than the rule formed from the root of the subtree.

Krishnan [22] provides a detailed analysis of the complexity of COMBO concluding that in the worst case COMBO, like other search based decompositional algorithms (Fu [11], [12]), is exponential. However Krishnan [22] concludes that COMBO will generally perform faster in practice due to the weight ordering approach.

2) The RF5 (Rule Extraction From Facts) Algorithm—Saito and R. Nakano [32]:

Rule Format: Scientific laws.
Quality: Extracted laws show high accuracy and fidelity.
Translucency: Decompositional.
Complexity: Linear in number of hidden units and inputs.
Portability: Not portable.

Saito and Nakano's RF5 (rule extraction from facts version 5) algorithm [32] extracts scientific laws of the form

$$y_t = c_0 + \sum_{i=1}^{h} c_i x_{t1}^{w_{i1}} \cdots x_{tn}^{w_{in}}$$

where each parameter c_i or w_{ij} is an unknown real number and h is an unknown integer. The technique is based on an ANN that uses "product units" (Durbin and Rumelhart [8]) in the hidden layer. Product units calculate a weighted product of input values where each input value is raised to power determined by a variable weight. Saito and Nakano [32] develop their own second-order learning algorithm called BPQ. After training each product unit represents a term in the above expression. Because it is not possible to determine *a priori* the optimum number of hidden layer product units several networks with different numbers of hidden layer units are trained and then a minimum description length (MDL) criterion (Rissanen [31]) is utilized to select the best law candidate from among the trained networks. The technique has been applied to artificial data with both integer and real valued exponents and to real world data including discovering Hagen–Ruben's law, Kepler's third law, and Boyle's law. Experiments showed that RF5 successfully discovered the underlying laws even if the data contained a small amount of noise.

3) RX—Setiono [34]:

Rule Format: Propositional.
Quality: Good.
Translucency: Decompositional.
Complexity: Not available.
Portability: Generally applicable to supervised feedforward networks.

RX generates rules from pruned trained feedforward networks by considering a small number of hidden unit activation values. Networks are constrained to a single hidden layer. If the number of inputs to a hidden units is sufficiently small, then rules describing a limited number of discrete activation values are extracted from this unit. Otherwise the hidden unit will be split, i.e., turned into a subnetwork which each output unit of this subnetwork using a discrete activation value as a target for training. A new hidden layer is inserted and the new subnetworks is trained and pruned.

In more detail, the RX algorithm consists of the following steps shown in Table I ([34], p. 208).

Step 3) of the RX algorithm assumes that the number of hidden units as well as the number of clusters in each hidden unit is small. If there are H hidden units in the pruned network, and Step 2) of RX finds C_i clusters in hidden unit $i = 1, 2, \cdots, H$, then there are $C_1 \times C_2 \cdots \times C_H$ possible combinations of clustered hidden unit activation values. For each of these combinations, the network output is computed using the weights of the connections from the hidden units to the output units. Rules that describe the network outputs in

TABLE I
THE RX ALGORITHM

1. Train and prune the neural network.
2. Discretize the activation values, generate rules that describe the network outputs.
3. Using the discretised activation values, generate rules that describe the network outputs.
4. For each hidden unit:
   a) If the number of input connections is fewer than an upper bound Uc, then extract rules to describe the activation values in terms of the inputs.
   b) Else form a subnetwork:
      i. Set the number of output units equal to the number of discrete activation values. Treat each discrete activation value as a target output.
      ii. Set the number of input units equal to the inputs connected to the hidden units.
      iii. Introduce a new hidden layer.
      iv. Apply RX to this subnetwork.
5. Generate rules that relate the inputs and the outputs by merging rules generated in steps 3 and 4.

terms of the clustered activation values are generated using the X2R algorithm [23].

RX has been tested by use of the splice-junction and sonar classification benchmark datasets. The author states that RX's accuracy and rule quality is good compared with C4.5 and M-of-N on the two datasets mentioned above.

4) Knowledge Extraction and Recurrent Neural Networks—Schellhammer et al. [33]:

Rule Format: DFA/State transition rules.
Quality: Good.
Translucency: Compositional (i.e., the process extracts rules from ensembles of neurons rather than individual neurons as in the purely decompositional approach).
Complexity: Not available.
Portability: Generally applicable to recurrent neural networks with Boolean input.

Schellhammer *et al.* [33] used Elman simple recurrent neural networks (SRN's) [9] trained on a natural language processing task. The task was to learn sequences of word categories in a text derived from a first-year school reader [20]. The grammar induced by the network was made explicit by extracting a deterministic finite-state automaton (DFA). A network with nine hidden units learned 69% of the data. By way of comparison, 48, 62, and 72% correct predictions could be obtained using bigram, trigram and four-gram models, respectively. At the end of training, the networks performed one pass through the training data without learning in order to recover hidden unit activations. Following k-means cluster analysis of the hidden unit activations, a DFA state transition table was constructed using the algorithm in Table II. DFA's (having up to 20 states) were prepared. The best (having 18 states) scored 60% on the training data. A recurrent network initialized with weights based on *a priori* knowledge of the text's statistics, learned slightly faster than the original network.

5) The Interval Analysis (IA) Algorithm—Filer et al. [10]:

Rule Format: Propositional with continuous or discrete input.

TABLE II
ALGORITHM USED FOR DFA EXTRACTION FROM ELMAN NETWORKS

1. K-means cluster analysis software was used to label the 642 hidden unit activation vectors with cluster numbers between 1 and k.
2. For every current input and previous state there is a transition to a new state with a resulting output. A transition table was created from this data.
3. If the same input lead to more than one transition from a given state, the transition having highest frequency was chosen. Similarly, if any transition brought about by a given input, generated more than one possible output, the most frequent output was chosen.
4. The transition rules so derived were used to construct deterministic FSA's having k states corresponding to the k clusters.
5. Each automaton was tested on the string of 643 categories used to train the original network. They were scored for total correct predictions, the fraction of missing transitions and score on the nonmissing transitions.
6. Low-frequency transitions (having less than l occurrences) were pruned from the automaton and the resulting automaton again tested for its performance on the original data sequence. Missing transitions were handled by jumping to a predefined "rescue" state and producing a predefined "rescue" output.

TABLE III
THE IA ALGORITHM

Repeat
   Identify a very small hypercube in input space which belongs to the target class.
   Iteratively enlarge and move this hypercube in input space until it accounts for as much input space that belongs to the desired target class as possible subject to the constraint that the hypercube cannot include any input space belonging to a class other than the desired target class.
Repeat
      Seed very small hypercubes at each face of the previously found hypercube and enlarge and move these hypercubes until they also account for as much of the remaining input space belonging to the desired target class as possible and do not overlap with any other hypercube.
      Until no newly added hypercube can be enlarged to greater than the minimum size.
Until all noncontiguous regions of input space belonging to the desired class are covered.

Quality: Extracted rule sets show high accuracy but low comprehensibility.
Translucency: Pedagogical.
Complexity: Not available.
Portability: Portable.

The IA (Table III) algorithm of Filer *et al.* [10] is a method of extracting rules from a trained MLP in the form of intervals of activation which neurons can take such that the neuron produces a desired output. Rules extracted using IA are of the form

IF $\forall 1 \leq i \leq n : x_i \in [a_i^{\min}, a_i^{\max}]$
THEN concept represented by the unit is true

where n is the input dimensionality and $[a_i^{\min}, a_i^{\max}]$ represents the interval of input space in the ith input dimension over which the neuron will produce the desired output. Thus a rule describes a hypercube in input space.

Fundamental aspects of the IA algorithm which contribute to the complexity of the algorithm include the determination of:

1) the initial center of each newly added hypercube;

TABLE IV
THE TREPAN ALGORITHM

```
TREPAN (training_examples, features)
  Queue := 0; /* sorted queue of nodes to expand */
  for each example E ∈ training_examples /*use trained ANN to label examples*/
    class label for E := ORACLE(E)
  initialize the root of the tree, T, as a leaf node
  put ⟨T, training_examples, {}⟩ into Queue
  while Queue is not empty and size(T) < tree_limit_size /*expand a node*/
    remove node N from head of Queue
    examples_N := examples of set stored with N
    constraints_N := constraint set stored with N use features to build a set of candidate splits
    use examples_N and calls to ORACLE(constraints_N) to evaluate splits
    S := best binary split
    search for best m-of-n split, S', using S as a seed
    search for best m-of-n split, S', using S as a seed
    make N an internal node with spit S'
    for each outcome s, of S' /*make children nodes*/
      make C, a new child node of N
      constraints_C := constraints_N ∪ {S' = s}
      use calls to ORACLE(constraints_C) to determine if C should remain a leaf, otherwise
        examplesC := members of examples_N with outcome s on split S'
        put ⟨C, examples_C, constraints_C⟩ into Queue
return T
```

2) the maximum size, (i.e., the intervals of activation in each input dimension);
3) final center position of each hypercube; and noncontiguous regions of input space.

However, the authors do not provide an analysis of the complexity of their algorithm.

The authors evaluate the approach on two datasets. The USER dataset (49 patterns, two continuous valued inputs, two output classes) and the pulmonary artery catheter (PAC) dataset (150 patterns, four continuous valued inputs, three output classes). For the USER data the trained MLP achieved 96% accuracy. IA extracted 68 rules (55% accuracy) for a smallest allowable hypercube of 10^{-3} and 947 rules (88% accuracy) for a smallest allowable hypercube of 10^{-5}. From the PAC dataset with 10% "noise" in the training patterns IA extracted 2193 rules (98% accuracy) from the trained MLP. When the noise level was increased to 20% IA extracted 2003 rules (85% accuracy) from the trained MLP. Unfortunately these results tend to suggest that, in its current form, IA appears to achieve high accuracy at the expense of rule comprehensibility.

6) TREPAN—Craven and Shavlik [5]:

Rule Format: Decision tree with M-of-N split tests at the nodes.
Quality: Extracted decision tree shows high accuracy, fidelity, and comprehensibility.
Translucency: Pedagogical.
Complexity: Polynomial in the sample size, the dimensionality of the input space and the maximum number of values for a discrete feature.
Portability: Suitable for any trained/learned model.

The TREPAN algorithm described by Craven and Shavlik [7] and Craven [5] (see Table IV), is a general purpose algorithm for extracting a decision tree from any learned model, (ANN or symbolic). TREPAN uses queries to induce a decision tree that approximates the concept acquired by a given inductive learning method.

TREPAN has been evaluated on the Cleveland Heart Disease database from the University California—Irvine machine learning repository, a variation of the gene promoter recognition problem used by Towell and Shavlik [42], and a problem in which the task was to recognize protein-coding regions in [6]. Analysis of results showed that TREPAN exhibits high fidelity with the model from which its trees are derived, exhibits high predictive accuracy, and is comparable in complexity with other decision trees trained on the same problem, e.g., C4.5 [30] and ID2-of-3 [25]. Craven and Shavlik point out that another significant advantage of this model over other rule extraction techniques is that it scales well to problems of higher dimensionality.

B. Rule Refinement Techniques

The following two techniques illustrate the developments which have occurred in the area of utilizing ANN's for rule refinement. In this process, the initial knowledge about a given problem domain is inserted into an ANN by programming some of the weights. After training the network on the available datasets, a "refined" set of rules is then extracted.

1) TopGen—Opitz and Shavlik [29]:

Rule Format: Propositional/M-of-N rules.
Quality: Better than KBANN (''sparser rule sets'').
Translucency: Decompositional.
Complexity: Dominated by muliple neural-network training
Portability: Specialized neural-network architecute, specialized training algorithm.

TopGen is a newer version of the knowledge-based artificial neural network (KBANN) rule refinement system. Although KBANN has previously been shown to be an effective theory-refinement system, it cannot add new domain knowledge during training. TopGen aims to overcome some of the limitations of KBANN. KBANN is a theory "refinement" system that

1) is capable of pruning an inserted rule set but not capable of adding new rules;
2) directly maps the dependency structure of the inserted rules to the network topology;
3) is largely "topology preserving" to ensure its ability to function as a theory refinement system;
4) assumes that the initial domain theory is basically correct and nearly complete. (KBANN does not handle "impoverished" domain theories well.)

It is also possible that during KBANN training:

1) the original concept may evolve as a distributed representation over several nodes/subtrees;

TABLE V
THE SALIENT POINTS OF THE TOPGEN ALGORITHM

Use KBANN's rule to network algorithm to create a first network topology
Train network and place on OPEN list.
Repeat until stopping criteria
Remove "best" network from OPEN list
Determine N best places to expand topology and generate N such networks
Add each generated network to the OPEN list
Train each network
Prune OPEN list to length M
Return the "best" network

2) the network learns new knowledge but represents this concept as a distributed representation.

Either of these would corrupt the initial domain theory and make rule extraction difficult and is discouraged.

The aim of TopGen is to expand KBANN during training so that it is able to learn the training examples without needlessly corrupting the initial rules. TopGen is a heuristically guided approach to determining the best place to add new nodes to the network. The computational expense is justified in terms of the human expert's willingness to wait for an extended period of time for better predictive accuracy.

Table V shows the salient points of the complete TopGen algorithm [29]. In essence, TopGen decreases false negatives by adding new rules. It decreases false positives by adding new nodes to the network. TopGen also uses a weight decay term which tends to preserve initial knowledge. In particular, weights that represent initial knowledge decay toward their initial value while weights that represent added knowledge decay toward zero.

In summary, the TopGen network which generalizes best on the corresponding validation set is selected as the best network. Opitz and Shavlik recognize that the inductive bias of the algorithm may lead to TopGen selecting networks that overfit the validation set. Opitz and Shavlik provide results in terms of predictive accuracy and number of nodes added to the network however it does not compare the initial and refined rule sets in terms of rule quality criteria such as accuracy, fidelity, comprehensibility, etc. TopGen is important because it overcomes an obvious limitation of KBANN in not being able to extend a relatively weak initial domain theory.

2) Extraction of Finite-State Automata from Recurrent Networks—Giles and Omlin [15]:

Rule Format: DFA/State transition rules.

Quality: Extracted DFA exhibits high accuracy and fidelity.

Translucency: Compositional (i.e., the granularity is at the level of ensembles of neurons rather than individual neurons).

Complexity: Not available. (While no formal results exist on the computational complexity of DFA extraction, the state space of trained recurrent networks tends to be sparsely populated, i.e., only a small subset of all available partitions are generally visited. Thus, the search tree tends to be shallow. This makes DFA extraction "tractable.")

Portability: Portable (The cluster analysis used in this method is independent of the particular network architecture. The only requirements are: 1) a state space and 2) external inputs.)

TABLE VI
GILES AND OMLIN'S METHOD FOR EXTRACTING DFA FROM RECURRENT NETWORKS

1. Divide the output of each of the N state neurons into q intervals, (quantization levels). This results in q^N partitions of the hidden state unit space.
2. Starting in a defined initial network state generate a search tree with the initial state as its root and the number of successors of each node equal to the number of symbols in the input alphabet. (Links between nodes correspond to transitions between DFA states.)
3. Search the tree by presenting all strings up to a certain length in alphabetical order starting with length 1. This search is performed in breadth-first order. Make a path from one partition to another. When:
a) a previously visited partition is reached, then only the new transition is defined between the previous and the current partition, i.e., no new DFA state is created and the search tree is pruned at that node;
b) an input causes a transition immediately to the same partition, then a loop is created and the search tree is pruned at that node.
4. Terminate the search when no new DFA states are created from the string set initially chosen and all possible transitions from all DFA states have been extracted.
5. For each resulting path, if the output of the response neuron is greater than 0.5 the DFA state is accepting; otherwise the DFA state is rejecting.

Work in the area of rule refinement and recurrent networks has centered on the ability of recurrent networks to learn the rules underlying a regular language, (where a regular language is the smallest class of formal languages in the Chomsky hierarchy) [19]. Cleeremans *et al.* [4], Williams and Zipser [44], and Elman [9] showed that recurrent networks were capable of being trained such that the behavior of the trained network emulated a given DFA. Cleeremans *et al.* [4] concluded that the hidden unit activations represented past histories and that clusters of these activations represented the states of the generating automaton. Giles *et al.* [13] extended the work of Cleeremans and described a technique for extracting complete deterministic finite-state automata from second-order dynamically driven recurrent networks and is described in Table VI above.

The networks used by Giles and Omlin [14], [15] have N recurrent hidden units labeled S_j; K special nonrecurrent input units labeled I_k; and $N^2 \times K$ real-valued weights labeled W_{ijk}. The values of the hidden neurons are referred to collectively as state vectors $\mathbf{S}$ in the finite N-dimensional space $[0, 1]^N$. In this context, a second-order recurrent network is taken to mean that the weights W_{ijk} modify a product of the hidden (S_j) and input (I_k) neurons. This allows a direct mapping of $\{\text{state, input}\} \Rightarrow \{\text{next state}\}$ and means the network has the representational potential of at least a finite-state automata [14], [15].

The extracted DFA depends on the following.

1) The quantization level, q, chosen. (While different DFA's will be extracted for different values of q, Giles and Omlin do not state how the quantization parameter q is chosen. However, their overall aim is to extract

DFA's with the smallest possible quantization level which explains the training data.)

2) The order in which strings are presented (which leads to different successors of a node visited by the search tree).

Giles and Omlin [15] state that these distinctions are usually not significant as they employ a minimization strategy [19] which guarantees a unique minimal representation for any extracted DFA. Thus DFA's extracted under different initial conditions may collapse into equivalence classes [13].

Rule insertion for known DFA transitions is achieved by programming some of the initial weights of a second-order recurrent network with N state neurons. The rule insertion algorithm assumes $N > N_s$ where N represents the number of neurons in the network and N_s is the number of states in the DFA to be represented. In the recurrent networks used by Giles and Omlin [14], [15] the network changes state $\mathbf{S}$ at time $t+1$ according to the equations

$$S^{(t+1)} = g(\Xi) \tag{1}$$

$$\Xi i = \sum_{j,k} W_{ijk} S_j^{(t)} I_k^{(t)} \tag{2}$$

where g is a sigmoid discriminant function.

To encode a known transition $\delta(s_j, a_k) = s_i$ Giles and Omlin [15] arbitrarily identify DFA states s_j and s_i with state neurons S_j and S_i, respectively. This transition can be represented by having S_i have a high output, ($\approx$1), and S_i have a low output ($\approx$0), after the input symbol a_k has entered the network via input neuron I_k.

Setting W_{ijk} to a large positive value will ensure that $s_i^{(t+1)}$ will be high, and setting W_{jjk} to a large negative value will ensure that $s_j^{(t+1)}$ will be low. All other weights are set at small random values.

To program the response neuron to indicate whether the resulting DFA state is an accepting or rejecting state the weight W_{0ie} is set large and positive if s_i is an accepting state, and large and negative if s_i is a rejecting state (where e is a special symbol that marks the end of an input string.)

Network training proceeds after all known transitions are inserted into the network by encoding the weights according to the above method. After training, the refined rules/DFA is extracted using the method described above. Giles and Omlin [15] conclude that network initialization reduces training time and improves generalization on the example problems studied.

3) Cascade ARTMAP—Integrating Neural Computation and Symbolic Knowledge Processing—Tan [35]:

Rule Format:	Propositional with discrete input.
Quality:	Good.
Translucency:	Decompositional.
Complexity:	Linear in the number of recognition catagories.
Portability:	Relies on specific architecture, viz., cascade ARTMAP (adaptive resonance theory mapping).

The Cascade ARTMAP architecture, a generalization of the fuzzy ARTMAP architecture [3], and associated rule insertion and extraction procedures allow for the refinement of symbolic knowledge given in propositional rule format.

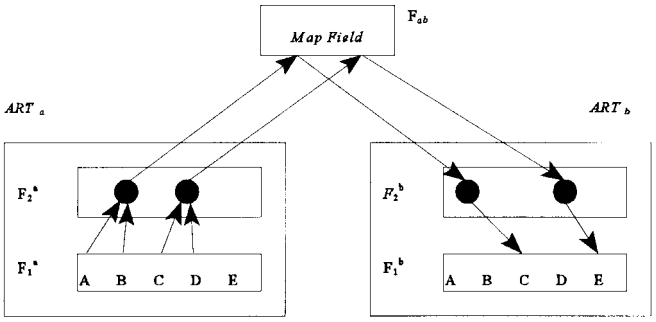

Fig. 3. Cascade ARTMAP representation of initial rule base.

Cascade ARTMAP allows multistep inferencing through the internal representation of intermediate concepts and a forward chaining (rule cascading) inferencing strategy.

Rule insertion proceeds in two phases. In the first phase all rules that are to be used to initialize the network are parsed to derive a *symbol table* where the elements of the symbol table are all input attributes plus the consequents of rules. For instance, consider the simple two-level rule cascade below where A, B, and D are input attributes, C is an intermediate concept, and E is an output attribute.

Rule 1: IF A and B THEN C.

Rule 2: IF C and D THEN E.

This leads to the Cascade ARTMAP representation in Fig. 3.

During learning new recognition categories, (F_2^a nodes) can be added dynamically to supplement the domain knowledge used to initialize the network. The "self stabilizing" property of Cascade ARTMAP ensures that the meanings of units do not change thus preserving the inserted knowledge during training.

Extracting rules after training is a straightforward process. Rules are represented by a combination of nodes in the F_2^a and F_1^b layers. Each committed node in the F_2^a layer represents the antecedent of a rule with the specific antecedent conditions being the symbols in the F_1^a layer that are connected to F_2^a layer node. The rule consequent is the symbol in the F_1^b layer that is connected to the F_2^a layer (antecedent) node. To generate concise rule sets the rule extraction step includes techniques for rule pruning, antecedent pruning, and recognition category pruning.

Tan [35] evaluates Cascade ARTMAP on two problem domains.

1) Animal recognition. The task is to distinguish among seven types of animals based on 21 attributes. The initial knowledge base consists of 15 rules with four intermediate concepts and had a three-level rule inference depth.
2) DNA promoter recognition. The task is to determine whether an input pattern contains a gene promoter. One hundred six input patterns each consisting of 228 attributes and an initial domain theory that classified only half the patterns correctly were used in the experiments.

Results indicate the following.

1) Use of *a priori* symbolic knowledge always improves system performance, especially with a small training set.
2) The Cascade ARTMAP rule refinement algorithm produces performance superior to this of other machine

learning systems including KBANN, ID-3, Backprop MLP, and KNN.

3) Rules extracted from Cascade ARTMAP are more accurate and comprehensible than the N-of-M rules extracted from KBANN networks.

C. Techniques Which Use Existing Knowledge to Initialize an ANN

The following two techniques illustrate the developments which have occurred in the area of using existing knowledge to initialize an ANN i.e., the first step in the total rule-refinement process. Since neither of these two examples include a corresponding rule-extraction process, the classification taxonomy used with the preceding techniques is not applicable.

1) Deterministically Encoding Fuzzy Finite-State Automata Into Recurrent Networks—Omlin et al. [28]: Omlin *et al.* [28] extend the work of Giles and Omlin [14], [15], and Omlin and Giles [27] by showing how a recurrent network can be constructed such that it acts as a fuzzy finite-state automata, (FFA), i.e., a recognizer for a fuzzy regular language. This work is purely about initializing networks to behave as FFA and does not include refinement of the initialized networks through training or extraction of the refined FFA's after training. Future work includes investigating whether FFA's can be learned through training on example strings and how weighted production rules are represented in deterministic trained networks.

In this technique a fuzzy regular grammar, $\tilde{G}$, is defined as $\tilde{G} = \langle S, N, T, P \rangle$ where S is the start symbol, N and T are nonterminal and terminal symbols, respectively, and P are productions of the form $A \xrightarrow{\theta} a$ or $A \xrightarrow{\theta} aB$ where $A, B \in N$, $a \in T$, and $0 < 2 \leq 1$.

Unlike in the case of DFA's, where strings belong or do not belong to some regular language, strings of a fuzzy language have graded membership of the regular language $L(G^*)$.

The encoding algorithm proposed by Omlin *et al.* [28] involves converting FFA's into equivalent DFA's that compute fuzzy string membership. The fuzzy FFA states are transformed into crisp DFA states. A membership label μ_i with $0 < \mu_i \leq 1$ is associated with each accepting DFA state; nonaccepting DFA states have label $\mu_i = 0$. The membership of a string in the regular language $L(G^*)$ is equal to the membership of the last visited DFA state.

The network structure used to encode the FFA's is similar to that used by Giles and Omlin [15] to encode DFA's except that the state neurons are connected to a linear output neuron which computes string membership. The weights of the connections from state neurons to output neuron are set to the membership labels of the DFA states.

Omlin *et al.* [28] conclude that deterministic recurrent networks are computationally rich enough models to represent FFA's and show how FFA's can be encoded in recurrent networks that compute string membership functions with arbitrary accuracy.

2) Initialization of Neural Network by Means of Decision Trees—Ivanova and Kubat [21]: Ivanova and Kubat [21] describe a method for utilizing prior knowledge about the problem domain by initializing a feedforward network with knowledge captured by means of a decision tree (see Table VII). Although this method does not include a method for extracting rules from the final trained network it is considered in this section because it provides an effective technique for incorporating prior rule based knowledge into a neural network.

TABLE VII
INITIALIZING A NEURAL NETWORK BY MEANS OF A DECISION TREE

Use a subset of training patterns to induce a decision tree.
Convert tree → rules
accept the quantization of attribute values produced by decision tree
construct rules as condition ⇒ consequent
condition = conjunct of nodes in path from root to leaf nodes
consequent = class label attached to leaf node
remove redundant rules
for each attribute
construct a quantization interval
attach a label to each quantization interval which represents a Boolean variable
rewrite the rules in terms of these Boolean variables
Convert rules → network connecting only regular links.
Fully interconnect adjacent layers with the additional links set to small initial value ξ.
Calculate weights along regular links so the network approximates the decision tree.
Slightly perturb all weights.
Convert step functions to sigmoids and attribute values into fuzzy membership functions.
Train (using backprop) with all training examples.

The work is based on the following premises.

1) An approximate logical description of the concept (in the form of a decision tree) can be acquired by a machine learning technique and then easily transformed into a neural net topology to be trained by backpropagation.
2) The network can further improve classification accuracy over the decision tree.
3) Threshold units can represent AND, OR, NOT, and M-of-N concepts.
4) Replacement of threshold units with sigmoid activation functions results in more complex modeling behavior.
5) The class of concepts that can be recognized by a neural network is incomparably larger than that for decision trees where each concept is characterized by a DNF expression.

The network produced by the technique has an input layer, a hidden layer of AND threshold logic units and an output layer of OR threshold logic units. Classification is "winner take all" with the pattern classified as an example of the ith concept where I is the index of the OR node with the highest output.

The system has been tested on artificial and real world data sets including plane partitioning into axis parallel and nonaxis parallel regions and sleep stage classification. The system has been tested against C4.5, LVQ, and MLP which learned from *tabula rasa*. Results tend to support the hypotheses that:

1) network training after initialization is better than network learning alone;

2) initialization plus training gives better results than purely symbolic learning (as compared to C4.5)

VI. Research Questions and the Future Directions in Extracting Knowledge from ANN's

A. A Formal Model of Rule Extraction

While rule extraction from trained feedforward ANN's is only one facet of the total problem of extracting the knowledge embedded within ANN's, it is nonetheless an area which continues to attract considerable attention. Golea [18] sees as one of the main challenges in this particular area, the formulation of a consistent theoretical basis for what has been, until recently, a disparate collection of empirical results. To this end he has proposed a formal model for analyzing the rule-extraction problem which is an adaptation of the well-known probably approximately correct (PAC) learning model [43]. Applied in the context of extracting Boolean if–then–else rules, this model provides the basis for proving that extracting simple rules from trained feedforward ANN's is computationally an NP-hard problem. Hence an avenue for further investigation is to determine if this (or a similar) model could be used to analyze for the computational complexity of extracting other forms of rules such as fuzzy rules and probabilistic rules as well as that of extracting FSM's from trained recurrent neural networks. Ultimately one of the overall goals of this process is to determine under what conditions, if any, the knowledge-extraction problem is tractable.

B. Extracting Knowledge in Other Forms

Previous surveys of ANN rule-extraction techniques together with the critiques presented above, illustrate that there is an increasingly rich and diverse range of schemas being employed to represent the knowledge extracted from ANN's. Nonetheless, there remains considerable scope for further exploration and development. One such area, for example, is where ANN's are used in system control and management functions. In particular a distinguishing characteristic of this application of knowledge-extraction techniques is that the output from the ANN is real-valued (continuous) data as distinct from nominal-valued output used in classification problems.

A second such example is in situations where an ANN has been trained on the type of time-series data which characterizes the financial markets. In this case a desired goal of the knowledge-extraction process is an abstract representation of the market's dynamical behavior which can then be used as the basis for decision-making (e.g., whether or not to buy or sell a particular stock).

C. The Use of Prior Knowledge for Network Training and Rule Extraction

"Pedagogical" or learning-based rule-extraction techniques are attractive because they make no assumption about the underlying neural-network architecture and are offered as an alternative to decompositional algorithms [7]. However, more recently, Golea [17] pointed out that the time complexity of Craven and Shavlik [7] algorithm is not polynomial in the worst case. Furthermore, it was shown in [17] that rule-extraction from single perceptrons can be, but is not always, easier than extracting rules from two-layer networks.

Unfortunately, the situation is further complicated as the following discussion explains: a pedagogical rule-extraction algorithm accepts as input a data set D with two subsets: The training data (D1) and the data generated by the neural network via generalization (D2). The latter data set is typically larger and crucial for the evaluation of the pedagogical rule-extraction technique. In order to achieve good "fidelity," the pedagogical rule-extraction method has to learn D2 particularly well to approximate the neural network with high fidelity.

A decompositional algorithm accepts as input the neural network N plus the training data D1, or at least knowledge about the coding scheme used for D1. For instance, decompositional techniques such as Lap require sparse binary coding. This knowledge is used to constrain the search for a set of inputs that can activate a hidden or output unit above threshold irrespective of other inputs.

Lets compare a decompositional method with a symbolic learning algorithm. The symbolic algorithm can be "knowledge-intensive" or "analytical." That is, it can use background knowledge and predefined concepts. If we are aiming at a fair comparison of these techniques, we have to compare the decompositional method with the knowledge-intensive symbolic learning technique because the decompositional algorithm accepts a "representation" as input: the hidden and output units of the trained neural network.

To make this point clearer assume that the task is recognize the object "chair," the symbolic algorithm might use the predicates "edge(pos_1, pos_2)" and "bar(pos_1, pos_2, pos_3, pos_4)" where the "pos" variables can take any value. A system that uses the generic predicates "edge" and "bar" has an advantage because these can easily be combined to form "leg" and "seat" representations which are part of a "chair." The crucial point is that the decompositional algorithm is disadvantaged because the hidden and output unit representations it takes as input are primitive compared to the prior knowledge of the symbolic algorithm. It is important to note that the generic predicates are not only an advantage for learning: The symbolic system can do question answering, i.e., can answer questions like "How many legs has a chair" and "What are the parts of a chair."

From this perspective, a decompositional algorithm is inherently disadvantaged compared to a symbolic induction algorithm. However, there are at least two solutions to the problem: 1) the use of connectionist knowledge representation systems which approximate the "expressiveness" of a symbolic algorithm or 2) modular networks, where one network preprocesses for another network. For instance, one network recognizes "edges" and "bars" and another combines these.

VII. Summary and Conclusion

The preceding survey and associated discussion has highlighted the rich diversity of mechanisms and procedures now available to extract and represent the knowledge embedded within trained ANN's. In addition, the discussion has also

highlighted the effectiveness and utility of such techniques in a broad range of problem domains. While the field of ANN knowledge-extraction is one which continues to attract considerable interest, the preceding discussion has also argued the case for using an established learning model (such as PAC learning) to provide a consistent theoretical basis for what has been, until now, a disparate collection of empirical results. Furthermore, useful work remains to be done to determine under what conditions, if any, the knowledge-extraction problem is tractable.

ACKNOWLEDGMENT

The authors would like to acknowledge the constructive criticisms made by three anonymous reviewers on an earlier version of this paper.

REFERENCES

[1] R. Andrews, J. Diederich, and A. B. Tickle, "A survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-Based Syst.*, vol. 8, no. 6, pp. 373–389, 1995.

[2] R. Andrews, R. Cable, J. Diederich, S. Geva, M. Golea, R. Hayward, C. Ho-Stuart, and A. B. Tickle, "An evaluation and comparison of techniques for extracting and refining rules from artificial neural networks," *QUT NRC*, Feb. 1996.

[3] G. A. Carpenter and A. W. Tan, "Rule extraction: From neural architecture to symbolic representation," *Connection Sci.*, vol. 7, no. 1, pp. 3–27, 1995.

[4] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite-state automata and simple recurrent networks," *Neural Comput.*, vol. 1, no. 3, pp. 372–381, 1989.

[5] M. Craven, "Extracting comprehensible models from trained neural networks," Ph.D. dissertation, Univ. Wisconsin, Madison, WI, 1996.

[6] M. W. Craven and J. W. Shavlik, "Learning to predict reading frames in E. coli sequences," in *Proc. 26th Hawaii Int. Conf. Syst. Sci.*, Wailea, HI, 1993, pp. 773–782.

[7] ———, "Using sampling and queries to extract rules from trained neural networks, machine learning," in *Proc. 11th Int. Conf.*, Amherst, MA, 1994, pp. 73–80.

[8] R. Durbin and D. Rumelhart, "Product units: A computationally powerful and biologically plausible extension," *Neural Comput.*, vol. 1, no. 1, pp. 133–166, 1989.

[9] J. L. Elman, "Finding structure in time," *Cognitive Sci.*, vol. 14, pp. 179–211, 1990.

[10] R. Filer, I. Sethi, and J. Austin, "A comparison between two rule extraction methods for continuous input data," in *Proc. NIPS'97 Rule Extraction From Trained Artificial Neural Networks Wkshp.*, Queensland Univ. Technol., 1996, pp. 38–45.

[11] L. M. Fu, "Rule learning by searching on adapted nets," in *Proc. 9th Nat. Conf. Artificial Intell.*, Anaheim, CA, 1991, pp. 590–595.

[12] ———, "Rule generation from neural networks," *IEEE Trans. Syst., Man, Cybern.*, vol. 28, pp. 1114–1124, 1994.

[13] C. L. Giles, C. B. Miller, D. Chen, H. Chen, G. Z. Sun, and Y. C. Lee, "Learning and extracting finite-state automata with second-order recurrent neural networks," *Neural Comput.*, vol. 4, pp. 393–405, 1992.

[14] C. L. Giles and C. W. Omlin, "Rule refinement with recurrent neural networks," in *Proc. IEEE Int. Conf. Neural Networks*, San Francisco, CA, Mar. 1993, pp. 801–806.

[15] ———, "Extraction, insertion, and refinement of symbolic rules in dynamically driven recurrent networks," *Connection Sci.*, vol. 5, nos. 3/4, pp. 307–328, 1993.

[16] ———, "Rule revision with recurrent networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 8, p. 183, 1996.

[17] M. Golea, "On the complexity of rule extraction from neural networks and network querying," in *Proc. Rule Extraction From Trained Artificial Neural Networks Wkshp.*, Society for the Study of Artificial Intelligence and Simulation of Behavior Workshop Series (AISB'96), Univ. Sussex, Brighton, U.K., Apr. 1996, pp. 51–59.

[18] ———, "On the complexity of extracting simple rules from trained neural nets," 1997, to appear.

[19] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley, 1979.

[20] M. A. Hume, *The Happy Way to Reading.* London, U.K.: Blackie & Son, 1950.

[21] I. Ivanova and M. Kubat, "Initialization of neural networks by means of decision trees," *Knowledge-Based Syst.*, vol. 8, no. 6, pp. 333–344, 1995.

[22] R. Krishnan, "A systematic method for decompositional rule extraction from neural networks," in *Proc. NIPS'97 Rule Extraction From Trained Artificial Neural Networks Wkshp.*, Queensland Univ. Technol., 1996, pp. 38–45.

[23] H. Liu and S. T. Tan, "X2R: A fast rule generator," in *Proc. IEEE Conf. Syst., Man, Cybern.* New York: IEEE Press, 1995, pp. 1631–1635.

[24] F. Maire, "A partial order for the M-of-N rule extraction algorithm," *IEEE Trans. Neural Networks*, vol. 8, pp. 1542–1544, Nov. 1997.

[25] P. M. Murphy and M. J. Pazzani, "ID2-of-3: Constructive induction of M-of-N concepts for discriminators in decision trees," in *Proc. 8th Int. Machine Learning Wkshp.*, Evanston, IL, 1991, pp. 183–187.

[26] C. W. Omlin and C. L. Giles, "Extraction of rules from discrete-time recurrent neural networks," *Neural Networks*, vol. 9, no. 1, pp. 41–52, 1996.

[27] ———, "Constructing deterministic finite-state automata in recurrent neural networks," *J. ACM*, vol. 43, no. 6, pp. 937–972, 1996.

[28] C. W. Omlin, K. Thornber, and C. L. Giles, "Fuzzy finite-state automata can be deterministically encoded into recurrent neural networks," *IEEE Trans. Fuzzy Syst.*, 1998, to be published.

[29] D. W. Optiz and J. W. Shavlik, "Dynamically adding symbolically meaningful nodes to knowledge-based neural networks," *Knowledge-Based Syst.*, vol. 8, no. 6, pp. 301–311, 1995.

[30] R. Quinlan, *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan-Kaufmann, 1993.

[31] J. Rissanen, *Stochastic Complexity in Statistical Inquiry.* Singapore: World, 1989.

[32] K. Saito and R. Nakano, "Law discovery using neural networks," in *Proc. NIPS'96 Rule Extraction From Trained Artificial Neural Networks Wkshp.*, Queensland Univ. Technol., 1996, pp. 62–69.

[33] I. Schellhammer, J. Diederich, M. Towsey, and C. Brugman, "Knowledge extraction and recurrent neural networks: An analysis of an Elman network trained on a natural language learning task," QUT-NRC Tech. Rep. 97-IS1, 1997.

[34] R. Setiono, "Extracting rules from neural networks by pruning and hidden unit splitting," *Neural Comput.*, vol. 9, pp. 205–225, 1997.

[35] A. W. Tan, "Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing," in *IEEE Trans. Neural Networks*, vol. 8, pp. 23–250, 1997.

[36] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, K. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang, "The MONK's problems: A performance comparison of different learning algorithms," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-91-197, Dec. 1991.

[37] S. B. Thrun, "Extracting provably correct rules from artificial neural networks," Inst. Informatik III, Univ. Bonn, Germany, Tech. Rep. IAI-TR-93-5, 1994.

[38] A. B. Tickle, R. Hayward, and J. Diederich, "Recent developments in techniques for extracting rules from trained artificial neural networks," *Herbstschule Konnektionismus (HeKonn'96)*, Münster, Germany, Oct. 1996.

[39] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich, "Rule extraction from trained artificial neural networks," *Neural Network Analysis, Architectures, and Applications*, A. Browne, Ed. Bristol, U.K.: Inst. Phys. Publishing, 1997, pp. 61–99.

[40] A. B. Tickle, M. Golea, R. Hayward, and J. Diederich, "The truth is in there: Current issues in extracting rules from trained feedforward artificial neural networks," in *Proc. 1997 IEEE Int. Conf. Neural Networks (ICNN'97)*, vol. 4, Houston, TX, June 1997, pp. 2530–2534.

[41] A. B. Tickle, "Machine learning, neural networks and information security: Techniques for extracting rules from trained feedforward artificial neural networks and their application in an information security problem domain," Ph.D. dissertation, 1997.

[42] G. Towell and J. Shavlik, "The extraction of refined rules from knowledge-based neural networks," *Machine Learning*, vol. 131, pp. 71–101, 1993.

[43] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[44] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, pp. 270–280, 1989.

Alan B. Tickle received the B.Sc. and M.Sc degrees from the University of Queensland, Australia. He received the Ph.D. degree from Queensland University of Technology, Australia.

His research interests include application of machine learning and data mining techniques in the area of communications network security.

Robert Andrews received the M.I.T. degree from Queensland University of Technology, Australia, in 1995 and is currently completing work toward the Ph.D. degree.

His research interests include knowledge extraction from trained artificial neural networks.

Mostefa Golea received the B.Sc degree from the University of Constantine, Algeria, in 1986, and the M.Sc and Ph.D. degrees from the University of Ottawa, Canada, in 1989, and 1993, respectively.

In 1994 he was an Invited Research Fellow at Fujitsu Labs, Japan, and then a Research Fellow at Queensland University of Technology from 1995 to 1996 and the Institute of Advanced Studies, Australian National University, from 1996 to 1997. His research interests include computational learning theory, neural networks, the sample and time complexity of learning systems, applications of machine learning, and object-oriented design and analysis.

Joachim Diederich received the master's degree in psychology form the University of Muenster, Germany, in 1983, the Ph.D. degree in computational linguistics from the University of Bielefeld, Germany, in 1985, and the habilitaion degree in computer science from the University of Hamburg, Germany, in 1995.

Prior to joining Queensland University of Technology, Australia, he was with the German National Research Center for Information Technology (GMD), St. Augustin, Germany, and the International Computer Science Institute (ICSI), Berkeley, CA.