# Constructive Theory Refinement in Knowledge Based Neural Networks


Rajesh Parekh & Vasant Honavar
Artificial Intelligence Research Group
Department of Computer Science
226 Atanasoff Hall
Iowa State University
Ames IA 50011. U.S.A.
{parekh|honavar}@cs.iastate.edu



*Abstract*— **Knowledge based artificial neural networks offer an approach for connectionist theory refinement. We present an algorithm for refining and extending the domain theory incorporated in a knowledge based neural network using constructive neural network learning algorithms. The initial domain theory comprising of propositional rules is translated into a knowledge based network of threshold logic units (threshold neuron). The domain theory is modified by dynamically adding neurons to the existing network. A constructive neural network learning algorithm is used to add and train these additional neurons using a sequence of labeled examples. We propose a novel hybrid constructive learning algorithm based on the Tiling and Pyramid constructive learning algorithms that allows knowledge based neural network to handle patterns with continuous valued attributes. Results of experiments on two non-trivial tasks (the *ribosome binding site prediction* and the *financial advisor*) show that our algorithm compares favorably with other algorithms for connectionist theory refinement both in terms of generalization accuracy and network size.**


## I. Introduction

INDUCTIVE learning systems [1], [2], [3] attempt to learn concept descriptions (or knowledge) from a sequence of labeled examples. The use of domain specific knowledge or a domain theory about the concept being learned (when such domain knowledge is available) can potentially enhance the performance of the inductive learning system. Hybrid learning systems that exploit domain knowledge in inductive learning can potentially outperform those that rely entirely on data-driven induction from examples. In many practical applications of interest (e.g., scientific knowledge discovery), the available domain knowledge is often *incomplete* or even *inaccurate*. Against this background, data-driven theory refinement systems that extend or modify domain knowledge are clearly of interest.

Artificial neural networks [1], [4], because of their potential for parallelism and noise tolerance, offer an attractive approach for automated knowledge acquisition. In particular, constructive neural network learning algorithms, [5], [6], [7], [8] since they obviate the need for an *ad-hoc*, *a-priori* choice of the network topology, but instead adaptively recruit neurons as needed to improve classification accuracy, lend themselves well to incorporation of prior knowledge into the learning process. The domain theory can be translated into an initial network topology and new rules can be incorporated and inaccuracies in the existing rules (if any) can be corrected by dynamically adding new neurons to the network. These new neurons can be trained using a sequence of labeled examples. This process is illustrated in Fig. 1.

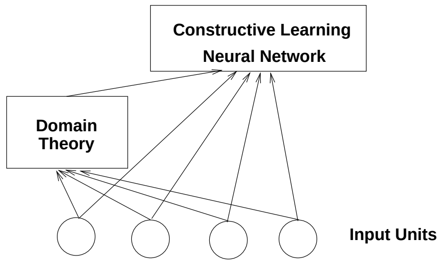


Fig. 1. **Constructive Learning in Knowledge Based Networks.**

Against this background we discuss a constructive approach to theory theory refinement using knowledge based neural networks. The rest of this paper is organized as follows: Section II outlines the process of incorporating the domain theory into the initial network topology and the constructive learning algorithm that is used to dynamically grow the knowledge based network. It describes a new hybrid Tiling-Pyramid constructive learning algorithm that uses an adaptive vector quantization based on the tiling algorithm [7], [9] in conjunction with the multi-category pyramid learning algorithm [7]. Section III presents the results of our experiments with the *Financial Advisor Rule Base* [10], [11] and two datasets from the Human Genome Project (*ribosome binding sites* and *DNA promoter sequences*). [1] Section IV concludes with a summary of the paper, a brief discussion of related work, and an outline of some promising directions for future research.

Rajesh Parekh is currently with the Allstate Research and Planning Center in Menlo Park, California.


Honavar's research was partially supported through grants from National Science Foundation (NSF IRI-9409580) and the John Deere Foundation.


[1] These datasets are available at the University of Wisconsin Madison at ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/datasets/.

## II. Constructive Theory Refinement Using a Knowledge-Based Neural Network

### A. Embedding the Domain Theory in the Neural Network

We use a symbolic knowledge encoding procedure to translate a domain theory in the form of propositional rules into a network of threshold neurons using *rules-to-networks* algorithm of Towell and Shavlik [11], [12]. The procedure involves rewriting the rules into a format that highlights the hierarchical structure of the domain theory. In particular the disjuncts are expressed as a set of rules that each have only one antecedent. This modified set of rules can be mapped to an *AND-OR* graph which in turn can be directly translated into a network of threshold neurons.

Consider for example the following propositional rules of a domain theory.

$$
\begin{aligned}
A &:- \quad B, C, D \\
A &:- \quad D, \neg E
\end{aligned}
$$

The rules are rewritten in the following format.

$$
\begin{aligned}
A &:- \quad A^{'} \\
A^{'} &:- \quad B, C, D \\
A &:- \quad A^{''} \\
A^{''} &:- \quad D, \neg E
\end{aligned}
$$

The *AND-OR* graph corresponding to the modified set of rules is shown in Fig. 2. The equivalent network of bipolar threshold neurons (with outputs 1 and $-1$) is shown in Fig. 3. Note that the threshold neurons $A^{'}$ and $A^{''}$ implement the logical *AND* function and the threshold neuron $A$ implements the logical *OR* function.

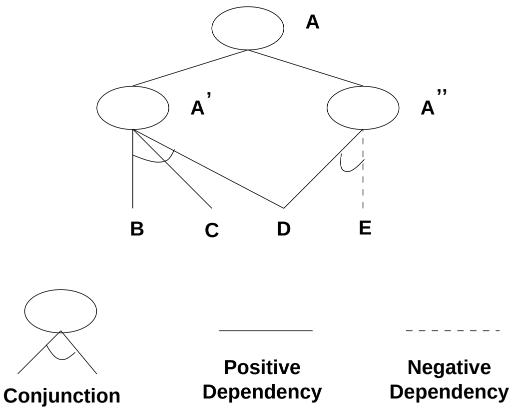


Fig. 2. *AND-OR* **Graph Representation of Knowledge Rules.**

Further, the magnitudes of the connection weights can be set to appropriately to satisfy the different propositional rules as shown in Fig. 4. Note that this threshold neuron implements the propositional rule "*if* $a - 4b > 6$ *then* $c$".

Using the approach outlined above, the initial neural network topology corresponding to the simple financial advisor rule base (due to [10]) of Table I is shown in Fig. 5. Each threshold neuron in the network computes a bipolar hardlimiting function (i.e., the threshold neuron's outputs are 1 and $-1$) of the weighted sum of its inputs. The neurons in the first hidden layer encode the rules 6–9 of the rule base. The only threshold neuron in the second hidden layer computes the logical *and* function and encodes rule 4 and so on.

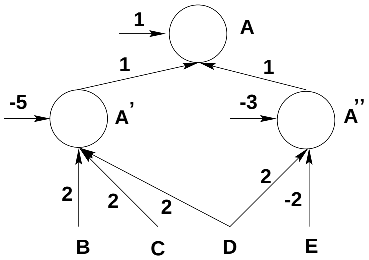


Fig. 3. **Neural Network Implementation of Knowledge Rules.**

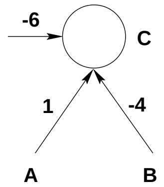


Fig. 4. **Threshold Neuron Implementing an** *If-Then* **Propositional Rule.**

TABLE I

**Financial Advisor Rule Base.**

| | | | |
|---|---|---|---|
| 1 | if (sav_adeq and inc_adeq) | then | invest_stocks |
| 2 | if dep_sav_adeq | then | sav_adeq |
| 3 | if assets_hi | then | sav_adeq |
| 4 | if (dep_inc_adeq and earn_steady) | then | inc_adeq |
| 5 | if debt_lo | then | inc_adeq |
| 6 | if (sav $\geq$ dep * 5000) | then | dep_sav_adeq |
| 7 | if (assets $\geq$ income * 10) | then | assets_hi |
| 8 | if (income $\geq$ 25000 + dep * 4000) | then | dep_inc_adeq |
| 9 | if (debt_pmt $<$ income * 0.3) | then | debt_lo |

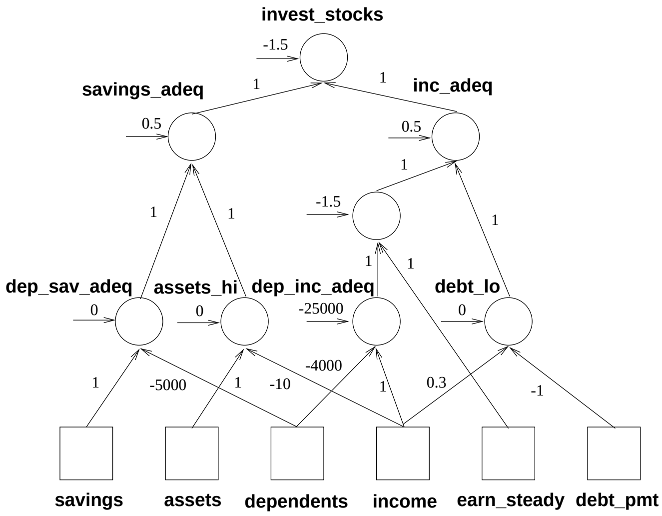


Fig. 5. **Embedding the Financial Advisor Domain Theory in a Neural Network.**

### B. A Hybrid Constructive Learning Algorithm

The proposed approach to data-driven, knowledge-guided theory refinement uses a constructive neural network learning algorithm [6] to augment the domain theory that is encoded by the initial network topology. It is possible to use any of the constructive neural network learning algorithms described in [7] for this purpose. However, this paper focuses on a hybrid algorithm that combines the features of the multi-category versions [7], [6], [9] of the Tiling [13] and the Pyramid [4] algorithms.

The Pyramid algorithm successively adds new layers of threshold neurons to the network. Each newly added layer becomes the network's new output layer. The neurons of the new layer are connected to the input layer and all previously added layers of the network. The convergence of the Pyramid algorithm is proved by demonstrating that the total classification error in the training set decreases with each added layer. The Pyramid algorithm as originally proposed by Gallant [4] assumes the patterns to be binary or bipolar. The extension of the Pyramid algorithm to handle real valued attributes requires a pre-processing of the pattern set. The individual patterns could be *normalized* or *projected* onto a parabolic surface, or discretized to guarantee convergence [7].

The algorithm proposed in this paper relies on discretization of continuous valued attributes. Discretization (or quantization) involves a mapping of the pattern space $\mathcal{R}^N$ to an equivalent bipolar (or binary) representation $\{-1, 1\}^{N'}$ (typically, $N' > N$). Quantization algorithms e.g., the Learning Vector Quantizer (LVQ) [14], are often used in conjunction with supervised machine learning algorithms. The interested reader is referred to [15] for a survey of several approaches to quantization.

We use an adaptive quantization algorithm that dynamically constructs an appropriate sized layer of threshold neurons to perform vector quantization which is based on the MTiling algorithm [7], [9]. MTiling, is a multicategory version of the Tiling algorithm [13]. It trains a group of $M$ master neurons (where $M$ is the number of output classes) using a perceptron style learning algorithm. If the master neuron can not correctly classify all training patterns then ancillary neurons are added to the layer and trained to achieve a *faithful representation* of the pattern set. A faithful representation of the pattern has the property that no two patterns belonging to different output classes have the same representation. It is easy to see that the MTiling algorithm can be used as a *vector quantizer*. The output of each threshold neuron is either 1 or $-1$ and the output vector (combined outputs of the $M$ master neurons and the $K$ ancillary neurons) can be treated as a code book vector representation of the input pattern. Note that the quantized outputs provide a faithful representation of the training patterns Thus, the MTiling algorithm can be used as an efficient adaptive vector quantization algorithm. In particular, a single tiling layer can be constructed using the set of training patterns and the output of this layer in response to each pattern can be treated as the quantized representation of that pattern. The quantized dataset can then be used to construct a network using tower, pyramid, or any other constructive learning algorithm.

The Tiling-Pyramid algorithm proposed in this paper for constructive theory refinement uses the MTiling algorithm to generate a quantized representation of the training patterns and trains a multi-category pyramid or the MPyramid algorithm to construct a neural network for pattern classification (Fig. 6). This choice was motivated by the results of experiments [16] that compared the Tiling-Pyramid and Tiling-Cascade learning algorithms with MPyramid, MCascade, and MTiling. The experiments showed that Tiling-Pyramid and Tiling-Cascade significantly outperformed MPyramid and MCascade algorithms on many real-world datasets.

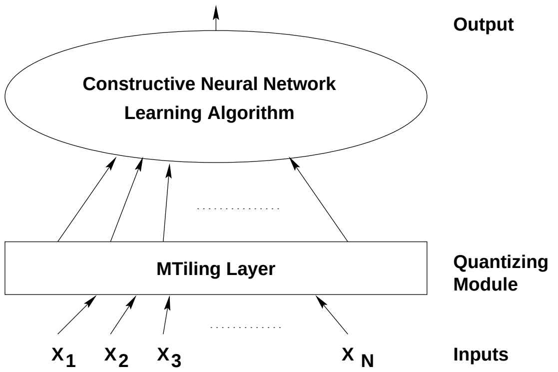


Fig. 6. **Block Diagram of a Hybrid Constructive Network.**

## III. Experimental Results

We report the results of experiments on the *ribosome binding sites* and the *E. coli promoter sequence* datasets from the Human Genome Project and the *financial advisor* rule base. The former two datasets comprise of an imperfect domain theory and a set of labeled examples. In these domains the input is a short segment of DNA nucleotides and the goal is to learn to predict whether these DNA segments contain an important site (such as a ribosome binding site or a promoter) or not. The ribosome binding set's domain theory contains 17 rules. Additionally, there are 1877 labeled training examples. The promoter dataset contains a set of 31 rules and 936 labeled training examples. The financial advisor rule base is shown in Table I. Following the procedure used by Fletcher and Obradović [11], a set of 5500 labeled examples (500 for training and 5000 for testing) are randomly generated using the financial advisor rule base.

We used the Tiling-Pyramid constructive learning algorithm described in section II to augment the initial domain knowledge. The hybrid network was trained using the thermal perceptron [17]. Each threshold neuron was trained for 500 epochs with the initial weights chosen randomly between $-1$ and 1, the learning rate $\eta$ held constant at 1 and the initial temperature $T_0$ set to 1.0. The network was allowed to train to zero classification errors and the network size and generalization accuracy on the test set were recorded.

### *A. Human Genome Project Datasets*

We performed ten-fold cross validation based training on the ribosome binding set and the promoter datasets using exactly the same folds used in the experiments performed

TABLE II
EXPERIMENTS WITH THE RIBOSOME DATASET.

| Rules alone | Tiling-Pyramid | | TopGen | | REGENT | |
|---|---|---|---|---|---|---|
| *Test %* | *Test %* | *Size* | *Test %* | *Size* | *Test %* | *Size* |
| 87.3 ± 2.0 | 90.3 ± 1.8 | 23 ± 0.0 | 90.9 | 42.1 ± 9.3 | 91.8 | 70.1 ± 25.1 |

TABLE III
EXPERIMENTS WITH THE PROMOTERS DATASET.

| Rules alone | Tiling-Pyramid | | TopGen | | REGENT | |
|---|---|---|---|---|---|---|
| *Test %* | *Test %* | *Size* | *Test %* | *Size* | *Test %* | *Size* |
| 77.5 ± 5.0 | 96.3 ± 1.8 | 34 ± 0.0 | 94.8 | 40.2 ± 3.3 | 95.8 | 74.9 ± 38.9 |

by Opitz and Shavlik[2]. On each of the ten runs we first recorded the training and test accuracy of the original network representing the domain theory. The domain theory was refined using the hybrid Tiling-Pyramid algorithm to add dynamically add new threshold neurons to the network. Training was continued until the network attained 100% accuracy on the training set. The network size and generalization accuracy were then measured. We report the average generalization accuracy and the average network size (along with the standard deviations where available) over the ten runs for the ribosome dataset in Table II and for the promoter dataset in Table III.

We see from the results shown in the Tables II and III that the Tiling-Pyramid based constructive theory refinement method generalizes well from the labeled examples. The generalization performance of the refined domain theory (represented by the trained neural network) is significantly better than that of the original set of rules. Furthermore, our approach compares favorably with TopGen and REGENT [18] on both the datasets. In terms of generalization accuracy Tiling-Pyramid performs slightly worse than TopGen and REGENT on the ribosome dataset and slightly better on the promoter dataset. Our approach trains a single network as against a pool networks evaluated by TopGen and REGENT. As a result, our experiments take only about 10 minutes of CPU time on the ribosome binding site dataset and less than 2 minutes of CPU time on the promoter dataset. When compared with the training times for TopGen and REGENT (which are reported to be several days of CPU time [18]) we see that Tiling-Pyramid offers a significant advantage over TopGen and REGENT. (However, we must note that TopGen and REGENT appear to have been designed to exploit available computing resources and come up with hypotheses that have good generalization performance. Computational efficiency appears to not have been a major focus of their design).

A comparison of the number of hidden nodes in the resulting networks constructed by the Tiling-Pyramid, TopGen, and REGENT shows that Tiling-Pyramid is able to come up with considerably smaller networks as compared to TopGen and REGENT. Smaller networks tend to generalize better under otherwise similar scenarios [1].

### *B. Financial Advisor Dataset*

As described earlier the financial advisor dataset comprises of 5500 patterns that are generated at random to satisfy the rules mentioned in Table I. Incomplete domain knowledge was modeled by pruning certain rules and their antecedents from the original rule base (as described in [11]). For example, if *sav_adeq* was selected as the pruning point, then the rules for *sav_adeq*, *dep_sav_adeq*, and *assets_hi* are eliminated from the rule base. In other words rules 2, 3, 6, and 7 are pruned. Furthermore, rule 1 is modified to read *"if (inc_adeq) then invest_stocks"*. The initial network is constructed from this modified rule base and is then augmented using constructive learning. Our experiments follow those performed by Fletcher and Obradović [11]. In Table IV we summarize the average generalization (on the 5000 test patterns) and the average network size over 25 runs for 6 different pruning points. The generalization accuracy of the corresponding network prior to the theory refinement (i.e., based on rules alone) is also reported. In all except for the *assets_hi* rule we see a substantial increase in generalization accuracy after theory refinement. Since the generalization accuracy of the network without the *assets_hi* rule is already significantly high, the slight worsening in generalization performance after refining the theory can be attributed to over-fitting. In Table V we show the results for the experiments with the HDE algorithm that were reported by Fletcher and Obradović[3] [11].

The experimental results (in Tables IV and V) demonstrate that the performance of the hybrid Tiling-Pyramid algorithm compares favorably with that of the HDE algorithm on the financial advisor rule base in terms of both the generalization accuracy and the final size of the trained network.

[2] These folds are available with the dataset.

[3] Note that the standard deviations for the results with these experiments were not available.

TABLE IV

Financial Advisor Rule Base (*Tiling-Pyramid*).

| Pruning point | Tiling-Pyramid | | Rules alone |
|---|---|---|---|
| | Test % | Size | Test % |
| dep_sav_adeq | 92.1 ± 2.03 | 23.3 ± 3.71 | 52.4 |
| assets_hi | 98.7 ± 0.42 | 10 ± 0.0 | 99.4 |
| dep_inc_adeq | 87 ± 2.39 | 30.2 ± 3.57 | 67.4 |
| debt_lo | 94 ± 1.32 | 22.9 ± 2.36 | 81.2 |
| sav_adeq | 92.7 ± 1.57 | 21.1 ± 3.69 | 87.6 |
| inc_adeq | 84.5 ± 2.48 | 31.3 ± 3.53 | 69.4 |

TABLE V

Financial Advisor Rule Base (HDE).

| Pruning point | HDE | | Rules alone |
|---|---|---|---|
| | Test % | Hidden Units Constructed | Test % |
| dep_sav_adeq | 92.7 | 31 | 75.1 |
| assets_hi | 92.4 | 23 | 93.4 |
| dep_inc_adeq | 85.8 | 25 | 84.5 |
| debt_lo | 84.7 | 30 | 61.7 |
| sav_adeq | 92.2 | 19 | 90.9 |
| inc_adeq | 81.2 | 32 | 64.6 |

## IV. Discussion

Theory refinement systems can be broadly classified into two categories: *symbolic* and *neural network* based approaches. Some symbolic theory refinement algorithms use decision tree induction for theory revision. Examples of such systems include RTLS [19], EITHER [20], PTR [21], and TGCI [22]. Some symbolic theory refinement algorithms use *Inductive Logic Programming* (ILP) based techniques [23]. Examples of such systems which use first order predicate logic for knowledge representation include FOCL [24] and FORTE [25].

Towell and Shavlik proposed the KBANN (knowledge based artificial neural network) learning algorithm for connectionist theory refinement [12], [26]. KBANN uses a *rules-to-network* algorithm to construct an *AND-OR* graph representation of the initial domain knowledge and translates this graph to an appropriate neural network topology. KBANN then uses the standard backpropagation learning algorithm [4] to refine the domain knowledge. The approaches described by Fu [27] and Katz [28] are similar to the KBANN algorithm. The KBANN learning algorithm is demonstrated to perform better than several other machine learning algorithms on domains such as promoter and splice-junction datasets [12], [26]. However, KBANN is limited by the fact that it does not modify the network topology. The TopGen [29] and REGENT [18] algorithms were designed to add new neurons to the KBANN network thereby extending the realm of network topologies considered by KBANN. TopGen heuristically determines effective places in the network where new nodes might be added (based on measured generalization performance) whereas REGENT uses a genetic algorithm to search the space of neural network architectures, guided by a fitness function which is based on observed generalization performance. TopGen and REGENT reportedly outperform KBANN in terms of generalization performance on several problems [18], [29].

The generalization performance of Tiling-Pyramid compares very favorably with that of TopGen and REGENT on both the ribosome binding site and promoter datasets. The hybrid Tiling-Pyramid algorithm generates significantly smaller networks as compared to both TopGen and REGENT. The Tiling-Pyramid algorithm proposed in this paper is considerably simpler than both TopGen and REGENT. We construct a single network of threshold neurons as opposed to a population of networks explored by TopGen and REGENT. Consequently, Tiling-Pyramid is orders of magnitude faster than TopGen and REGENT as demonstrated by our experiments.

TopGen and REGENT allow weight changes to occur in the part of the network that incorporates the original domain theory, thereby risking the possibility that the weight updates wipe out the initial domain theory, especially in cases where the examples are drawn from a subset of the entire domain. On the other hand, theory revision in Tiling-Pyramid is performed by constructively adding new neurons to the initial network. The weights of the neurons that encode the initial domain theory are not altered in this process. This has the added advantage of potentially simplifying the task of knowledge extraction and analysis (e.g., identifying the differences between the initial theory and the modified theory).

Fletcher and Obradović's algorithm starts with an initial network representing the domain theory and modifies this theory by training a single hidden layer of threshold neurons using the labeled training data [11]. It uses the *hyperplane detection* from *examples* (HDE) algorithm [30] to construct the hidden layer. Each hidden neuron corresponds to a hyperplane. Fletcher and Obradović's algorithm maps these hyperplanes to a set of threshold neurons and then then trains the output neuron using the pocket algorithm [4]. Our approach is similar to the one taken by Fletcher and Obradović. Instead of constructing a single hidden layer Tiling-Pyramid builds a network of one or more hidden layers (if necessary) on top of the initial network representing the domain theory (see Fig. 1). This provides a more general framework for incorporating domain knowledge into any constructive neural network learning algorithm. Furthermore, the hybrid approach used in Tiling-Pyramid lends itself to the use of other network construction algorithms in place of the Pyramid algorithm. Since the performance of different constructive learning algorithms often differs quite significantly for different datasets [7], this flexibility is likely to be of use in practice.

Experimental results demonstrate that the performance of the Tiling-Pyramid algorithm compares favorably with those reported by Fletcher and Obradović on the financial advisor rule base in terms of both the generalization accuracy and the network size. Furthermore, the generalization accuracy after data-driven theory refinement are significantly superior to that of the imperfect domain theory.

In summary, the Tiling-Pyramid algorithm presented in this paper demonstrates the effectiveness of constructive neural network algorithms in data-driven theory refinement. More systematic experimental evaluation of the Tiling-Pyramid algorithm on a broad range of knowledge discovery tasks such as plant genome analysis and carcinogenecity prediction is in progress. We conclude by identifying some interesting directions for future research.

Constructive neural network learning algorithms typically train the network until the number of errors on the training set is reduced to zero. This can lead to over-fitting of the training data which in turn can hurt the generalization performance of the network. It is of interest to study the performance of the Tiling-Pyramid algorithm (and other constructive learning algorithms) when the network's generalization performance on a *hold-out* set of examples is used to determine when to stop training.

Extraction of rules from the trained neural networks is of significant interest in datamining and knowledge discovery applications [31], [32], [33]. We have not yet explored rule extraction from Tiling-Pyramid networks. We conjecture that such networks lend themselves well to knowledge extraction. In particular, the use threshold neurons permits more straightforward rule extraction than in the case of *sigmoid* neurons that are typically used in backpropagation type algorithms. The fact that Tiling-Pyramid algorithm *builds on* the initial domain theory is likely to enhance the comprehensibility of changes to the domain theory introduced by the data-driven theory refinement process.

The type of rules that can be currently incorporated into the Tiling-Pyramid network is limited to propositional rules. This is a limitation that our approach shares with most neural network based approaches to theory refinement. Also missing from Tiling-Pyramid are mechanisms for quantifying uncertainty associated with the rules. These and related issues merit further investigation.

## References


[1] T. Mitchell, *Machine Learning*, McGraw Hill, New York, 1997.

[2] P. Langley, *Elements of Machine Learning*, Morgan Kaufmann, Palo Alto, CA, 1995.

[3] V. Honavar, "Machine learning: Principles and applications," in *Encyclopedia of Electrical and Electronics Engineering*, J. Webster, Ed. Wiley, New York, 1998, To appear.

[4] S. Gallant, *Neural Network Learning and Expert Systems*, MIT Press, Cambridge, MA, 1993.

[5] V. Honavar and L. Uhr, "Generative learning structures and processes for connectionist networks," *Information Sciences*, vol. 70, pp. 75–108, 1993.

[6] V. Honavar, "Structural learning," in *Encyclopedia of Electrical and Electronics Engineering*, J. Webster, Ed. Wiley, New York, 1998, To appear.

[7] R. G. Parekh, J. Yang, and V. G. Honavar, "Constructive neural network learning algorithms for multi-category real-valued pattern classification," Tech. Rep. ISU-CS-TR97-06, Department of Computer Science, Iowa State University, 1997, (Submitted).

[8] J. Yang, R. Parekh, and V. Honavar, "DistAl: An inter-pattern distance-based constructive learning algorithm," Tech. Rep. ISU-CS-TR 97-05, Iowa State University, 1997.

[9] J. Yang, R. Parekh, and V. Honavar, "MTiling - a constructive neural network learning algorithm for multi-category pattern classification," in *Proceedings of the World Congress on Neural Networks'96*, San Diego, 1996, pp. 182–187.

[10] G. F. Luger and W. A. Stubblefield, *Artificial Intelligence and the Design of Expert Systems*, Benjamin/Cummings, Redwood City, CA, 1989.

[11] J. Fletcher and Z. Obradović, "Combining prior symbolic knowledge and constructive neural network learning," *Connection Science*, vol. 5, no. 3,4, pp. 365–375, 1993.

[12] G. G. Towell, J. W. Shavlik, and M. O. Noordwier, "Refinement of approximate domain theories by knowledge-based neural networks," in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990, pp. 861–866.

[13] M. Mézard and J. Nadal, "Learning feed-forward networks: The tiling algorithm," *J. Phys. A: Math. Gen.*, vol. 22, pp. 2191–2203, 1989.

[14] T Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, Germany, 1989.

[15] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," in *Proceedings of the Twelfth International Conference on Machine Learning*, San Fransisco, CA, 1995, pp. 194–202.

[16] R. Parekh, *Constructive learning: Inducing grammars and neural networks*, Ph.D. thesis, Department of Computer Science, Iowa State University, Ames, IA, 1998.

[17] M. Frean, "A thermal perceptron learning rule," *Neural Computation*, vol. 4, pp. 946–957, 1992.

[18] D. W. Opitz and J. W. Shavlik, "Connectionist theory refinement: Genetically searching the space of network topologies," *Journal of Artificial Intelligence Research*, vol. 6, pp. 177–209, 1997.

[19] A. Ginsberg, "Theory reduction, theory revision, and retranslation," in *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990, pp. 777–782, AAAI/MIT Press.

[20] D. Ourston and R. J. Mooney, "Theory refinement: Combining analytical and empirical methods," *Artificial Intelligence*, vol. 66, pp. 273–310, 1994.

[21] M. Kopel, R. Feldman, and A. Serge, "Bias-driven revision of logical domain theories," *Journal of Artificial Intelligence Research*, vol. 1, pp. 159–208, 1994.

[22] S. Donoho and L. Rendell, "Representing and restructuring domain theories: A constructive induction approach," *Journal of Artificial Intelligence Research*, vol. 2, pp. 411–446, 1995.

[23] S. Muggleton, *Inductive Logic Programming*, Academic Press, San Diego, 1992.

[24] M. Pazzani and D. Kibler, "The utility of knowledge in inductive learning," *Machine Learning*, vol. 9, pp. 57–94, 1992.

[25] B. Richards and R. Mooney, "Automated refinement of first-order horn-clause domain theories," *Machine Learning*, vol. 19, pp. 95–131, 1995.

[26] G. Towell and J. Shavlik, "Knowledge-based artificial neural networks," *Artificial Intelligence*, vol. 70, no. 1–2, pp. 119–165, 1994.

[27] L. M. Fu, "Integration of neural heuristics into knowledge-based inference," *Connection Science*, vol. 1, pp. 325–340, 1989.

[28] B. F. Katz, "Ebl and sbl: A neural network synthesis," in *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, 1989, pp. 683–689.

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

[30] E. Baum and K. Lang, "Constructing hidden units using examples and queries," in *Advances in Neural Information Processing Systems, vol. 3*, R. Lippmann, J. Moody, and D. Touretzky, Eds., San Mateo, CA, 1991, pp. 904–910, Morgan Kaufmann.

[31] G. Towell and J. Shavlik, "Extracting rules from knowledge-based neural networks," *Machine Learning*, vol. 13, pp. 71–101, 1993.

[32] L. M. Fu, "Knowledge based connectionism for refining domain theories," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 1, 1993.

[33] M. Craven, *Extracting Comprehensible Models from Trained Neural Networks*, Ph.D. thesis, Department of Computer Science, University of Wisconsin, Madison, WI, 1996.