Inductive Inference of Tree Automata by Recursive Neural Networks

P. Frasconi[2], M. Gori[1], M. Maggini[1], E. Martinelli[1], and G. Soda[2]

[1] Dipartimento di Ingegneria dell'Informazione
Università di Siena
Via Roma, 56 - 53100 Siena (Italy)
[2] Dipartimento di Sistemi e Informatica
Università di Firenze
Via S. Marta, 3 - 50138 Firenze (Italy)

Abstract. Recurrent neural networks are powerful learning machines capable of processing sequences. A recent extension of these machines can conveniently be used to process also general data structures like trees and graphs, which opens the doors to a number of new very interesting applications previously unexplored.
In this paper, we show that when the problem of learning is restricted to purely symbolic data structures, like trees, the continuous representation developed after learning can also be given a symbolic interpretation. In particular, we show that a proper quantization of the neuron activation trajectory makes it possible to induce tree automata. We present preliminary experiments for small-size problems that, however, are very promising, especially when considering that this methodology is very robust with respect to accidental or malicious corruption of the learning set.

1 Introduction

The art of building intelligent systems and modeling cognitive processes has been hovering, pendululum-like, between symbolic artificial intelligence and subsymbolic models like neural networks, for over three decades. The resurgence of interest in connectionist models of the past decade, however, has generated many blind debates on the supposed "actual nature" of intelligence.

M. Boden [1] has recently given her view on these debates by a very colorful comparison of the two different approaches to intelligence with the color of the horse in the film *The Wizard of Oz.* Her conclusion is that, like in the film, "... the pretty creature was *visibly* the same horse, changing colour as it trotted along." That is, "AI is one beast, like the Wizard's pony." Somewhere located in this story, this paper faces the inductive inference of tree automata, a purely symbolic task, by adaptive computation, a processing scheme taking place in a continuous domain.

Inductive inference is nicely reviewed concerning both problem definition and methodology in seminal papers by D. Angluin [2] and K-S. Fu and T.L. Booth [3], who also reviewed the case of tree grammars, considered in this paper [4].

The idea that symbols can emerge from subsymbolic representations through proper quantization of the neuron activation trajectory is not new in its own. In particular, a number of papers have focused on the inductive inference of finite state automata by using different recurrent neural network architectures (see e.g. [5,6]). Although the complex nonlinear dynamics of neural networks often departs to some extent from the corresponding automata behavior, especially for long sequences, proper architectures and conditions can be identified in which, under the given state space quantization, the automata offer a perfect representation of the neural dynamics [7,8].

In this paper, we go one step further and face the problem of inducing the rule(s) that generate a given set of trees. This entails a new crucial step, namely the definition of connectionist models capable of processing data structures. Sperduti and Starita [9] have proposed a connectionist-based architecture, based on the concept of generalized recurrent neuron, for classifying data structures. These approaches to processing data structures, however, were not conceived for inductive inference of grammars, as their adaptive scheme acts by modifying the weights of a sort of cryptic black box.

In this paper, we propose two algorithms for the extraction of tree automata from the learned configuration. In particular, the second one produces tree automata with *don't care* conditions, thus simplifying the corresponding representation significantly. These algorithms are based on proper quantizations of the state trajectory of the recursive networks associated with the given training set. We also introduce *third-order recursive neural networks*, and show that they are very well-suited for processing binary trees and that they inherit the basic features of the recursive networks used to process strings. Our preliminary experimental results are very promising. In particular, we show the the rule hidden in the *traffic policeman problem*, a nice cognitive graphical test defined in this paper, can be inferred successfully from examples.

2 Recursive Networks for Processing of Data Structures

In this section, we review briefly the basic idea proposed in [9] concerning adaptive processing of DOAGs (Directed Acyclic Graphs) [10]. The recursive networks considered in this paper process data structures beginning from a fixed initial state. The examples can be regarded as a collection of pairs composed of DOAGs with their own targets. Formally, let d^-, $d^+ \in R$ be such that $[d^-, d^+] \subset [\underline{d}, \overline{d}]$ and define $\mathcal{D} \doteq \{(\mathcal{U}_l, d_l)\,,\ l = 1, \ldots, L\}$, where $\mathcal{U}_l$ is a DOAG and $d_l \in \{d^-, d^+\}$ its corresponding target.
$\mathcal{U}_l$ is a directed acyclic graph, where the set of nodes leaving to a given node is ordered. For all DOAGs, there exists at least a special node, called *supersource* and denoted by $\mathbf{U}_{ls}$, that receives no inputs from other nodes.

Let o be the maximum outdegree of the given DOAGs. The dependence of node v from its children $ch[v]$ can be expressed by *pointer matrices* $A_r \in \mathcal{R}^{n,n}$, $r = 1, \ldots o$. Similarly, the information attached to the nodes can be prop-

agated by weight matrix $B \in \mathcal{R}^{n,m}$. Hence, the parameters of the adaptive model are $\boldsymbol{\theta}_f \doteq \{A_1, \ldots, A_o\}$ and $\boldsymbol{\theta}_g \doteq B$. The state is updated according to

$$\boldsymbol{X}_v = \boldsymbol{\sigma}\left(\sum_{k=1}^{o} A_k \cdot q_k^{-1}\boldsymbol{X}_v + B \cdot \boldsymbol{U}_v\right), \tag{1}$$

where q_k^{-1} is an operator which returns $\boldsymbol{X}_v$'s children and $\boldsymbol{\sigma}$ is a vectorial sigmoidal function. This equation is a straightforward extension of *first-order recurrent neural networks*, the only difference being in the generalized form of processing taking place in the "pseudo-time" dimension v. The output is determined by a general two-layer perceptron so as to generate any desired map of the state to the output according to

$$\boldsymbol{Y}_v = \boldsymbol{\sigma}\left(D\boldsymbol{\sigma}\left(C\boldsymbol{X}_v\right)\right). \tag{2}$$

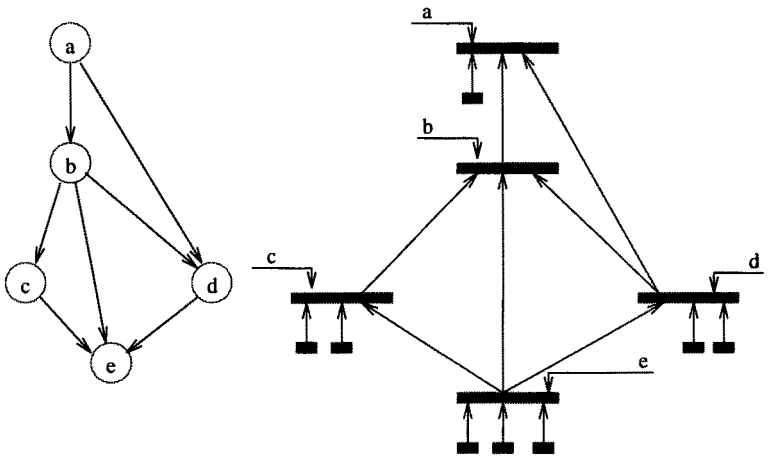


Fig. 1. A DOAG with the corresponding encoding network. Note that $\bar{o} = 2$ (graph outdegree), and that the nil pointers are represented by proper frontier (initial) states. The nodes must be presented according to a *topological sort*, e.g. e, c, d, b, a.

Fig. 1 is a pictorial representation of the computation taking place in the recursive neural network. In Fig. 1, all the recursive neurons are represented by the layer they belong to, whilst a proper notation is used to represent the nil pointer. Each nil pointer is associated with a *frontier state*, which is in fact an initial state that turns out to be useful to terminate recursive equation 1.

High-order neural networks, proposed mainly by Giles and associates for both static networks [11] and recurrent networks [12], are very interesting models especially for dealing with symbolic tasks. One can easily conceive high-order networks for processing data structure as an extension of second-order recurrent

networks. For instance, in the special case of binary trees, one can introduce *third-order networks* based on $\boldsymbol{\theta} \doteq \{w_{ijkl}\}$ as follows

$$X_{i,v} = \sigma\left(\sum_{j,k,l} w_{ijkl} \cdot q_L^{-1} X_{j,v} \cdot q_R^{-1} X_{k,v} \cdot U_{l,v}\right). \tag{3}$$

In this equation q_L^{-1} and q_R^{-1} are operators which return $\boldsymbol{X}_v$'s left and right children, respectively. The extension to the general case of $(o+1)$-dimensional networks is straightforward. In the case $\overline{o} = 1$, commonly used in the literature to carry out sequence processing, these models reduce to second-order recurrent networks.

According to the classical connectionist learning optimization-based approach, the output-target data fitting is measured by means of the cost function. The process of learning the given data structures consists of determining the parameters of the defined adaptive models.

3 Extraction of Symbolic Rules from Tree Automata

The computation of recursive networks rely on on a set of real-valued state and input variables which are processed using a set of continuous operators such as multipliers, adders and sigmoidal functions. When a recursive network parses a symbolic data structure, its state vector $\boldsymbol{X}_v \in R^n$ describes complex trajectories that encode the network memory about the processed nodes. These trajectories are generated applying for each node the network transition function to the o-uple of state vectors produced while processing its descendants. The state transition function $\phi(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_o, \boldsymbol{U}_v) : [R^n]^o \times R^m \to R^o$ depends on the node label $\boldsymbol{U}_v$. For symbolic tasks, the input alphabet is finite and can be encoded by a finite sets of vectors $\mathcal{U} = \{U_1, \ldots, U_K\}$, each corresponding to a symbol of the input alphabet. As a consequence, the network transition function defines a set of K different maps, which transform o vectors in R^n onto a point in R^n. This computation can be thought of as an extension of Iterated Function Systems (IFSs)[1].

The behavior resulting from the application of this set of non-linear maps may be very complex and may reveal a deep recursive structure. The generated trajectories may be much more complex than that of recurrent network analyzing time sequences because of the high dimensionality of the map input.

In order to give a *symbolic* interpretation of the continuous computation of the network, one can introduce an approximation based on the quantization of the continuous state trajectories. This process can be regarded as a *symbolic projection* of the regions of the continuous state space onto a set of discrete

[1] An IFS is defined as a set of S transformations Φ_s on a metric space $\mathcal{X}$. For each time step we can pick up one of these transformations to map a point $x \in \mathcal{X}$ to $\Phi_{s(t)}(x) \in \mathcal{X}$. The *attractor* of the IFS is the set $\mathcal{A} \subset \mathcal{X}$ such that $\forall a \in \mathcal{A}$ and $s \in \{1, \ldots, S\} \Rightarrow \Phi_s(a) \in \mathcal{A}$.

states. In particular, because of the nature of the computation performed by recursive networks, we propose a method to approximate the network behaviour by means of *Frontier to Root Automata* (FRA).

Definition 1. (see e.g. [14])
A FRA $\mathcal{A}$ is a 5-uple $\{\Sigma, S, s_0, M, F\}$ where $\Sigma = \{\sigma_1, \ldots, \sigma_K\}$ is the input alphabet used to label the nodes, S is a finite set of q states, s_0 is the initial state used to label the frontier nodes, $M : \Sigma \times S^o \rightarrow S$ is the state transition function and F defines the set of accepting states.

The FRA computation is carried out starting from the leaves of the graph which are supposed to have frontier nodes labeled with state s_0 as descendants; then for any node the corresponding state is computed as $M(\sigma, s_1, \ldots, s_o)$ being σ the node label and $< s_1, \ldots, s_k >$ the states computed for the node descendants. Obviously the computation must take place following the ordering of the nodes by descending values of their depth.

The symbolic interpretation of a recursive neural network in terms of a FRA emerges by partitioning the state space into a set of regions that are associated to the finite set of states of the automaton. The choice of the number of regions is crucial, since it defines the amount of loss in the memory capacity with respect to that of the network.
The state space can be partitioned by using the K-MEAN algorithm for the points of the state space visited by the neural network, while parsing a given set of data structures. This technique defines convex regions in the state space and has the property of using the higher resolution in the regions that are mostly visited. The representative point of each region may be chosen as the centroid of the corresponding cluster. Possible measures of the resolution are the minimum distance between two centroids and the number of clusters. If we use more clusters to approximate the network trajectories, we obtain a more detailed description and, consequently, the extracted machine is likely to have a larger number of states. The extraction of a complete FRA from the associated recursive network $\mathcal{N}$ with training set set $\mathcal{D}$ can be carried out by the following algorithm:

Algorithm 1 FRA extraction
$\mathcal{X} \leftarrow$ StateVectors($\mathcal{N}, \mathcal{D}$);
States $\leftarrow$ 2;
repeat
  $\mathcal{C} \leftarrow$ GeneratePartition($\mathcal{X}$,States);
  $s_0 \leftarrow$ GetRegion(X_0,$\mathcal{C}$);
  for ($< s_1, \ldots, s_o, \sigma > \in S^o \times \Sigma$)
    $X \leftarrow \phi(C(s_1, \mathcal{C}), \ldots, C(s_o, \mathcal{C}), U(\sigma))$;
    $s \leftarrow$ GetRegion(X,$\mathcal{C}$);
    $M(s_1, \ldots, s_o, \sigma) \leftarrow s$;
  for ($s \in S$)
    F(s) $\leftarrow$ GetNearestTarget($C(s, \mathcal{C})$);
  Errors $\leftarrow$ CompareIO($\mathcal{N}$,$< \Sigma, S, s_0, M, F >$,$\mathcal{D}$);
  States $\leftarrow$ States+1;
until (Errors<>0)

This algorithm extracts FRAs with increasing number of states (i.e. memory capacity) until it finds a machine that produces a perfect approximation of the Input/Output behaviour of the recursive network on the set of examples used for the extraction process. The comparison (CompareIO) can be carried out for each node (full I/O equivalence) or only on those nodes for which an output target is provided (target I/O equivalence).

The algorithm computes the state space partition $\mathcal{C}$ (GeneratePartition) using the state trajectories points $\mathcal{X}$ obtained while processing the graphs of the learning set (StateVectors). The continuous transition function of the recursive network $\phi()$ is computed and quantized for any combination of the centroids of the partition regions ($C(s,\mathcal{C})$). The output corresponding to each state s is obtained by computing the network output when its state is initialized with the centroid $C(s,\mathcal{C})$ and by finding the nearest target defined on the leaning set (GetNearestTarget).

The previous algorithm produces a FRA in terms of its full state transition map that is represented by a table with n^oK entries. The number of entries becomes huge when the number of states increases and can make the extraction process impractical. Moreover, most of these transitions are not contained in the training examples; this is often due to constraints in the data structure that make some combinations of states and inputs impossible.

In order to reduce the computational complexity of the extraction process, Algorithm 1 can be modified so as to produce *don't care* conditions for certain entries in the FRA state transition table. In many cases this choice corresponds to a drastic reduction of the valid transition rules, thus obtaining simpler solutions. The algorithm extracts only those state transitions that are activated while processing the data graphs in the learning set. The extraction algorithm turns out to be updated as follows:

Algorithm 2 FRA extraction (*don't cares*)

$\mathcal{X} \leftarrow$ StateVectors($\mathcal{N}, \mathcal{D}$);
States $\leftarrow$ 2;
repeat
  $\mathcal{C} \leftarrow$ GeneratePartition($\mathcal{X}$,States);
  $s_0 \leftarrow$ GetRegion(X_0,$\mathcal{C}$);
  for ($Graph \in \mathcal{D}$)
    for ($Node \in Graph$)
      $< s_1, \ldots, s_o > \leftarrow < State(Node.child[1]), \ldots, State(Node.child[o]) >$
      $X \leftarrow \phi(C(s_1), \ldots, C(s_o), U(Node.\sigma))$;
      $s \leftarrow$ GetRegion(X,$\mathcal{C}$);
      $M(s_1, \ldots, s_o, \sigma) \leftarrow s$;
  for ($s \in S$)
    F(s) $\leftarrow$ GetNearestTarget($C(s)$);
  Errors $\leftarrow$ CompareIO($\mathcal{N}$,$< \Sigma, S, s_0, M, F >$,$\mathcal{D}$);
  States $\leftarrow$ States+1;
until (Errors<>0)

The main difference between the two algorithms resides in the loops where the transition extraction is performed. In the case of Algorithm 2 only the transitions in the learning set $\mathcal{D}$ are considered, while all the others are set to *don't care*.

4 Inference of Tree Automata: Experimental Results

In this section we show the experimental results we found when using recursive networks and the FRA extraction algorithm to infer tree automata. The first two experiments concern binary trees, while the third one involves a graph with a higher outdegree, but a fixed number of nodes. In all these experiments the network weights were initialized with random values in the interval [-0.5,0.5]. Input symbols were coded by unitary vectors (i.e. vectors having an entry equal to 1 and all other entries equal to 0). The learning and the test sets were generated randomly. Table 1 reports the complete setup for the three experiments. The rules used to generate the examples are the following:

1. **No b** : Given a set of binary trees with nodes whose labels are in the alphabet $\{a, b\}$, positive trees are those with no b labels, regardless of the the tree structure.
2. **f(a,b)** : Given a set of binary trees with nodes whose labels are in the alphabet $\{a, b, f\}$, positive trees are those containing sub-term $f(a, b)$ and following the rule: labels a and b can be found only in leaves, while term f must have two children. An example for this class of trees is the "expression" $f(f(b, a), f(a, f(a, b)))$ that corresponds to a tree with depth 4.
3. **The traffic policeman problem:** It is related to a cognitive task in which a traffic policeman can stop a car depending on his gestures (see Fig. 2). The policeman *stops* the car when he raises one of his arms or when he holds the red sign in one of his hands. The policeman can be represented by a graph which is constructed by inspection of the relative positions of its body components and colors. An empty position results in a nil pointer assigned to the corresponding direction link for the node. Thus the rule can simply be translated into the graphical formalism: one of the pointers NW or NE from the root node (1) must not be nil or one of the hand labels (nodes 7,9,10,11 if linked) must be r.

Table 2 summarizes the results for these experiments. In all the experiments all the training examples were learned perfectly. From the last column, the advantage of using the *don't care*-based extraction algorithm is clear, especially for the traffic policeman problem.

Figure 3 shows the extraction process for the *No-b trees* problem. The plot shows the states generated while parsing the 50 examples of the training set. Although the state distribution is quite complex, a simple partition into 3 regions turns out to be sufficient for extracting a FRA that has the same I/O behaviour as the network on the root nodes (the only nodes with targets). The resulting FRA has three states: s_0 is the initial state, s_2 codes the presence of the b symbol,

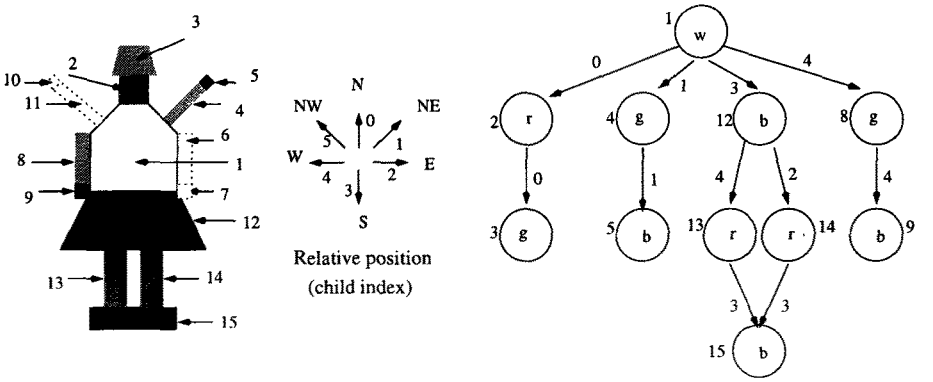


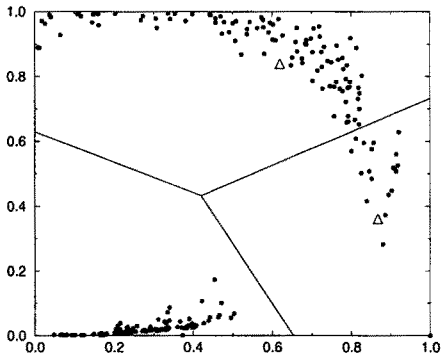
Fig. 2. The traffic policeman problem: Each component of the figure can be labeled with one out of four colors ([b]lack, [g]reen, [r]ed, [w]hite).

Problem	Network	Symbols	Learning	Test	Depth	Nodes
No b	3rd 2-21	{a/*,b/*}	13-27	32-68	[4,6]	[10,33]
f(a,b)	3rd 2-21	{a/0,b/0,f/2}	18-32	75-125	[2,5]	[3,15]
Policeman	1lr 2-37	{b/-,g/-,r/-,w/-}	193-207	243-257	[4,4]	[11,11]

Table 1. Experimental setup. The first column reports the problem rule; the second column specifies the neural network architecture used for the experiments reporting the network model (3d = third-order, 1lr = feedforward with one layer), the number of state neurons and the number of learnable parameters; the third column reports the symbols used to label the nodes and the constraints on the number of required descendants (* = no constraint, - = depends on node position); the fourth and fifth columns report the number of positive and negative graphs contained in the learning and test sets; the last two columns show the ranges for the depth and the number of nodes of the examples.

Problem	Learning	Test	States	Transitions
No b	100%	100%	3	18/18
f(a,b)	100%	100%	8	40/192
Policeman	100%	100%	5	64/62500

Table 2. Experimental results. The first two columns report the accuracy of the inferred rule on the training and test sets, respectively; the third column reports the number of states of the extracted FRA; the last column represents the number of extracted transitions with respect to the total entries in the FRA state table.



$\Rightarrow$

$< s_L, s_R >$	a	b
$< s_0, s_0 >$	s_1	s_2
$< s_0, s_1 >$	s_1	s_2
$< s_0, s_2 >$	s_2	s_2
$< s_1, s_0 >$	s_1	s_2
$< s_1, s_1 >$	s_1	s_2
$< s_1, s_2 >$	s_2	s_2
$< s_2, s_0 >$	s_2	s_2
$< s_2, s_1 >$	s_2	s_2
$< s_2, s_2 >$	s_2	s_2

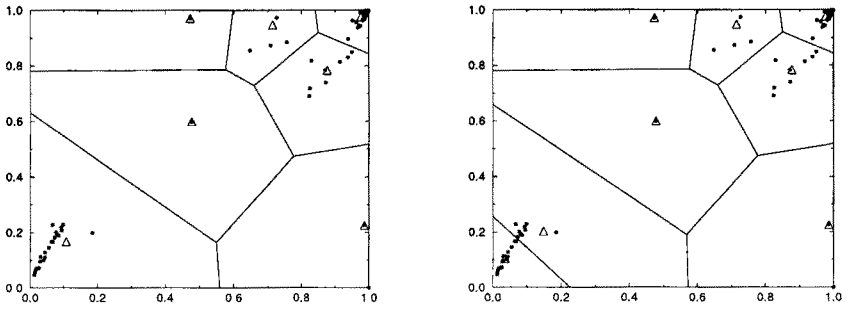
$\text{Out}(s_0) = -$
$\text{Out}(s_1) = +$
$\text{Out}(s_2) = -$

Fig. 3. FRA extraction for the No-b problem. The plot shows the state vectors, the state space partitions and the corresponding centroids. The table reports the FRA transition rules and the state/output map.

s_1 is the accepting state. The FRA state becomes s_2 whenever symbol b is read in a tree node.

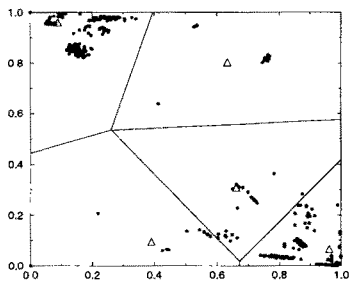
For the $f(a, b)$ experiment, the extracted FRA is more complex. Only the transitions that are compatible with the term constraints are present. In particular, from the initial state only the transitions with symbols a and b are defined, while from all the other states only transitions with symbol f are extracted. The 8 state FRA can be minimized to an equivalent 4 state FRA that implements exactly the classification rule. The four states represent a leaf containing symbol a, a leaf labeled with b, an internal node having a sub-tree containing term $f(a, b)$, and an internal node which has no subtree containing $f(a, b)$. In Fig. 4 it is shown that 8 regions are required by the automata extraction process, since the extraction with 7 clusters is not successful, as the two states near the origin are not equivalent.

Figure 5 reports the extraction results for the traffic policeman problem. The extraction is performed using 5 regions. It is easy to see that the classification rule was extracted successfully by inspection of the developed state transitions. Because of the fixed structure of the graphs, it can be seen that transitions 1 through 5 are always applied to the root node. Transitions 1 and 2 deal with a raised arm (s_{NW} and s_{NE} not nil, i.e. s_0); transitions 4 and 5 correspond to the case of a red sign in the hand with both arms down (respectively in the left and right hand); transition 3 represents a *non-stopping* policeman (state s_2 is not accepting). Transition 6 says whether the color of a leaf is red (s_3). Transition 7 (8) is used to propagate the presence (absence) of a red sign in a hand to the root node when the arm is down (through the s_S link). The other transitions are necessary to manage the other nodes and reflect the independence on the component color.



(a) (b)

Fig. 4. FRA extraction for the $f(a, b)$ problem: (a) state space partition using 7 states. This resolution is not sufficient to satisfy the network-FRA I/O equivalence condition. (b) State space partition using 8 states. The network I/O behaviour is exactly approximated by the extracted FRA.



	$< s_N, s_{NE}, s_E, s_S, s_W, s_{NW} >$	b	g	r	w
1	$< s_1, s_0, s_3, s_2, s_0, s_1 >$	s_1	s_1	s_1	s_1
2	$< s_1, s_1, s_0, s_2, s_3, s_0 >$	s_1	s_1	s_1	s_1
3	$< s_1, s_0, s_3, s_2, s_3, s_0 >$	s_2	s_2	s_2	s_2
4	$< s_1, s_0, s_3, s_2, s_4, s_0 >$	s_4	s_1	s_4	s_1
5	$< s_1, s_0, s_4, s_2, s_3, s_0 >$	s_4	s_4	s_4	s_4
6	$< s_0, s_0, s_0, s_0, s_0, s_0 >$	s_1	s_1	s_3	s_1
7	$< s_0, s_0, s_0, s_3, s_0, s_0 >$	s_4	s_4	s_4	s_4
8	$< s_0, s_0, s_0, s_1, s_0, s_0 >$	s_3	s_3	s_3	s_3
9	$< s_1, s_0, s_0, s_0, s_0, s_0 >$	s_1	s_1	s_1	s_1
10	$< s_3, s_0, s_0, s_0, s_0, s_0 >$	s_1	s_1	s_1	s_1
11	$< s_0, s_1, s_0, s_0, s_0, s_0 >$	s_1	s_1	s_1	s_1
12	$< s_0, s_3, s_0, s_0, s_0, s_0 >$	s_1	s_1	s_1	s_1
13	$< s_0, s_0, s_0, s_0, s_0, s_3 >$	s_1	s_1	s_1	s_1
14	$< s_0, s_0, s_0, s_0, s_0, s_1 >$	s_1	s_1	s_1	s_1
15	$< s_0, s_0, s_4, s_0, s_4, s_0 >$	s_2	s_2	s_2	s_2
16	$< s_0, s_0, s_3, s_0, s_3, s_0 >$	s_2	s_2	s_2	s_2

Fig. 5. FRA extraction from the traffic policeman problem. The plot shows the state vectors, the state space partitions, and the corresponding centroids. The table reports the FRA transition rules. States s_1 and s_4 are accepting.

5 Conclusions

In this paper we have shown that recursive neural networks, used for processing data structures, can be given an intriguing symbolic interpretation in terms of tree automata. On the basis of the experience gained from some preliminary experiments, we have shown that proper quantization algorithms can be conceived that make it possible to infer tree automata from examples. It is worth mentioning that, at the moment, we do not have evidence to state that the proposed methodology can challenge symbolic approaches to inductive inference. Our own feeling is that the proposed adaptive model can in fact hardly compete in a purely symbolic domain, like the inductive inference of tree automata. On the other hand, unlike most symbolic approaches to inductive inference, the one we propose is likely to be very robust with respect to accidental or malicious modifications of the training set, since the optimization process on which learning relies is inherently robust. Finally, Boden's moral on artificial intelligence, mentioned in the introduction, seems to suggest that the new "color" emerging from connectionist models could play his own role, particularly in the case of noisy examples: "when lighting conditions change, the Wizard's pony could trot along with this new color.

Acknowledgements

We thank Alessandro Sperduti (Dipartimento di Informatica, Università di Pisa) for fruitful discussions on recursive neural networks and their links with tree automata.

References

1. M. Boden, "Horses of a different colour?," in *Artificial Intelligence and Neural Networks* (V. Honavar and L. Uhr, eds.), pp. 3–19, Academic Press, 1994.
2. D. Angluin and C. Smith, "Inductive inference: Theory and methods," *Computing Surveys*, vol. 15, no. 3, pp. 237–269, 1983.
3. K.-S. Fu and T. Booth, "Grammatical inference: Introduction and survey - part i," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 5, pp. 95–111, January 1975.
4. K.-S. Fu and T. Booth, "Grammatical inference: Introduction and survey - part ii," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 5, pp. 409–423, May 1975.
5. R. Watrous and G. Kuhn, "Induction of finite-state languages using second-order recurrent networks," *Neural Computation*, vol. 4, no. 3, pp. 406–414, 1992.
6. C. Giles, C. Miller, D. Chen, G. Sun, H. Chen, and Y. Lee, "Learning and extracting finite state automata with second-order recurrent neural networks," *Neural Computation*, vol. 4, no. 3, pp. 393–405, 1992.
7. P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite state automata in recurrent radial basis function networks," *Machine Learning*, vol. 23, pp. 5–32, 1996.

8. C. Omlin and C. Giles, "Constructing deterministic finite-state automata in recurrent neural networks," *Journal of the ACM*, vol. 43, no. 6, pp. 937–972, 1996.
9. A. Sperduti and T. Starita, "Supervised neural networks for classification of structures," *IEEE Transactions on Neural Networks*. to appear.
10. M. Arbib and Y. Given'on, "Algebra automata i: Parallel programming as a prolegomena to the categorical approach," *Information and Control*, vol. 12, pp. 331–345, 1968.
11. C. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Applied Optics*, vol. 26, no. 23, p. 4972, 1987.
12. C. Miller and C. Giles, "Experimental comparison of the effect of order in recurrent neural networks," *Int. Journal of Pattern Recognition and Artificial Intelligence*, 1993. Special Issue on Applications of Neural Networks to Pattern Recognition (I. Guyon Ed.).
13. E. Sontag and H. Sussman, "Backpropagation separates when perceptrons do," in *International Joint Conference on Neural Networks*, vol. 1, (Washington DC), pp. 639–642, IEEE Press, June 1989.
14. J. Thatcher, "Tree automata: An informal survey," in *Current Trends in the Theory of Computing* (A. Aho, ed.), pp. 143–172, Prentice-Hall, Inc.