# On the Implementation of Frontier-to-Root Tree Automata in Recursive Neural Networks

Marco Gori, *Senior Member, IEEE*, Andreas Küchler, and Alessandro Sperduti, *Member, IEEE*

***Abstract*— In this paper we explore the node complexity of recursive neural network implementations of frontier-to-root tree automata (FRA). Specifically, we show that an FRAO (Mealy version) with $m$ states, $l$ input–output labels, and maximum rank $N$ can be implemented by a recursive neural network with $O(\sqrt{\frac{(\log l+\log m)lm^N}{\log l+N\log m}})$ units and four computational layers, i.e., without counting the input layer. A lower bound is derived which is tight when no restrictions are placed on the number of layers. Moreover, we present a construction with three computational layers having node complexity of $O((\log l+\log m)\sqrt{lm^N})$ and $O((\log l+\log m)lm^N)$ connections. A construction with two computational layers is given that implements any given FRAO with a node complexity of $O(lm^N)$ and $O((\log l+N\log m)lm^N)$ connections. As a corollary we also get a new upper bound for the implementation of finite-state automata (FSA) into recurrent neural networks with three computational layers.**

***Index Terms*—Automata implementation, knowledge injection, node complexity, recursive neural networks, tree automata.**

## I. Introduction

IT has recently been demonstrated that neural networks are not limited to the processing of static patterns and sequences; in fact, they can also naturally deal with more complex data structures, like labeled trees and graphs. For example, it has already been shown how neural networks can represent labeled trees [1] and labeled directed graphs [2], and how learning algorithms for recurrent neural networks can be extended to the processing of labeled directed graphs [3]–[7]. The ability to represent and classify these type of data structures is fundamental in a number of different applications such as medical and technical diagnoses, molecular biology, automated reasoning, software engineering, geometrical and spatial reasoning, and pattern recognition. Some application perspectives, utilizing this type of networks which we will put here under the collective name *recursive neural networks*, already emerged, e.g., in computational chemistry [8], [9], in logo recognition [10], and in search control in theorem proving [11].

The *node complexity* measures the resource consumption—here mainly the number of nodes—required for neural networks to implement certain classes of functions [12]. In this paper we explore the node complexity of *frontier-to-root tree automata* (FRAO) implementations into recursive neural networks. Our work is mainly driven by the following motivations.

1) *Theoretical Interest:* the class of recursive neural networks may be viewed as a generalization of the well-known *recurrent* neural networks. There are significant results on the node complexity of recurrent neural network implementations of *finite-state automata* (FSA) [13], [14]. Recently, recursive neural networks have been proven to possess the computational power of at least frontier-to-root automata [15], [16]. This machine model for tree processing is known to be a generalization of the FSA concept for sequence processing [17], [18]. These results raise the question about the node complexity of recursive neural network implementations of FRAO. Can the previous methods and results be lifted from sequence to tree processing? Furthermore, the consideration of recursive neural networks might also give new insights on the relationship between recurrent networks and FSA.
2) *Application Perspectives:* considerable results are reported on the *injection* of domain knowledge—*a priori* knowledge available in form of formal languages or inferred by symbolic machine learning approaches—into neural networks and its inductive *refinement* by common learning algorithms (for a general framework see [19]; for the case of recurrent neural networks see [20]–[22]. Thus, it might be useful to explore methods to insert knowledge that is given in form of tree grammars (or by the corresponding tree automaton) into recursive neural networks. The node complexity plays an important role especially for those applications (e.g., real-time systems, embedded systems, controllers) where neural networks have to be realized in hardware under strict space constraints.

We give upper bounds on the number of units needed to implement a given FRA with output (FRAO) in four, three, and two layers recursive networks. Whereas the bound for four layers networks constitutes a generalization of the bound discussed in [14] for the implementation of FSA in recurrent networks, the bound for networks with three computational layers produces, as a special case, a new bound on the implementation of FSA in recurrent networks. Moreover, the bounds for networks with two and three computational layers are constructive and, thus, they may turn out to be useful for practical applications. Their constructive proof is based

Manuscript received June 25, 1998; revised June 17, 1999 and July 20, 1999.

M. Gori is with the Dipartimento di Ingegneria dell'Informazione, Università di Siena, 53100 Siena, Italy.

A. Küchler is with the Department of Neural Information Processing, University of Ulm, 89069 Ulm, Germany.

A. Sperduti is with the Dipartimento di Informatica, Università di Pisa, I-56125 Pisa, Italy.

Publisher Item Identifier S 1045-9227(99)09186-9.

on the encoding of each possible input configuration to the FRAO by a different integer in a predefined interval. This integer is then processed by the repeated application of the *telescopic* technique [23], [24], so called because it consists of the progressive activation of a set of units according to the magnitude of the processed integer. On the basis of this progressive encoding, it is possible to implement both the transition and output function of the desired FRAO.

By using the techniques developed by Alon *et al.* [13], and Horne and Hush [14] we are able to derive a lower bound on the node complexity (in the case when no restrictions are placed on the number of layers) which is tight to the upper bound for four computational layers.

This paper is organized as follows. First, we introduce recursive neural networks, tree automata and their encoding by Boolean functions. Then, we present the upper bounds for the implementation of FRAO's in recursive networks with four, three, and two computational layers, respectively (Section III). Where appropriate, we discuss the relationships with the insertion of FSA in standard recurrent networks. Section IV is devoted to the derivation of the lower bound. Finally, we argue that the bound for three computational layers can be improved by solving the *optimal coding problem*, i.e., an optimization problem which, unfortunately, we conjecture to be NP-complete.

## II. Background

### A. Tree Automata

The computation model of *tree automata* is well understood and applied in several fields of computer science. There are several types of tree automata and several ways to introduce them [17], [18]. Here it is convenient to define an analogous extension to what is known as *Mealy machines* [25] in the case of sequence processing, i.e., automata that map trees to trees of the same shape.

*Definition 1:* A (deterministic) *frontier-to-root tree automaton*[1] *with output* (FRAO) is a system $\mathcal{A}_t = (\mathcal{U}, \mathcal{O}, \mathcal{X}, \delta, \omega, x_f)$ where

- $\mathcal{U}$ is the finite *input* ranked alphabet with maximum rank $N$;
- $\mathcal{O}$ is the finite *output* ranked alphabet;
- $\mathcal{X}$ is a finite set of states;
- $\delta : \mathcal{X}^N \times \mathcal{U} \to \mathcal{X}$ is the transition function which maps a tuple in $\mathcal{X}^N \times \mathcal{U}$ into a state $x = \delta(x_{q_1}, \cdots, x_{q_N}, u) \in \mathcal{X}$;
- $\omega : \mathcal{X}^N \times \mathcal{U} \to \mathcal{O}$ is the output function where the rank of the output symbol is constrained by the rank of the input symbol;
- $x_f$ is the *frontier* state.

The given ranked input alphabet $\mathcal{U}$ defines a domain of *rooted labeled ordered trees*. We implicitly assume that trees are extended and filled-up to the maximum rank $N$. This can be achieved by virtually adding *empty tree* nodes to the frontier. A given input tree is processed in a bottom-up manner (from the frontier to the root), i.e., the computation performed by the automaton can inductively be described by the following.

1) Each frontier node of the input tree is assigned the state $x_f$.
2) For any node with a label $u$ the state $\delta(x_{q_1}, \cdots, x_{q_N}, u)$ is assigned, where $x_{q_1}, \cdots, x_{q_N}$ are the states already assigned to the offsprings of that node, and the associated output value is $\omega(x_{q_1}, \cdots, x_{q_N}, u)$.

By this mode of processing, an FRAO maps trees to trees of the same shape where the labels of the input tree are transformed to symbols (with the same rank) of the output alphabet. In order to clarify the exposition, in the following we will use $\delta_u(x_{q_1}, \cdots, x_{q_N})$ to denote $\delta(x_{q_1}, \cdots, x_{q_N}, u)$, and $\omega_u(x_{q_1}, \cdots, x_{q_N})$ to denote $\omega(x_{q_1}, \cdots, x_{q_N}, u)$. Further, we assume that the transition and the output function are defined on the whole domain.

Note that, the above definition of FRAO can implement any frontier-to-root tree automaton (operating as language *recognizer*) where no output function is defined, but a subset $F$ of $\mathcal{X}$ is defined as the *set of final states*. In this case, an input tree is *accepted* by $\mathcal{A}_t$ iff the automaton can enter a final state upon encountering the root. Such automaton can be simulated by an FRAO by defining

$$\omega_u(x_{q_1}, \cdots, x_{q_N}) = \begin{cases} 1, & \text{if } \delta_u(x_{q_1}, \cdots, x_{q_N}) \in F \\ 0, & \text{otherwise.} \end{cases}$$

The *recognized language* of $\mathcal{A}_t$ is then defined by the set of trees whose root nodes are mapped to the output value one. The so-called *regular tree languages* is the corresponding class of languages that can be recognized by frontier-to-root automata [17], [18].

In Fig. 1 we give an example of an automaton, showing the processing by the automaton on a specific input tree.

### B. Recursive Neural Networks

The processing of trees[2] by using neural networks can be understood by analogy with tree automata. In fact, given a vertex $v$ belonging to the input tree, a recursive neural network computes two functions defined in the following way:

$$x(v) = t\big(x_{\mathrm{ch}[v]}, u(v)\big) \tag{1}$$

$$o(v) = z(x(v), u(v)) \tag{2}$$

where $x(v)$ is the state associated with $v$, $x_{\mathrm{ch}[v]}$ is the set of states[3] associated with the children of $v$, $u(v)$ is the label attached to the vertex $v$, and $o(v)$ is the output generated by the network after the presentation of vertex $v$.

Of course, $t(\cdot)$ can be related to the transition function $\delta(\cdot)$ in tree automata, whereas $z(\cdot)$ can be related to the output function $\omega(\cdot)$. In fact, since we are interested in tree automata with output, we will use the following definition for the output

[1] The term *bottom-up tree automaton* is synonymously used in the literature.

[2] Recursive neural networks can actually deal with a more expressive class of structures, namely *acyclic directed graphs* [5]. In [5] processing of cyclic graphs have been proposed as well, however, up to now, no experimental results on cyclic graphs has been produced. So the analysis presented in this paper can be considered as a first step toward a more comprehensive study on the computational and complexity capabilities of recursive neural networks.

[3] The states are ordered according to the order defined on the children.

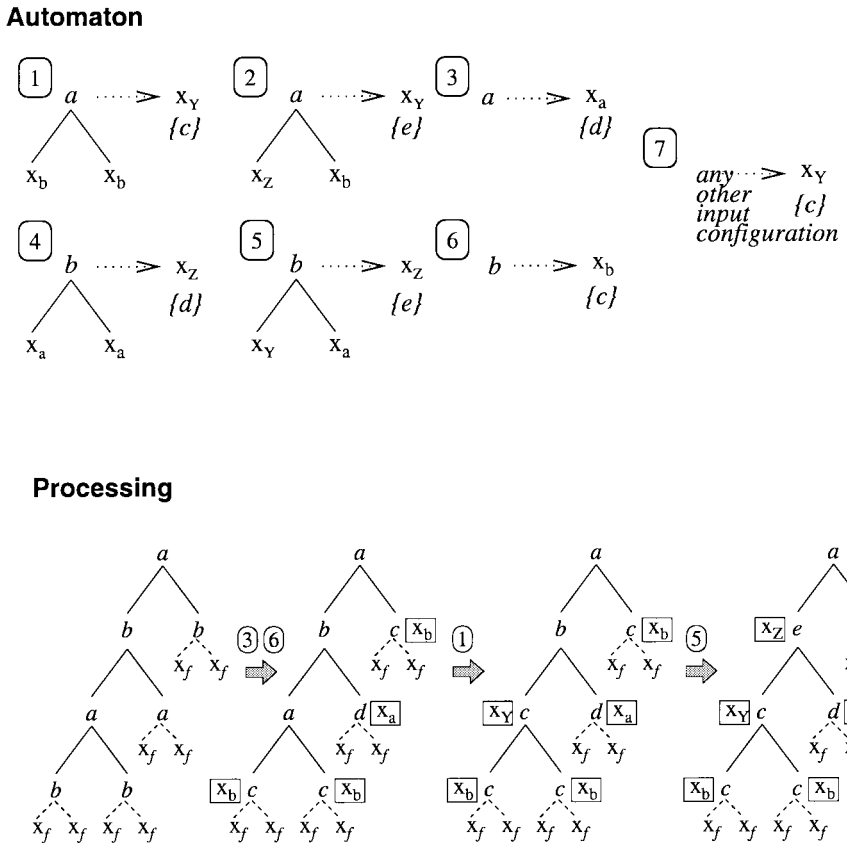


Fig. 1. Top: example of FRAO with $\mathcal{U} = \{a, b\}$, $\mathcal{O} = \{c, d, e\}$, and $\mathcal{X} = \{x_a, x_b, x_Y, x_Z\}$. Both $\delta$ and $\omega$ are represented in graphical form (with the output of $\omega$ in brackets). Each transition rule is numbered and it is constituted by an input structure (on the left of the arrow), with root labeled by a letter of the input alphabet and children labeled by allowed states, and an output state (on the right of the arrow). Bottom: example of processing of the FRAO over a tree. The tree on the left side is the input tree, where the frontier state $x_f$ is represented explicitly where needed. Given this input tree, by applying transition rules 3 and 6, the second tree is obtained, where to each node on the frontier the label is changed according to the applied transition rule and a state, represented into a square, is associated. From this tree the processing proceeds through the application of transition rules 1, 5, 2, till the final tree on the right side is obtained.

function:

$$o(v) = g\big(x_{\mathrm{ch}[v]}, u(v)\big). \tag{3}$$

The actual neural adaptive processing of data structures based on (1) and (3) takes place once we introduce a parametric representation in which the weights can be estimated from examples by a gradient descent technique. Specifically, the state transition function $t(\cdot)$ and the output function $g(\cdot)$ are approximated by feedforward neural networks, leading to the parametric representation

$$x(v) = t\big(x_{\mathrm{ch}[v]}, u(v), W\big)$$
$$o(v) = g\big(x_{\mathrm{ch}[v]}, u(v), V\big)$$

where $W$ and $V$ are connection weights.

A typical example of realization for $t(\cdot)$ is given by

$$x(v) = f\left(\sum_{i=1}^{n} w_i \lambda_i + \sum_{j=1}^{N} \hat{w}_j x(vj) + \theta\right) \tag{4}$$

where $f$ is a sigmoidal function, $w_i$ are the weights associated with the label space, i.e., $\{0,1\}^n$, $\hat{w}_j$ are the weights associated with the subgraphs spaces, $\theta$ is the bias, $\boldsymbol{\lambda}$ is the vector representing the input label $u(v)$, and $x(v1), \cdots, x(vN)$ are the encoded representations of $v$'s children, i.e., if $z$ is the $j$th child of $v$, then $x(vj) = x(z)$. A richer representation of the state can be given by a set of $m$ recursive neurons

$$\mathbf{x}(v) = \mathbf{F}\left(\mathbf{W}\boldsymbol{\lambda} + \sum_{j=1}^{N} \hat{\mathbf{W}}_j \mathbf{x}(vj)\right) + \boldsymbol{\theta}) \tag{5}$$

where $\mathbf{x}(v) \in I\!R^m$, $\mathbf{F}_i(\mathbf{q}) = f(q_i)$, $\boldsymbol{\lambda} \in I\!R^n$, $\boldsymbol{\theta} \in I\!R^m$, $\mathbf{W} \in I\!R^{m \times n}$, $\mathbf{x}(vj) \in I\!R^m$, $\hat{\mathbf{W}}_j \in I\!R^{m \times m}$.

Concerning the output function $g(\cdot)$, it can be defined as a set of $k$ standard neurons taking on input the state representation for $v$. An example, which agrees with the form of the FRAO output function, is

$$\mathbf{o}(v) = \mathbf{F}\left(\mathbf{L}\boldsymbol{\lambda} + \sum_{j=1}^{N} \mathbf{V}_j \mathbf{x}(vj) + \boldsymbol{\beta}\right) \tag{6}$$

where $\mathbf{o}(v) \in I\!R^k$, $\mathbf{L} \in I\!R^{k \times n}$ is the label input–output weight matrix, $\mathbf{V}_j \in I\!R^{k \times m}$ are the state input–output matrices, and $\boldsymbol{\beta} \in I\!R^k$ is the vector of the bias terms defining $g(\cdot)$.

In Fig. 2 we have given a pictorial representation of how a recursive network with a single hidden unit generates the so called *encoding network* given a tree in input. The encoding network constitutes a generalization of the encoding network obtained in *recurrent* neural network by unrolling the network in time. In fact, in the special case of linear chains, i.e., $N = 1$,

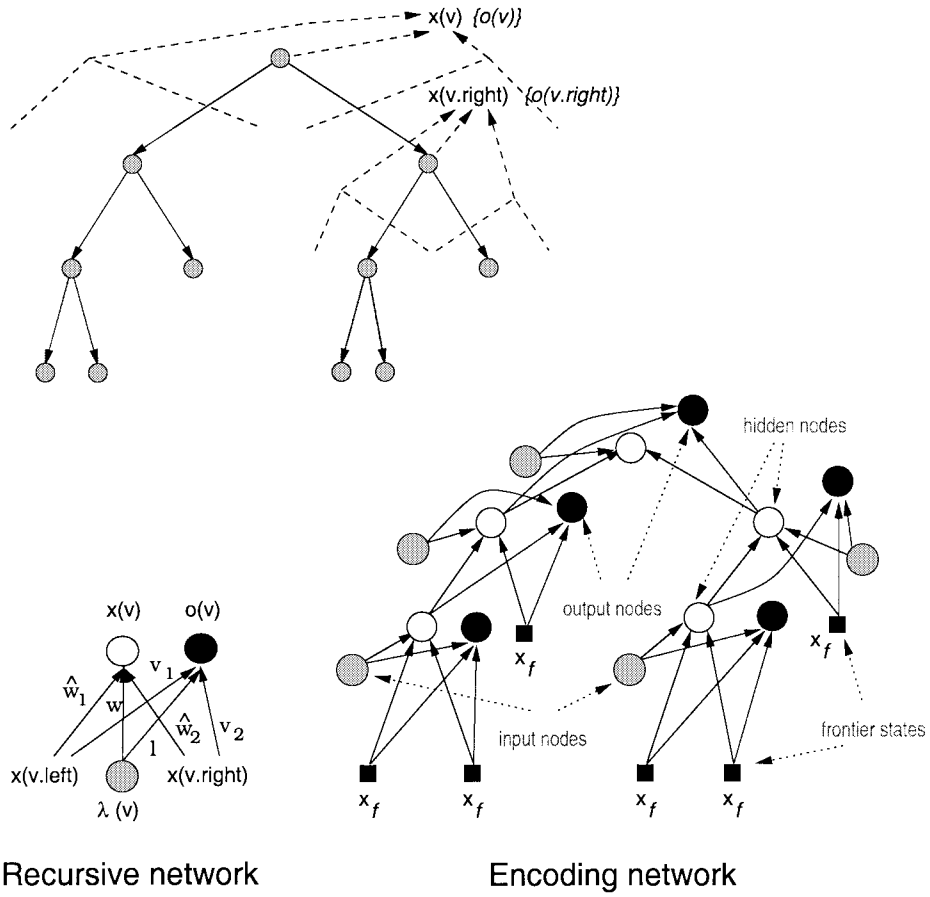


Fig. 2. An example tree (upper half) being processed by a recursive neural network with one input, hidden, and output node (lower left). The encoding network (lower right) shows the network unfolded according to the input tree.

the above equations exactly correspond to the general state space equations of *recurrent* neural networks.

The above examples of realizations for $t(\cdot)$ and $g(\cdot)$ are not exhaustive since, in general, they can be implemented by any feedforward neural network. In particular, in this paper we will use general feedforward neural networks with across-levels connections.

For more details on recursive neural networks and associated learning algorithms see [4] and [5].

### C. Encoding of FRA by Boolean Functions

Following the idea by Horne and Hush [14] an FRAO can be realized by Boolean functions. Let $\mathcal{A}_t = (\mathcal{U}, \mathcal{O}, \mathcal{X}, \delta, \omega, x_f)$ be a given FRAO, $m = |\mathcal{X}|$ be the number of states, $l = \max(|\mathcal{U}|, |\mathcal{O}|)$ be the alphabet size and $N$ be the maximum rank found in $\mathcal{U}$. The $m$ states can be encoded in $\lceil \log m \rceil$ bits, the $l$ labels in $\lceil \log l \rceil$ bits. Hence, the state transition function and the output function may be grouped together and implemented by a Boolean function of the type

$$\iota : \{0,1\}^{\lceil \log l \rceil + N \lceil \log m \rceil} \rightarrow \{0,1\}^{\lceil \log l \rceil + \lceil \log m \rceil}. \qquad (7)$$

By taking this point of view known results on the node complexity of neural networks (e.g., [12], [23], and [24]) can be used to derive propositions about the node complexity of FRAO implementations in recursive networks.

In the following, for the sake of simplicity, we will drop the ceilings, since it can easily be shown that all bounds to be presented in this paper hold even if $m$ and $l$ are not powers of two.

## III. Upper Bounds

The question we address in this section is to determine how many units we need to implement a tree automaton in a recursive neural network. In order to answer this question we consider hard threshold units, i.e., units where the output function is defined as

$$\operatorname{sgn}(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

Since a sigmoid function can approximate a step function to an arbitrary degree of precision by augmenting the modulus of the associated weight vector, if we demonstrate that a recursive network with threshold gates can implement any FRAO, then the upper bound results hold for recursive networks with sigmoids as well. In the following we will give upper bounds for recursive networks with four, three, and two computational layers.

### A. Background Knowledge

In this section we introduce Lemmas which will be used to demonstrate the upper bounds.

The first lemma is due to Lupanov [26] and it states the amount of nodes needed by a network with four computational layers of perceptrons to implement an arbitrary Boolean logic function:

*Lemma 1:* Arbitrary Boolean logic functions of the form $f : \{0,1\}^x \rightarrow \{0,1\}^y$ can be implemented in a network of perceptrons with four computational layers and with a node complexity of $O(\sqrt{\frac{y2^x}{x-\log y}})$.

If the number of allowed layers is two, then the following lemma by Horne and Hush [12], gives an upper bound on the node complexity.

*Lemma 2:* Arbitrary Boolean logic functions of the form $f : \{0,1\}^x \rightarrow \{0,1\}^y$ can be implemented in a network of perceptrons with two computational layers and a node complexity of $O(2^x + y)$.

The above lemmas are useful for deriving relevant results, however, their utilization does not lead to a constructive proof. A constructive proof would be very desirable since it will eventually show how to insert a tree automata into a recursive neural network. The inserted tree automata can then be refined on the basis of training examples using one of the suggested algorithms for training recursive neural networks. The following lemma introduces the *telescopic* technique [24], which will be used for building constructive upper bounds for networks with two and three computational layers.

*Lemma 3 (Telescopic):* Let the integer interval $[0,n]$ be divided into $k+1$ subintervals $[\beta_0, \beta_1 - 1]$, $[\beta_1, \beta_2 - 1], \cdots, [\beta_{k-1}, \beta_k - 1]$, $[\beta_k, n]$, where $\beta_0 = 0 < \beta_1 < \beta_2 < \cdots < \beta_k < n$. Let $y_i = \mathrm{sgn}(\sum_{j=1}^{n} x_j - \beta_i)$, for all $i = 1, \cdots, k$. Then

$$\sum_{j=1}^{k} (\alpha_j - \alpha_{j-1}) y_j = \alpha_m$$

if $\sum_{j=1}^{n} x_j \in [\beta_m, \beta_{m+1} - 1]$, where, $\alpha_0 = 0$, and $\alpha_1, \cdots, \alpha_k$ are arbitrary real numbers.

### B. Networks with Four Computational Layers

It is known that finite-state automata with $m$ states and a binary alphabet can be inserted in a recurrent neural networks with four computational layers and $O(\sqrt{m})$ units [14]. This result was obtained by exploiting Lemma 1. Using the same lemma we can prove the following result.

*Theorem III.1 (Four Layers Implementation):* Any FRAO with $m$ states, $l$ input–output labels, and maximum rank $N$, can be implemented by a recursive neural network with four computational layers and a node complexity of $O(\sqrt{\frac{(\log l + \log m) l m^N}{\log l + N \log m}})$.

If $l < m$ the node complexity can be simplified to $O(\sqrt{\frac{l m^N}{N}})$.

*Proof:* The state transition and output function of a FRAO may be viewed (see Section II.C) as a Boolean Function with input dimension $x = \log l + N \log m$ and output dimension $y = \log l + \log m$. Thus, by applying Lemma 1 and absorbing the term $\log(\log l + \log m)$ into $(\log l + N \log m)$, the stated bound is obtained. □

### C. Networks with Three Computational Layers

If the network is constrained to only have three computational layers, the following theorem holds.

*Theorem III.2 (Three Layers Insertion):* Any FRAO with $m$ states, $l$ input–output labels, and maximum rank $N$, can be implemented by a recursive neural network with three computational layers and a node complexity of $O((\log l + \log m)\sqrt{l m^N})$ and a number of connections of $O((\log l + \log m) l m^N)$.

*Proof:* The three layers insertion theorem can be proved by exploiting the so-called *telescopic technique* which was used to build efficient feedforward networks for the computation of symmetric functions (by Siu *et al.* [24], [23]). This technique is based on Lemma 3. The basic idea is to exploit this lemma by mapping each distinct input configuration $c_j = (u^j, x^{(j,1)}, \cdots, x^{(j,N)})$ into a distinct integer $n_j$. Any injective map (i.e., a one-to-one map) from the input space $\{0,1\}^{\log l + N \log m}$ to the integer interval $[0, l m^N - 1]$ will suffice. Specifically, given the input bits $b_0, \cdots, b_\phi$, where $\phi = \log l + N \log m - 1$, we consider the standard binary encoding of integer numbers

$$e(b_0, \cdots, b_\phi) = \sum_{i=0}^{\phi} b_i 2^i.$$

On the basis of this encoding, each output bit $j$ ($j \in [0, \cdots, \log l + \log m - 1]$) can be expressed as a Boolean function $\mathcal{F}_j$ of $e(b_0, \cdots, b_\phi)$. This means that each $\mathcal{F}_j$ can be characterized by a set of $I_j$ disjoint subintervals in $[0, l m^N - 1]$, say $X_1^j = [\xi_1^j, \tilde{\xi}_1^j]$, $X_2^j = [\xi_2^j, \tilde{\xi}_2^j], \cdots, X_{I_j}^j = [\xi_{I_j}^j, \tilde{\xi}_{I_j}^j]$, where (for $k = 1, \cdots, I_j$) $\xi_k^j$ and $\tilde{\xi}_k^j$ are integers, $\xi_{k+1}^j > \tilde{\xi}_k^j + 1$, and $\xi_k^j \leq \tilde{\xi}_k^j$, such that $\mathcal{F}_j(b_0, \cdots, b_\phi) = 1$ if and only if for some $k$, $\xi_k^j \leq e(b_0, \cdots, b_\phi) \leq \tilde{\xi}_k^j$. Note that the number of these intervals for a given $j$ can be at most $\lceil \frac{l m^N}{2} \rceil$.

Let us consider a given $j$ and the realization of the corresponding Boolean function $\mathcal{F}_j$. Since $j$ is fixed, for avoiding a cumbersome notation, in the following, we omit explicit reference to the $j$ index when the presence of the index is clear from the context. To optimize the number of units to be used in the realization of $\mathcal{F}_j$, it is convenient to partition the $[0, l m^N - 1]$ interval into $\sigma$ subintervals $\Omega_p = [\xi_{(p-1)\eta+1}, \xi_{p\eta+1} - 1]$ $(p = 1, \cdots, \sigma)$, i.e., $\Omega_1 = [\xi_1, \xi_{\eta+1} - 1]$, $\Omega_2 = [\xi_{\eta+1}, \xi_{2\eta+1} - 1], \cdots, \Omega_\sigma = [\xi_{(\sigma-1)\eta+1}, l m^N - 1]$, each containing the same number $\eta$ of intervals $X_i$, where $\eta \leq \lceil \frac{l m^N}{2\sigma} \rceil$. Thus, the $p$-th subinterval $\Omega_p$, will contain the intervals $X_{(p-1)\eta+1}$, $X_{(p-1)\eta+2}, \cdots$, $X_{p\eta}$ (see Fig. 3).

The first layer of neurons will contain $\sigma$ units computing the values $x_p$, where for $p = 1, \cdots, \sigma$

$$x_p = \mathrm{sgn}\left( \sum_{y=0}^{\phi} b_y 2^y - \xi_p \right).$$

The second layer of neurons consists of $2\eta$ units computing the two sets of values $S_h$ and $s_h$ $(h = 1, \cdots, \eta)$, defined as

$$S_h = \mathrm{sgn}\left( T_h - \sum_{y=0}^{\phi} b_y 2^y \right), \quad s_h = \mathrm{sgn}\left( \sum_{y=0}^{\phi} b_y 2^y - t_h \right)$$

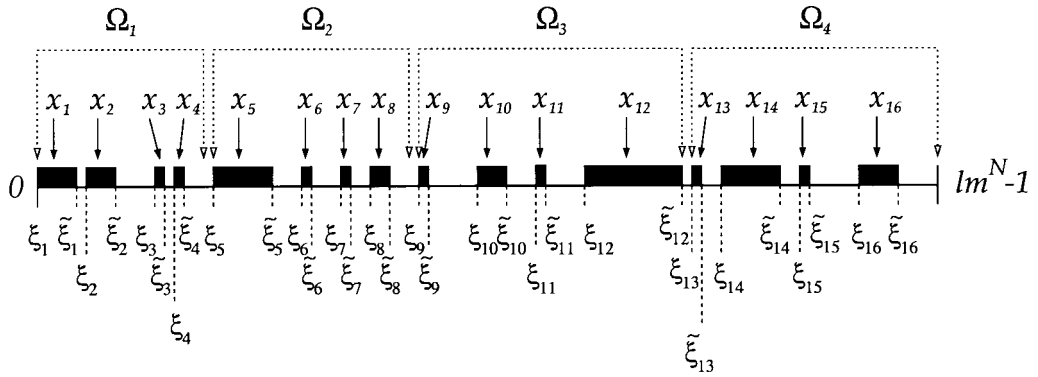


Fig. 3. Graphical visualization of the intervals and indices involved in the demonstration of the theorem for recursive networks with three computational layers. The black subintervals represent encoded input configurations for which the $\mathcal{F}_j$ Boolean function outputs one.

where $T_h$ and $t_h$ (i.e., two telescopic sums) are computed combining the outputs of the first layer as follows:

$$\begin{aligned} T_h &= \tilde{\xi}_h x_1 + (\tilde{\xi}_{\eta+h} - \tilde{\xi}_h)x_2 + (\tilde{\xi}_{2\eta+h} - \tilde{\xi}_{\eta+h})x_3 \\ &\quad + \cdots + (\tilde{\xi}_{(\sigma-1)\eta+h} - \tilde{\xi}_{(\sigma-2)\eta+h})x_\sigma, \\ t_h &= \xi_h x_1 + (\xi_{\eta+h} - \xi_h)x_2 + (\xi_{2\eta+h} - \xi_{\eta+h})x_3 \\ &\quad + \cdots + (\xi_{(\sigma-1)\eta+h} - \xi_{(\sigma-2)\eta+h})x_\sigma. \end{aligned}$$

Thus, the values $T_h$ and $t_h$ contain the $\eta$ upper and lower bounds of the fine-grained subintervals inside the coarse interval which is preselected by the first layer. $S_h$ and $s_h$ are conjunctively combined to implement the membership test on the fine-grained interval against the overall net input. The function $\mathcal{F}_j$ can then be computed (by one unit constituting the third layer) as

$$\mathcal{F}_j(b_0, \cdots, b_\phi) = \operatorname{sgn}\left(\sum_{h=1}^{\eta} 2(S_h + s_h) - 2\eta - 1\right).$$

The correctness of the above construction can be verified by observing that when $\sum_{y=0}^{\phi} b_y 2^y \in \Omega_k$, $k = 1, \cdots, \sigma$, by the Telescopic Lemma, we have $(h = 1, \cdots, \eta)$

$$T_h = \tilde{\xi}_{(k-1)\eta+h}, \quad t_h = \xi_{(k-1)\eta+h}.$$

Consequently, by definition of $S_h$ and $s_h$, we have that

$$S_h + s_h = \begin{cases} 2, & \text{if } h = i \quad \text{and} \quad \sum_{y=0}^{\phi} b_y 2^y \in X_{(k-1)\eta+i} \\ 1, & \text{otherwise} \end{cases}$$

Hence, when $\sum_{y=0}^{\phi} b_y 2^y \in X_{(k-1)\eta+i}$, the network correctly outputs one, i.e., $\operatorname{sgn}(\sum_{h=1}^{\eta} 2(S_h + s_h) - 2\eta - 1) = \operatorname{sgn}(2\eta + 2 - 2\eta - 1) = 1$. Otherwise, there is no $h$ such that $\sum_{y=0}^{\phi} b_y 2^y \in X_{(k-1)\eta+h}$, and the network outputs zero, i.e., $\operatorname{sgn}(\sum_{h=1}^{\eta} 2(S_h + s_h) - 2\eta - 1) = \operatorname{sgn}(2\eta + 1 - 2\eta - 1) = 0$.

The number of units in the network computing $\mathcal{F}_j$ is thus given by

$$\underbrace{\sigma}_{\text{1st layer}} + \underbrace{2\eta}_{\text{2nd layer}} + \underbrace{1}_{\text{3rd layer}}.$$

Since $\eta \leq \lceil \frac{lm^N}{2\sigma} \rceil$, the total number of units is at most $\sigma + \lceil \frac{lm^N}{\sigma} \rceil + 1$. The minimum for this function is obtained by setting $\sigma = \lceil \sqrt{lm^N} \rceil$. In conclusion, all the functions $\mathcal{F}_j$ can be computed by a network having $\log l + \log m$ subnetworks, each of $2\sqrt{lm^N} + 1$ units.

Regarding the number of connections, each subnetwork implementing a single $\mathcal{F}_j$ needs the following number of parameters

$$\underbrace{2(\log l + N \log m)\lceil \sqrt{lm^N} \rceil}_{\text{input}\to\text{1st layer+input}\to\text{2nd layer}} + \underbrace{\lceil \sqrt{lm^N} \rceil^2}_{\text{1st}\to\text{2nd layer}} + \underbrace{\lceil \sqrt{lm^N} \rceil}_{\text{2nd}\to\text{3rd layer}} + \underbrace{2\lceil \sqrt{lm^N} \rceil + 1}_{\text{thresholds}}.$$

Hence, the whole network needs $O((\log l + \log m)lm^N)$ connections. □

Note that the result stated in the theorem gives a direct upper bound to the insertion of FSA's in recurrent networks with three computational layers as well.

*Corollary 1:* Any FSA having $m$ states, and $l$ input–output labels, can be implemented by a recurrent neural network with three computational layers, a node complexity of $O((\log l + \log m)\sqrt{lm})$ and a number of connections of $O((\log l + \log m)lm)$.

Moreover, by setting $N = 1$, the proof of Theorem III.2 can be used as a constructive procedure for implementing the FSA.

### D. Networks with Two Computational Layers

It is known that networks with two computational layers have the computational power of FRAO's while networks with one layer cannot simulate arbitrary FRAO's ([15], [16]). We present two different ways to prove the following theorem.

*Theorem III.3 (Two Layers Implementation):* Any FRAO with $m$ states, $l$ input–output labels, and maximum rank $N$, can be implemented by a recursive neural network with two computational layers and a node complexity of $O(lm^N)$ and $O((\log l + N \log m)lm^N)$ connections.

*Proof (Alternative A):* Horne and Hush [12] argued that any Boolean function can be transformed into disjunctive normal form and both disjunctions and conjunctions can be implemented by a perceptron. Using Lemma 2 and noting that there can be at most $2^x$ conjunctions, we get a node complexity of $O(2^x + y) = O(lm^N)$ and at most $O((x + y)2^x) = O((\log l + N \log m)lm^N)$ connections. □

*Proof (Alternative B):* We modify the proof for Theorem III.2 by applying the Telescopic Lemma 3 to construct a one-level selection of the relevant subinterval.

Again, let $\mathcal{F}_j$ be characterized by a set of $I_j$ disjoint subintervals in $[0, lm^N - 1]$, say $X_1^j = [\xi_1^j, \tilde{\xi}_1^j], X_2^j = [\xi_2^j,$

$\tilde{\xi}_2^j], \cdots, X_{I_j}^j = [\xi_{I_j}^j, \tilde{\xi}_{I_j}^j]$, where (for $k = 1, \cdots, I_j$) $\xi_k^j$ and $\tilde{\xi}_k^j$ are integers, $\xi_{k+1}^j > \tilde{\xi}_k^j + 1$, and $\xi_k^j \leq \tilde{\xi}_k^j$, such that $\mathcal{F}_j(b_0, \cdots, b_\phi) = 1$ if and only if for some $k$, $\xi_k^j \leq e(b_0, \cdots, b_\phi) \leq \tilde{\xi}_k^j$.

The first layer consists of $I_j$ output gates, computing $x_k = \operatorname{sgn}(\sum_{y=0}^{\phi} b_y 2^y - \xi_k^j)$ for each $k, 1 \leq k \leq I_j$. The output unit of the second layer computes

$$\mathcal{F}_j(b_0, \cdots, b_\phi) = \operatorname{sgn}\left(\sum_{k=1}^{I_j} (\tilde{\xi}_k^j - \tilde{\xi}_{k-1}^j) x_k - \sum_{y=0}^{\phi} b_y 2^y\right).$$

It can be easily verified that the above construction works as desired. Assume $\sum_{y=0}^{\phi} b_y 2^y \in X_p$ for some $p$, $1 \leq p \leq I_j$. It follows from the Telescopic Lemma 3 that

$$\sum_{k=1}^{I_j} (\tilde{\xi}_k^j - \tilde{\xi}_{k-1}^j) x_k = \tilde{\xi}_p^j$$

where we define $\tilde{\xi}_0^j = 0$. If $\mathcal{F}_j(b_0, \cdots, b_\phi) = 1$, then $\xi_p^j \leq \sum_{y=0}^{\phi} b_y 2^y \leq \tilde{\xi}_p^j$ and hence, $\operatorname{sgn}(\tilde{\xi}_p^j - \sum_{y=0}^{\phi} b_y 2^y) = 1$. If $\mathcal{F}_j(b_0, \cdots, b_\phi) = 0$, then $\tilde{\xi}_p^j < \sum_{y=0}^{\phi} b_y 2^y \leq \xi_{p+1}^j - 1$ and hence, $\operatorname{sgn}(\tilde{\xi}_p^j - \sum_{y=0}^{\phi} b_y 2^y) = 0$.

The number of gates in the circuit is $I_j + 1$. Since $I_j$ can be at most $\lceil \frac{lm^N}{2} \rceil$ all the function $\mathcal{F}_j$ can be computed by a network having $\log l + \log m$ subnetworks, yielding a node complexity of $O((\log l + \log m) l m^N)$ and a number of connections of $O((\log^2 l + N \log^2 m) l m^N)$.

However, the hidden nodes (there can be at most $\lceil \frac{lm^N}{2} \rceil$) can be shared among all functions $\mathcal{F}_j$ which improves the node complexity to $O(lm^N + \log l + \log m) = O(lm^N)$ and the number of connections to $O((\log l + N \log m) l m^N)$. □

Note that Proof (Alternative B) gives also a construction scheme to implement FSA into recurrent networks with two computational layers:

*Corollary 2:* Any FSA having $m$ states, and $l$ input–output labels, can be implemented by a recurrent neural network with two computational layers and a node complexity of $O(lm)$ and a number of connections of $O((\log l + \log m) l m)$.

*Remarks:* The usage of sigmoid neurons might lead to a further reduction in the node complexity. This hypothesis is supported by the observation that for certain Boolean functions the size of the implementing network can be reduced by at least a logarithmic factor by using continuous (e.g., sigmoid) instead of threshold units [23].

## IV. A Lower Bound on the Node Complexity

Next we will deal with the questions about the minimum number of neurons that are required to implement arbitrary FRAO into recursive networks.

The following definition specifies the case when two automata are considered as "really" different.

*Definition 2:* Two FRAO $\mathcal{A}_t^j = (\mathcal{U}^j, \mathcal{O}^j, \mathcal{X}^j, \delta^j, \omega^j, x_f^j)$ $(j = 1, 2)$ with maximum rank $N$ are *divergent* if there exists an input tree $t$ for which there exists a node in $t$, labeled $u$, such that its direct offsprings are assigned the states $x_1^j, \cdots, x_N^j$ by the two machines, and

$$\omega_u(x_1^1, \cdots, x_N^1) \neq \omega_u(x_1^2, \cdots, x_N^2)$$

i.e., the two automata respond with two different output trees (i.e., differing at least for the output generated in correspondence of the node labeled $u$) on the same input.

A first naive consideration would lead to $\Omega(\log l + \log m)$ since $l$ different input symbols and $m$ states can effectively be encoded by $\log l + \log m$ units.

Here we use the techniques shown by Alon *et al.* [13] and Horn and Hush [14] (lower bounds on the node complexity for recurrent network implementations of FSA) to derive a lower bound for the node complexity of recursive neural network implementations of FRAO.

Let $K(m)$ be the smallest number such that every FRAO with $m$ or less states can be implemented by a recursive network using $K(m)$ or fewer neurons. Let $L(m)$ be the number of pairwise *divergent* (see Definition 2) FRAO with $m$ or fewer states and $U(z)$ be the number of different $m$-state FRAO that can be built from a recursive network using $z$ neurons. Obviously

$$U(K(m)) \geq L(m) \tag{8}$$

and by deriving a good upper bound for $U(z)$ and a lower bound $L(m)$ we are able to compute a lower bound for $K(m)$. We begin with a simple extension of a result proved by Horne and Hush ([12] and [14, Proof of Theorem 2]; for technical details see [16, Lemma 8]).

*Lemma 4:* The number $U(z)$ of different $m$-state FRAO (with $l$ input–output labels and maximum rank $N$) that can be built from a recursive network using $z$ neurons can be bounded to $O(y 2^{xz^2})$, where $x = \log l + N \log m$ and $y = \log l + \log m$.

The number $L(m)$ of pairwise divergent $m$-state FRAO can be bounded according to the following lemma.

*Lemma 5 (Number of Divergent FRAO):* If $m$ is prime then there exists a system of $L(m) = (lm)^{(l-1)m^N} (l^{m^N} - l^{m^{N-1}})/m$ (pairwise) divergent $m$-state FRAO's with $l$ input–output labels and maximum rank $N$.

*Proof:* Assume $m$ being prime and consider the following system of $m$-state FRAO's, defined by

1) $\delta_{u_0}(x_{q_1}, \cdots, x_{q_N}) = x_{(q_N+1) \bmod m}$;
2) given any states $x_{q_1}, \cdots, x_{q_{N-1}} \in \mathcal{X}$, $\omega_{u_0}(x_{q_1}, \cdots, x_{q_{N-1}}, x_0), \cdots, \omega_{u_0}(x_{q_1}, \cdots, x_{q_{N-1}}, x_{m-1})$, are not all equal;
3) for $j = 1, \cdots, l-1$,
   $\delta_{u_j}(x_{q_1}, \cdots, x_{q_N})$ arbitrary,
   $\omega_{u_j}(x_{q_1}, \cdots, x_{q_N})$ arbitrary.

First of all, the system of $m$-state FRAO's contains $(lm)^{(l-1)m^N} (l^{m^N} - l^{m^{N-1}})/m$ different automata since, for input labels different by $u_0$, the number of different functions from a domain of $(l-1)m^N$ elements to one of $ml$ elements is exactly $(lm)^{(l-1)m^N}$. Moreover, we have to consider the number of different functions which satisfy condition 2). This number is equal to the number of functions from a domain of $m^N$ elements to a domain of $l$ elements, minus the number of functions which violates condition 2), which can easily be demonstrated to be $l^{m^{N-1}}$. Finally, the factor $m^{-1}$ is justified by the fact that if two automata are not divergent, then they are identical, provided that the states $x_{q_j}$ of one are relabeled

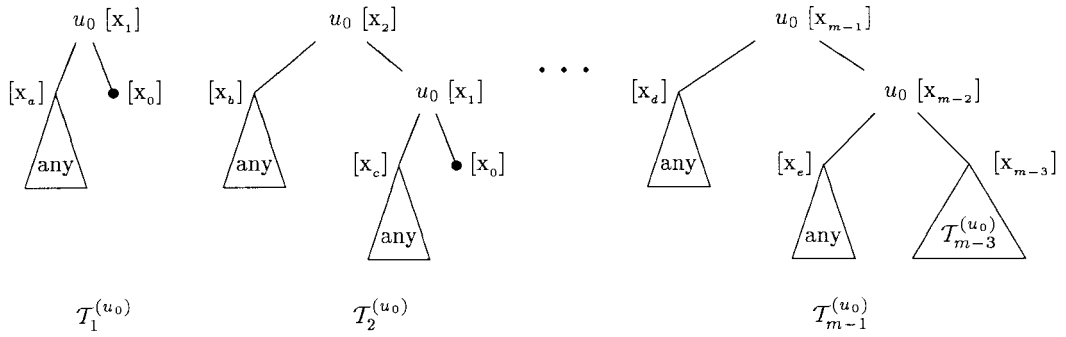


Fig. 4. Examples of $\mathcal{T}_j^{(u_0)}$ sets for binary trees.

as states $x_{q_{(j+d) \bmod m}}$ of the other for a suitable value of $d$ $(0 \leq d \leq m-1)$.

Suppose automata $\mathcal{A}_t^{(1)}$ and $\mathcal{A}_t^{(2)}$ belong to the class of FRAO's defined above and that they are not divergent, i.e., there exist states $x^{(1)}$ and $x^{(2)}$ such that if state $x^{(s)}$ $(s=1,2)$ is used as frontier state, then, given the same (arbitrary) tree as input, both automata generate the same output tree. Let us rename the internal states of both automata, so that $x^{(1)} = x^{(2)} = x_0$, and recursively define the sets of trees $\mathcal{T}_j^{(u_0)}$, $j = 0, \cdots, m-1$, in the following way:

$(j=0)$: $\mathcal{T}_0^{(u_0)}$ contains the void tree;

$(j=1, \cdots, m-1)$: $\mathcal{T}_j^{(u_0)}$ is the set of trees where the root is labeled $u_0$ and whose rightmost subtree belongs to $\mathcal{T}_{j-1}^{(u_0)}$;

In Fig. 4, examples of $\mathcal{T}_j^{(u_0)}$ sets are shown for binary trees.

By definition of the class of FRAO, it follows that giving in input to both $\mathcal{A}_t^{(1)}$ and $\mathcal{A}_t^{(2)}$ a tree belonging to $\mathcal{T}_j^{(u_0)}$, the final state for both will be $x_j$. Moreover, since the two automata are supposed not to be divergent and since $j \in [0, \cdots, m-1]$, for each $(x_{q_1}, \cdots, x_{q_N}) \in \mathcal{X}^N$

$$\begin{aligned} \omega_{u_0}^{(1)}(x_{q_1}, \cdots, x_{q_N}) &= \omega_{u_0}^{(2)}(x_{q_1}, \cdots, x_{q_N}) \\ &= \omega_{u_0}(x_{q_1}, \cdots, x_{q_N}). \end{aligned}$$

Now consider the families $\mathcal{F}_j^{u_k}$ of trees which have the root labeled $u_k$ $(k = 1, \cdots, l-1)$, a tree in $\mathcal{T}_j^{(u_0)}$ $(j = 0, \cdots, m-1)$ as rightmost subtree of the root, and arbitrary subtrees for the remaining $N-1$ subtrees of the root. By presenting these trees to both $\mathcal{A}_t^{(1)}$ and $\mathcal{A}_t^{(2)}$, it follows that, for each $(x_{q_1}, \cdots, x_{q_N}) \in \mathcal{X}^N$

$$\begin{aligned} \omega_{u_j}^{(1)}(x_{q_1}, \cdots, x_{q_N}) &= \omega_{u_j}^{(2)}(x_{q_1}, \cdots, x_{q_N}) \\ &= \omega_{u_j}(x_{q_1}, \cdots, x_{q_N}). \end{aligned}$$

Finally, consider any $j_0 \in [0, \cdots, m-1]$ and a tree $t$ obtained as follows:

1) select any tree $t_{\text{top}} \in \mathcal{T}_j^{(u_0)}$ for some $j \neq 0$;
2) attach to the rightmost node of $t_{\text{top}}$ as the rightmost subtree (which is void in $t_{\text{top}}$) any tree $t_{\text{bottom}} \in \mathcal{F}_{j_0}^{u_k}$.

When presenting the rightmost subtree of root$(t_{\text{bottom}})$ to $\mathcal{A}_t^{(1)}$ and $\mathcal{A}_t^{(2)}$, both machines assign state $x_{j_0}$ to the root of the subtree, and arbitrary states $x_{q_0}, \cdots, x_{q_{N-2}}$ to the roots of the remaining $N-1$ subtrees. Then, after presenting $t_{\text{bottom}}$, $\mathcal{A}_t^{(1)}$ will assign state $x_y = \delta_{u_k}^{(1)}(x_{q_0}, \cdots, x_{q_{N-2}}, x_{j_0})$ to root$(t_{\text{bottom}})$, while $\mathcal{A}_t^{(2)}$ will assign state $x_z = \delta_{u_k}^{(2)}(x_{q_0}, \cdots, x_{q_{N-2}}, x_{j_0})$. Finally, after presenting $t$ to both machines, $\mathcal{A}_t^{(1)}$ will assign state $x_{(y+j) \bmod m}$ to root$(t)$, while $\mathcal{A}_t^{(2)}$ will assign, to the same node, state $x_{(z+j) \bmod m}$. Let $\Delta = (z-y) \bmod m$. We will show that $\Delta = 0$, thus proving the theorem.

By definition of $\delta_{u_0}(\cdot)$, and since $\mathcal{A}_t^{(1)}$ and $\mathcal{A}_t^{(2)}$ are supposed not to be divergent

$$\begin{aligned} &\omega_{u_0}(x_{q_1}, \cdots, x_{q_{(y+j) \bmod m}}) \\ &\quad = \omega_{u_0}(x_{q_1}, \cdots, x_{q_{(y+\Delta+j) \bmod m}}), \qquad \text{for any value of } j. \end{aligned} \tag{9}$$

Specifically

$$\begin{aligned} \omega_{u_0}(x_{q_1}, \cdots, x_{q_y}) &= \omega_{u_0}(x_{q_1}, \cdots, x_{q_{(y+\Delta) \bmod m}}) \\ &= \omega_{u_0}(x_{q_1}, \cdots, x_{q_{(y+2\Delta) \bmod m}}) = \cdots \end{aligned} \tag{10}$$

which can be summarized by

$$\begin{aligned} &\omega_{u_0}(x_{q_1}, \cdots, x_{q_y}) \\ &\quad = \omega_{u_0}(x_{q_1}, \cdots, x_{q_{(y+s\Delta) \bmod m}}) \qquad \text{for all integer } s. \end{aligned} \tag{11}$$

If we suppose that $\Delta \neq 0$, since $m$ is prime, the above equation implies that for any state $x \in \mathcal{X}$, $\omega_{u_0}(x_{q_1}, \cdots, x) = \omega_{u_0}(x_{q_1}, \cdots, x_{q_y})$, which contradicts equation 2) of the FRAO's family definition. This means that $\Delta = 0$ and $\delta_{u_j}^{(1)}(\cdot) = \delta_{u_j}^{(2)}(\cdot)$ for each $j$, i.e., both automata have identically defined transition functions. This together with the assumption that both machines are not divergent implies that both machines also must have identically defined output functions which is a contradiction to their definition and completes the proof. □

We are now ready to formulate the central proposition of this section.

*Theorem IV.1 (Lower Bound, Node Complexity):* The node complexity required to implement an arbitrary FRAO with $m$ states, $l$ input–output labels and maximum rank $N$ in a recursive neural network is $\Omega(\sqrt{\frac{(\log l + \log m) l m^N}{\log l + N \log m}})$.

*Proof:* By combining the results of Lemma 4 with Lemma 5 according to (8) we get

$$cy 2^{xz^2} \geq (l^{m^N} - l^{m^{N-1}})(lm)^{(l-1)m^N} > (lm)^{(l-1)m^N}$$

where $c$ is a suitable constant, $x = \log l + N \log m$, $y = \log l + \log m$ and $z = K(m)$. By substituting $x$ and $y$ and simplifying the above equation we can find positive constants

TABLE I
SUMMARY OF RESULTS ON THE UPPER AND LOWER BOUNDS OF THE NODE COMPLEXITY OF RECURSIVE NEURAL NETWORK IMPLEMENTATIONS OF FRONTIER-TO-ROOT AUTOMATA WITH $m$ STATES, $l$ INPUT–OUTPUT LABELS, AND MAXIMUM RANK $N$. THE LAST ROW SHOWS THAT NODE COMPLEXITY PROPOSITIONS ON THE IMPLEMENTATION OF FINITE-STATE AUTOMATA IN RECURRENT NEURAL NETWORKS ARE OBTAINED AS A SPECIAL CASE ($N = 1, l = 2$). * DENOTES PREVIOUSLY KNOWN RESULTS [14] NOT DERIVED IN THIS PAPER. FOR RECURRENT NETWORKS ALL THE RESULTS, INCLUDING THE ONE DEMONSTRATED IN THIS PAPER, ARE OBTAINED USING HARD THRESHOLD UNITS AND BINARY ALPHABETS

| # layers | recursive networks | recurrent networks |
|---|---|---|
| arbitrary | $\Omega\left(\sqrt{\frac{(\log l+\log m)lm^N}{\log l+N\log m}}\right)$ | $\Omega(\sqrt{m})$ * |
| 4 | $O\left(\sqrt{\frac{(\log l+\log m)lm^N}{\log l+N\log m}}\right)$ | $O(\sqrt{m})$ * |
| 3 | $O((\log l+\log m)\sqrt{lm^N})$ | $O((\log m)\sqrt{m})$ |
| 2 | $O(lm^N)$ | $O(m)$ * |

$c,'$ $m_0$, $l_0$ such that for $m \geq m_0$, $l \geq l_0$

$$z > \sqrt{\frac{(l-1)m^N(\log l+\log m)-\log c-\log(\log l+\log m)}{(\log l+N\log m)}}$$
$$\geq c'\sqrt{\frac{(\log l+\log m)lm^N}{\log l+N\log m.}} \qquad \square$$

## V. CAN WE IMPROVE THE THREE LAYERS CONSTRUCTION?

The Three layers theorem (see Section III-C) is based on the construction of a network which is actually composed of a set of $\log l+\log m$ disjoint subnetworks, each computing a different bit of output. One possibility to improve the construction is to build subnetworks which share hidden units (as presented by the construction involving two layers in Section III-D). However, even using disjoint subnetworks, it must be noted that the complexity of the construction depends on the numbers $I_j$, $j=0,\cdots,\log l+\log m-1$, i.e., the number of intervals which characterize the $\mathcal{F}_j$ functions, and that these numbers depend on which binary code we have chosen for representing the states of the automaton. Hence, the question is whether can we control the size of the numbers $I_j$ by choosing a suitable binary encoding of the automaton states.

*Definition 3 (Optimal Coding Problem):* Given a state assignment

$$\mathcal{A}=\left\{\left(x^i,\beta_0^i\beta_1^i\cdots\beta_{\log m-1}^i\right)\right\}_{i=1,\cdots,m}$$

where at each state $x^i$ is assigned a different binary string $\beta_0^i\beta_1^i\cdots\beta_{\log m-1}^i$, we want to solve the following minimization problem:

$$\arg\min_{(\mathcal{A},\pi(b_0b_1\cdots b_\phi))} \sum_{j=0}^{\log l+\log m-1} \times\sqrt{\sum_{i=0}^{lm^N-1}\mathcal{F}_j\left(e_\pi^{-1}(i)\right)\left(1-\mathcal{F}_j\left(e_\pi^{-1}(i-1)\right)\right)} \qquad (12)$$

where $\mathcal{A}$ is any valid state assignment, $\pi(b_0b_1\cdots b_\phi)$ is any permutation of the input binary string[4], $e_\pi^{-1}(i)$ is the function that given an integer $i$ returns its binary representation according to the indexes order given by permutation $\pi$, and $\forall j$, $\mathcal{F}_j(e_\pi^{-1}(-1))=0$. Note that, given a state assignment $\mathcal{A}$ and a permutation $\pi$, $\sum_{i=0}^{lm^N-1}\mathcal{F}_j(e_\pi^{-1}(i))(1-\mathcal{F}_j(e_\pi^{-1}(i-1)))$ is exactly $I_j$.

Unfortunately, the optimal coding problem is very difficult to solve since it involves a search among all the possible permutations of the input bits. We conjecture that the problem is NP-complete.

## VI. CONCLUSION

We have given upper bounds on the necessary number of computational units in recursive networks, with four, three, and two computational layers, required to implement *frontier-to-root tree automata*. The bound we give for networks with three (two) computational layers is very interesting, since it is constructive and may suggest practical ways for inserting tree automata into recursive networks. This could be used for refining, through learning on a set of examples, the inserted automaton. From a more general point of view, in this paper we have shown that working in a more general setting, i.e., considering trees as input domain, yields automatically results on sequences as a special case (see also Table I). So we obtained a novel constructive upper bound on implementing finite-state automata into three layers recurrent neural networks as a special case of the more general three layers bound. Finally, we argued that the three layers construction may actually be improved by solving the *optimal coding problem* which, however, we conjecture to be NP-complete. An alternative way of improving the construction would be to have the hidden units to contribute to the computation of each output unit instead of a single one, or to use sigmoidal units. Especially a better encoding of the automaton states can lead to a relevant improvement in the construction of automata belonging to a restricted family. Hidden units sharing can further improve the complexity, however, for the moment it is not clear to these authors how the telescopic technique should be modified in order to take advantage of the sharing.

Two major questions still remain to be addressed, i.e., the precision in the weight representation and the possibility to *learn* the type of networks we have described here. Concerning the first point, the construction we suggested for recursive networks with three computational layers, seems to be robust to the precision of the weight representation since the larger the number of states, the larger the interval of integers used for representing them is. This is true as long as the errors in weight representation can be modeled by a Gaussian process with zero mean. Of course, if the errors are correlated, the cumulated error can lead the telescopic implementation to fail. The second point is more tricky, since the demonstration that there exists a recursive network able to represent a give FRAO does not mean that that network can be easily learned, if not at all. The construction we gave is only applicable

[4]Recall that the input string is obtained by concatenating the $\log l$ bits representing the input label with the $N\log m$ bits representing the input offspring states coded according to $\mathcal{A}$. So $\phi=\log l+N\log m$.

if the FRAO is completely specified. However, it can be used to insert *a priori* knowledge into the recursive network if some structural information about the desired FRAO is known. It remains an open problem to judge whether training a preconfigured network can lead to a network able to obtain good generalization performances.

## REFERENCES


[1] J. B. Pollack, "Recursive distributed representations," *Artificial Intell.*, vol. 46, nos. 1–2, pp. 77–106, 1990.

[2] A. Sperduti, "Labeling RAAM," *Connection Sci.*, vol. 6, no. 4, pp. 429–459, 1994.

[3] A. Sperduti, A. Starita, and C. Goller, "Learning distributed representations for the classification of terms," in *Proc. Int. Joint Conf. Artificial Intell.*, 1995, pp. 509–515.

[4] C. Goller and A. Küchler, "Learning task-dependent distributed structure-representations by backpropagation through structure," in *IEEE Int. Conf. Neural Networks*, 1996, pp. 347–352.

[5] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Trans. Neural Networks*, vol. 8, pp. 714–735, 1997.

[6] A. Sperduti, "Neural networks for processing data structures," in *Adaptive Processing of Sequences and Data Structures, Lecture Notes in Artificial Intelligence*, M. Gori and C. L. Giles, Eds., vol. 1387. New York: Springer-Verlag, ch. 5, 1998, pp. 121–144.

[7] P. Fasconi, "An introduction to learning structured information," in *Adaptive Processing of Sequences and Data Structures, Lecture Notes in Artificial Intelligence*, M. Gori and C. L. Giles, Eds., vol. 1387 New York: Springer-Verlag, ch. 4, 1998, pp. 99–120.

[8] A. M. Bianucci, A. Micheli, A. Sperduti, and A. Starita, "Quantitative structure-activity relationships of benzodiazepines by recursive cascade correlation," in *Proc. 1998 IEEE Int. Joint Conf. Neural Networks*, 1998, pp. 117–122.

[9] T. Schmitt and C. Goller, "Relating chemical structure to activity: An application of the neural folding architecture," in *Proc. 5th Int. GI-Wkshp. Fuzzy-Neuro Syst.'98*, 1998.

[10] E. Francesconi, P. Frasconi, M. Gori, S. Marinai, J. Q. Sheng, G. Soda, and A. Sperduti, "Logo recognition by recursive neural networks," in *Proc. 2nd IAPR Wkshp. Graphics Recognition (GREC'97)*, 1997, pp. 144–151.

[11] C. Goller, "A connectionist approach for learning search-control heuristics for automated deduction systems," Ph.D. dissertation, Tech. Univ. Munich, Germany, Faculty Comput. Sci., 1997.

[12] B. G. Horne and D. R. Hush, "On the node complexity of neural networks," *Neural Networks*, vol. 7, no. 9, pp. 1413–1426, 1994.

[13] N. Alon, A. K. Dewdney, and T. J. Ott, "Efficient simulation of finite automata by neural nets," *J. ACM*, vol. 38, no. 2, pp. 495–514, Apr. 1991.

[14] B. G. Horne and D. R. Hush, "Bounds on the complexity of recurrent neural-network implementations of finite-state machines," *Neural Networks*, vol. 9, no. 2, pp. 243–252, 1996.

[15] A. Sperduti, "On the computational power of recurrent neural networks for structures," *Neural Networks*, vol. 10, no. 3, pp. 395–400, 1997.

[16] A. Küchler, "On the correspondence between neural folding architectures and tree automata," Faculty Comput. Sci., Univ. Ulm, Germany, Ulmer Informatik-Berichte 98-06, May 1998.

[17] F. Gécseg and M. Steinby, *Tree Automata*, Akadémiai Kiadó, Budapest, Hungary, 1984.

[18] F. Gécseg and M. Steinby, "Tree languages," in *Handbook of Formal Languages*, G. Rozenberg and A. Salomaa, Eds., vol. 3 New York: Springer-Verlag, pp. 1–68, 1996.

[19] J. W. Shavlik, "Combining symbolic and neural learning," *Machine Learning*, vol. 14, pp. 321–331, 1994.

[20] C. W. Omlin and C. L. Giles, "Constructing deterministic finite-state automata in recurrent neural networks," *J ACM*, vol. 43, no. 2, pp. 937–972, 1996.

[21] C. W. Omlin and C. L. Giles, "Stable encoding of large finite-state automata in recurrent neural networks with sigmoid discriminants," *Neural Comput.*, vol. 8, no. 4, pp. 675–693, 1996.

[22] P. Frasconi, M. Gori, and G. Soda, "Recurrent neural networks and prior knowledge for sequence processing: A constrained nondeterministic approach," *Knowledge-Based Syst.*, vol. 8, no. 6, pp. 313–332, 1995.

[23] K.-Y. Siu, V. Roychowdhury, and T. Kailath, *Discrete Neural Computation*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[24] K.-Y. Siu, V. Roychowdhury, and T. Kailath, "Depth-size tradeoffs for neural computation," *IEEE Trans. Comput.*, vol. 40, pp. 1402–1412, Dec. 1991.

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

[26] O. Lupanov, "Circuits using threshold elements," *Sov. Phys.—Doklady*, vol. 17, no. 2, pp. 91–93, 1972.


**Marco Gori** (S'88–M'91–SM'97) received the Laurea degree in electronic engineering from Università di Firenze, Italy, in 1984, and the Ph.D. degree in 1990 from Università di Bologna, Italy.

From October 1988 to June 1989 he was a Visiting Student at the School of Computer Science, McGill University, Montreal, Canada. In 1992, he became an Associate Professor of Computer Science at Università di Firenze and, in November 1995, he joint the University of Siena, Italy. His main research interests include pattern recognition (especially document processing) and neural networks (especially links with symbolic models).

Dr. Gori was the general chairman of the Second Workshop of Neural Networks for Speech Processing held in Firenze in 1992, organized the NIPS'96 postconference workshop on "Artificial Neural Networks and Continuous Optimization: Local Minima and Computational Complexity," and coorganized the Caianiello Summer School on "Adapting Processing of Sequences" held in Salerno (IT) on September 1997. He serves as a Program Committee member of several workshops and conferences mainly in the area of Neural Networks. He acted as Guest Coeditor of the Neurocomputing Journal for the special issue on recurrent neural networks (July 1997). He is an Associate Editor of many journals including the IEEE TRANSACTIONS ON NEURAL NETWORKS, *Pattern Recognition*, *Neurocomputing*, and *Neural Computing Survey*. He is the Italian chairman of the IEEE Neural Network Council (R.I.G.) and is a member of the Executive Board of the European Neural Networks Council.

**Andreas Küchler** received the M.Sc. degree in computer science from the University of Erlangen, Germany, in 1994. Currently, he is a Ph.D. candidate at the Faculty of Computer Science, University of Ulm, Germany.

From 1994 to 1999, he held a position as Research Scientist at the Department of Neural Information Processing, University of Ulm, Germany. His research interests include neural networks, inductive grammatical inference, classification and regression of structural data, and the combination of analog and discrete models of computation and learning.

**Alessandro Sperduti** (M'98) received the Laurea and Doctoral degrees in 1988 and 1993, respectively, all in computer science, from the University of Pisa, Italy.

In 1993 he was at the International Computer Science Institute, Berkeley, supported by a postdoctoral fellowship. In 1994 he moved back to the Computer Science Department, University of Pisa, where he was Assistant Professor, and is presently Associate Professor. His research interests include pattern recognition, image processing, neural networks, and hybrid systems. In the field of hybrid systems his work has focused on the integration of symbolic and connectionist systems. He contributed to the organization of several workshops on this subject and he served also in the program committee of conferences on Neural Networks. He is the author of more than 60 refereed papers mainly in the areas of neural networks, fuzzy systems, pattern recognition, and image processing. Moreover, he gave several tutorials within international schools and conferences, such as IJCAI'97 and IJCAI'99. He acted as Guest Coeditor of the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING for a special issue on Connectionist Models for Learning in Structured Domains.