Simple strategies to encode tree automata in sigmoid recursive neural networks

Rafael C. Carrasco and Mikel L. Forcada[*]
{carrasco,mlf}@dlsi.ua.es
Departament de Llenguatges i Sistemes Informàtics
Universitat d'Alacant, E-03071 Alacant (Spain)
Fax: +34-96-5909326
Tel.: +34-96-5903772

November 2, 2001

Abstract

Recently, a number of authors have explored the use of recursive recursive neural nets (RNN) for the adaptive processing of trees or tree-like structures. One of the most important language-theoretical formalizations of the processing of tree-structured data is that of deterministic finite-state tree automata (DFSTA). DFSTA may easily be realized as RNN using discrete-state units such as the threshold linear unit. A recent result by Šíma (*Neural Network World* **7**(1997)679–686) shows that any threshold linear unit operating on binary inputs can be implemented in an analog unit using a continuous activation function and bounded real inputs. The constructive proof finds a scaling factor for the weights and re-estimates the bias accordingly. In this paper, we explore the application of this result to simulate DFSTA in sigmoid RNN (that is, analog RNN using monotonically growing activation functions), and also present an alternative scheme for *one-hot*

[*]Corresponding author

encoding of the input that yields smaller weight values and therefore works at a lower saturation level.

Keywords: tree automata, recursive neural networks, neural computation, analog neural networks

1 Introduction

During the last decade, a number of authors have explored the use of analog recursive neural nets (RNN) for the adaptive processing of data laid out as trees or tree-like structures such as directed acyclic graphs. In this arena, Frasconi, Gori and Sperduti [5] have recently established a rather general formulation of the adaptive processing of structured data, which focuses on directed ordered acyclic graphs (which includes trees); Sperduti and Starita [18] have studied the classification of structures (directed ordered graphs, including cyclic graphs) and Sperduti [18] has studied the computational power of recursive neural nets as structure processors.

One of the most important language-theoretical formalizations of the processing of tree-structured data is that of deterministic finite-state tree automata (DFSTA), also called deterministic *frontier-to-root* or *ascending* tree automata[19, 7]. DFSTA may easily be realized as RNN using discrete-state units such as the threshold linear unit (TLU). Sperduti, in fact, [17] has recently shown that Elman-style [3] RNN using TLU may simulate DFSTA, and provides an intuitive explanation (similar to that expressed by Kremer [10] for the special case of deterministic finite automata) why this should also work for sigmoid networks: incrementing the gain of the sigmoid function should lead to an arbitrarily precise simulation of a step function. We are, however, unaware of any attempt to establish a finite value of this gain such that exact simulation of a DFSTA may indeed be performed by an analog RNN.

A recent result by Šíma [16] shows that any TLU operating on binary inputs can be simulated by an analog unit using a continuous activation function and bounded real inputs. A TLU is a neuron that computes its output by applying a threshold or step activation function

$$g_H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

to a biased linear combination of its binary inputs.

The corresponding analog neuron works with any activation function $g(x)$ having two different finite limits a and b when $x \to \pm\infty$ and for given input and output tolerances. The constructive proof finds a scaling factor for the weights —basically, a value for the gain of the analog activation function— and uses the same scaling factor on a shifted value of the bias. In this paper, we define three possible ways of encoding DFSTA in discrete-state RNN using TLU and then explore the application of Šíma's result to turn the discrete-state RNN into a sigmoid RNN simulating the original DFSTA (with *sigmoid* meaning an analog activation function that is monotonically growing). In addition, we present an alternative scheme for analog simulation that yields smaller weight values than Šíma's scheme for both discrete-state cases and therefore works at a lower saturation level; our goal is to find the smallest possible scaling factor guaranteeing correct behavior. This last approach, which assumes a *one-hot* encoding of inputs, is a generalization of the approach used in [2] for the stable encoding of a family of finite-state machines (FSM) in a variety of sigmoid discrete-time recurrent neural networks and similar in spirit to previous work by Omlin and Giles [13, 14] for deterministic finite automata (a class of FSM) and a particular discrete-time recurrent neural network (DTRNN) architecture (the second-order DTRNN used by Giles et al. [6]).

In the following section, tree automata and recursive networks are introduced. Section 3 describes three different schemes to encode recursive neural networks in discrete-state RNN using TLU. The main result by Šíma[16] is presented in Section 4 together with a similar construction for the case of exclusive (also called *one-hot*) encoding of the input. Section 5 describes the conversion of discrete-state RNN into and their sigmoid counterparts and the different schemes are evaluated by comparing the magnitude of the resulting weight values. Finally, we present our conclusions in the last section.

2 Tree automata and recursive neural networks

Before we explore how neural networks can simulate tree automata we need to specify the notation for trees and describe the architecture of recursive neural networks.

2.1 Trees and finite-state machines

We will denote with Σ a *ranked alphabet*, that is, a finite set of symbols $\Sigma = \{\sigma_1, ..., \sigma_{|\Sigma|}\}$ with an associated function $r : \Sigma \to \mathbb{N}$ giving the rank of the symbol.[1] The subset of symbols in Σ having rank m is denoted with Σ_m. The set of Σ-trees, Σ^T, is defined as the set of strings (made of symbols in Σ augmented with the parenthesis and the comma) representing ordered labeled trees or, recursively,

1. $\Sigma_0 \subset \Sigma^T$ (any symbol of rank 0 is a single-node tree in Σ^T).
2. $f(t_1, ..., t_m) \in \Sigma^T$ whenever $m > 0$, $f \in \Sigma_m$ and $t_1, ..., t_m \in \Sigma^T$ (a tree having a root node with a label of f rank m and m children $t_1 \dots t_m$ which are valid trees of Σ^T belongs to Σ^T).

[1] The rank may be defined more generally as a relation $r \subseteq \Sigma \times \mathbb{N}$; both formulations are equivalent if symbols having more than one possible rank are split.

A *deterministic finite-state tree automaton* (DFSTA) is a five-tuple $A = (Q, \Sigma, r, \Delta, F)$, where $Q = \{q_1, \dots, q_{|Q|}\}$ is the finite set of *states*, $\Sigma = \{\sigma_1, \dots, \sigma_{|\Sigma|}\}$ is the alphabet of *labels*, ranked by function r, $F \subseteq Q$ is the subset of *accepting states* and $\Delta = \{\delta_0, \delta_1, \dots, \delta_M\}$ is a finite collection of *transition functions* of the form $\delta_m : \Sigma_m \times Q^m \to Q$, for $m \in [0, M]$ with M the *maximum rank* or valence of the DFSTA.

For all trees $t \in \Sigma^T$, the result $\delta(t) \in Q$ of the operation of DFSTA A on a tree $t \in \Sigma^T$ is defined as

$$\delta(t) = \begin{cases} \delta_0(a) & \text{if } t = a \in \Sigma_0 \\ \delta_m(f, \delta(t_1), ..., \delta(t_m)) & \text{if } t = f(t_1, ..., t_m),\ 0 < m \leq M,\ f \in \Sigma_m \\ \text{undefined} & \text{otherwise} \end{cases} \tag{2}$$

In other words, the state $\delta(t)$ associated to a given tree t depends on the label of the root node (f) and also on the states that the NDFSTA associates to its children $(\delta(t_1), \delta(t_2), \dots, \delta(t_m))$.

By convention, undefined transitions lead to unaccepted trees. That is, M is the maximum number of children for any node of any tree in $L(A)$.

As usual, the language $L(A)$ recognized by a DFSTA A is the subset of Σ^T defined as

$$L(A) = \{t \in \Sigma^T : \delta(t) \in F\}. \tag{3}$$

One may generalize this definition so that the DFSTA produces an output label from an alphabet $\Gamma = \{\gamma_1, \dots, \gamma_{|\Gamma|}\}$ at each node visited, so that it acts like a (structure-preserving) finite-state tree transducer; two generalizations are possible, which correspond to the classes of finite-state string transducers known as Mealy and Moore machines[9, 15]:

- *Mealy tree transducers* are obtained by replacing the subset of accepting states F in the definition of a DFSTA by a collection of output functions $\Lambda = \{\lambda_0, \dots \lambda_M\}$, one for each possible rank, $\lambda : \Sigma_m \times Q^m \to \Gamma$.

- *Moore tree transducers* are obtained by replacing F by a single output function whose only argument is the new state: $\lambda : Q \to \Gamma$.

Conversely, a DFSTA can be regarded as a particular case of Mealy or Moore machine operating on trees whose output functions return only two values ($\Gamma = \{\text{yes}, \text{no}\}$).

2.2 Neural architectures

Here we define two recursive neural architectures that are similar to that used in related work as that of Frasconi, Gori and Sperduti [5], Sperduti [17] and Sperduti and Starita [18]. We find it convenient to talk about Mealy and Moore neural networks to define the way in which these networks compute their output, using the analogy with the corresponding finite-state tree transducers. The first architecture is a high-order Mealy RNN and the second one is a first-order Moore RNN[2].

2.2.1 A high-order Mealy recursive neural network

A high-order Mealy recursive neural network consists of two sets of single-layer neural networks, the first one to compute the next state (playing the role of the collection Δ of transition functions in a finite-state tree transducer) and the second one to compute the output (playing the role of the collection Λ of output functions in a Mealy finite-state tree transducer).

The *next-state function* is realized as a collection of $M + 1$ high-order single-layer networks, one for each possible rank $m = 0, \ldots, M$, having n_X neurons and $m+1$ input ports: m for the input of subtree state vectors, each

[2]The remaining two combinations are high-order Moore RNN (which may easily be shown to have the same computational power as their Mealy counterparts) and first-order Mealy machines (which need an extra layer to compute arbitrary output functions, see [2]).

of dimensionality n_X, and one for the input of node labels, represented by a vector of dimensionality n_U.

The node label input port takes input vectors equal in dimensionality to the number of input symbols, that is $n_U = |\Sigma|$. In particular, if μ is a node in the tree with label $l(\mu)$ and $\mathbf{u}[\mu]$ is the input vector associated with this node, the component $u_k[\mu]$ is equal to 1 if the input symbol at node μ is σ_k and 0 for all other input symbols (*one-hot* or exclusive encoding).

For a node μ with label $l(\mu) \in \Sigma_m$ and children $\nu_1, ..., \nu_m$ the next state $\mathbf{x}[\mu]$ is computed by the corresponding $m+1$-th order single-layer neural net as follows:

$$x_i[\mu] = g\left(w_i^m + \sum_{k=1}^{n_U} \sum_{j_1=1}^{n_X} \cdots \sum_{j_m=1}^{n_X} w_{ij_1j_2\ldots j_mk}^m \, x_{j_1}[\nu_1] x_{j_2}[\nu_2] \cdots x_{j_m}[\nu_m] u_k[\mu]\right) \tag{4}$$

where w_i^m represents the bias for the network of rank m and $i = 1, \ldots, n_X$. If μ is a leaf, i.e., $l(\mu) \in \Sigma_0$ the expression above for the component $x_i[\mu]$ reduces to

$$x_i[\mu] = g\left(w_i^0 + \sum_{k=1}^{n_U} w_{ik}^0 u_k[\mu]\right), \tag{5}$$

that is, there is a set of $|\Sigma|$ weights of type $\mathbf{w}_k^0 = (w_{1k}^0, ..., w_{n_xk}^0)$ which play the role of the initial state in recurrent networks [4].

The output function is realized as a collection of $M + 1$ high-order single-layer networks having n_Y units and the same input structure. The output function for a node of rank m is evaluated as

$$y_i[\mu] = g(v_i^m + \sum_{k=1}^{n_U} \sum_{j_1=1}^{n_X} \cdots \sum_{j_m=1}^{n_X} v_{ij_1j_2\ldots j_mk}^m \, x_{j_1}[\nu_1] x_{j_2}[\nu_2] \cdots x_{j_m}[\nu_m] u_k[\mu]) \tag{6}$$

where $i = 0, \ldots, n_Y$ and $m = 1, \ldots M$.

2.2.2 A first-order Moore recursive neural network

The first-order Moore recursive neural network has a collection of $M + 1$ next-state functions (one for each rank m) of the form

$$x_i[\mu] = g\left(w_i^m + \sum_{k=1}^{n_U} w_{ik}^{mu} u_k[\mu] + \sum_{p=1}^{m} \sum_{j=1}^{n_X} w_{ij}^{mx_p} x_j[\nu_p] \right) \quad i = 1, \ldots, n_X \qquad (7)$$

having the same structure of input ports as its high-order counterpart, and a single output function of the form

$$y_i[\mu] = g\left(v_i + \sum_{j=1}^{n_X} v_{ij} x_j[\mu] \right) \quad i = 1, \ldots, n_Y \qquad (8)$$

taking n_X inputs and producing n_Y outputs.

3 Encoding tree automata in discrete-state recursive neural networks

We will present three different ways to encode finite-state recursive transducers in discrete-state RNN using TLU as activation functions. The first two use the discrete-state version of the high-order Mealy RNN described in Section 2.2.1 and the third one uses the discrete-state version of first-order Moore RNN described in Section 2.2.2. The first two encodings are straightforward; the third one is explained in more detail.

All of the encodings are based on an exclusive or *one-hot* encoding of the states of the finite-state transducers ($n_X = |Q|$): the RNN is said to be in state i when component x_i of the n_X-dimensional state vector $\mathbf{x}$ takes a high value ($x_i = 1$) and all other components take a low value ($x_j = 0,\ j \neq i$). In addition, exclusive encoding of inputs ($n_U = |\Sigma|$) and outputs ($n_Y = |\Gamma|$) is used.

Each one of these discrete-state RNN encodings will be converted in Section 5 into sigmoid RNN encodings by using each of the two strategies described in Section 4.

3.1 A high-order Mealy encoding using biases

Assume that we have a Mealy finite-state tree transducer $A = (Q, \Sigma, r, \Gamma, \Delta, \Lambda)$. Then, a discrete-state high-order Mealy RNN with weights

$$W^m_{ij_1,\ldots,j_m f} = \begin{cases} 1 & \text{if } \delta(f, q_1, \ldots, q_m) = q_i \\ 0 & \text{otherwise} \end{cases}, \tag{9}$$

bias $W^m_i = -1/2,\ m \in [0, M]$, output weights

$$V^m_{kj_1,\ldots,j_m f} = \begin{cases} 1 & \text{if } \lambda_m(f, q_1, \ldots, q_m) = \gamma_k \\ 0 & \text{otherwise} \end{cases}, \tag{10}$$

and bias $V^m_i = -1/2,\ m \in [0, M]$ behaves as a stable simulator for the finite-state tree transducer (note that uppercase letters are used to designate weights in discrete-state neural nets). This would also be the case if biases V^m_i were set to any value in $(0, 1)$; the value $-1/2$ happens to be the best value for the conversion into a sigmoid RNN.

3.2 A high-order Mealy encoding using no biases

A second possible encoding (the tree-transducing counterpart of a string-transducer encoding described in [12, 2]), which uses no bias, has the next-state weights

$$W^m_{ij_1,\ldots,j_m f} = \begin{cases} 1 & \text{if } \delta(f, q_1, \ldots, q_m) = q_i \\ -1 & \text{otherwise} \end{cases} \tag{11}$$

and all biases $W^m_i = 0$, and the output weights

$$V^m_{kj_1,\ldots,j_m f} = \begin{cases} 1 & \text{if } \lambda_m(f, q_1, \ldots, q_m) = \gamma_k \\ -1 & \text{otherwise} \end{cases} \tag{12}$$

and all biases $V^m_i = 0,\ m \in [0, M]$ (as in the case of the biased construction, the encoding also works if biases are set to any value in $(-1, 1)$, with 0 being the optimal value for conversion into a sigmoid RNN).

3.3 A first-order Moore encoding

Consider a Moore finite-state tree transducer of the form $A = (Q, \Sigma, r, \Gamma, \Delta, \lambda)$. Before encoding this DFSTA in a first-order RNN we need to split its states first; state-splitting has been found necessary to implement arbitrary finite-state machines in first-order discrete-time recurrent neural networks [8, 1, 2] and has also been recently described by Sperduti [17] for the encoding of DFSTA in RNN.

A DFSTA $A' = (Q', \Sigma, r, \Gamma, \Delta', \lambda')$, which can easily be shown to be equivalent to A is easily constructed by using the method described by Sperduti [17] as follows:

- The new set of states Q' is the subset of $\bigcup_{m=0}^{M} \Sigma_m \times Q^m$ defined as
$$Q' = \{(f, q_{j_1}, q_{j_2}, \ldots, q_{j_m}) : \delta_m(f, q_{j_1}, q_{j_2}, \ldots, q_{j_m}) \in Q\}, \qquad (13)$$
- The next-state functions in the new set $\Delta' = \{\delta'_0, \delta'_1, \ldots, \delta'_M\}$, $\delta'_m : \Sigma_m \times (Q')^m \rightarrow Q'$ are defined as follows:
$$\forall a \in \Sigma_0, \;\; \delta'(a) = (a) \qquad (14)$$
and
$$\forall f \in \Sigma_m, \; m > 0, \; \forall q'_{j_k} \in Q', \qquad (15)$$
$$\delta'_m(f, q'_{j_1}, q'_{j_2}, \ldots, q'_{j_m}) = (f, \delta(q'_{j_1}), \delta(q'_{j_2}), \ldots, \delta(q'_{j_m})) \qquad (16)$$
where the shorthand notation
$$\delta(q') = \delta_m(f, q_{j_1}, q_{j_2}, \ldots, q_{j_m}) \qquad (17)$$
has been used.
- Finally, the new output function $\lambda' : Q' \rightarrow \Gamma$ is defined as follows:
$$\forall q' = (f, q_{j_1}, q_{j_2}, \ldots, q_{j_m}) \in Q', \;\; \lambda'(q') = \lambda(\delta_m(f, q_{j_1}, q_{j_2}, \ldots, q_{j_m})). \qquad (18)$$

The split DFSTA A' is then encoded into a discrete-state RNN in eqs. (7) and (8) by choosing its parameters as follows:[3]

- $n_X = |Q'|$;
- $W^{mu}_{ik} = 1$ if there exist $q'_{j_1}, \dots, q'_{j_m}$ such that $\delta'_m(\sigma_k, q'_{j_1}, \dots, q'_{j_m}) = q'_i$ and zero otherwise;
- $W^{mx_p}_{ij} = 1$ if there exist $q'_{i_1}, \dots q'_{i_{p-1}}, q'_{i_{p+1}}, \dots, q'_{i_m}$ and σ_l such that $\delta'_m(q'_{i_1}, \dots q'_{i_{p-1}}, q'_j, q'_{i_{p+1}}, \dots, q'_{i_m}) = q'_i$ and zero otherwise;
- $W^m_i = -(m + \frac{1}{2})$;
- $V_{ij} = 1$ if $\lambda'(q'_j) = \gamma_i$ and zero otherwise;
- $V_i = -\frac{1}{2}$;

It is not difficult to show that the operation of this discrete-state RNN is equivalent to that of the corresponding DFSTA A', and therefore to that of A (as was the case with the previous constructions, different values for the biases are also possible but the ones shown happen to be optimal for conversion into a sigmoid RNN).

4 Stable simulation of discrete-state units on analog units

4.1 Using Šíma's theorem

The following is a restatement of a theorem by Šíma [16] which is included for convenience. Only the notation has been slightly changed in order to adapt it to the present study.

[3]Remember that uppercase letters are used to denote the weights in discrete-state RNN.

A threshold linear unit (TLU) is a neuron computing its output as

$$y(\mathbf{x}) = g_H(W_0 + \sum_{j=1}^{n} W_j x_j) \tag{19}$$

where g_H the threshold activation function, the W_j ($j = 0, \ldots, n$) are real-valued weights and $\mathbf{x} = (x_1, ..., x_n) \in \{0,1\}^n$ is a binary input vector.

Consider an analog neuron with weights $w_0, w_1, ...w_n$ having an activation function g with two different limits $a = \lim_{x\to-\infty} g(x)$ and $b = \lim_{x\to\infty} g(x)$.

Now, let $W = \max_{1\le j\le n}\{|W_j|\}$, $\xi = \min\{|W_0 + \sum_{j=1}^{n} W_j x_j| > 0 : \mathbf{x} \in \{0,1\}^n\}$ and

$$\delta_{\max} = \frac{\xi}{2nW}. \tag{20}$$

The magnitude $\delta_{\max}$ will be called the *maximum input tolerance.*

Finally, let also the mapping τ_ϵ be defined as:

$$\tau_\epsilon(x) = \begin{cases} 0 & \text{if } x \in (a-\epsilon, a+\epsilon) \\ 1 & \text{if } x \in (b-\epsilon, b+\epsilon) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{21}$$

This mapping classifies the output of an analog neuron into three categories: *low* (0), *high* (1), or *forbidden* (undefined). Then, let $r_\delta : \{0,1\} \to \mathbb{R}$ be the inverse mapping $r_\delta(k) = \{x \in \mathbb{R} : \tau_\delta(x) = k\}$. Šíma's theorem states that, for any input tolerance δ such that $0 < \delta < \delta_{\max}$ and for any output tolerance ϵ such that $0 < \epsilon \le \delta$, there exists an analog neuron with activation function g and weights $w_0, \ldots, w_n \in \mathbb{R}$ such that, for all $\mathbf{x} \in \{0,1\}^n$,

$$g_H(W_0 + \sum_{j=1}^{n} W_j x_j) = \tau_\epsilon(g(w_0 + \sum_{j=1}^{n} w_j r_\delta(x_j))) \tag{22}$$

According to the constructive proof of the theorem [16] a set of sufficient conditions for the above equation to hold is

$$w_0 = H(W_0 + \frac{\xi}{2} - \frac{a}{b-a}\sum_{j=1}^{n} W_j) \tag{23}$$

and

$$w_j = \frac{HW_j}{b-a} \ \ \forall j = 1, \ldots, n \tag{24}$$

with

$$H > \frac{2}{\xi - 2n\delta W} \max\{-\alpha, \beta\} \tag{25}$$

where α and β are such that $|g(x) - a| < \epsilon$ for all $x < \alpha$ and $|g(x) - b| < \epsilon$ for all $x > \beta$ and $|\alpha|$ and $|\beta|$ are as small as possible. That is, Šíma's prescription simply scales the weights of the TLU to get those of the analog network and does the same with the bias but only after shifting it conveniently to avoid a zero value for the argument of the activation function.

Note that inputs to the analog unit are allowed to be within δ of 0 and 1 whereas outputs are allowed to be within ϵ of a and b. When constructing a *recursive network*, the outputs of one analog unit are normally used as inputs for another analog unit and therefore the most natural choice is $a = 0$, $b = 1$, and $\epsilon \leq \delta$. This choice is compatible, for instance, with the use of the logistic function $g_L(x) = 1/(1+\exp(-x))$ whose limits are exactly $a = 0$ and $b = 1$ but not with other activation functions such as the hyperbolic tangent $\tanh(x)$. In particular, for the case $g = g_L$, eqs. (23) and (24) reduce to

$$w_0 = H(W_0 + \frac{\xi}{2}) \tag{26}$$

and

$$w_j = HW_j \ \ \forall j = 1, \ldots, n \tag{27}$$

and (25) becomes

$$H > \frac{2}{\xi - 2n\delta W} g_L^{-1}(1-\epsilon) \tag{28}$$

In the following, we rederive simple sufficient conditions for stable simulation of a TLU by an analog unit which are suitable for any strictly growing activation function but restricted to exclusive encoding of the input (whereas Šíma's construction is valid for any binary input vector). The simplicity of the prescriptions allows for an alternate straightforward worst-case analysis that leads to weights that are, in most common situations, smaller than those obtained by direct application of Šíma's theorem.

4.2 Using a simple scheme for exclusive encoding of the input

The conditions for stable simulation of finite-state machines (FSM) in DTRNN have been studied, following an approach related to that of Šíma[16], by Carrasco et al. [2] (see also [12, 11]; these conditions assume the special but usual case of one-hot or exclusive encoding of the input and strictly growing activation functions. These assumptions, together with a worst-case analysis, allow one to obtain a prescription for the choice of suitable weights for stable simulation that works at lower saturation levels than the general scheme (eqs. 23–25). Usually, the prescription can be realized as a single-parameter scaling of all of the weights in the TLU including the bias; this scaling is equivalent to finding a *finite* value of the gain of the sigmoid function which ensures correct behavior.

Note that, in the case of exclusive encoding (as the ones used in Section 3, there are only n possible inputs: the binary vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ ($\mathbf{b_i}$ being the vector whose i-th component is one and the rest are zero). Therefore, the argument $W_0 + \sum_{j=1}^{n} W_j x_j$ of (22) may only take n different values $W_0 + W_i$ for $i = 1, ..., n$ (for the binary input $\mathbf{x} = \mathbf{b}_i$, however, the analog neuron with input tolerance δ may receive input vectors in $r_\delta(\mathbf{b}_i) = \{\mathbf{x} \in \mathbb{R}^n : x_i \in (1-\delta, 1+\delta) \wedge x_j \in (-\delta, \delta),\ \forall j \neq i\}$). This property of exclusive encoding makes it possible to formulate a condition such that (22) holds for all possible inputs $\mathbf{x} = \mathbf{b}_i$. Two cases have to be distinguished:

1. $g_H(W_0 + W_i) = 1$, that is, $W_0 + W_i \geq 0$. In this case, (22) holds if $g(w_0 + \sum_{j=1}^{n} w_j x_j) \in (b-\epsilon, b+\epsilon)$ for all $\mathbf{x} \in r_\delta(\mathbf{b}_i)$. As g is strictly growing, we may also write $w_0 + \sum_{j=1}^{n} w_j x_j > g^{-1}(b-\epsilon)$. Obviously, the

minimum value of $w_0 + \sum_j w_j x_j$ in $r_\delta(\mathbf{b}_i)$ is bounded by $w_0 + w_i - n\delta w$, with $w = \max_{1 \leq j \leq n}\{|w_j|\}$. Therefore,

$$w_0 + w_i - n\delta w > g^{-1}(b - \epsilon) \tag{29}$$

is a sufficient condition for the analog neuron to simulate the corresponding TLU with input $\mathbf{b}_i$.

2. $g_H(W_0 + W_i) = 0$, that is, $W_0 + W_i < 0$. A similar argument leads to the sufficient condition

$$w_0 + w_i + n\delta w < g^{-1}(a + \epsilon). \tag{30}$$

For instance, if we choose $w_i = HW_i$ (and, therefore, $w = HW$), then eqs. (29) and (30) are fulfilled either if

$$H > \frac{g^{-1}(b - \epsilon)}{|W_0 + W_i| - n\delta W} \tag{31}$$

and $W_0 + W_i - n\delta W > 0$ or

$$H > \frac{-g^{-1}(a + \epsilon)}{|W_0 + W_i| - n\delta W} \tag{32}$$

and $W_0 + W_i + n\delta W > 0$. In order to compare with eqs. (23–25), the last two conditions may be written as a single, more restrictive pair of conditions

$$H > \frac{\max\{-g^{-1}(a + \epsilon), g^{-1}(b - \epsilon)\}}{\min_{i=1,\ldots,n} |W_0 + W_i| - n\delta W} \tag{33}$$

and

$$\delta < \min_{i=1,\ldots,n} \frac{|W_0 + W_i|}{nW}. \tag{34}$$

The simple choice $w_i = HW_i$ is not adequate in case that $W_0 + W_i = 0$, but this case does not appear in any of the encodings proposed in Section 3.

5 Encoding tree automata in sigmoid recursive neural networks

As mentioned before, the theorem in section (4) leads to the natural choice $a = 0$, $b = 1$ in addition to $\epsilon \leq \delta$ when applying it to neurons in recursive neural networks. Due to its widespread use, we will consider and compare in this section various possible encodings using the logistic function $g_L = 1/(1 + \exp(-x))$ although results for different activation functions having $a = 0$ and $b = 1$ may also be obtained. Indeed, monotonic growth of the function along the real line is enough for the following derivation (as it was the case for eqs. (29) and (30)). In our case, we want to simulate a DFSTA with a sigmoid RNN.

Consider first the high-order Mealy RNN architecture. As the input x_j in (22) is, in the case of the RNN described in (4), the product of m outputs $x_{j_1} \cdots x_{j_m}$, each one in the range $(0, \epsilon) \cup (1 - \epsilon, 1)$, the product is always in the range $(0, \epsilon(1-\epsilon)^{m-1}) \cup ((1-\epsilon)^m, 1)$. In other words, there is a forbidden region between $\epsilon(1 - \epsilon)^{m-1}$ and $(1 - \epsilon)^m$. It is not difficult to show that $1 - (1 - \epsilon)^m \geq \epsilon(1 - \epsilon)^{m-1}$ with the equality holding only if $m = 1$ or $\epsilon = 0$. Therefore, the conditions

$$\delta = 1 - (1 - \epsilon)^m \tag{35}$$

and $\delta < \delta_{\max}$ suffice for our purposes, as $(\delta, 1 - \delta) \subseteq (\epsilon(1 - \epsilon)^{m-1}, ((1 - \epsilon)^m)$. If we want to use the same scaling factor for all weights and all possible ranks m, we can use $m = M$.

Consider now the first-order Moore RNN architecture. In this case, there are no products, and the conditions

$$\delta = \epsilon \tag{36}$$

and $\delta < \delta_{\max}$ are sufficient.

The previous section describes two different schemes to simulate discrete-state neurons taking exclusive input vectors in sigmoid neurons. This section describes the application of these two schemes to the three recursive neural network architectures described in Section 2.2.

5.1 Using Sima's prescription

Šíma's construction (Section 4.1) gives for the biased high-order Mealy RNN in Section 3.1 the following: $n = (n_X)^m$, $\xi = \frac{1}{2}$, $W = 1$, $\delta_{\max} = \frac{1}{4(n_X)^m}$ and, therefore, $w_i^m = -H/4$, $w_{ij_1j_2\cdots j_mk}^m = HW_{ij_1j_2\cdots j_mk}^m$, $v_i^m = -H/4$, and $v_{ij_1j_2\cdots j_mk}^m = HV_{ij_1j_2\cdots j_mk}^m$ with

$$H > \frac{4\log\frac{1-\epsilon}{\epsilon}}{1-4(n_X)^m(1-(1-\epsilon)^m)}, \tag{37}$$

where the condition (35) has been applied[4], together with $1-(1-\epsilon)^m < \delta_{\max}$, a condition that ensures a positive value of H. As shown in (37), the minimum value allowed for H depends both on n_X and ϵ. For a given architecture, n_X and the maximum value of m (M) are fixed, so only ϵ can be changed. There exists at least one value of ϵ that allows one to choose the minimum value of H needed for stable simulation. In this sense, minimization of H by choosing an appropriate ϵ can be performed as in [2] and leads to the values shown in Table 1 (minimum required H as a function of n_X and M). The weights obtained grow slower than $\log(mn_x^m)$ with m and n_X, and, as can be seen, are inordinately large and lead therefore to a very saturated analog RNN.

Applying Šíma's construction to the biasless high-order Mealy RNN (Section 3.2), we get $n = (n_X)^m$, $\xi = 1$, $W = 1$, $\delta_{\max} = \frac{1}{2(n_X)^m}$ and, therefore, $w_i^m = H/2$, $w_{ij_1j_2\cdots j_mk}^m = HW_{ij_1j_2\cdots j_mk}^m$, $v_i^m = H/2$, and $v_{ij_1j_2\cdots j_mk}^m =$

[4]We have used $n = (n_X)^m$; equations (4) and (6) have $n_U(n_X)^m$ terms but, due to the exclusive encoding of the inputs, $(n_U-1)(n_X)^m$ of terms are identically zero with no uncertainty at all.

$HV^m_{ij_1j_2\dots j_mk}$ with

$$H > \frac{2\log\frac{1-\epsilon}{\epsilon}}{1-2(n_X)^m(1-(1-\epsilon)^m)}, \tag{38}$$

together with $1-(1-\epsilon)^m < \delta_{\max}$ (for positive values of H). The weights obtained by searching for the minimum H satisfying the conditions are shown in Table 2; as can be seen, weights (which show the same asymptotic behavior as the ones in the previous construction) are smaller but still too large to avoid saturation.

Finally, applying Šíma's construction[5] to the first-order Moore RNN in Section 3.3, we have, for a next-state function of rank m, $n = mn_X$, $\xi = \frac{1}{2}$, $W = 1$, $\delta_{\max} = \frac{1}{4mn_X}$, and, accordingly, weights are:

- $w^{mu}_{ij} = H$ if there exist $q'_{j_1}, \dots, q'_{j_m}$ such that $\delta'_m(\sigma_j, q'_{j_1}, \dots, q'_{j_m}) = q'_i$ and zero otherwise;
- $w^{mx_p}_{ij} = H$ if there exist $q'_{i_1}, \dots q'_{i_{p-1}}, q'_{i_{p+1}}, \dots, q'_{i_m}$ and σ_l such that $\delta'_m(q'_{i_1}, \dots q'_{i_{p-1}}, q'_j, q'_{i_{p+1}}, \dots, q'_{i_m}) = q'_i$ and zero otherwise
- $w^m_i = (-m + \frac{3}{4})H$;

with

$$H > \frac{4\log\frac{1-\epsilon}{\epsilon}}{1-4(mn_X)\epsilon} \tag{39}$$

(where (36) has been used), together with $\epsilon < \frac{1}{4(mn_X)}$.

For the output function, $n = n_X$, $\xi = \frac{1}{2}$, $W = 1$, $\delta_{\max} = \frac{1}{4n_X}$, and accordingly, weights are

- $v_{ij} = H$ if $\lambda'(q'_j) = \gamma_i$ and zero otherwise;

[5]Šíma's construction can be applied provided that we consider each possible $W^m_i + \sum_{k=1}^{n_U} W^{mu}_{ik} u_k[\mu]$ in the next-state function as a different bias W_0 with $\mathbf{u}[\mu] \in \{0,1\}^n$ and choose the safest prescription (valid for all possible values of the bias). In the present case, this bias has always the value $-(m-\frac{1}{2})$, and $n = mn_X$ (number of additive terms in (19)).

- $v_i = -\frac{H}{4}$,

with

$$H > \frac{4 \log \frac{1-\epsilon}{\epsilon}}{1 - 4(n_X)\epsilon}. \tag{40}$$

provided that $\epsilon < \frac{1}{4n_X}$ where we have used $\delta < \epsilon$.

If we want a single value of H to assign weights both to all of the next-state functions and the output function, we have to use

$$H > \frac{4 \log \frac{1-\epsilon}{\epsilon}}{1 - 4n\epsilon}. \tag{41}$$

with $n = \max(n_X, Mn_X) = Mn_X$. The weights obtained by searching for the minimum H satisfying the conditions are shown in Table 3; they grow slower than $\log(Mn_X)$, and, as can be seen, they are equal or smaller than the ones for the biased high-order construction, but larger than those for the biased construction. However, the fact that state splitting leads to larger values of n_X for automata having the same transition function has to be taken into account.

5.2 Using the encoding for exclusive inputs

If we choose $w_i = HW_i$ for all weights, including biases we obtain for the biased high-order Mealy encoding in Section 3.1, by substituting in eqs. (31) and (32),

$$H > \frac{2g_L^{-1}(1-\epsilon)}{1 - 2n_X^m(1-(1-\epsilon)^m)} \tag{42}$$

together with $1 - (1-\epsilon)^m < \frac{1}{2n_X^m}$, which happens to be the same expression as the one obtained in the previous section by using Šíma's construction on the biasless encoding (Section 3.2); results are shown in Table 2. The results for $M = 1$ are obviously identical to those reported for second-order discrete-time recurrent neural networks using the biased construction in [2].

If we instead apply our alternate encoding to the biasless high-order Mealy construction in Section 3.2 we get

$$H > \frac{g_L^{-1}(1-\epsilon)}{1-(n_X)^m(1-(1-\epsilon)^m)} \tag{43}$$

together with $1-(1-\epsilon)^m < \frac{1}{(n_X)^m}$, which, after suitable minimization of H, leads to the best possible weights of all encodings. Weights grow with m and n_X slower than $\log(mn_X^m)$; some results are shown in Table 4. As in the previous case, the results for $M=1$ are obviously identical to those reported for second-order discrete-time recurrent neural networks using no biases in [2].

Finally, we apply our alternate encoding scheme to the first-order Moore construction in Section 3.3. Now $n = mn_X$ and

$$H > \frac{g_L^{-1}(1-\epsilon)}{|W_0 + W_i| - n\epsilon W} \tag{44}$$

(the particular form of (33) in this case) has to be valid for all combinations $W_0 + W_i$. As W_0 can take any value in

$$\{W_i^m + W_{ik}^{mu} : i = 1, \ldots, n_X;\ k = 1, \ldots, n_U\},$$

and W_i can take any value in

$$\{W_{ij}^{mx_p} : i = 1, \ldots, n_X;\ j = 1, \ldots, n_X;\ p = 1, \ldots, m\},$$

the minimum value of $|W_0 + W_i|$ is then, for binary, exclusive values of all state vectors, equal to $\frac{1}{2}$. Therefore,

$$H > \frac{2g_L^{-1}(1-\epsilon)}{1-2mn_X\epsilon} \tag{45}$$

together with $\delta < \frac{1}{2mn_X}$, which, after suitable minimization, leads weights that grow with m and n_X slower than $\log(mn_X)$; values are shown in Table 5. The values are smaller than the ones obtained with Šíma's construction for the same first-order network but are still very large, especially if one considers that splitting leads to very large values of n_X.

6 Conclusion

We have studied four strategies to encode deterministic finite-state tree automata (DFSTA) on high-order sigmoid recursive neural networks (RNN) and two strategies to encode them in first-order sigmoid RNN. These six strategies are derived from three different strategies to encode DFSTA in discrete-state RNN (that is, RNN using threshold linear units) by applying two different weight mapping schemes to convert each one of them into a sigmoid RNN. The first mapping scheme is the one described by Šíma[16]. The second one is an alternate scheme devised by us. All of the strategies yield analog RNN with a very simple "weight alphabet" containing only three weights all of which are proportional to a single parameter H. The best results (i.e., smallest possible value of H, as would be desired in a derivative-based learning setting) are obtained by appling the alternate scheme to a biasless discrete-state high-order RNN (it has to be mentioned that Šíma's mapping yields larger weights in all cases but is more general and would also work with distributed encodings which allow the construction of smaller RNN). In all of the constructions, the values of H suggest that, even though *in principle* RNN with finite weights are able to simulate exactly the behavior of DFSTA, it will *in practice* be very difficult to learn the exact finite-state behavior from examples because of the very small gradients present when weights reach adequately large values.

Smaller weights are obtained at the cost of enlarging the size of the RNN due to exclusive encoding of states and inputs (Šíma's result also works for distributed encodings).

Acknowledgements: This work has been supported by the Spanish Comision Interministerial de Ciencia y Tecnología through grant TIC97-0941.

References

[1] R. Alquézar and A. Sanfeliu. An algebraic framework to represent finite state automata in single-layer recurrent neural networks. *Neural Computation*, 7(5):931–949, 1995.

[2] R. C. Carrasco, M. L. Forcada, M. Ángeles Valdés-Muñoz, and Ramón P. Ñeco. Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units. *Neural Computation*, 12, 2000. In press.

[3] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[4] M. L. Forcada and R. C. Carrasco. Learning the initial state of a second-order recurrent neural network during regular-language inference. *Neural Computation*, 7(5):923–930, 1995.

[5] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive data structures processing. *IEEE Transactions on Neural Networks*, 9(5):768–786, 1998.

[6] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. Learning and extracted finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[7] R. C. Gonzalez and M. G. Thomason. *Syntactical pattern recognition.* Addison-Wesley, Menlo Park, CA, 1978.

[8] M. W. Goudreau, C. L. Giles, S. T. Chakradhar, and D. Chen. First-order vs. second-order single layer recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(3):511–513, 1994.

[9] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation.* Addison–Wesley, Reading, MA, 1979.

[10] S. C. Kremer. On the computational power of Elman-style recurrent networks. *IEEE Transactions on Neural Networks*, 6(4):1000–1004, 1995.

[11] S. C. Kremer, R. P. Ñeco, and M. L. Forcada. Constrained second-order recurrent networks for finite-state automata induction. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *Proceedings of the 8th International Conference on Artificial Neural Networks ICANN'98*, volume 2, pages 529–534, London, 1998. Springer.

[12] R. P. Ñeco, M. L. Forcada, R. C. Carrasco, and M. A. Valdés-Muñoz. Encoding of sequential translators in discrete-time recurrent neural nets. In *Proceedings of the European Symposium on Artificial Neural Networks ESANN'99*, pages 375–380, 1999.

[13] C. W. Omlin and C. L. Giles. Constructing deterministic finite-state automata in recurrent neural networks. *Journal of the ACM*, 43(6):937–972, 1996.

[14] C. W. Omlin and C. L. Giles. Stable encoding of large finite-state automata in recurrent neural networks with sigmoid discriminants. *Neural Computation*, 8:675–696, 1996.

[15] A. Salomaa. *Formal Languages*. Academic Press, New York, NY, 1973.

[16] J. Šíma. Analog stable simulation of discrete neural networks. *Neural Network World*, 7:679–686, 1997.

[17] A. Sperduti. On the computational power of neural networks for structures. *Neural Networks*, 10(3):395–400, 1997.

[18] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[19] J. W. Thatcher. Tree automata: An informal survey. In A.V. Aho, editor, *Currents in the theory of computing*. Prentice-Hall, Englewood-Cliffs, NJ, 1973.

Rafael C. Carrasco was born in Alacant, Spain in 1963. He received the B.Sc. degree in Physics in 1987, and the Ph.D. in Physics in 1991, both from the University of Valencia, Spain, and the Ph.D. in Computer Engineering in 1997 from the University of Alacant, Spain. In 1992, Dr. Carrasco joined the Department of Languages and Information Systems (Departament de Llenguatges i Sistemes Informàtics) of the University of Alacant, where he is currently an Associate Professor and head of the Depratment. His research interests are in recurrent neural networks and statistical grammatical inference.

Mikel L. Forcada was born in Caracas, Venezuela in 1963. He received the B.Sc. degree in Chemistry in 1986, and the Ph.D. in Physics in 1991, both from the University of Alacant, Spain. In 1993, Dr. Forcada joined the Department of Languages and Information Systems (Departament de Llenguatges i Sistemes Informàtics) of the University of Alacant, where he is currently an Associate Professor and director of the Department's graduate program on language engineering and pattern recognition. His research interests are in recurrent neural networks, grammatical inference, nearest-neighbor methods in pattern recognition, and, more recently, machine translation. He is a member of the IEEE and its Computer Society, of the International Neural Network Society, and of the International Association for Machine Translation.

	$M = 1$	$M = 2$	$M = 3$	$M = 4$	$M = 5$
$n_X = 2$	18.29	25.20	30.36	34.84	38.95
$n_X = 3$	20.39	29.03	35.90	42.09	47.89
$n_X = 4$	21.84	31.80	39.77	47.14	54.13
$n_X = 5$	22.94	33.72	42.73	51.03	58.93

Table 1: *Minimum values of the scaling factor H as a function of the number of state units n_X and the maximum rank M for Šíma's construction as applied to the biased high-order Mealy RNN (Section 3.1).*

	$M = 1$	$M = 2$	$M = 3$	$M = 4$	$M = 5$
$n_X = 2$	7.18	10.91	13.56	15.84	17.93
$n_X = 3$	8.37	12.88	16.38	19.51	22.43
$n_X = 4$	9.15	14.24	18.33	22.05	25.57
$n_X = 5$	9.73	15.28	19.83	24.01	27.96

Table 2: *Minimum values of the scaling factor H as a function of the number of state units n_X and the maximum rank M for Šíma's construction as applied to the biasless high-order Mealy RNN (Section 3.2).*

	$M = 1$	$M = 2$	$M = 3$	$M = 4$	$M = 5$
$n_X = 2$	18.29	21.84	23.83	25.21	26.27
$n_X = 3$	20.39	23.83	25.77	27.13	28.18
$n_X = 4$	21.84	25.21	27.13	28.48	29.52
$n_X = 5$	22.94	26.27	28.18	29.52	30.54

Table 3: *Minimum values of the scaling factor H as a function of the number of state units n_X and the maximum rank M for Šíma's construction as applied to the biasless first-order Moore RNN (Section 3.3).*

	$M=1$	$M=2$	$M=3$	$M=4$	$M=5$
$n_X=2$	2.00	4.56	5.95	7.12	8.18
$n_X=3$	3.12	5.60	7.40	8.98	9.69
$n_X=4$	3.59	6.30	8.39	10.26	12.03
$n_X=5$	3.93	6.83	9.14	11.25	13.24

Table 4: *Minimum values of the scaling factor H as a function of the number of state units n_X and the rank M for the alternate construction construction as applied to the biasless Mealy RNN (Section 3.2).*

	$M = 1$	$M = 2$	$M = 3$	$M = 4$	$M = 5$
$n_X = 2$	7.18	9.15	10.20	10.92	11.47
$n_X = 3$	8.37	10.20	11.21	11.92	12.45
$n_X = 4$	9.15	10.92	11.92	12.61	13.14
$n_X = 5$	9.73	11.47	12.45	13.14	13.67

Table 5: *Minimum values of the scaling factor H as a function of the number of state units n_X and the rank M for the alternate construction construction as applied to the first-order Moore RNN (Section 3.3).*