Encoding nondeterministic finite-state tree automata in sigmoid recursive neural networks

Mikel L. Forcada and Rafael C. Carrasco

Departament de Llenguatges i Sistemes Informàtics
Universitat d'Alacant, E-03071 Alacant (Spain)
Phone: +34-96-590-3400 ext. 3384
Fax: +34-96-590-9326
mlf@dlsi.ua.es

Encoding nondeterministic finite-state tree automata in sigmoid recursive neural networks*

Mikel L. Forcada and Rafael C. Carrasco

Departament de Llenguatges i Sistemes Informàtics
Universitat d'Alacant, E-03071 Alacant (Spain)
{mlf,carrasco}@dlsi.ua.es

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 finite-state tree automata (FSTA). In many cases, the number of states of a nondeterministic FSTA (NFSTA) recognizing a tree language may be smaller than that of the corresponding deterministic FSTA (DFSTA) (for example, the language of binary trees in which the label of the leftmost k-th order grandchild of the root node is the same as that on the leftmost leaf). This paper describes a scheme that directly encodes NFSTA in sigmoid RNN.

Keywords: nondeterministic tree automata; recursive neural networks; stable encoding, computational power of neural networks.

* Work supported by the Spanish Comisión Interministerial de Ciencia y Tecnología through grant TIC97-0941.

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 [10] have studied the classification of structures (directed ordered graphs, including cyclic graphs) and Sperduti [10] 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 finite-state tree automata (FSTA), also called *frontier-to-root* or *ascending* tree automata[6, 11]. Deterministic FSTA (DFSTA) may easily be realized as RNN using discrete-state units such as the threshold linear unit (TLU). Sperduti, in fact, [9] has recently shown that Elman-style [3] RNN using TLU may simulate DFSTA, and provides an intuitive explanation (similar to that expressed by Kremer [7] 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. More recently, we [1] have shown that a finite value of this gain is possible such that exact simulation of a DFSTA may indeed be performed by various analog RNN architectures. This paper adds a new encoding to those described in [1], which can be used to directly encode nondeterministic FSTA (NFSTA); it is the case that in many cases, the number of states of a nondeterministic FSTA (NFSTA) recognizing a tree language may be much smaller than that of the corresponding DFSTA (for example, the language of binary trees in which the label of the leftmost k-th order grandchild of the root node is the same as that on the leftmost leaf); therefore, there may exist very compact recursive neural network encodings of a family of tree language recognizers. Tree automata include string automata such as Mealy and Moore machines as a special case. For a detailed study of the encoding of string automata in recurrent neural networks, see [2].

In the following section, tree languages and tree automata are introduced. Section 3 describes a particular recursive neural network architecture, a high-order Elman-like sigmoid recursive neural network. Section 4 describes the encoding of a nondeterministic FSTA into that RNN architecture. Finally, we present our conclusions in the last section.

2 Tree Languages and Tree Automata

We will denote with Σ a *ranked alphabet*, that is, a finite set of symbols $\Sigma = \{\sigma_1, ..., \sigma_{|\Sigma|}\}$ with an associated function $r : \Sigma \rightarrow \mathbb{N}$ 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 $t_1 \ldots t_m$ which are valid trees of Σ^T belongs to Σ^T).

A *nondeterministic finite-state tree automaton* (NFSTA) is a five-tuple $A = (Q, \Sigma, r, \Delta, F)$, where $Q = \{q_1, \ldots, q_{|Q|}\}$ is the finite set of *states*, $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ is the alphabet of *labels*, ranked by function r, $F \subseteq Q$ is the subset of *accepting states* and $\Delta = \{\delta_0, \delta_1, \ldots, \delta_M\}$ is a finite collection of *transition functions* of the form $\delta_m : \Sigma_m \times Q^m \to 2^Q$, for $m \in [0, M]$ with M the *maximum rank* or valence of the NFSTA. For all trees $t \in \Sigma^T$, the result $\delta(t) \in 2^Q$ of the operation of NFSTA A on a tree $t \in \Sigma^T$ is defined as

$$\delta(t) = \begin{cases} \delta_0(a) & \text{if } t = a \in \Sigma_0 \\ \bigcup_{q_i \in \delta(t_i), 1 < i \leq m} \delta_m(f, q_1, ..., q_m) & \text{if } t = f(t_1, ..., t_m),\ 0 < m \leq M,\ f \in \Sigma_m \\ \text{undefined} & \text{otherwise} \end{cases} \tag{1}$$

(M is the maximum number of children for any node of any tree in $L(A)$). Deterministic FSTA are a special case of NFSTA, namely, when $\delta_m : \Sigma_m \times Q^m \to Q$.

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

$$L(A) = \{t \in \Sigma^T : \delta(t) \cap F \neq \emptyset\}. \tag{2}$$

3 A Recursive Neural Net to Encode Tree Automata

Here we define a recursive neural architecture that is similar to those used in related work as that of Frasconi, Gori and Sperduti [5], Sperduti [9] and Sperduti and Starita [10].

A high-order Elman-like recursive neural network consists of one set of single-layer neural networks which computes the next state (playing the role of the collection Δ of transition functions in a finite-state tree transducer) and one single-layer feedforward neural network with a single output node which detects the existence of an accepting state in the set of states active after processing a tree. The schematics of this recursive neural network architecture are shown in Fig. 1

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

[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.

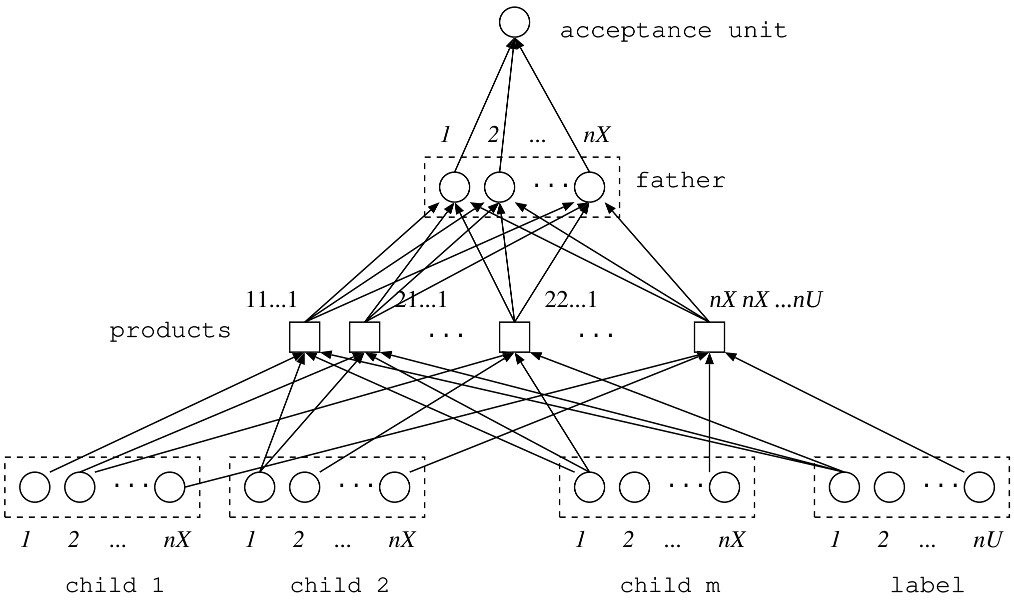

Fig. 1. A high order Elman-like recursive neural network having m state ports with n_X units, one label port with n_U units, and a single output unit (the acceptance unit).

and $m+1$ input ports: m for the input of subtree state vectors, each 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 $\sigma_k^m \in \Sigma_m$ (the k-th symbol of rank m) 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:

$$x_i[\mu] = g\left(w_i^m + \sum_{k=1}^{n_U} \sum_{j_1=1}^{n_X} \cdots \sum_{j_m=1}^{n_X} w_{ij_1 j_2 \ldots j_m k}^m \, x_{j_1}[\nu_1] x_{j_2}[\nu_2] \cdots x_{j_m}[\nu_m] u_k[\mu]\right) \tag{3}$$

where w_i^m represents the bias for the network of rank m and $i = 1, \ldots, n_X$ and the $w_{ij_1 j_2 \cdot j_m k}^m$ are the weights for that network. If μ is a leaf, i.e., $l(\mu) \in \Sigma_0$ the expression above for the component $x_i[\mu]$ reduces to

$$x_i[\mu] = g\left(w_i^0 + \sum_{k=1}^{n_U} w_{ik}^0 u_k[\mu]\right), \tag{4}$$

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

Acceptance is expressed by a single unit connected to all of the state units:

$$y[\mu] = g\left(v + \sum_{j=1}^{n_X} v_j x_{j[\mu]}\right). \tag{5}$$

4 Encoding Tree Automata In Recursive Neural Networks

4.1 Using Threshold Linear Units

Here, we present a way to encode a NFSTA in a RNN as the one defined above using TLU as activation functions ($g = \theta$, with $\theta(x) = 1$ if $x \geq 0$ and 0 otherwise). The encoding is based on the following scheme for states. Each of the M next-state networks in the RNN will have $n_X = |Q|$ state units and $n_U = |\Sigma_m|$ and will be interpreted as being in state $q_i \in Q$ after reading tree μ if $x_i[\mu]$ is high, and as not being in state $q_j \in Q$ if $x_j[\mu]$ is low. In this way, the RNN, even if it is a deterministic device, may be interpreted as being in more than one state of the NFSTA.

Next-state networks: Weights are chosen as follows: biases v_i^m on all state units of all of the next-state are equal to $-1/2$, to bring state units to zero if they do not receive any sufficiently positive contribution. Weights $w^m_{ij_1 j_2 \ldots j_m k}$ are 1 if $q_i \in \delta_m(\sigma^m_k, q_{j_1}, q_{j_2}, \ldots, q_{j_m})$ and σ^m_k is the k-th symbol of Σ_m, and 0 otherwise. In this way, the unit i representing state q_i will be high when (a) the k-th symbol of Σ_m labels the node, (b) the sets of states at the m children nodes include each one at least state q_{j_m} (that is unit x_{j_m} is high), and (c) $q_i \in \delta_m(\sigma^m_k, q_{j_1}, q_{j_2}, \ldots, q_{j_m})$. In that case, unit i receives a positive contribution (perhaps not the only one) and becomes high. If there is not any such contribution, it remains low. This construction, therefore, emulates the next-state functions δ_m of the NFSTA.

Acceptance unit: This unit has a bias $v = -1/2$ to keep it low in the absence of sufficiently positive inputs. Weights v_j are 1 if $q_j \in F$ and zero otherwise. In this way, if at least one of the states at the root node of a tree is an acceptance state, the output will be high, but zero otherwise.

4.2 Using Sigmoid Units

Our main hypothesis is that substituting in the previous construction the step function θ by a scaled version of the logistic sigmoid function

$$g_L(Hx) = \frac{1}{1 + \exp(-Hx)} \tag{6}$$

where H is a positive gain, there is a finite value of H that ensures a correct NFSTA behavior of the resulting RNN (obviously, in the limit $H \to \infty$ the behavior is correct because one recovers the step function θ). We will look for the smallest value of H that guarantees correct behavior.

For a binary interpretation of the output of all sigmoid units, we will use the following criterion. Two special values, $\epsilon_0, \epsilon_1 \in [0, 1]$, $\epsilon_0 < \epsilon_1$ will be defined so that the outputs of all units will be taken to be *low* if they are in $[0, \epsilon_0]$, *high* if they are in $[\epsilon_1, 1]$ and *forbidden* otherwise.

To ensure correct behavior in all cases, we have to ensure correct behavior in the worst cases, that is, when low and high values are the farthest possible from the ideal values 0 and 1 respectively.

Conditions for the next-state network: For a next state function of rank m two worst cases are possible. Let us study a typical transition $\delta_m(\sigma^m_k, q_{j_1}, q_{j_2}, \ldots, q_{j_m})$ leading at a node t; on the one hand, we want all $x_i[t]$ such that $q_i \in \delta_m(\sigma^m_k, q_{j_1}, q_{j_2}, \ldots, q_{j_m})$ to be high whenever the corresponding $x_{j_k}[t-1]$, $1 \leq k \leq m$ are high at the children nodes and $u_k[t] = 1$, and we want all other $x_{i'}[t]$ to be low. To ensure $x_i[t] \geq \epsilon_1$ even in the worst case, when it only receives a single positive contribution from units that have the weakest possible high value, ϵ_1, one gets:

$$g_L\left(H(\epsilon_1)^m - H/2\right) > \epsilon_1 \tag{7}$$

Ensuring $x_i[t] \leq \epsilon_0$ even in the worst case is more complex. One has to first consider each possible *configuration* of the m state ports (each possible configuration of low and high states), then consider which one would be the worst case for that configuration, and finally look for the worst configuration.

Since we want $x_i[t]$ to be low, the worst case in each configuration $\mathbf{k}$ is the one in which $x_i[t]$ receives the largest possible positive contribution from state combinations $(q_{l_1}, q_{l_2}, \ldots q_{l_m})$ not currently present in children but corresponding to allowed transitions into state q_i $(q_i \in \delta_m(\sigma_k^m, q_{l_1}, q_{l_2}, \ldots q_{l_m}))$. The worst possible contribution of each state combination that is not currently present (*low contribution* in the following) is a product of ϵ_0 for each state not present (at least one) and 1 for each state present (that is, each of these contributions ranks from ϵ_0 to ϵ_0^k). For each possible combination $(q_{l_1}, q_{l_2}, \ldots q_{l_m})$ of high and low units in each port in a given port configuration $\mathbf{k}$, the worst case is that all the possible low contributions correspond to allowed transitions and thus contribute noise to the desired low value of $x_i[t]$. We will call the sum of all possible low contributions of a configuration $\mathbf{k}$ the *worst low contribution* of that configuration, $C(\mathbf{k}, n_X, m, \epsilon_0)$. The worst case for any RNN having m state ports with n_X units is the maximum worst low contribution for all possible combinations of state, $C^*(n_X, m, \epsilon_0) = \max_{\mathbf{k}} C(\mathbf{k}, n_X, m, \epsilon_0)$. We have not been able to obtain this value analytically but instead we have performed an exhaustive search for this maximum over all $2^{n_X m}$ combinations[2]: each configuration $\mathbf{k}$ is defined by a vector $\mathbf{k} = (k_1, k_2, \ldots, k_m)$ specifying the number $k_i \in [1, n_X]$ $(i \in [1, m])$ of low units in each port and its worst contribution may easily be shown to be

$$C(\mathbf{k}, n_X, m, \epsilon_0) = \prod_{i=1}^{m}(n_X - k_i + k_i\epsilon_0) - \prod_{i=1}^{m}(n_X - k_i) \tag{8}$$

(of course many contributions are equal due to symmetries). The condition for a low value of $x_i[t]$ is, then, in the worst case,

$$g_L\left(H(C^*(n_X, m, \epsilon_0) - 1/2)\right) < \epsilon_0. \tag{9}$$

Conditions for the acceptance unit: If the output of the acceptance unit has to be high, the worst case occurs when only one state unit is contributing and its contribution is the weakest possible. The condition is therefore

$$g_L\left(H(\epsilon_1 - 1/2)\right) > \epsilon_1. \tag{10}$$

The worst case for a low value of the output of the acceptance unit is that all acceptance state units have the worst low value (ϵ_0) and all states but one are acceptance states. The condition is therefore

$$g_L\left(H((n_X - 1)\epsilon_0 - 1/2)\right) < \epsilon_0 \tag{11}$$

[2] Analytical upper bounds to H are however easily obtained for the oversimplified (impossible) worst case in which all contributions are equal to ϵ_0, and there are $n_X^m - 1$ of them; these bounds are inordinately high and therefore of no interest.

Values of H: The values of H we are looking for are the smallest values of H compatible with values of ϵ_0 and such that ϵ_1 such that $0 < \epsilon_0 < \epsilon_1 < 1$ and all conditions (7)–(11) are fulfilled. It happens to be the case that the most stringent condition is (9); the minimum value of H determined on that one may then be substituted in (7) to obtain the value of ϵ_1, because this value will always ensure that condition (10) is fulfilled.

Table 1 shows values of H for a set of representative values of M (maximum m for a given NFSTA) and n_X. The values are very high; the sigmoid RNN works almost like a discrete-state RNN because neurons are usually very saturated; the reader is however, reminded of the fact that these values of H are (a) derived from a worst-case-based study (which considers the maximum possible number of transitions); (b) used to scale a particular discrete-state RNN encoding of the NFSTA; smaller values of H may therefore still guarantee correct NFSTA behavior for many NFSTA. If, instead of using a single scaling parameter H specialized parameters were used for each unit (as in [8]), weight values would get even smaller.

	$M=1$	$M=2$	$M=3$	$M=4$
$n_X=2$	7.18	7.38	8.47	9.25
$n_X=3$	8.36	10.22	11.93	14.25
$n_X=4$	9.15	11.92	14.52	17.04
$n_X=5$	9.73	13.14	16.36	19.48

Table 1. *General values of the scaling factor H as a function of the number of state units n_X and the maximum rank M for NFSTA encoding on a high-order Elman recursive neural network.*

5 Conclusion

We have studied a strategy to encode nondeterministic finite-state tree automata (NFSTA) on a high-order sigmoid recursive neural network (RNN) architecture. This strategy has been derived from a strategy to encode a NFSTA in a discrete-state RNN by first turning it into a sigmoid RNN and then looking for the smallest value of the gain H of the sigmoid that ensures correct NFSTA behavior, using a worst-case criterion. It has to be noted that the values of H obtained are derived from general worst cases that may not occur in general, and therefore sigmoid RNN having a smaller gain may still show correct NFSTA behavior. The values of H obtained suggest that, even though *in principle* RNN with finite weights can simulate exactly the behavior of NFSTA, 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.

References

1. Rafael C. Carrasco and Mikel L. Forcada. Simple strategies to encode tree automata in analog recursive neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2000. Accepted for publication.
2. Rafael C. Carrasco, Mikel 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. Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A general framework for adaptive data structures processing. *IEEE Transactions on Neural Networks*, 9(5):768–786, 1998.
6. R.C. Gonzalez and M.G. Thomason. *Syntactical pattern recognition*. Addison-Wesley, Menlo Park, CA, 1978.
7. Stefan C. Kremer. On the computational power of Elman-style recurrent networks. *IEEE Transactions on Neural Networks*, 6(4):1000–1004, 1995.
8. Stefan C. Kremer. *A Theory of Grammatical Induction in the Connectionist Paradigm*. PhD thesis, Department of Computer Science, University of Alberta, Edmonton, Alberta, 1996.
9. Alessandro Sperduti. On the computational power of neural networks for structures. *Neural Networks*, 10(3):395–400, 1997.
10. Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.
11. 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.