Learning deterministic regular grammars from stochastic samples in polynomial time*

Rafael C. Carrasco and Jose Oncina
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante
E-mail: (carrasco, oncina)@dlsi.ua.es

Running head: Learning stochastic regular grammars

Abstract

In this paper, the identification of stochastic regular languages is addressed. For this purpose, we propose a class of algorithms which allow for the identification of the structure of the minimal stochastic automaton generating the language. It is shown that the time needed grows only linearly with the size of the sample set and a measure of the complexity of the task is provided. Experimentally, our implementation proves very fast for application purposes.

Resumé

Dans cet article, on étudie l'identification de langages réguliers stochastiques. Pour ce but, nous proposons une classe d'algorithmes lesquels permettent la identification de la structure de l'automate stochastique minime que engendre le langage. On trouve que le temps nécessaire croissait linéairement avec la taille de l'échantillon et on donne une mesure de la complexité de la identification. Expérimentalement, notre implementation est très rapide, ce que le rend très intéressant pour des applications.

*Work partially supported by the Spanish CICYT under grant TIC97–0941.

1 Introduction

Identification of stochastic regular languages (SRL) represents an important issue within the field of grammatical inference. Indeed, in most applications —as speech recognition, natural language modeling, and many others— the learning process involves noisy or random examples. The assumption of stochastic behavior has important consequences for the learning process. Indeed, Gold (1967) introduced the criterion of *identification in the limit* for successful learning of a language. He also proved that regular languages cannot be identified if only *text* (i.e., a sample containing only examples of strings in the language) is given, but they can be identified if a complete presentation is provided. A *complete presentation* is sample containing strings classified as belonging (positive examples) or not (negative examples) to the language. In practice, negative examples are usually scarce or difficult to obtain. As proved by Angluin (1988), stochastic samples (i.e., samples generated according to a given probability distribution) can compensate the lack of negative data, although they do not enlarge the class of languages that can be identified.

Some attempts to find suitable learning procedures using stochastic samples have been made in the past. For instance, Maryanski and Booth (1977) used a chi-square test in order to filter regular grammars provided by heuristic methods. Although convergence to the true one was not guaranteed, acceptable grammars (i.e., statistically close to the sample set) were always found. The approach of van der Mude and Walker (1978) merges variables in a stochastic regular grammar, where Bayesian criteria are applied. In that paper (van der Mude & Walker 1978), convergence to the true grammar was not proved and the algorithm was too slow for application purposes.

In the recent years, neural network models were used in order to identify regular languages (Smith & Zipser 1989; Pollack 1991; Giles et al. 1992; Watrous & Kuhn 1992) and they have also been applied to the problem of stochastic samples (Castaño, Casacuberta & Vidal 1993). However, these methods share the serious drawback that long computational times and vast sample sets are needed. Hidden Markov models are used by Stolcke and Omohundro (1993). In order to maximize the probability of the sample, they include *a priori* probabilities penalizing the size of the automaton.

Oncina and García (1992) proposed an algorithm, similar to the one presented by Lang (1992) which allows for the correct identification in the limit of any regular language if a complete presentation is given. Moreover, the time needed by this algorithm in order to output a hypothesis grows polynomially with the size of the sample, and a linear time complexity was found experimentally. In this paper, we follow a similar approach and develop the algorithm `rlips`(Regular Language Inference from Probabilistic Samples) which builds the prefix tree automaton from the sample and evaluates at every node the relative probabilities of the transitions coming out from the node. Next, it compares pairs of nodes, following a well defined order (essentially, that of the levels in the prefix tree acceptor or lexicographical order). Equivalence of the nodes is accepted if they generate —within statistical uncertainty— the same stochastic language. The process ends when further comparison is not possible.

A preliminary version of the algorithm was already presented in Carrasco and Oncina (1994). Here we develop a modified version which allows us to prove that, with probability one, the algorithm identifies the correct structure of the automaton generating the language.

Meanwhile, an algorithm with a different learning model (the PAC model) and some connection points with ours has been proposed by Ron, Singer and Tishby (1995). The differences between both approaches will be commented in the next section, as well as the differences between stochastic and non-stochastic identification. Some definitions will be introduced in section 3. A more detailed description of our algorithm can be found in section 4, which is proved to be correct in section 5. Finally, results and discussion will be presented in section 6.

2 Identification of stochastic languages

At this point, it is worthwhile to remark on some differences between the identification process of stochastic and non-stochastic regular languages. Identification in the limit means that only finitely many changes of hypothesis take place before a correct one is found. Non-stochastic regular languages form a recursively enumerable set of classes $R = \{L_1, L_2, \ldots\}$ and a simple enumerative procedure identifies in the limit R provided that a complete sample S is provided. A complete sample presents all strings classified as belonging or not to the language. If L_r is the true hypothesis, there is only a finite number of incorrect L_k preceding L_r, and for all of them a counterexample exists in S. Therefore, by choosing as hypothesis the first L_k consistent with the first n strings in S, all incorrect languages will be rejected provided that n is large enough (say $n > N$). Obviously, the hypothesis is changed finitely many times (at most N times). Of course, negative examples play a relevant role, since they may be necessary in order to reject languages whose only difference with L_r lies on $L_k - L_r$ (and they may exist because an order which respects inclusion is not generally possible).

In contrast, samples of stochastic languages contain only examples which appear repeatedly, according to a probability distribution $p(w|L)$ giving the probability of the string w in the language L. There are no negative examples in the sample and therefore, no explicit information about strings such that $p(w|L) = 0$.

However, the statistical regularity is able to compensate for the lack of negative examples (Angluin, 1988). In particular, stochastic regular languages with rational probabilities are identifiable with probability one, by simply using enumerative algorithms. Because enumerative methods are experimentally unfeasible, the search of fast and reliable algorithms for identification becomes a challenging task.

A widespread measure for the success in learning a probability distribution is the Kullback-Leibler distance or *relative entropy* (see Cover & Thomas 1991). One can use this measure, for instance, in the reduced problem of learning the bias p of a coin. A traditional approach is to estimate p with $\hat{p}$, the rate between the number of heads and the number of tosses. It is also possible to define a procedure in order to identify the bias p, provided that p is rational. However, except for very simple rational values of p the estimation $\hat{p}$ gives better results (in terms of relative entropy) than the identification procedure. A typical result is shown in Figs. 1 and 2.

The situation changes when the number of possible outcomes in the experiment is infinite. The *support* of L is the subset $R_L = \{w \in \mathcal{A}^* : p(w|L) > 0\}$ of non-zero probability strings. For most languages, R_L is infinite and thus, there are strings whose probability is as small as desired. Therefore, many strings in R_L will not be represented by a finite sample and their probability will be incorrectly estimated as zero, leading to a large relative entropy. According to this, we have chosen to identify the structure of the canonical generator of the language and then, estimate the transition probabilities (which are a finite set of numbers) from the sample. Note that it is not enough to identify the *support* R_L, as the minimal acceptor for R_L is often smaller than the canonical generator for L.

In this way, we will find that the relative entropy between our model and the true distribution decreases very fast as the sample size grows, something that cannot be achieved without a good estimation of the probabilities of all the strings in R_L, especially for those not contained in the finite sample. It is important to remark that we make no assumption about the underlying stochastic automaton. In contrast, the algorithm of Ron, Singer and Tishby (1995) assumes that the states in the automaton are distinguishable at a given degree μ and only outputs acyclic automata (in particular automata whose transitions go from states in level d to states in level $d+1$). Although one can always find an acyclic automaton close to the target one, our approach identifies the structure even when cycles are present.

3 Definitions

Let $\mathcal{A}$ be a finite alphabet, $\mathcal{A}^*$ the free monoid of strings generated by $\mathcal{A}$ and λ the empty string. The length of $w \in \mathcal{A}^*$ will be denoted as $|w|$. For $x, y \in \mathcal{A}^*$, if $w = xy$ we will also write $y = x^{-1}w$. The expression $x\mathcal{A}^*$ denotes the set of strings which contain x as a prefix. On the other hand,

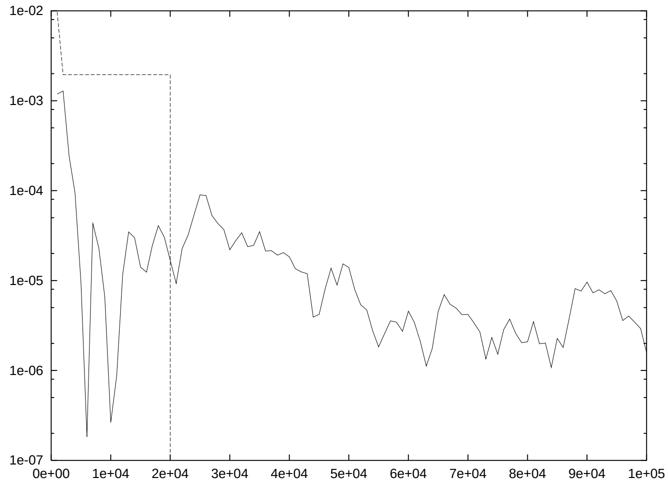

Figure 1: Typical plot of the relative entropy (bits) between $p = 0.875$ and the experimental bias as a function of the number of tosses. Continuous line: estimation. Dotted line: identification procedure (drops to zero after 20000 experiments).

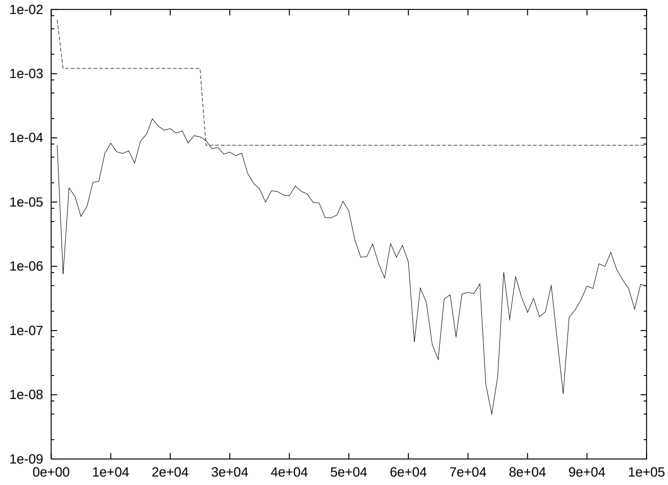

Figure 2: Same as Fig. 1 with a bias $p = 0.62$. Identification takes place too late for practical purposes.

$x < y$ in *lexicographical order* if either $|x| < |y|$ or $|x| = |y|$ and x precedes y alphabetically.

A *stochastic language* L is defined by a probability density function over $\mathcal{A}^*$ giving the probability $p(w|L)$ that the string $w \in \mathcal{A}^*$ appears in the language. The probability of any subset $X \subset \mathcal{A}^*$ is given by

$$p(X|L) = \sum_{x \in X} p(x|L), \tag{1}$$

and the *identity* of stochastic languages is interpreted as follows:

$$L_1 = L_2 \Leftrightarrow p(w|L_1) = p(w|L_2) \; \forall w \in \mathcal{A}^*, \tag{2}$$

or, equivalently,

$$L_1 = L_2 \Leftrightarrow p(w\mathcal{A}^*|L_1) = p(w\mathcal{A}^*|L_2) \; \forall w \in \mathcal{A}^*. \tag{3}$$

In other approaches (Ron, Singer, Tishby 1995) a minimal difference $\mu > 0$ between the probabilities is assumed. However, we will make no assumption of this kind about the probability distribution.

A *stochastic regular grammar* (SRG), $G = (\mathcal{A}, V, S, R, p)$, consists of a finite alphabet $\mathcal{A}$, a finite set of variables V —one of which, S, is referred to as the starting symbol—, a finite set of derivation rules R with either of the following structures:

$$\begin{array}{lcl} X & \rightarrow & aY \\ X & \rightarrow & \lambda \end{array} \tag{4}$$

where $a \in \mathcal{A}$, $X, Y \in V$, and a real function $p : R \rightarrow [0, 1]$ giving the probability of the derivation. The sum of the probabilities for all derivations from a given variable X must be equal to one. The form of Eq. (4), although formally different, is equivalent to other ones used in the literature, as in Fu (1982). The stochastic grammar G is *deterministic* if for all $X \in V$ and for all $a \in \mathcal{A}$ there is at most one $Y \in V$ such that $p(X \rightarrow aY) \neq 0$.

Every stochastic deterministic regular grammar G defines a *stochastic deterministic regular language* (SDRL), L_G, through the probabilities $p(w|L_G) = p(S \Rightarrow w)$. The probability $p(S \Rightarrow w)$ that the grammar G generates the string $w \in \mathcal{A}^*$ is defined in a recursive way:

$$\begin{array}{rcl} p(X \Rightarrow \lambda) & = & p(X \rightarrow \lambda) \\ p(X \Rightarrow aw) & = & p(X \rightarrow aY) p(Y \Rightarrow w) \end{array} \tag{5}$$

where Y is the only variable satisfying $p(X \rightarrow aY) \neq 0$ (if such variable does not exist, then $p(X \rightarrow aY) = 0$).

A *stochastic deterministic finite automaton* (SDFA), $A = (Q^A, \mathcal{A}, \delta^A, q_I^A, p^A)$, consists of an alphabet $\mathcal{A}$, a finite set of nodes $Q^A = \{q_1, q_2, \dots q_n\}$, with $q_I^A \in Q^A$ the initial node, a transition function $\delta^A : Q^A \times \mathcal{A} \rightarrow Q^A$ and a probability function $p^A : Q^A \times \mathcal{A} \rightarrow [0, 1]$ giving the probability $p(q_i, a)$ that symbol a follows after a prefix leading to state q_i. The probability $p^A(q_i, \lambda)$ is defined as

$$p^A(q_i, \lambda) = 1 - \sum_{a \in \mathcal{A}} p^A(q_i, a) \tag{6}$$

and represents the probability that the string ends at node q_i or, equivalently, an end of string symbol follows the prefix. The constraint $p^A(q_i, \lambda) \geq 0$ holds for all correctly defined SDFA. As usual, the transition function is extended to $\mathcal{A}^*$ as $\delta^A(q_i, aw) = \delta^A(\delta^A(q_i, a), w)$.

Every SDFA A defines a SDRL, L_A, through the probabilities $p(w|L_A) = \pi^A(q_I^A, w)$, defined recursively as

$$\begin{array}{rcl} \pi^A(q_i, \lambda) & = & p^A(q_i, \lambda) \\ \pi^A(q_i, aw) & = & p^A(q_i, a)\pi^A(\delta^A(q_i, a), w) \end{array} \tag{7}$$

If $\delta^A(q_i, a)$ is undefined, then $\pi^A(\delta^A(q_i, a), w) = 0$.

A comparison of equations (5) and (7) shows the equivalence between SDRG and SDFA. In case the SDRG contains no *useless symbols* (Hopcroft and Ullman 1979), the probabilities of the strings sum up to 1:

$$p(\mathcal{A}^*|L_G) = \sum_{w \in \mathcal{A}^*} p(w|L_G) = 1 \tag{8}$$

The *quotient* $x^{-1}L$ is the stochastic language defined by the probabilities of the strings in L starting with x, conveniently normalized:

$$p(w|x^{-1}L) = \frac{p(xw|L)}{p(x\mathcal{A}^*|L)} \tag{9}$$

If $p(x\mathcal{A}^*|L) = 0$, then by convention $x^{-1}L = \emptyset$ and $p(w|x^{-1}L) = 0$. Note that $\lambda^{-1}L = L$.

If L is a SDRL, the *canonical generator* $M = (Q^M, \mathcal{A}, \delta^M, q_I^M, p^M)$ is defined as:

$$\begin{array}{rcl} Q^M & = & \{x^{-1}L \neq \emptyset : x \in \mathcal{A}^*\} \\ \delta^M(x^{-1}L, a) & = & (xa)^{-1}L \\ q_I^M & = & \lambda^{-1}L \\ p^M(x^{-1}L, a) & = & p(a\mathcal{A}^*|x^{-1}L) \end{array} \tag{10}$$

The automaton M is the minimal SDFA generating L, and its construction is supported by the following facts which allow us to extend the Myhill-Nerode theorem (Hopcroft & Ullman 1979) for stochastic automata:

1. The automaton M is finite and not larger than any other automaton $A = (Q^A, \mathcal{A}, \delta^A, q_I^A, p^A)$ generating L. By writing $q_x = \delta^A(q_I, x)$, and making repeated use of (7) one gets from the definition (9),

$$p(w|x^{-1}L_A) = \frac{\pi^A(q_I, xw)}{\pi^A(q_I, x\mathcal{A}^*)} = \frac{\pi^A(q_x, w)}{\pi^A(q_x, \mathcal{A}^*)} = \pi^A(q_x, w). \tag{11}$$

As the number of different values for q_x is bounded by $|Q^A|$, the size of the automaton A, so is the number of different languages $x^{-1}L$, and therefore $|Q^M| \leq |Q^A|$.

2. The transition function δ^M is well defined, i.e.,

$$x^{-1}L = y^{-1}L \Rightarrow \delta^M(x^{-1}L, a) = \delta^M(y^{-1}L, a). \tag{12}$$

Indeed, for all $w \in \mathcal{A}^*$

$$p(w|a^{-1}x^{-1}L) = \frac{p(aw|x^{-1}L)}{p(a\mathcal{A}^*|x^{-1}L)} = \frac{p(xaw|L)}{p(xa\mathcal{A}^*|L)} = p(w|(xa)^{-1}L). \tag{13}$$

and therefore $(xa)^{-1}L = a^{-1}x^{-1}L$. With this, Eq. (12) is straightforward. In addition, the previous relation allows one to write $\delta^M(q_I, w) = w^{-1}L$.

3. The automaton M generates $L_M = L$. In fact, it is easier to prove $\pi^M(x^{-1}L, w\mathcal{A}^*) = p(w\mathcal{A}^*|x^{-1}L)$ for all $x, w \in \mathcal{A}^*$, which includes the initial state ($x = \lambda$) as a special case. The equation trivially holds for all x when $w = \lambda$. According to (7)

$$\pi^M(x^{-1}L, aw\mathcal{A}^*) = p^M(x^{-1}L, a)\pi^M((xa)^{-1}L, w\mathcal{A}^*) \tag{14}$$

Finally, by induction in w and using (10), one gets

$$\pi^M(x^{-1}L, aw\mathcal{A}^*) = p(a\mathcal{A}^*|x^{-1}L)p(w\mathcal{A}^*|(xa)^{-1}L) = p(aw\mathcal{A}^*|L). \tag{15}$$

In order to identify the canonical generator, we need to define the *prefix set* and the *short-prefix set* of L as:

$$\mathrm{Pr}(L) = \{x \in \mathcal{A}^* : x^{-1}L \neq \emptyset\} \tag{16}$$

$$\mathrm{Sp}(L) = \{x \in \mathrm{Pr}(L) : x^{-1}L = y^{-1}L \Rightarrow x \leq y\} \tag{17}$$

Note that $x^{-1}L \neq y^{-1}L$ for all $x, y \in \mathrm{Sp}(L)$ such that $x \neq y$, and therefore, the strings in $\mathrm{Sp}(L)$ are representatives of the states in the canonical generator M. Accordingly, we will use them in the construction of M and add transitions of the type $\delta(x, a) = xa$, except when xa is not a short prefix. In order to deal with these undefined transitions we will use the *kernel* and the *frontier set* of L, defined respectively as:

$$K(L) = \{\lambda\} \cup \{xa \in \mathrm{Pr}(L) : x \in \mathrm{Sp}(L) \wedge a \in \mathcal{A}\} \tag{18}$$

$$F(L) = K(L) - \mathrm{Sp}(L) \tag{19}$$

Note that $K(L)$ has size at most $1 + |M||\mathcal{A}|$ and contains $\mathrm{Sp}(L)$ as a subset.

Our aim is to identify the canonical generator from random examples. A *stochastic sample* S of the language L is an infinite sequence of strings generated according to the probability distribution $p(w|L)$. We denote with S_n the sequence of the n first strings (not necessarily different) in S, which will be used as input for the algorithm. The number of occurrences in S_n of the string x will be denoted with $c_n(x)$, and for any subset $X \subset \mathcal{A}^*$,

$$c_n(X) = \sum_{x \in X} c_n(x). \tag{20}$$

The sequence S_n defines a stochastic language L_n with the probabilities

$$p(x|L_n) = \frac{1}{n} c_n(x). \tag{21}$$

The *prefix tree automaton* of S_n is a SDFA, $T_n = (Q^T, \mathcal{A}, \delta^T, q_I^T, p^T)$, which generates L_n and can be interpreted as a model for the target language L assigning to every string the experimental probability. Formally,

$$\begin{array}{rcl} Q^T & = & \mathrm{Pr}(L_n) \\ \delta^T(x,a) & = & \left\{ \begin{array}{ll} xa & \text{if } xa \in \mathrm{Pr}(L_n) \\ \emptyset & \text{otherwise} \end{array} \right. \\ q_I^T & = & \lambda \\ p^T(x,a) & = & \frac{c_n(xa\mathcal{A}^*)}{c_n(x\mathcal{A}^*)} \end{array} \tag{22}$$

Probabilities of the type $p^T(x, \lambda)$ are evaluated according to (6).

4 The inference algorithm

We define the boolean function $\texttt{equiv}_L : K(L) \times K(L) \to \{\text{true}, \text{false}\}$ as

$$\texttt{equiv}_L(x,y) = \text{true} \Leftrightarrow x^{-1}L = y^{-1}L. \tag{23}$$

Note that $\texttt{equiv}_L$ is an equivalence relation for the strings in the kernel $K(L)$. We will make use of the following lemma:

Lemma 1 *Given L, a SDRL, the structure of the canonical generator of L is isomorphic to:*

$$\begin{array}{rcl} Q & = & \mathrm{Sp}(L) \\ q_I & = & \lambda \\ \delta(x,a) & = & y \end{array} \tag{24}$$

where, for every $(x,a) \in \mathrm{Sp}(L) \times \mathcal{A}$, y is the only string in $\mathrm{Sp}(L)$ *such that* $\texttt{equiv}_L(xa, y)$.

Proof. Let $\Phi : Q \to Q^M$ be defined as $\Phi(x) = x^{-1}L$. The mapping Φ is an isomorphism if $\delta^M(\Phi(x), a) = \Phi(\delta(x,a))$, which means $(xa)^{-1}L = y^{-1}L$. Therefore, Φ is isomorphism if and only if y is a string in $\mathrm{Sp}(L)$ satisfying

$\texttt{equiv}_L(xa, y)$ and, according to definition (17), y is unique. Note that $x \in \mathrm{Sp}(L) \Rightarrow xa \in K(L)$, and $\texttt{equiv}_L$ remains well defined. ■

The next lemma shows that the problem of inferring the structure of the canonical generator can be reduced to that of learning the correct function $\texttt{equiv}_L$.

Lemma 2 *The structure of the canonical generator of L can be obtained from $\texttt{equiv}_L$ and any $D \subset \mathrm{Pr}(L)$ such that $K(L) \subset D$ with the algorithm depicted in Fig. 3, which gives $\mathrm{Sp}(L)$ and $F(L)$ as byproducts.*

Proof.(sketch) Induction in the number of iterations shows that $\mathrm{Sp}^{[i]} \subset \mathrm{Sp}(L)$, $F^{[i]} \subset F(L)$ and $W^{[i]} \subset K(L)$, where the super-index denotes the result after i iterations. On the other hand, if xa is in $K(L)$, induction in the length of the string shows that xa eventually enters the algorithm. Following Lemma 1, for every $x \in \mathrm{Sp}(L)$, if xa is also in $\mathrm{Sp}(L)$, then $\delta(x, a) = xa$. However, if $xa \notin \mathrm{Sp}(L)$, there exists $y \in \mathrm{Sp}(L)$ such that $\texttt{equiv}_L(xa, y)$ and $\delta(x, a) = y$. ■

The algorithm 3 performs a branch and bound process following the prefix tree. Every time a short prefix x is found (a string which has no shorter equivalent string) the possible continuations xa are added as candidates for elements in Sp. On the contrary, if x is not a short prefix, no string is added and only the corresponding transition is stored.

One can replace the subset D with $\mathrm{Pr}(L_n) \subset \mathrm{Pr}(L)$, which contains $K(L)$ when n large enough. On the other hand, $\texttt{equiv}_L$ is always well defined because the function is never called out of its domain. As $x \in K(L)$ and $y \in \mathrm{Sp}(L)$, the algorithm makes at most $|K(L)| \times |\mathrm{Sp}(L)|$ calls to $\texttt{equiv}_L$. Thus, the global complexity of the algorithm is $\mathcal{O}(|\mathcal{A}||M|^2)$ times the complexity of function $\texttt{equiv}_L$.

5 Convergence of the algorithm

In order to evaluate the equivalence relation $x^{-1}L = y^{-1}L$, we will use a variation of (3) which improves[1] convergence:

$$L_1 = L_2 \Leftrightarrow p(a\mathcal{A}^*|z^{-1}L_1) = p(a\mathcal{A}^*|z^{-1}L_2) \; \forall a \in \mathcal{A}, z \in \mathcal{A}^* \tag{25}$$

Taking into account (10), the above relation means that for all $z \in \mathcal{A}^*$ and $a \in \mathcal{A} \cup \{\lambda\}$

$$p^M((xz)^{-1}L, a) = p^M((yz)^{-1}L, a) \tag{26}$$

In practice, L is unknown and function $\texttt{equiv}_L(x, y)$, defined as $x^{-1}L = y^{-1}L$, is replaced with the experimental function $\texttt{compatible}_n(x, y)$, which checks $x^{-1}L_n = y^{-1}L_n$ instead. This means using p^T instead of p^M in (26).

[1]This method allows one to distinguish different probabilities faster, as more information is always available about a prefix than about the whole string.

```
algorithm rlips
input: D ⊂ Pr(L) such that K(L) ⊂ D
output: Q^M = Sp (short prefix set)
        F (frontier set)
        δ^K (transition function)

begin algorithm
  Sp = {λ}      (short prefix set)
  F = ∅      (frontier set)
  W = 𝒜      (candidate strings)
  do ( while W ≠ ∅ )
    x = min W
    W = W − {x}
    if ∃y ∈ Sp : equiv_L(x, y) then
      F = F ∪ {x}      [x is not a short prefix]
      δ^M(w, a) = y     [with wa = x, a ∈ 𝒜, w ∈ 𝒜*]
    else
      Sp = Sp ∪ {x}     [x is a short prefix]
      W = W ∪ {xa ∈ D : a ∈ 𝒜}     [add new candidates]
      δ^M(w, a) = x      [with wa = x, a ∈ 𝒜, w ∈ 𝒜*]
    endif
  end do
end algorithm
```

Figure 3: Algorithm `rlips`.

As L_n is stochastic, a confidence range has to be defined for the difference between the probabilities in $x^{-1}L_n$ and $y^{-1}L_n$. There is a number of different statistical tests (Hoeffding 1963; Feller 1950; Anthony & Biggs 1992) leading to a class of algorithms rather than a single one. We have chosen the Hoeffding (1963) bound as described in the Appendix and implemented in function `different` (Fig. 5). It returns the correct answer with probability greater than $(1-\alpha)^2$, being α an arbitrarily small positive number. Because the number of checks grows when the size t of the prefix tree automaton grows, we will allow the parameter α to depend on n.

According to (26), compatibility of two states x and y in Q^T will be rejected if some $z \in \mathcal{A}^*$ is found such that the estimated transition probabilities from xz and yz are `different`. We will show that $\texttt{compatible}_n(x,y)$, as plotted in Fig. 4, returns in the limit of large n the same value as $\texttt{equiv}_L(x,y)$ for all $x, y \in K(L)$. Therefore, following Lemma 2, the correct structure of the canonical acceptor can be inferred in the limit, and the transition probabilities $p^M(x,a)$ defined in Eq. (10) can be evaluated from S_n by means of the experimental ones $p^T(x,a)$, defined in Eq. (22).

```
algorithm compatible_n
input: x, y (strings)
       T_n (prefix tree automaton)
output: boolean
begin algorithm
  do ( ∀z ∈ A*: xz ∈ Pr(L_n) ∨ yz ∈ Pr(L_n) )
     if different (c_n(xz), c_n(xzA*), c_n(yz), c_n(yzA*), α) then
        return FALSE
     endif
     do ( ∀a ∈ A )
        if different (c_n(xzaA*), c_n(xzA*), c_n(yzaA*), c_n(yzA*), α) then
           return FALSE
        endif
     end do
  end do
  return TRUE
end algorithm
```

Figure 4: Algorithm `compatible`. Function `different` is plotted in Fig. 5.

Theorem 3 *Let the parameter* α_n *in function* `different` *be such that the sum* $\sum_{n=0}^{\infty} n\,\alpha_n$ *is finite. Then, with probability one, function* $\texttt{equiv}_L(x,y)$ *and function* $\texttt{compatible}_n(x,y)$ *return the same value for any* $x, y \in K(L)$ *except for finitely many values of* n.

Proof. Following (26), the loop over z in function $\texttt{compatible}_n$ checks, for the subtrees rooted at x and y, if the transition probabilities $p^T(xz, a)$ and $p^T(yz, a)$ are similar (in the sense of function `different`) and also compares $p^T(xz, \lambda)$ with $p^T(yz, \lambda)$ at every node. There are at most $t_n - 1$ arcs plus t_n nodes in a subtree, and therefore, a maximum of $2t_n$ calls to `different` in $\texttt{compatible}_n$. Let A_n be the event $\texttt{equiv}_L(x,y) \neq \texttt{compatible}_n(x,y)$ and $p(A_n)$ its probability. As `different` works with a confidence level above $(1-\alpha_n)^2$, the probability $p(A_n)$ is smaller than $4\alpha_n\tau_n$, where τ_n is the expected size of the prefix tree automaton after n examples. According to the Borel-Cantelli lemma (Feller, 1950), if $\sum_n p(A_n) < \infty$ then, with probability one, only finitely many events A_n take place. As the expected size τ_n of the prefix tree automaton cannot grow faster than linearly with n, it is sufficient that $\sum_n n\,\alpha_n < \infty$ for $\texttt{compatible}_n(x,y)$ and $\texttt{equiv}_L(x,y)$ to return the same value, except for finietly many values of n. ■

An immediate consequence from the previous proof is that the complexity of $\texttt{compatible}_n$ is bounded by n and, according to the discussion at the end of the former section, the algorithm `rlips` works in time $\mathcal{O}(n|\mathcal{A}||M|^2)$. Therefore, the algorithm is, in the limit of large sample sets, linear with the

```
algorithm different
input: n, f, n', f', α
output: boolean
begin algorithm
  if n = 0 or n' = 0 then
    return FALSE
  endif
  return |f/n − f'/n'| > ε_α(n) + ε_α(n')
end algorithm
```

Figure 5: Algorithm `different`. Function ϵ_α is defined by Eq. 31.

size of the sample, and usually dominated by input/output processes.

Recall that `rlips` only needs $\texttt{compatible}_n(x, y)$ to be correct within the finite set $K(L) \times \mathrm{Sp}(L)$. Thus, with probability one, there exists an N such that all calls to $\texttt{compatible}_n$ with $n > N$ return the correct value and, then, `rlips` outputs the correct structure of the canonical generator.

6 A lower bound on the sample size for convergence

An interesting question is the number of examples necessary in order to correctly infer a SDFA. This number depends on the detailed structure of the automaton and the statistical tests being applied. However, a lower bound for any algorithm of the class described in this paper can be found.

For every pair of strings $x_1, x_2 \in \mathrm{Sp}(L)$ such that $x_1 \neq x_2$, a minimum number of examples needed in order to find $x_1^{-1}L \neq x_2^{-1}L$ will be denoted with $\gamma(x_1, x_2)$. Following (26), there exist $z \in \mathcal{A}^*$ and $a \in \mathcal{A} \cup \{\lambda\}$, such that

$$|p^M(x_1', a) - p^M(x_2', a)| \neq 0, \tag{27}$$

being $x_1' = x_1 z$ and $x_2' = x_2 z$. One cannot expect convergence to take place before the statistical error of the experimental range becomes smaller than the above difference. An algorithm-independent estimate of the error range is given by the sum of standard deviations $\sigma_1 + \sigma_2$ with

$$\sigma_i \simeq \sqrt{\frac{p^M(x_i', a)(1 - p^M(x_i', a))}{np(x_i'\mathcal{A}^*|L)}}, \tag{28}$$

where n is the number of examples in S_n.

Therefore, comparison of $p^M(x_1 z, a)$ and $p^M(x_2 z, a)$ cannot be expected

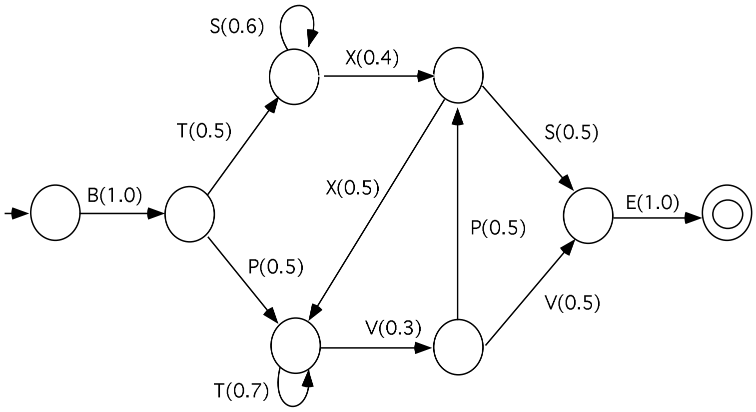

Figure 6: SFA corresponding to the Reber grammar

to be correct before $n > N(x_1, x_2, z, a)$ with

$$N(x_1, x_2, z, a) = \left(\frac{\sqrt{\frac{p^M(x_1', a)(1-p^M(x_1', a))}{p(x_1' \mathcal{A}^* | L)}} + \sqrt{\frac{p^M(x_2', a)(1-p^M(x_2', a))}{p(x_2' \mathcal{A}^* | L)}}}{p^M(x_1', a) - p^M(x_2', a)} \right)^2 \tag{29}$$

We may take now $\gamma(x_1, x_2) = \min_{(z,a)}\{N(x_1, x_2, z, a)\}$, because one string z and one symbol a are enough to find x_1 and x_2 not compatible. The most difficult comparison gives a lower bound for the difficulty of identifying the canonical generator:

$$\Gamma_1 = \max_{x,y \in \mathrm{Sp}(L)} \{\gamma(x, y) : y < x\} \tag{30}$$

A similar bound Γ_2 applies when $x \in K(L)$ and $y \in \mathrm{Sp}(L)$, but in this case it is enough to look for all $y < z$ where z is the only string in $\mathrm{Sp}(L)$ equivalent to x. As an example, the Reber grammar of Fig. 6, has a lower bound $\Gamma \simeq 330$ corresponding to $x_1 = BT$, $x_2 = BTX$ and $z = \lambda$ and compatible with the experimental results discussed in next section.

7 Results and discussion

The performance of the algorithm has been tested with a variety of grammars. For each grammar, different samples were generated with the canonical generator of the grammar and given as input for `rlips`. For instance, the Reber grammar (Reber 1967) of Fig. 6 has been used in order to compare `rlips` with previous works on neural networks which used this grammar as check (Castaño, Casacuberta & Vidal 1993).

In Fig. 7 we plot the average (after 10 experiments) number of nodes in the automaton found by `rlips` as a function of the size of the sample set generated by the Reber grammar. As seen in the figure, the number of states converges to the right value when the sample is large enough. In order to check that also the structure was correctly inferred, the relative entropy (Cover & Thomas 1991) between the hypothesis and the target grammar has been plotted in Fig. 8. For comparison, the relative entropy of the strings in the sample (or, equivalently, of the prefix tree automaton) is also plotted, and shows a much slower convergence. Indeed, if the symmeterized form is used, this latter distance becomes infinite.

As indicated by Fig. 7, when the number of examples available is small the algorithm tends to propose hypothesis which over-generalize the target language. In this range, because the estimations of the transition probabilities are not accurate, most pairs of states are taken to be equivalent and the automaton found contains fewer states than the correct one. However, when enough information is available, the algorithm always finds the correct structure of the canonical generator. The number of examples needed to achieve convergence is relatively small (about eight hundred) and consistent with the bound of previous section. This number compares rather favorably with the performance of recurrent neural networks (Castaño, Casacuberta & Vidal 1993) which cannot guarantee convergence for this grammar even after tens of thousands of examples. The algorithm behaved robustly with respect to the choice of parameter α, due to its logarithmic dependence on the parameter.

In Fig. 9, the average time needed by the algorithm is plotted as a function of the number of examples in the sample (dispersions are negligible in this picture). The linear complexity is observed and the algorithm proves very fast even for huge sample sets.

Fig. 10 shows the number of examples needed in order to identify 250 randomly generated automata. The correlation with Γ suggests that the bound $\Gamma = \max(\Gamma_1, \Gamma_2)$ obtained in previous section can be regarded as an indication of the difficulty in the identification. These experiments also showed that even some small automata can be difficult to identify, in the sense that huge samples are needed, if they contain quasi-equivalent states (states with almost identical transition probabilities) or states which are very unlikely reached from the initial state. Therefore, in order to keep the experiments with larger automata feasible, we excluded those with $\Gamma > 10^6$. With this restriction, `rlips` was able to correctly identify any randomly generated medium-size automata as the one depicted in Fig. 11, where $\Gamma \simeq 500000$ and identification was reached after 3 million examples.

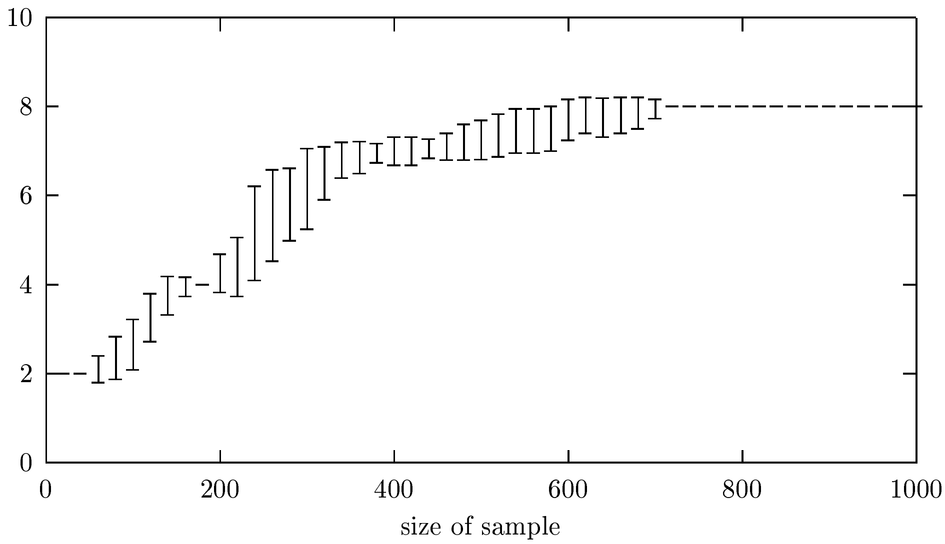

Figure 7: Number of nodes in the hypothesis for the Reber grammar as a function of the size of the sample.

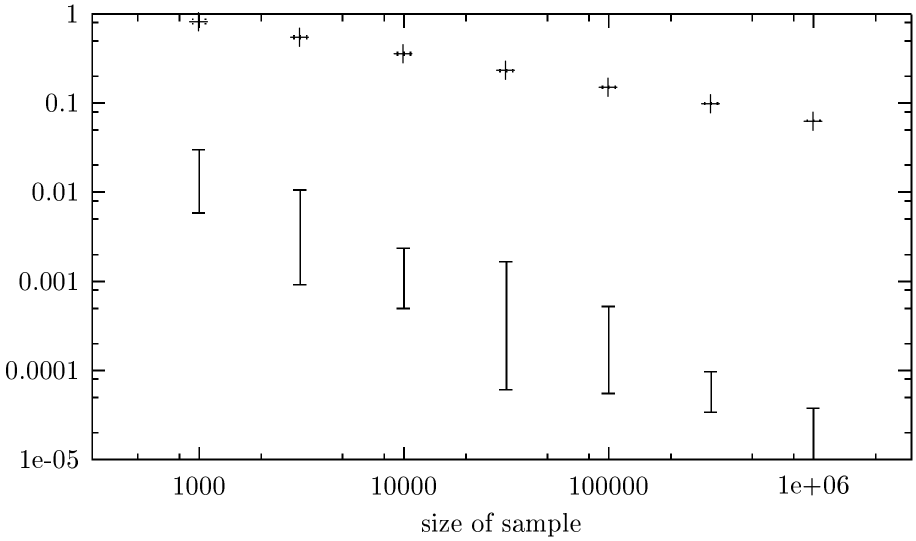

Figure 8: Lower dots: relative entropy (in bits) between the hypothesis and the target language. Upper dots: same between sample and target.

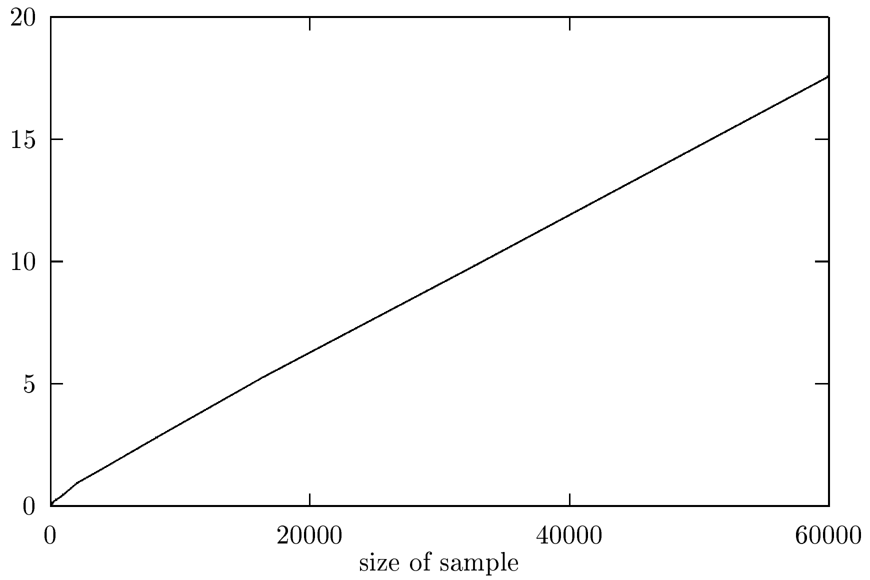

Figure 9: Time (in seconds) needed by our implementation of `rlips` running on a Hewlett-Packard 715 (40 MIPS) as a function of the size of the sample.

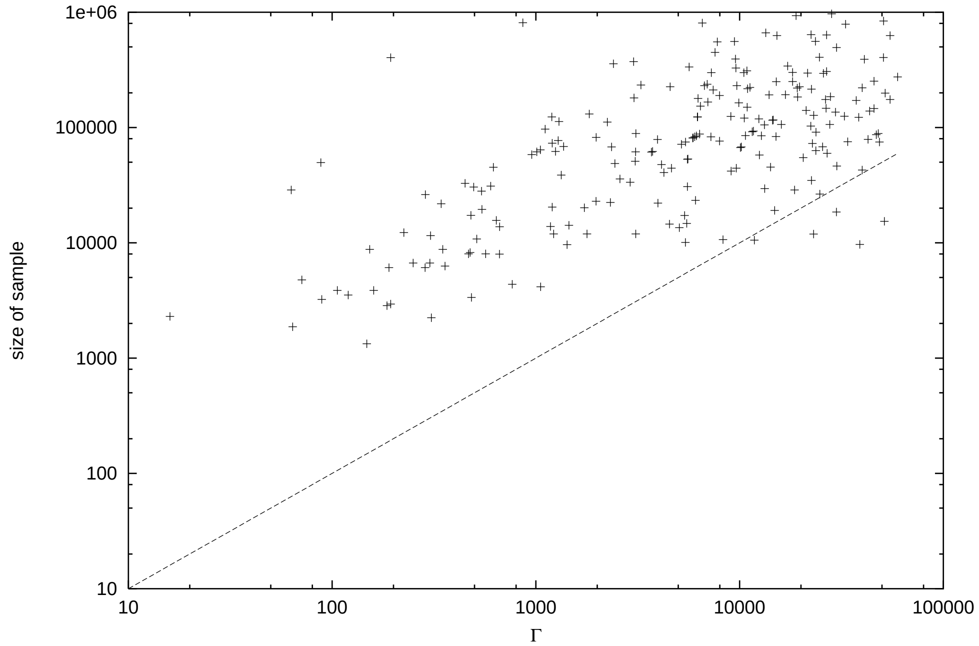

Figure 10: Sample size n needed for convergence as a function of Γ for 250 randomly generated automata. The line $n = \Gamma$ is plotted to guide the eye.

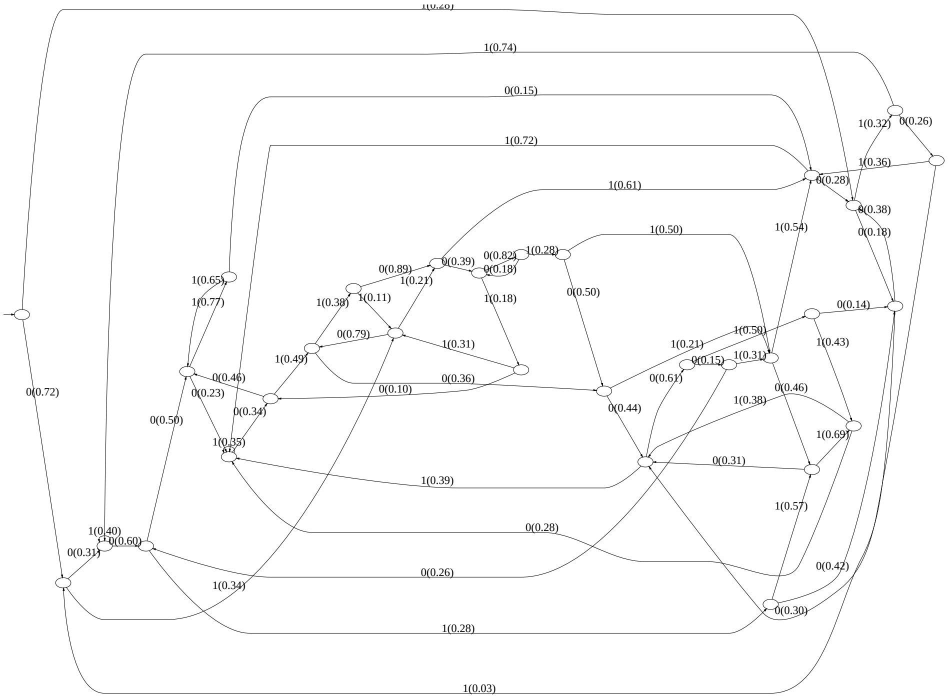

Figure 11: Medium-size automaton identified by `rlips` after 3 million examples.

8 Conclusions

An algorithm has been proposed which identifies the minimal stochastic automaton generating a deterministic regular language. Identification is achieved from stochastic samples of the strings in the language, and no negative examples are used. Experimentally, the algorithm needs very short times and comparatively small samples in order to identify the regular set. For large samples, linear time is needed (about one minute for a sample containing one million examples running on a Hewlett-Packard 715). The algorithm is suitable for recognition tasks where noisy examples or random sources are common. In this line, applications to speech recognition problems are planned.

Acknowledgments

The authors want to acknowledge useful suggestions from M.L. Forcada and E. Vidal.

Appendix

We have chosen the following bound, due to Hoeffding (1963), for the observed frequency f/m of a Bernoulli variable of probability p. Let $\alpha > 0$ and let

$$\epsilon_\alpha(m) = \sqrt{\frac{1}{2m} \log \frac{2}{\alpha}} \tag{31}$$

then, with probability greater than $1 - \alpha$,

$$\left| p - \frac{f}{m} \right| < \epsilon_\alpha(m) \tag{32}$$

Consistently, for every couple of Bernoulli variables with probabilities p and p' respectively, with probability greater than $(1-\alpha)^2$,

$$\begin{aligned} \left| \tfrac{f}{m} - \tfrac{f'}{m'} \right| &< \epsilon_\alpha(m) + \epsilon_\alpha(m') \quad \text{if} \quad p = p' \\ \left| \tfrac{f}{m} - \tfrac{f'}{m'} \right| &> \epsilon_\alpha(m) + \epsilon_\alpha(m') \quad \text{if} \quad |p - p'| > 2\epsilon_\alpha(m) + 2\epsilon_\alpha(m') \end{aligned} \tag{33}$$

and only one of the two conditionals stands for m and m' large enough, as $\epsilon_\alpha(m) \to 0$ when m grows. This is the check implemented through the logical function `different`, shown in Fig. 5. The return value will be correct for large samples with probability greater than $(1-\alpha)^2$. In our algorithm, α will depend polynomially on the size of the sample, but even in this case the implicit condition $\epsilon_{\alpha(t)}(c_n(x)) \to 0$ remains true, as the logarithm in Eq. (31) cannot compensate the growth in the denominator.

References

- Angluin, D.(1988): Identifying Languages from Stochastic Examples. Internal Report YALEU/DCS/RR–614.
- Anthony, M. and Biggs, N. (1992): *Computational Learning Theory.* Cambridge University Press, Cambridge.
- Carrasco, R.C. and Oncina, J. (1994): Learning stochastic regular grammars by means of a state merging method, in *Grammatical Inference and Applications,* (R.C. Carrasco and J. Oncina, Eds.). Lecture Notes in Artificial Intelligence **862**, Springer-Verlag, Berlin.
- Castaño, M.A., Casacuberta, F., Vidal, E. (1993): Simulation of Stochastic Regular Grammars through Simple Recurrent Networks, in *New Trends in Neural Computation* (Eds. J. Mira, J. Cabestany and A. Prieto). Springer Verlag, Lecture Notes in Computer Science **686**, 210–215.
- Cover, T.M and Thomas, J.A. (1991): *Elements of Information Theory.* John Wiley and Sons, New York.
- Feller, W. (1950): *An introduction to probability theory and its applications.* John Wiley and Sons, New York.
- Fu, K.S. (1982): *Syntactic Pattern Recognition and Applications.* Prentice Hall, Englewood Cliffs, N.J.
- Giles, C.L., Miller, C.B., Chen, D., Chen, H.H., Sun, G.Z., and Lee, Y.C. (1992): Learning and Extracting Finite State Automata with Second Order Recurrent Neural Networks. *Neural Computation* **4**, 393–405.
- Gold, E.M. (1967): Language identification in the limit. *Information and Control* **10**, 447–474.
- Hoeffding, W. (1963): Probability inequalities for sums of bounded random variables. *American Statistical Association Journal* **58**, 13–30.
- Hopcroft, J.E. and Ullman, J.D. (1979): *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading, Massachusetts (1979).
- Lang, K. (1992): Random DFA's can be Approximately Learned from Sparse Uniform Examples, in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory.*
- Maryanski, F. J. and Booth, T.L. (1997): Inference of Finite-State Probabilistic Grammars. *IEEE Transactions on Computers* **C26**, 521–536.
- Oncina, J. and García, P. (1992): Inferring Regular Languages in Polynomial Time, in *Pattern Recognition and Image Analysis* (N. Pérez de la Blanca, A. Sanfeliu and E. Vidal Eds.), World Scientific.
- Pollack, J.B. (1991): The Induction of Dynamical Recognizers. *Machine Learning* **7**, 227–252.
- Reber, A.S. (1967): Implicit Learning of Artificial Grammars. *Journal of Verbal Learning and Verbal Behaviour* **6**, 855–863.
- Ron, D., Singer, Y., Tishby, N. (1995): On the Learnability and Usage of Acyclic Probabilistic Finite Automata, in *Proceedings of the 8th Annual Conference on Computational Learning Theory (COLT'95)*, 31–40. ACM Press, New York, 1995.

- Smith, A.W. and D. Zipser, Z. (1989): Learning Sequential Structure with the Real-Time Recurrent Learning Algorithm. *International Journal of Neural Systems* **1**, 125–131.
- Stolcke A. and Omohundro,S. (1993): Hidden Markov Model Induction by Bayesian Model Merging, in *Advances in Neural Information Processing Systems 5* (C.L. Giles, S.J. Hanson and J.D. Cowan Eds.), Morgan Kaufman, Menlo Park, California.
- van der Mude, A., and Walker, A. (1978): On the Inference of Stochastic Regular Grammars. *Information and Control* **38**, 310–329.
- Watrous R.L., and Kuhn, G.M. (1992): Induction of Finite-state Languages Using Second-Order Recurrent Networks. *Neural Computation* **4**, 406–414.