An EM Approach to Grammatical Inference: Input/Output HMMs

Paolo Frasconi
Dipartimento di Sistemi
e Informatica
Università di Firenze
50139 Firenze (Italy)

Yoshua Bengio
Dept. Informatique et
Recherche Opérationnelle
Université de Montréal
Montreal, Qc H3C-3J7

Abstract

We propose a modular recurrent connectionist architecture for adaptive temporal processing. The model is given a probabilistic interpretation and is trained using the EM algorithm. This model can also be seen as an Input/Output Hidden Markov Model. *The focus of this paper is on sequence classification tasks. We demonstrate that EM supervised learning is well suited for solving grammatical inference problems. Experimental benchmark results are presented for the seven Tomita grammars, showing that these adaptive models can attain excellent generalization.*

1 Introduction

Challenging learning tasks, such as those related to language, can in principle be approached with recurrent neural networks. Recurrent networks can store and retrieve information in a flexible way. In particular, dynamical attractors can be used to implement reliable long-term memories. However, practical difficulties have been reported in training recurrent networks to perform tasks involving long-term dependencies. A mathematical analysis of the problem shows that either one of two conditions arises in such systems [1]. In the first situation, the dynamics of the network allow it to reliably store bits of information, but gradients vanish exponentially fast as one propagates them backward in time. In the second situation, gradients can flow backward but the system is locally unstable and cannot reliably store bits of information in the presence of input noise. In practice, even if the task does not exhibit very long durations in the dependencies to be captured, the solution found by backpropagation tends to reflect mostly *short-term* dependencies, thus leading to suboptimal training and generalization.

In this paper we focus on grammatical inference. In this task the learner is presented a set of labeled strings and is requested to infer a set of rules that define a formal language. It can be considered as a prototype for more complex language processing problems. However, even in the "simplest" case, i.e. regular grammars, the task can be proved to be NP-complete [2]. Many researchers [3, 4, 5] have approached grammatical inference with recurrent networks. These studies demonstrate that second-order networks can be trained to approximate the behavior of finite state automata (FSA). However, memories learned in this way appear to lack robustness and noisy dynamics become dominant for long input strings. This has motivated research to extract automata rules from the trained network [3, 5]. In many cases, it has been shown that the extracted automaton outperforms the trained network. Although FSA extraction procedures are relatively easy to devise for symbolic inputs, they may be more difficult to apply in tasks involving a subsymbolic or continuous input space, such as in speech recognition. Moreover, the complexity of the discrete state space produced by the FSA extraction procedure may grow intolerably if the continuous network has learned a representation involving chaotic attractors. Other researchers have attempted to encourage a finite-state representation via regularization [6] or by integrating clustering techniques in the training procedure [7].

Following previous work [8], in this paper we propose a training method that propagates a discrete distribution of targets rather than differential error information. We assume a discrete distribution of states and we introduce a modular recurrent architecture that associates subnetworks to discrete states. This new model can also be seen as a particular type of hidden Markov model with inputs as well as outputs, hence the name IOHMM (Input/Output HMM). With this probabilistic interpretation a training algorithm

1051-4651/94 $04.00 © 1994 IEEE

can be expressed as an estimation-maximization (EM) or generalized EM (GEM) algorithm, of Dempster, Laird, & Rubin [9]. In this way learning is decomposed into (1) temporal credit assignment and (2) learning a static mapping, from the state/input space to the next-state and output space. As in hidden Markov models (HMMs), the system propagates forward and backward a discrete distribution over the n states. HMMs however adjust their parameters using unsupervised learning, whereas we use EM in a supervised fashion. This new perspective has already been found successful in training static modular systems [10].

Although the architecture can deal with continuous input and output spaces, in this paper we investigate the modeling of symbolic finite state machines. Experimental results on a benchmark study (Tomita's grammars) demonstrate that the model can achieve very good generalization using few training examples.

2 Proposed Architecture: IOHMM

Essentially, IOHMMs (Input/Output Hidden Markov Models) are HMMs whose output and transition probabilities are not constant over time, but instead, are conditioned (i.e., function of) the current input from an input sequence. Like recurrent networks, IOHMMs can thus be used to map input sequences to output sequences, or more generally model the distribution of an output sequence variable y_1^T when given the value of an input sequence variable u_1^T.

Consider a dynamical discrete-state system based on the following state space description:

$$\begin{aligned} q_t &= f(q_{t-1}, \boldsymbol{u}_t) \\ \boldsymbol{y}_t &= g(q_t, \boldsymbol{u}_t) \end{aligned} \tag{1}$$

where $\boldsymbol{u}_t \in R^m$ is the input vector at time t, $\boldsymbol{y}_t \in R^r$ is the output vector, and $q_t \in \mathcal{Q} = \{1, 2, \ldots, n\}$ is a discrete state. If inputs and outputs are restricted to take on symbolic values, the above equations define a Mealy finite state machine. $f(\cdot)$ is referred to as *state transition function* and $g(\cdot)$ is the *output function*. In this paper, we consider a probabilistic version of these dynamics, where the current inputs and the current state distribution are used to estimate the state distribution and the output distribution for the next time step.

In order to permit restrictions on the transition graph of the system, we suppose that each state j has a set of successors $\mathcal{S}_j \subset \mathcal{Q}$.

The system defined by equations 1 can be modeled by the recurrent architecture depicted in Figure 1a. The basic modules are a set of *state networks* $\mathcal{N}_j, j = 1 \ldots n$ and a set of *output networks* $\mathcal{O}_j, j = 1 \ldots n$. Each one of the state and output network is uniquely associated to one of the states in $\mathcal{Q}$, and all networks share the same input $\boldsymbol{u}_t$. Each state network $\mathcal{N}_j$ has the task of predicting the next state representation, based on the current input and given that $q_{t-1} = j$. Similarly, each output network $\mathcal{O}_j$ predicts the output of the system, given the current state and input.

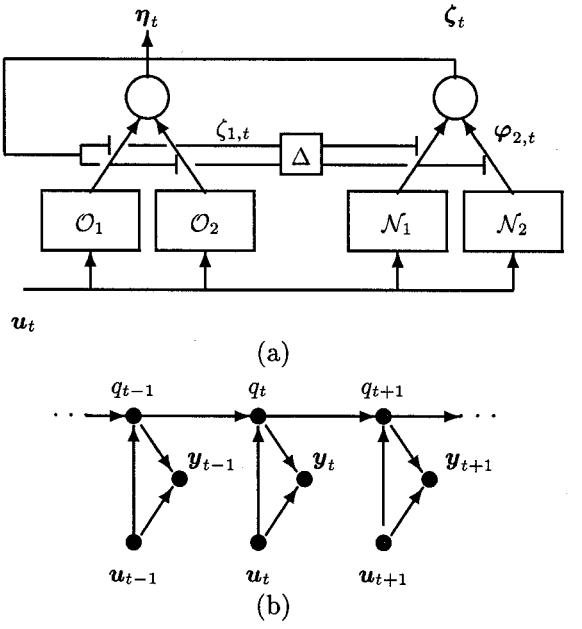

Figure 1. (a): The proposed recurrent architecture, or IOHMM, for a number of states $n = 2$. (b): Bayesian network describing the assumed conditional dependencies.

We suppose that state networks have a feedforward multilayered architecture. Each output $\varphi_{ij,t}$ of $\mathcal{N}_j$ is associated to one of the successors i of state j. Thus the last layer of $\mathcal{N}_j$ has as many units as the cardinality of $\mathcal{S}_j$. A softmax output function is used in the last layer, so that $\sum_i \varphi_{ij,t} = 1 \ \forall j, t$. $\boldsymbol{\zeta}_t \in R^n$ represents the internal state of the model and it is computed as a linear combination of the outputs of the state networks, gated by the previously computed internal state:

$$\zeta_{it} = \sum_{j=1}^{n} \zeta_{j,t-1} \varphi_{ij,t}. \tag{2}$$

Output networks are also feedforward and compete to compute the predicted output of the system $\boldsymbol{\eta}_t \in R^r$:

$$\boldsymbol{\eta}_t = \sum_{j=1}^{n} \zeta_{jt} \boldsymbol{\eta}_{jt} \tag{3}$$

where $\boldsymbol{\eta}_{jt} \in R^r$ is the output of network $\mathcal{O}_j$.

The above architecture can be easily interpreted as a probability model. Define the vector of indicator variables $\boldsymbol{z}_t$ as:

$$z_{it} = \begin{array}{ll} 1 & \text{if } q_t = i, \\ 0 & \text{otherwise.} \end{array} \tag{4}$$

Assuming a multinomial distribution for states we have

$$\mathrm{P}(\boldsymbol{z}_t) = \prod_{i=1}^{n} \zeta_{it}^{z_{it}} \tag{5}$$

so that $\zeta_{it} = \mathrm{P}(q_t = i \mid \boldsymbol{u}_1^t)$, having denoted with $\boldsymbol{u}_1^t$ the subsequence of inputs from time 1 to t, inclusively. $\boldsymbol{\eta}_0$ is chosen to be a vector of initial state probabilities. From equation (2) one recognizes that $\varphi_{ij,t} = \mathrm{P}(q_t = i \mid q_{t-1} = j, \boldsymbol{u}_t)$.

As in neural networks trained to minimize the output squared error (MSE), the output of this architecture can be interpreted as an expected "position parameter" for the probability distribution of the output sequence $\boldsymbol{y}_1^T$. However, in addition to being conditional on an input $\boldsymbol{u}_t$, this expectation is also conditional on the state q_t:

$$\boldsymbol{\eta}_t = E[\boldsymbol{y}_t \mid q_t, \boldsymbol{u}_t]. \tag{6}$$

The actual form of the output density, denoted $f_Y(\boldsymbol{y}_t; \boldsymbol{\eta}_t)$, will be chosen according to the task. For example a multinomial distribution is suitable for sequence classification, while a gaussian distribution is adequate for producing continuous outputs.

The random variables involved in the probabilistic interpretation of the proposed architecture have a joint probability $\mathrm{P}(\boldsymbol{u}_1^T, \boldsymbol{x}_1^T, \boldsymbol{y}_1^T)$. Without conditional independency assumptions, the amount of computation necessary to estimate the probabilistic relationships among these variables can quickly get intractable. Thus we introduce an independency model over this set of random variables. Rather than listing a set of conditional independency assumptions, we prefer to graphically express dependencies through a *Bayesian network* [11]. In particular, we assume that the directed acyclic graph $\mathcal{G}$ depicted in Figure 1b is a Bayesian network for the independency model associated to the variables $\boldsymbol{u}_1^T, \boldsymbol{x}_1^T, \boldsymbol{y}_1^T$. One of the most evident consequences of this independency model is that only the previous state and the current input are relevant to determine the next-state. This one-step memory property is analogue to the Markov assumption in hidden Markov models.

It is however important to remark some basic differences between IOHMMs and ordinary HMMs. In the latter models, states form a stationary (or homogeneous) Markov chain and data are modeled as if they were generated as a probabilistic output function of states or state transitions. In IOHMMs state transitions are conditional on the external input and thus states form a non-homogeneous Markov chain. This property allows to model input/output temporal relationships, resulting in an information processing style akin to that of recurrent networks. Consistently with this framework, we use the output portion $\boldsymbol{y}_1^T$ of the data as *targets* for the outputs of the model.

3 A Supervised Learning Algorithm

The learning algorithm for the proposed architecture is derived from the maximum likelihood principle. The training data are a set of P pairs of input/output sequences: $\mathcal{D} \stackrel{def}{=} \{(\boldsymbol{u}_1^{T_p}{}_{[p]}, \boldsymbol{y}_1^{T_p}{}_{[p]}); p = 1 \ldots P\}$. Let $\boldsymbol{\Theta}$ denote the vector of parameters obtained by collecting all the parameters of the architecture. The likelihood function is then given by [1]

$$L(\mathcal{D}; \boldsymbol{\Theta}) \stackrel{def}{=} \prod_{p=1}^{P} \mathrm{P}(\boldsymbol{y}_1^{T_p}{}_{[p]} \mid \boldsymbol{u}_1^{T_p}{}_{[p]}; \boldsymbol{\Theta}). \tag{7}$$

The output values (targets) may be specified intermittently. In particular, for sequence classification tasks one is only interested in the output $\boldsymbol{y}_T$, at the end of each string. Maximization of (7) will be treated as parameter estimation with missing data, where the missing variables are the state indicator variables $\boldsymbol{z}_t$ (describing the path in state space). The *complete data* will be denoted $\mathcal{D}_c \stackrel{def}{=} \{(\boldsymbol{u}_1^{T_p}{}_{[p]}, \boldsymbol{y}_1^{T_p}{}_{[p]} \boldsymbol{z}_1^{T_p}{}_{[p]}); p = 1 \ldots P\}$ and the corresponding complete-data likelihood is

$$L_c(\boldsymbol{\Theta}; \mathcal{D}_c) = \prod_{p} \mathrm{P}(\boldsymbol{y}_1^T, \boldsymbol{z}_1^T \mid \boldsymbol{u}_1^T; \boldsymbol{\Theta}). \tag{8}$$

Using the assumed conditional independency model and taking the logarithm, we can decompose the complete likelihood as:

$$\begin{aligned} l_c(\boldsymbol{\Theta}; \mathcal{D}_c) &= \log L_c(\boldsymbol{\Theta}; \mathcal{D}_c) = \\ &\sum_{p} \sum_{t=1}^{T} \sum_{i=1}^{N} z_{it} \log \mathrm{P}(\boldsymbol{y}_t \mid q_t = i, \boldsymbol{u}_t; \boldsymbol{\Theta}) + \\ &\sum_{j=1}^{N} z_{it} z_{j,t-1} \log \mathrm{P}(q_t = i \mid q_{t-1} = j, \boldsymbol{u}_t; \boldsymbol{\Theta}). \end{aligned} \tag{9}$$

[1] In the following, in order to simplify the notation, the sequence subscript $[p]$ may be omitted

Since $l_c(\Theta; \mathcal{D}_c)$ depends on the unknown state indicator variables, it cannot be maximized directly. Finding the correct values of these variables would solve the temporal credit assignment problem, since it would then suffice to extract from training data the static next-state and output mappings. In this situation the EM algorithms [9] can be used to decouple the temporal and static learning problems. The MLE optimization is solved by introducing the auxiliary function $Q(\Theta; \Theta^{(k)})$ and iterating the following two steps for $k = 1, 2, \ldots$:

Estimation: $Q(\Theta; \Theta^{(k)}) = E[l_c(\Theta; \mathcal{D}_c) \mid \mathcal{D}, \Theta^{(k)}]$
Maximization: $\Theta^{(k+1)} = \arg\max_{\Theta} Q(\Theta; \Theta^{(k)})$

3.1 The Expectation Step

The expectation of $l_c(\Theta; \mathcal{D}_c)$, conditional on $\mathcal{D}$ and the "old" parameters Θ' can be readily computed as:

$$\mathcal{Q}(\Theta; \Theta') = \sum_{p} \sum_{t=1}^{T} \sum_{i=1}^{N} \zeta'_{it} \log \mathrm{P}(\boldsymbol{y}_t \mid q_t = i, \boldsymbol{u}_t; \Theta) + \sum_{j=1}^{N} h'_{ij,t} \log \mathrm{P}(q_t = i \mid q_{t-1} = j, \boldsymbol{u}_t; \Theta) \quad (10)$$

where

$$h_{ij_t} \stackrel{def}{=} E[z_{it} z_{j,t-1} \mid \boldsymbol{u}_1^T, \boldsymbol{y}_1^T; \Theta]$$

In order to compute h_{ij_t} we introduce the following probabilities, borrowing from HMM literature:

$$\alpha_{it} \stackrel{def}{=} \mathrm{P}(\boldsymbol{y}_1^t \mid q_t = i; \boldsymbol{u}_1^t); \quad (11)$$

$$\beta_{it} \stackrel{def}{=} \mathrm{P}(\boldsymbol{y}_t^T \mid q_t = i, \boldsymbol{u}_t^T). \quad (12)$$

Using the assumed conditional independencies, it follows that

$$\beta_{it} = f_Y(\boldsymbol{y}_t; \boldsymbol{\eta}_{it}) \sum_{\ell} \varphi_{\ell i}(\boldsymbol{u}_{t+1}) \beta_{\ell,t+1} \quad (13)$$

and

$$\alpha_{it} = \frac{f_Y(\boldsymbol{y}_t; \boldsymbol{\eta}_{it})}{\zeta_{it}} \sum_{\ell} \varphi_{i\ell}(\boldsymbol{u}_t) \alpha_{\ell,t-1} \quad (14)$$

and after some further manipulations we obtain the following expression for $h_{ij,t}$:

$$h_{ij,t} = \frac{\beta_{it} \alpha_{j,t-1} \varphi_{ij}(\boldsymbol{u}_t) \zeta_{j,t-1}}{\sum_i \alpha_{iT} \zeta_{iT}}. \quad (15)$$

3.2 The Maximization Step

The EM algorithm requires, at each step, to find the maximum of $Q(\Theta; \Theta^{(k)})$. In general, if the networks have nonlinearities (e.g. hidden units with sigmoidal functions, or a softmax at their output to constrain the outputs to sum to 1) this may be quite difficult.

Table 1. Definitions of the seven Tomita grammars

Grammar	Definition
1	1^*
2	$(10)^*$
3	$1^{2n+1}0^{2m+1}$ is not a substring
4	000 is not a substring
5	# of 01's and 10's is even
6	# of 0's - # of 1's is a multiple of 3
7	$0^*1^*0^*1^*$

In these cases it is possible to use a generalized EM (GEM) algorithm, which simply increases Q, for example with gradient ascent. Although GEM is also guaranteed to converge to a maximum of the likelihood, its convergence may be significantly slower. A true EM algorithm can be used if the inputs are quantized and the subnetworks perform a simple look-up in a table of symbol probabilities. In this case we can show that $\frac{\partial Q(\theta, \theta^k)}{\partial \theta} = 0$ can be solved analytically [12]. Experiments have shown that learning with GEM can be somewhat sped up by using a stochastic update, i.e. updating the parameters at the end of each sequence rather than accumulating gradients over the whole training set.

4 Inference of Regular Grammars

In this section we report experimental results on a set of regular grammars introduced by Tomita [13] and afterwards used by other researchers to measure the accuracy of inference methods based on recurrent networks [7, 3, 6, 4, 5]. The grammars use the binary alphabet $\{0, 1\}$ and are reported in Table 1. For each grammar, Tomita also defined a small set of labeled strings to be used as training data. One of the difficulties of the task is to infer the proper rules (i.e., to attain perfect generalization) using these impoverished data.

Since the task is to classify each sequence in two classes (namely, accepted or rejected strings), we used a scalar output and we put supervision at the last time step T. The final output y_T was modeled as a Bernoulli variable, i.e. $f_Y(y_T; \eta_T) = \eta_T^{y_T}(1 - \eta_T)^{1-y_T}$, with $y_T = 0$ if the string is rejected and $y_T = 1$ if it is accepted. During test we adopted the criterion of accepting the string if $\eta_T > 0.5$. It is worth mentioning that final states cannot be used directly as targets —as done in [8]— since there can be more than one accepting or rejecting state. In [3] this problem is circumvented by appending a special "end" symbol to each string.

Table 2. Summary of experimental results on the seven Tomita's grammars (see text for explanation).

Grammar	Sizes		Convergence	Accuracies			
	$n^\star$	FSA min		Average	Worst	Best	W&K Best
1	2	2	.600	1.000	1.000	1.000	1.000
2	8	3	.800	.965	.834	1.000	1.000
3	7	5	.150	.867	.775	1.000	.783
4	4	4	.100	1.000	1.000	1.000	.609
5	4	4	.100	1.000	1.000	1.000	.668
6	3	3	.350	1.000	1.000	1.000	.462
7	3	5	.450	.856	.815	1.000	.557

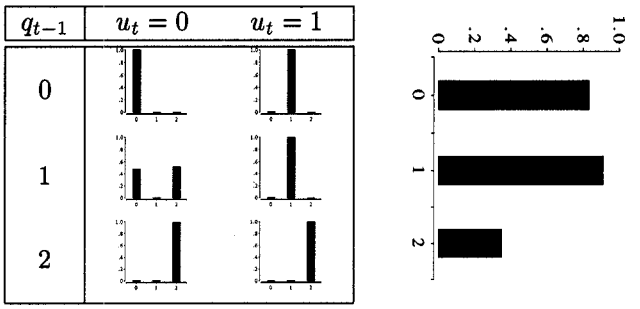

Figure 2. Learned transition probabilities for grammar # 7: Left: Transition probabilities; bar chart on row j and column k represents the discrete distribution $\mathrm{P}(q_t \mid q_{t-1} = j, u_t = k)$. Right: Probabilities of accepting the input string, $P(y_T = 1 \mid q_T)$. This network correctly classifies all the test strings.

In this application we did not apply external inputs to the output networks. This corresponds to modeling a Moore finite state machine. Thus each of such networks reduces to one unit fed by a bias input and the final prediction is simply a function of the last state reached by the model. Given the absence of prior knowledge about plausible state paths, we used an *ergodic* transition graph in which each state transition is allowed (i.e., $\mathcal{S}_j = \mathcal{Q}$ for all j). Each state network was composed of a single layer of n neurons with a softmax function at their outputs. Input symbols were encoded by two-dimensional index vectors (i.e., $\boldsymbol{u}_t = [1, 0]'$ for the symbol 0 and $\boldsymbol{u}_t = [0, 1]'$ for the symbol 1). The total number of free parameters is thus $2n^2 + n$.

In the experiments we measured convergence and generalization performance using different number of states for the IOHMM. For each setting we ran 20 trials with different seeds for the initial weights. We considered a trial successful if the trained network was able to correctly label all the training strings. In order to select the model size (i.e., the number of states n) we generated a small data set composed of 20 randomly selected strings of length $T \leq 12$, and we applied a cross-validation criterion. For each grammar we trained seven different architectures having $n = 2, 3 \ldots, 8$ and we selected the value $n^\star$ that yielded the best average accuracy on the cross-validation data set. For comparison, in Table 2 we also report for each grammar the number of states of the minimal recognizing FSA [13].

We tested the trained networks on a corpus of $2^{13} - 1$ binary strings of length $T \leq 12$. The final results are summarized in Table 2. The column "Convergence" reports the fraction of trials that succeeded to separate the training set. The next three columns report averages and order statistics (worst and best trial) of the fraction of correctly classified strings, measured on the successful trials. For each grammar these results refer to the model size $n^\star$ selected by cross-validation. Generalization was always perfect on grammars 1,4,5 and 6. For each grammar, the best trial also attained perfect generalization. These results compare very favorably to those obtained with second-order networks trained by gradient descent, when using the learning sets proposed by Tomita. For comparison, in the last column of Table 2 we reproduce the results reported by Watrous & Kuhn [5] in the best of five trials. Other researchers also obtained interesting results, although they are not directly comparable because of the use of larger training sets [7, 3] or different experimental conditions [6].

In most of the successful trials we observed that the model learned a "deterministic" behavior, i.e. the

transition probabilities were asymptotically converging either to 0 or to 1 (exact values 0 or 1 would require to develop infinite weights because of the softmax function). Thanks to this property we can trivially extract the recognizing FSA. Indeed, for grammars 1,4,5, and 6, we found that the trained networks behave exactly like the minimal recognizing FSA. In some cases, however, the network learned a different representation. In particular, for grammar 7 we found a model with three states that correctly classify all the test strings. This is interesting because the minimal FSA for grammar 7 has five states. We report the learned transition probabilities in Figure 2a, and the output probabilities in Figure 2b. One might wonder if such a representation is robust for longer input strings. To investigate this issue we generated 1000 random strings of length $T = 500$ and we found that the network still made no errors.

A potential training problem is the presence of local maxima in the likelihood function. For example, the number of converged trials for grammars 3, 4, and 5 is quite small and the difficulty of discovering the optimal solution might become a serious restriction for tasks involving a large number of states. In other experiments [8] we noticed that restricting the connectivity of the transition graph can significantly help to remove problems of convergence. Of course, this approach can be effectively exploited only if some prior knowledge about the state space is available. For example, applications of HMMs to speech recognition always rely on structured topologies.

5 Conclusions

We have presented a recurrent architecture for modeling discrete state dynamical systems. The architecture has a probabilistic interpretation, called Input/Output HMM, and is trained by an EM or GEM algorithm, using the state paths as missing data. Experiments suggest that the method is appropriate for solving grammatical inference problems, comparing favorably to second order nets trained by gradient descent in terms of generalization performance.

References

[1] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[2] D. Angluin and C. Smith, "Inductive inference: Theory and methods," *Computing Surveys*, vol. 15, no. 3, pp. 237–269, 1983.

[3] 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*, vol. 4, no. 3, pp. 393–405, 1992.

[4] J. B. Pollack, "The induction of dynamical recognizers," *Machine Learning*, vol. 7, no. 2, pp. 196–227, 1991.

[5] R. L. Watrous and G. M. Kuhn, "Induction of finite-state languages using second-order recurrent networks," *Neural Computation*, vol. 4, no. 3, pp. 406–414, 1992.

[6] M. Gori, M. Maggini, and G. Soda, "Insertion of finite state automata into recurrent radial basis function networks," Tech. Rep. DSI-17/93, Università di Firenze (Italy), 1993. (submitted).

[7] S. Das and M. C. Mozer, "A unified gradient-descent/clustering architecture for finite state machine induction," in *Advances in Neural Information Processing Systems 6* (J. Cowan, et al., eds.), Morgan Kaufmann, 1994.

[8] Y. Bengio and P. Frasconi, "Credit assignment through time: Alternatives to backpropagation," in *Advances in Neural Information Proc. Systems 6* (J. Cowan et al. eds.), Morgan Kaufmann, 1994.

[9] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum-likelihood from incomplete data via the EM algorithm," *Journal of Royal Statistical Society B*, vol. 39, pp. 1–38, 1977.

[10] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Computation*, vol. 6, pp. 181–214, 1994.

[11] J. Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[12] Y. Bengio and P. Frasconi, "An EM approach to learning sequential behavior," Tech. Rep. RT-DSI-11-94, Università di Firenze, May 1994.

[13] M. Tomita, "Dynamic construction of finite-state automata from examples using hill-climbing," in *Proc. of the Fourth Annual Cognitive Science Conference*, (Ann Arbor, MI), pp. 105–108, 1982.