Input/Output HMMs for Sequence Processing

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

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

September 4, 1995

Abstract

We consider problems of sequence processing and propose a solution based on a discrete state model in order to represent past context. We introduce a recurrent connectionist architecture having a modular structure that associates a subnetwork to each state. The model has a statistical interpretation we call *Input/Output Hidden Markov Model* (IOHMM). It can be trained by the EM or GEM algorithms, considering state trajectories as missing data, which decouples temporal credit assignment and actual parameter estimation.

The model presents similarities to hidden Markov models (HMMs), but allows us to map input sequences to output sequences, using the same processing style as recurrent neural networks. IOHMMs are trained using a *more discriminant learning* paradigm than HMMs, while potentially taking advantage of the EM algorithm.

We demonstrate that IOHMMs are well suited for solving grammatical inference problems on a benchmark problem. Experimental results are presented for the seven Tomita grammars, showing that these adaptive models can attain excellent generalization.

*also, AT&T Bell Laboratories, Holmdel, NJ

1 Introduction

For many learning problems, the data of interest have a significant sequential structure. Problems of this kind arise in a variety of applications, ranging from written or spoken language processing, to the production of actuator signals in control tasks, to multivariate time-series prediction. Feedforward neural networks are inadequate in many of these cases because of the absence of a memory mechanism that can retain past information in a flexible way. Even if these models include delays in their connections [1], the duration of the temporal contingencies that can be captured is fixed a priori by the architecture rather than being inferred from data. Furthermore, for some tasks, the appropriate size of the input window (or delays) varies during the sequence or from sequence to sequence. Recurrent neural networks, on the other hand, allow one to model arbitrary dynamical systems [2, 3] and can store and retrieve contextual information in a very flexible way, i.e., for durations that are not fixed a priori and that can vary from one sequence to another. In sequence analysis systems that can take context into account in a flexible manner (such as recurrent neural networks and HMMs), one finds some form of "state variable" or representation of past context. With a state-space representation the main computations can be divided into (1) updating the state or context variable (the state transition function), and (2) computing or predicting an output, given the current state (the output function).

Up to now, research efforts on supervised learning for recurrent networks have been almost exclusively focused on gradient descent methods and a continuous state-space. Numerous algorithms are available for computing the gradient. For example, the back-propagation through time (BPTT) algorithm [4, 5] is a straightforward generalization of back-propagation that allows one to compute the complete gradient in fully recurrent networks. The real time recurrent learning (RTRL) algorithm [6, 7, 8] is local in time and produces a partial gradient after each time step, thus allowing on-line weights updating. Another algorithm was proposed for training local feedback recurrent networks [9, 10]. It is also local in time, but requires computation only proportional to the number of weights, like back-propagation through time. Local feedback recurrent networks are suitable for implementing short-term memories but they have limited representational power for dealing with general sequences [11, 12].

However, practical difficulties have been reported in training recurrent neural networks to perform tasks

in which the temporal contingencies present in the input/output sequences span long intervals [13, 14, 15]. In fact, it can be proved that any parametric dynamical system with a non-linear recurrence (such as a recurrent neural network) will be increasingly difficult to train with gradient descent as the duration of the dependencies to be captured increases [13]. This is a problem with the gradient of the error function and thus it persists regardless of what gradient computation algorithm (such as RTRL or BPTT) is employed. A common heuristic solution is to start training on shorter sequences, and then incrementally train on longer sequences. In general, however, the rules needed to deal with long term dependencies might not be present in short sequences.

Previous work on alternative training algorithms [16, 13] suggests that the root of the problem lies in the essentially *discrete* nature of the process of storing contextual information for an indefinite amount of time. A potential solution to this problem is to propagate, backward in time, targets in state space, rather than differential error information. In order to gain some intuition about target propagation, suppose that an oracle is available that provides targets for each internal state variable, and for each time step. In this case learning would be reduced to a static learning problem, namely the problem of learning the next-state and the output mappings that define the behavior of the dynamical system using a state space representation. Of course, such an oracle is not available in general. It essentially supposes prior knowledge of an appropriate state representation. However, we can conceive an iterative approach based on two repeated steps: a first step approximates the oracle providing pseudo-targets, and a second step fits the parameters to the pseudo-target state trajectory and the output targets. In the absence of prior knowledge, the pseudo-target state trajectories can be randomly initialized. If each iteration is guaranteed to produce some improvements in the approximation of the "true" targets, then the process may converge to some useful solution with regard to the output targets specified by supervision. One of the first related approaches is probably the *moving target* algorithm by Rohwer [17]. The moving target approach consists in formulating supervised learning as an optimization problem in the joint space of temporal targets and adjustable parameters (connection weights). Rohwer proposed a solution based on gradient descent and demonstrated experimentally that some difficult credit assignment tasks could be solved. However, for more difficult problems, the method got stuck very often in local minima and no useful solution could be

obtained.

Extending previous work [18], in this paper we propose a statistical approach to target propagation, based on the EM algorithm. We consider a parametric dynamical system having n discrete states and we introduce a modular architecture, with subnetworks associated to discrete states. The architecture can be interpreted as a statistical model and can be trained by the EM or generalized EM (GEM) algorithms of Dempster, Laird, & Rubin [19], by considering the internal state trajectories as missing data. In this way learning is factored into a temporal credit assignment subproblem and a static learning subproblem that consists in fitting parameters to the next-state and output mappings defined by the estimated trajectories. In order to iteratively tune parameters with the EM or GEM algorithms, the system propagates forward and backward a discrete distribution over the n states, resulting in a procedure similar to the Baum-Welsh algorithm used to train standard hidden Markov models (HMMs) [20, 21, 22].

The main difference between standard HMMs and the model presented here, is that the former represent the distribution $P(\boldsymbol{y}_1^T)$ of output sequences $\boldsymbol{y}_1^T = \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T$, whereas the latter represents the *conditional* distribution $P(\boldsymbol{y}_1^T \mid \boldsymbol{u}_1^T)$ of output sequences given input sequences $\boldsymbol{u}_1^T = \boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_T$. The model presented here is therefore called *Input/Output HMM*, or IOHMM. IOHMMs are trained by maximizing the conditional likelihood $P(\boldsymbol{y}_1^T \mid \boldsymbol{u}_1^T)$. This is a supervised learning problem since the output sequence $\boldsymbol{y}_1^T$ plays the role of a desired output in response to the input sequence $\boldsymbol{u}_1^T$. If the output represents classification decisions to be made when the input sequence is given, this approach is more discriminant than standard HMMs (trained to maximize the likelihood of the observations). For example, in applications of HMMs to isolated word recognition (a special case of sequence classification), a separate model is constructed for each word (class) and trained on instances of that class only. This type of training is said to be not discriminant because each model (in our example, each word model) is trained independently: we try to model the type of observations (here, acoustic) representative of that class (here, word). Instead, discriminant training strategies do not attempt to build the best model of observations for each class, but rather focus on on the differences between the type of observations for each class, in order to better predict whether a given observation belongs to one class or another. Thus, models trained by discriminant approaches can be expressed with less degrees of freedom, since they concentrate the use of parameters

on the decision surface between the classes, rather than on the distribution of data everywhere. Another advantage of more discriminant training criteria is that they tend to be more robust to incorrectness of the model, and for this reason sometimes perform better [24, 25].

Both the input and output sequences can be multivariate, discrete or continuous. Thus IOHMMs can perform sequence regression ($\boldsymbol{y}$ continuous) or classification ($\boldsymbol{y}$ discrete). For example, in a task such as phoneme recognition, $\boldsymbol{u}_1^T$ may be a sequence of acoustic vectors (such as cepstral parameters) and $\boldsymbol{y}_1^T$ may consist of a discrete sequence of phonetic labels. In sequence classification tasks (such as isolated word recognition), the output can be the label $\boldsymbol{y}_T$ of a class, defined only at the end of each sequence. Other potential fields of application are robot navigation, system identification, and time-series prediction. For example, for economic time-series, the input sequences could be different economic time-series, the output sequence could be a prediction for the future values of some of these variables, and the hidden states could represent different regimes of the economy (e.g., business cycles). For applications such as handwriting or speech recognition, the output sequence (e.g., a sequence of characters or phonemes) does not have to be synchronized with the input sequence (e.g., pen trajectory or acoustic sequence).

Like in HMMs, using Markov assumptions, the distribution of outputs given the inputs can be factored into sums of products of two types of factors, output probabilities and transition probabilities:

1. $P(\boldsymbol{y}_t \mid x_t, \boldsymbol{u}_t)$ is the output distribution given the state x_t and input $\boldsymbol{u}_t$ at time t. This specifies the output function of the dynamical system.

2. $P(x_t \mid x_{t-1}, \boldsymbol{u}_t)$ is the matrix of transition probabilities at time t, conditioned on current input $\boldsymbol{u}_t$. This specifies the state transition function of the dynamical system.

Therefore, a simple way to obtain an IOHMM from an HMM is to make the output and transition probabilities function of an input $\boldsymbol{u}_t$ at each time step t. The output distribution $P(\boldsymbol{y}_t \mid \boldsymbol{u}_t)$ is obtained as a mixture of probabilities [26], in which each component is conditional on a particular discrete state, and the mixing proportions are the current state probabilities, conditional on the input. Hence, the model has also interesting connections to the mixture of experts (ME) architecture by Jacobs, Jordan, Nowlan & Hinton [27]. Like in the mixture of experts, sequence regression is carried out by associating different

modules to different states and letting each module fit the data (e.g., compute the expected value of the output given the state and input, $E[\boldsymbol{y}_t \mid x_t, \boldsymbol{u}_t]$) during the interval of time when it receives credit. As in the mixture of experts, the task decomposition is smooth. Unlike the related approach of [28], the gating (or switching) between experts is provided by expert modules (one per state i) computing the state transition distribution $P(x_t \mid x_{t-1=}i, \boldsymbol{u}_t)$ (conditioned on the current input).

Another connectionist model extending hidden Markov models to process discrete input and output streams was proposed in [23], for modeling the distribution of an output sequence y given an input sequence u.

Other interesting related models are the various hybrids of neural networks and HMMs that have been proposed in the literature (such as [29, 30, 31, 32]). As in IOHMMs with neural networks for modeling the transition and output distributions, for all of these models, the strictly neural part of the model is feedforward (or has a short horizon), whereas the HMM is used to represent the longer-term temporal structure (and in the case of speech, the prior knowledge about this structure).

Experiments on artificial tasks [18] have shown that a simplified version of the approach presented here can deal with long-term dependencies more effectively than recurrent networks trained with back-propagation through time or other alternative algorithms. The model used in [18] has very limited representational capabilities and can only map an input sequence to a final discrete state. In the present paper we describe an extended architecture that allows one to fully specify both the input and output portions of data, as required by the supervised learning paradigm. In this way, general sequence processing tasks can be addressed, such as production, classification, or prediction.

The paper is organized as follows. Section 2 is devoted to a circuit description of the architecture and its statistical interpretation. In section 3 we derive the equations for training IOHMMs. In particular, we present an EM version of the learning algorithm that can be used for discrete inputs and linear subnetworks, and a GEM version for general multilayered subnetworks. In section 4 we compare IOHMMs to other related models for sequence processing, such as standard HMMs and other recurrent Mixture of Experts architectures. In section 5 we analyze from a theoretical point of view the learning capabilities of the model in the presence of long-term dependencies, arguing that improvements can be achieved by adding

an extra term to the likelihood function and/or by constraining the state transitions of the model. Finally, in section 6 we report experimental results for a classical benchmark study in grammatical inference (Tomita's grammars). The results demonstrate that the model can achieve very good generalization using few training examples.

2 Input/Output Hidden Markov Models

2.1 The Proposed Architecture

Whereas recurrent networks usually have a continuous state space, in IOHMMs we will consider a probability distribution over a *discrete state* dynamical system, based on the following state space description:

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

where $\boldsymbol{u}_t \in \boldsymbol{R}^m$ is the input vector at time t, $\boldsymbol{y}_t \in \boldsymbol{R}^r$ is the output vector, and $x_t \in \mathcal{V} = \{1, 2, \ldots, n\}$ is a discrete state. These equations define a generalized Mealy finite state machine, in which inputs and outputs may take on continuous values. $f(\cdot)$ is referred to as the *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 output distribution and the state distribution for the next time step.

Admissible state transitions will be specified by a transition graph $\mathcal{G} = \{\mathcal{V}, \vec{\mathcal{E}}\}$, whose vertices correspond to the model's states. $\mathcal{G}$ describes the topology of the underlying Markov model. A transition from state j to state i is admissible if and only if there exists an edge $e_{ij} \in \vec{\mathcal{E}}$. We define the set of successors for each state j as: $\mathcal{S}_j \stackrel{\text{def}}{=} \{i \in \mathcal{V} : \exists e_{ij} \in \vec{\mathcal{E}}\}$.

The proposed system is illustrated in Figure 1. The architecture is composed by 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 networks is uniquely associated to one of the states in $\mathcal{V}$, and all networks share the same input $\boldsymbol{u}_t$. Each state network $\mathcal{N}_j$ has the task of predicting the next state distribution $P(x_t \mid x_{t-1=}i, \boldsymbol{u}_t)$, based on the current input and given that the previous state $x_{t-1} = j$. Similarly, each output network $\mathcal{O}_j$ computes some parameters of

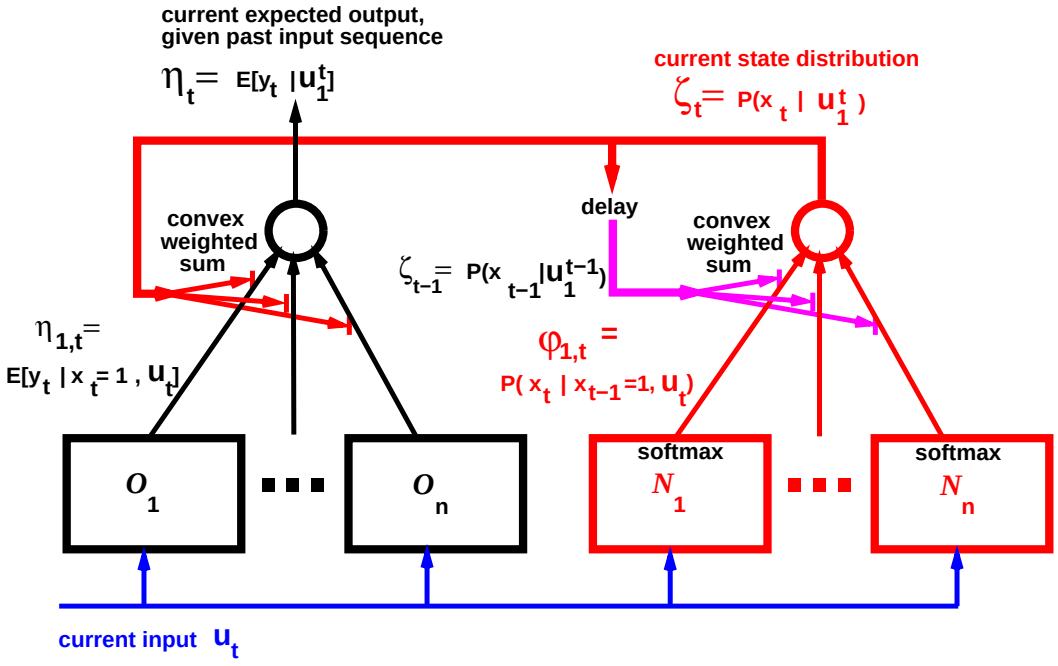


Figure 1: The proposed architecture: IOHMM and recurrent mixture of experts.

the distribution for the current output of the system, $P(\boldsymbol{y}_t \mid x_{t=}i, \boldsymbol{u}_t)$, given the current state and input. Typically, the output networks compute the expected output value, $\boldsymbol{\eta}_{i,t} = E[\boldsymbol{y}_t \mid x_t = i, \boldsymbol{u}_t]$.

All the subnetworks are assumed to be static, i.e., they are defined by means of algebraic mappings $N_j(\boldsymbol{u}_t; \boldsymbol{\theta}_j)$ and $O_j(\boldsymbol{u}_t; \boldsymbol{\vartheta}_j)$, where $\boldsymbol{\theta}_j$ and $\boldsymbol{\vartheta}_j$ are vectors of adjustable parameters (e.g., connection weights). We assume that these functions are differentiable with respect to their parameters. The ranges of the functions $N_j()$ may be constrained in order to account for the underlying transition graph $\mathcal{G}$. Each output $\varphi_{ij,t}$ of the state subnetwork $\mathcal{N}_j$ 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 $\mathcal{S}_j$. For convenience of notation, we suppose that $\varphi_{ij,t}$ are defined for each $i, j = 1, \ldots, n$ and we impose the condition $\varphi_{ij,t} = 0$ for each i not belonging to $\mathcal{S}_j$. To guarantee that the variables $\varphi_{ij,t}$ are positive and summing to 1, the *softmax* function [33] is used in the last layer:

$$\varphi_{ij,t} = \frac{\mathrm{e}^{a_{ij,t}}}{\sum_{\ell \in \mathcal{S}_j} \mathrm{e}^{a_{\ell j,t}}}, \quad j = 1, \ldots, n, \quad i \in \mathcal{S}_j \tag{2}$$

where $a_{ij,t}$ are intermediate variables that can be thought of as the activations (e.g, weighted sums) of the

output units of subnetwork $\mathcal{N}_j$. In this way,

$$\sum_{i=1}^{n} \varphi_{ij,t} = 1 \ \ \forall j, t. \tag{3}$$

and φ can be given a probabilistic interpretation. As shown in Figure 1, the outputs $\boldsymbol{\varphi}$ of the state networks are used to recursively compute at each time step the vector $\boldsymbol{\zeta}_t \in \boldsymbol{R}^n$, which represents the current "memory" of the system, and can be interpreted as the current state distribution, given the past input sequence. This "memory" variable is computed as a linear combination of the outputs of the state networks, gated by its value at the previous time step:

$$\boldsymbol{\zeta}_t = \sum_{j=1}^{n} \zeta_{j,t-1} \, \boldsymbol{\varphi}_{j,t} \tag{4}$$

where $\boldsymbol{\varphi}_{j,t} = [\varphi_{1j,t}, \ldots, \varphi_{nj,t}]'$.

Output networks compete to predict the global output of the system $\boldsymbol{\eta}_t \in \boldsymbol{R}^r$:

$$\boldsymbol{\eta}_t = \sum_{j=1}^{n} \zeta_{j,t} \boldsymbol{\eta}_{j,t} \tag{5}$$

where $\boldsymbol{\eta}_{j,t} \in \boldsymbol{R}^r$ is the output of subnetwork $\mathcal{O}_j$.

At this level of description, we do not need to further specify the internal architecture of the state and output subnetworks, as long as they compute a differentiable function of their parameters.

2.2 A Probabilistic Model

As hinted above, this connectionist architecture can be also interpreted as a probability model. To simplify, we assume here a multinomial distribution for the state variable x_t, i.e., a probability is computed for each possible value of the state variable x_t. Let us consider $\boldsymbol{\zeta}_t$, the main variable of the temporal recurrence $\boldsymbol{\zeta}_t = \sum_{j=1}^{n} \zeta_{j,t-1} \boldsymbol{\varphi}_{j,t}$. If we initialize the vector $\boldsymbol{\zeta}_0$ to positive numbers summing to 1, it can be interpreted as a vector of initial state probabilities. Because (4) is a convex sum, $\boldsymbol{\zeta}_t$ is a vector of positive numbers summing to 1 for each t, and it will be given the following interpretation:

$$\zeta_{i,t} = \mathrm{P}(x_t = i \mid \boldsymbol{u}_1^t) \tag{6}$$

having denoted by $\boldsymbol{u}_1^t$ the subsequence of inputs from time 1 to t, inclusively. When making certain conditional independence assumptions described in the next section, equation (4) then has the following

probabilistic interpretation:

$$P(x_t = i \mid \boldsymbol{u}_1^t) = \sum_{j=1}^{n} P(x_t = i \mid x_{t-1=}j, \boldsymbol{u}_t) P(x_{t-1=}j \mid \boldsymbol{u}_1^{t-1}) \tag{7}$$

i.e., the subnetworks $\mathcal{N}_j$ compute transition probabilities conditioned on the input $\boldsymbol{u}_t$:

$$\varphi_{ij,t} = \mathrm{P}(x_t = i \mid x_{t-1} = j, \boldsymbol{u}_t) \tag{8}$$

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

$$\boldsymbol{\eta}_{i,t} = E[\boldsymbol{y}_t \mid x_t = i, \boldsymbol{u}_t]. \tag{9}$$

The total probability $P(\boldsymbol{y}_t \mid \boldsymbol{u}_t; \boldsymbol{\vartheta})$ is obtained as a mixture of the probabilities $P(\boldsymbol{y}_t \mid x_{t=}i, \boldsymbol{u}_t; \boldsymbol{\vartheta}_i)$, which are conditional to the present state [1]. For example, with a Gaussian output model for each subnetwork (corresponding to a mean squared error criterion), the output distribution is in fact a mixture of Gaussians. In general, the state distribution predicted by the set of state subnetworks provides the mixing proportions:

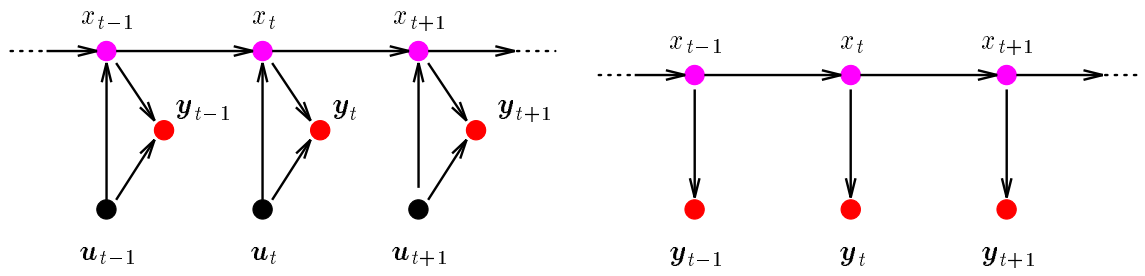
$$P(\boldsymbol{y}_t \mid \boldsymbol{u}_1^t) = \sum_i P(x_t = i \mid \boldsymbol{u}_1^t) P(\boldsymbol{y}_t \mid x_{t=}i, \boldsymbol{u}_t) \tag{10}$$

where the actual form of the output densities $P(\boldsymbol{y}_t \mid x_{t=}i, \boldsymbol{u}_t)$ will be chosen according to the task. For example a multinomial distribution is suitable for sequence classification, or for symbolic mutually exclusive outputs. Instead, a Gaussian distribution is adequate for producing continuous outputs. In the first case we use a softmax function at the output of subnetworks $\mathcal{O}_j$; in the second case we use linear output units for the subnetworks $\mathcal{O}_j$.

2.3 Conditional Dependencies

The random variables (for input, state, and output) 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 as-

[1] To simplify the notation, we write the probability $\mathrm{P}(X = x)$ that the discrete random variable X takes on the value x as $\mathrm{P}(x)$, unless this introduces some ambiguity. Similarly, if X is a continuous variable we use $\mathrm{P}(x)$ to denote its probability density.



(a) (b)

Figure 2: (a): Bayesian network expressing conditional dependencies among the random variables in the probabilistic interpretation of the recurrent architecture. (b): Bayesian network for a standard HMM.

sumptions, 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. For any three random variables A, B and C, we say that A is conditionally independent on B given C, written $I(A, C, B)$, if $\mathrm{P}(A = a \mid B = b, C = c) = \mathrm{P}(A = a \mid C = c)$ for each pair (a, c) such that $\mathrm{P}(A = a, C = c) > 0$. A dependency model M is a mapping that assigns truth values to "independence" predicates of the form $I(A, C, B)$. Rather than listing a set of conditional independency assumptions, we prefer to express dependencies using a graphical representation. A dependency model M can be represented by means of a directed acyclic graph (DAG), called *Bayesian network* of M. A formal definition of Bayesian networks can be found in [34]. In practice a Bayesian network is constructed by allocating one node for each variable in $\mathcal{S}$ and by creating one edge $A \to B$ for each variable A that is believed to have a direct causal impact on B.

Assumption 1 *We suppose that the DAG $\mathcal{G}$ depicted in Figure 2a is a Bayesian network for the dependency model M associated to the variables $\boldsymbol{u}_1^T, x_1^T, \boldsymbol{y}_1^T$.*

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. In fact, the Bayesian network for HMMs can be obtained by simply removing the $\boldsymbol{u}_t$ nodes and arcs from them (see Figure 2b). However, there are other basic

differences between this architecture and standard HMMs, both in terms of computing style and learning. These differences will be further discussed in 4.1.

3 A Supervised Learning Algorithm

The learning algorithm for the proposed architecture is derived from the maximum likelihood principle. An extension that takes into account priors on the parameters is straightforward and will not be discussed here. We will discuss here the case where the training data are a set of P pairs of input/output sequences, independently sampled from the same distribution:

$$\mathcal{D} \stackrel{\text{def}}{=} (\mathcal{U}, \mathcal{Y}) \stackrel{\text{def}}{=} \{(\boldsymbol{u}_1^{T_p}(p), \boldsymbol{y}_1^{T_p}(p)); p = 1 \ldots P\}.$$

Let $\boldsymbol{\Theta}$ denote the vector of parameters obtained by collecting all the parameters $\boldsymbol{\theta}_j$ and $\boldsymbol{\vartheta}_i$ of the architecture. The likelihood function is then given by[2]:

$$L(\boldsymbol{\Theta}; \mathcal{D}) \stackrel{\text{def}}{=} \mathrm{P}(\mathcal{Y} \mid \mathcal{U}; \boldsymbol{\Theta}) = \prod_{p=1}^{P} \mathrm{P}(\boldsymbol{y}_1^{T_p}(p) \mid \boldsymbol{u}_1^{T_p}(p); \boldsymbol{\Theta}). \tag{11}$$

The output values (used here as targets) may also be specified intermittently. For example, in sequence classification tasks, one is only interested in the output $\boldsymbol{y}_T$ at the end of each sequence. The modification of the likelihood to account for intermittent targets is straightforward. According to the maximum likelihood principle, the optimal parameters are obtained by maximizing (11). The optimization problem arising from this formulation of learning can be addressed in the framework of parameter estimation with *missing data*, where the missing variables are the state paths $\mathcal{X} = \{x_1^{T_p}(p); p = 1 \ldots P\}$ (describing a path in state space, for each sequence). Let us first briefly describe the EM algorithm.

3.1 The EM Algorithm

EM (*estimation-maximization*) is an iterative approach to maximum likelihood estimation (MLE), originally proposed in [19]. Each iteration is composed of two steps: an estimation (E) step and a maximization

[2]In the following, in order to simplify the notation, the sequence index p may be omitted.

(M) step. The aim is to maximize the log-likelihood function $l(\boldsymbol{\Theta}; \mathcal{D}) = \log L(\boldsymbol{\Theta}; \mathcal{D})$ where $\boldsymbol{\Theta}$ are the parameters of the model and $\mathcal{D}$ are the data. Suppose that this optimization problem would be simplified by the knowledge of additional variables $\mathcal{X}$, known as missing or hidden data. The set $\mathcal{D}_c = \mathcal{D} \cup \mathcal{X}$ is referred to as the complete data set (in the same context $\mathcal{D}$ is referred to as the incomplete data set). Correspondingly, the log-likelihood function $l_c(\boldsymbol{\Theta}; \mathcal{D}_c)$ is referred to as the complete data likelihood. $\mathcal{X}$ is chosen such that the function $l_c(\boldsymbol{\Theta}; \mathcal{D}_c)$ would be easily maximized if $\mathcal{X}$ were known. However, since $\mathcal{X}$ is not observable, l_c is a random variable and cannot be maximized directly. Thus, the EM algorithm relies on integrating over the distribution of $\mathcal{X}$, with the auxiliary function

$$Q(\boldsymbol{\Theta}; \hat{\boldsymbol{\Theta}}) = E_{\mathcal{X}} \left[l_c(\boldsymbol{\Theta}; \mathcal{D}_c) \mid \mathcal{D}, \hat{\boldsymbol{\Theta}} \right] \tag{12}$$

which is the expected value of the complete data log-likelihood, given the observed data $\mathcal{D}$ and the parameters $\hat{\boldsymbol{\Theta}}$ computed at the end of the previous iteration. Intuitively, computing Q corresponds to filling in the missing data using the knowledge of observed data and previous parameters. The auxiliary function is deterministic and can be maximized. An EM algorithm thus iterates the following two steps, for $k = 1, 2, \ldots$, until a local maximum of the likelihood is found:

$$\begin{aligned} &\text{Estimation:} && \text{Compute } Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(k)}) = E_{\mathcal{X}}[l_c(\boldsymbol{\Theta}; \mathcal{D}_c) \mid \mathcal{D}, \boldsymbol{\Theta}^{(k)}] \\ &\text{Maximization:} && \text{Update the parameters as } \boldsymbol{\Theta}^{(k+1)} = \arg\max_{\boldsymbol{\Theta}} Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(k)}) \end{aligned} \tag{13}$$

In some cases, it is difficult to analytically maximize $Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}^{(k)})$, as required by the M step of the above algorithm, and we are only able to compute a new value $\boldsymbol{\Theta}^{(k+1)}$ that produces an increase of Q. In this case we have a so called generalized EM (GEM) algorithm:

$$\begin{gathered} \text{Update the parameters as } \boldsymbol{\Theta}^{(k+1)} = M(\boldsymbol{\Theta}^{(k)}) \text{ where } M(\cdot) \text{ is such that} \\ Q(M(\boldsymbol{\Theta}^{(k)}); \boldsymbol{\Theta}^{(k)}) \geq Q(\boldsymbol{\Theta}^{(k)}; \boldsymbol{\Theta}^{(k)}); \end{gathered} \tag{14}$$

The following theorem guarantees the convergence of EM and GEM algorithms to a (possibly local) maximum of the (incomplete data) likelihood:

Theorem 1 (Dempster et al. [19]) *For each GEM algorithm*

$$L(M(\boldsymbol{\Theta}); \mathcal{D}) \geq L(\boldsymbol{\Theta}; \mathcal{D}) \tag{15}$$

where the equality holds if and only if

$$Q(M(\boldsymbol{\Theta});\boldsymbol{\Theta}) = Q(\boldsymbol{\Theta};\boldsymbol{\Theta}). \tag{16}$$

3.2 EM for Training IOHMMs

In order to apply EM to IOHMMs we begin by noting that the variable $\mathcal{X}$, representing the paths in state space, is not observed. Knowledge of this variable would allow one to decompose the temporal learning problem into $2n$ *static*[3] learning subproblems. Indeed, if x_t was known the state probabilities $\zeta_{i,t}$ would reduce to either 0 or 1 and it would be possible to independently train each instance of the subnetworks at different time steps[4], without taking into account any temporal dependency (taking into account only the sharing of parameters across different times). This observation allows us to link EM learning to the target propagation approach discussed in the introduction. Note that if we used a Viterbi-like approximation (i.e., considering only the most likely path), we would indeed have $2n$ static learning problems at each epoch. Actually, we can think of the E step of EM as an approximation of the oracle that provides target states for each time t, based on averaging over all values of $\mathcal{X}$, where the distribution of $\mathcal{X}$ is conditioned on the values of the parameters at the previous epoch. The M step then fits the parameters, for the next epoch, to the estimated trajectories.

In the sequel of this section we derive the learning equations for our architecture. Let us define the *complete data* as

$$\mathcal{D}_c \stackrel{\text{def}}{=} (\mathcal{U}, \mathcal{Y}, \mathcal{X}) \stackrel{\text{def}}{=} \{(\boldsymbol{u}_1^{T_p}(p), \boldsymbol{y}_1^{T_p}(p), x_1^{T_p}(p)); p = 1 \ldots P\}.$$

The corresponding complete data likelihood is

$$L_c(\boldsymbol{\Theta};\mathcal{D}_c) = P(\mathcal{Y}, \mathcal{X} \mid \mathcal{U}) = \prod_{p=1}^{P} \mathrm{P}(\boldsymbol{y}_1^{T_p}(p), x_1^{T_p}(p) \mid \boldsymbol{u}_1^{T_p}(p); \boldsymbol{\Theta}) \tag{17}$$

that can be decomposed as follows[5]:

$$\mathrm{P}(\boldsymbol{y}_1^T, x_1^T \mid \boldsymbol{u}_1^T; \boldsymbol{\Theta}) \quad = \quad \mathrm{P}(\boldsymbol{y}_T, x_T \mid \boldsymbol{y}_1^{T-1}, x_1^{T-1}, \boldsymbol{u}_1^T; \boldsymbol{\Theta}) \mathrm{P}(\boldsymbol{y}_1^{T-1}, x_1^{T-1} \mid \boldsymbol{u}_1^T; \boldsymbol{\Theta})$$

[3] static, as in feedforward networks, by opposition to dynamic, e.g., involving back-propagation through time.

[4] credit would be assigned to only one transition network and one output network.

[5] again we omit p.

$$= \quad \mathrm{P}(\boldsymbol{y}_T, x_T \mid x_{T-1}, \boldsymbol{u}_T; \boldsymbol{\Theta}) \mathrm{P}(\boldsymbol{y}_1^{T-1}, x_1^{T-1} \mid \boldsymbol{u}_1^{T-1}; \boldsymbol{\Theta}). \tag{18}$$

The last equality follows from the conditional independency model that we have assumed. Iterating the decomposition we obtain the following factorization of the complete likelihood:

$$L_c(\boldsymbol{\Theta}; \mathcal{D}_c) = P(\mathcal{Y}, \mathcal{X} \mid \mathcal{U}) = \prod_{p=1}^{P} \prod_{t=1}^{T_p} \mathrm{P}(\boldsymbol{y}_t, x_t \mid x_{t-1}, \boldsymbol{u}_t; \boldsymbol{\Theta}).$$

Let us define the vector of indicator variables $\boldsymbol{z}_t$ as follows: $z_{i,t} = 1$ if $x_t = i$, and $z_{i,t} = 0$ otherwise. Since the state distribution is multinomial, $E[z_{i,t} \mid \boldsymbol{u}_1^t] = P(z_{i,t} \mid \boldsymbol{u}_1^t) = P(x_t = i \mid \boldsymbol{u}_1^t) = \zeta_{i,t}$. Using indicator variables, we rewrite the complete data likelihood as follows:

$$\begin{aligned} L_c(\boldsymbol{\Theta}; \mathcal{D}_c) &= \prod_{p=1}^{P} \prod_{t=1}^{T} \mathrm{P}(\boldsymbol{y}_t \mid x_t, \boldsymbol{u}_t; \boldsymbol{\Theta}) \mathrm{P}(x_t \mid x_{t-1} \boldsymbol{u}_t; \boldsymbol{\Theta}) \\ &= \prod_{p=1}^{P} \prod_{t=1}^{T} \prod_{i=1}^{n} \prod_{j=1}^{n} \mathrm{P}(\boldsymbol{y}_t \mid x_t = i, \boldsymbol{u}_t; \boldsymbol{\Theta})^{z_{i,t}} \ \mathrm{P}(x_t = i \mid x_{t-1} = j, \boldsymbol{u}_t; \boldsymbol{\Theta})^{z_{i,t} z_{j,t-1}}. \end{aligned}$$

Taking the logarithm we obtain the following expression for the complete data log-likelihood:

$$\begin{aligned} l_c(\boldsymbol{\Theta}; \mathcal{D}_c) &= \log L_c(\boldsymbol{\Theta}; \mathcal{D}_c) \\ &= \sum_{p=1}^{P} \sum_{t=1}^{T} \sum_{i=1}^{n} z_{i,t} \log \mathrm{P}(\boldsymbol{y}_t \mid x_t{=}i, \boldsymbol{u}_t; \boldsymbol{\Theta}) + \sum_{j=1}^{n} z_{i,t} z_{j,t-1} \log \mathrm{P}(x_t{=}i \mid x_{t-1} \ Eqj, \boldsymbol{u}_t; \boldsymbol{\Theta}). \end{aligned} \tag{19}$$

Since $l_c(\boldsymbol{\Theta}; \mathcal{D}_c)$ depends on the unknown state variable $\mathcal{X}$ we cannot maximize directly (19). If $\mathcal{X}$ was given, the temporal credit assignment problem would be solved. To complete the training there would only remain to learn from data the static mappings that produce the output and the state transitions. In this situation EM helps to decouple static and temporal learning.

3.2.1 The Estimation Step

We can compute the expected value of $l_c(\boldsymbol{\Theta}; \mathcal{D}_c)$ with respect to the distribution of the paths $\mathcal{X}$, given the data $\mathcal{D}$ and the "old" parameters $\hat{\boldsymbol{\Theta}}$:

$$\begin{aligned} Q(\boldsymbol{\Theta}; \hat{\boldsymbol{\Theta}}) &= E_{\mathcal{X}}[l_c(\boldsymbol{\Theta}; \mathcal{D}_c) \mid \mathcal{U}, \mathcal{Y}, \hat{\boldsymbol{\Theta}}] \\ &= \sum_{p=1}^{P} \sum_{t=1}^{T} \sum_{i=1}^{n} E_{\mathcal{X}}[z_{i,t} \mid \boldsymbol{u}_1^T, \boldsymbol{y}_1^T, \hat{\boldsymbol{\Theta}}] \log \mathrm{P}(\boldsymbol{y}_t \mid x_t = i, \boldsymbol{u}_t; \boldsymbol{\Theta}) \\ &+ \sum_{j} E_{\mathcal{X}}[z_{i,t} z_{j,t-1} \mid \boldsymbol{u}_1^T, \boldsymbol{y}_1^T, \hat{\boldsymbol{\Theta}}] \log \mathrm{P}(x_t = i \mid x_{t-1} = j, \boldsymbol{u}_t; \boldsymbol{\Theta}) \\ &= \sum_{p=1}^{P} \sum_{t=1}^{T} \sum_{i=1}^{n} \hat{g}_{i,t} \log \mathrm{P}(\boldsymbol{y}_t \mid x_t = i, \boldsymbol{u}_t; \boldsymbol{\Theta}) + \sum_{j=1}^{n} \hat{h}_{ij,t} \log \mathrm{P}(x_t = i \mid x_{t-1} = j, \boldsymbol{u}_t; \boldsymbol{\Theta}) \end{aligned} \tag{20}$$

where $g_{i,t} \stackrel{\text{def}}{=} P(x_t = i \mid \boldsymbol{u}_1^T, \boldsymbol{y}_1^T; \boldsymbol{\Theta})$, and the $h_{ij,t} \stackrel{\text{def}}{=} P(x_t = i, x_{t-1} = j \mid \boldsymbol{u}_1^T, \boldsymbol{y}_1^T; \boldsymbol{\Theta})$ are the elements of the autocorrelation matrix for consecutive state pairs. The hat in $\hat{g}_{i,t}$ and $\hat{h}_{ij,t}$ means that these variables are computed using the old parameters $\hat{\boldsymbol{\Theta}}$.

In order to compute $h_{ij,t}$ and $g_{i,t}$ we introduce the following probabilities, borrowing the notation from the HMM literature:

$$\alpha_{i,t} \stackrel{\text{def}}{=} \mathrm{P}(\boldsymbol{y}_1^t, x_t = i \mid \boldsymbol{u}_1^t); \tag{21}$$

$$\beta_{i,t} \stackrel{\text{def}}{=} \mathrm{P}(\boldsymbol{y}_{t+1}^T \mid x_t = i, \boldsymbol{u}_t^T). \tag{22}$$

$\alpha_{i,t}$ can be rewritten as follows:

$$\begin{aligned} \alpha_{i,t} &= \mathrm{P}(\boldsymbol{y}_1^t, x_t = i \mid \boldsymbol{u}_1^t) \\ &= \sum_\ell \mathrm{P}(\boldsymbol{y}_1^t, x_t = i, x_{t-1} = \ell \mid \boldsymbol{u}_1^t) \\ &= \sum_\ell \mathrm{P}(\boldsymbol{y}_t \mid \boldsymbol{y}_1^{t-1} x_t = i, x_{t-1} = \ell, \boldsymbol{u}_1^t) \mathrm{P}(x_t = i \mid \boldsymbol{y}_1^{t-1}, x_{t-1} = \ell, \boldsymbol{u}_1^t) \mathrm{P}(\boldsymbol{y}_1^{t-1}, x_{t-1} = \ell \mid \boldsymbol{u}_1^t) \end{aligned} \tag{23}$$

and thus, using the conditional independence assumptions graphically depicted in Figure 2 and previously discussed, we obtain

$$\alpha_{i,t} = P(\boldsymbol{y}_t \mid x_{t=} i, \boldsymbol{u}_t) \sum_\ell \varphi_{i\ell}(\boldsymbol{u}_t) \alpha_{\ell,t-1}. \tag{24}$$

where $P(\boldsymbol{y}_t \mid x_{t=} i, \boldsymbol{u}_t)$ specifies the output distribution in state i and $\varphi_{i\ell}(\boldsymbol{u}_t)$ specifies the next state distribution in state i, both conditioned on the current input. This recursion is initialized with the initial state probabilities $\alpha_{i,0} \stackrel{\text{def}}{=} P(x_0 = i)$ (which can fixed a priori or learned as extra parameters, as in HMMs). In general, we will constrain the model to end up in one of several final states from the set $\mathcal{F}$, so the likelihood $L(\boldsymbol{\Theta}; \mathcal{D}_p)$ for a sequence p can be written in terms of the α's:

$$L = \mathrm{P}(\boldsymbol{y}_1^T \mid \boldsymbol{u}_1^T) = \sum_{i \in \mathcal{F}} \mathrm{P}(\boldsymbol{y}_1^T, x_T = i \mid \boldsymbol{u}_1^T) = \sum_i \alpha_{i,T}. \tag{25}$$

Similarly, a backward recursion can be established for $\beta_{i,t}$:

$$\begin{aligned} \beta_{i,t} &= \mathrm{P}(\boldsymbol{y}_{t+1}^T \mid x_t = i, \boldsymbol{u}_t^T) \\ &= \sum_\ell \mathrm{P}(\boldsymbol{y}_{t+1}^T, x_{t+1} = \ell \mid x_t = i, \boldsymbol{u}_t^T) \\ &= \sum_\ell \mathrm{P}(\boldsymbol{y}_{t+1} \mid \boldsymbol{y}_{t+2}^T, x_{t+1} = \ell, x_t = i, \boldsymbol{u}_t^T) \mathrm{P}(\boldsymbol{y}_{t+1}^T \mid x_{t+1} = \ell, x_t = i, \boldsymbol{u}_t^T) \mathrm{P}(x_{t+1} = \ell \mid x_t = i, \boldsymbol{u}_t^T) \end{aligned} \tag{26}$$

and thus using the conditional independence assumptions:

$$\beta_{i,t} = \sum_{\ell} P(\boldsymbol{y}_{t+1} \mid x_{t+1=}l, \boldsymbol{u}_t)\varphi_{\ell i}(\boldsymbol{u}_{t+1})\beta_{\ell,t+1}. \tag{27}$$

where the backward recursion is initialized with $\beta_{i,T} = 1$ if $i \in \mathcal{F}$ and 0 otherwise. It is maybe useful to remark how these distributions are computed. $P(\boldsymbol{y}_t \mid x_{t=}i, \boldsymbol{u}_t)$ is obtained by running subnetwork $\mathcal{O}_i$ on the input vector $\boldsymbol{u}_t$, and plugging the target vector $\boldsymbol{y}_t$ and $\mathcal{O}_i$'s output $\boldsymbol{\eta}_{i,t}$ into the algebraic expression of the output distribution. $\varphi_{i\ell}(\boldsymbol{u}_t)$ is simply the ℓ-th output of subnetwork $\mathcal{N}_i$, fed with the input vector $\boldsymbol{u}_t$.

The transition posterior probabilities $h_{ij,t}$ can be expressed in terms of α and β:

$$\begin{aligned}
h_{ij,t} &= \mathrm{P}(x_{t=}i, x_{t-1=}j \mid \boldsymbol{y}_1^T \boldsymbol{u}_1^T) \\
&= \mathrm{P}(x_{t=}i, x_{t-1=}j, \boldsymbol{y}_1^T \mid \boldsymbol{u}_1^T)/\mathrm{P}(\boldsymbol{y}_1^T \mid \boldsymbol{u}_1^T) \\
&= \mathrm{P}(\boldsymbol{y}_{t+1}^T \mid x_{t=}i, x_{t-1=}j, \boldsymbol{y}_1^t, \boldsymbol{u}_1^T)\mathrm{P}(\boldsymbol{y}_t \mid x_{t=}i, x_{t-1=}j, \boldsymbol{y}_1^{t-1}, \boldsymbol{u}_1^T)\mathrm{P}(\boldsymbol{y}_1^{t-1}, x_{t-1=}j \mid \boldsymbol{u}_1^T)\mathrm{P}(x_{t=}i \mid x_{t-1=}j, \boldsymbol{y}_1^{t-1}, \boldsymbol{u}_1^T)/L \\
&= \mathrm{P}(\boldsymbol{y}_t | x_{t=}i, \boldsymbol{u}_t)\alpha_{j,t-1}\beta_{i,t}\varphi_{ij}(\boldsymbol{u}_t)/L
\end{aligned} \tag{28}$$

where the last equation is obtained using the conditional independency assumptions and L is the conditional likelihood (equation 25). The state posterior probabilities $g_{i,t}$ can be obtained by summing h's, with $\sum_j h_{ij,t}$, or directly:

$$\begin{aligned}
g_{i,t} &= \mathrm{P}(x_{t=}i \mid \boldsymbol{y}_1^T \boldsymbol{u}_1^T) \\
&= \mathrm{P}(x_{t=}i, \boldsymbol{y}_1^T \mid \boldsymbol{u}_1^T)/L \\
&= \mathrm{P}(\boldsymbol{y}_{t+1}^T \mid x_t = i, \boldsymbol{y}_1^t, \boldsymbol{u}_1^T)\mathrm{P}(\boldsymbol{y}_1^t, x_{t=}i \mid \boldsymbol{u}_1^T)/L \\
&= \alpha_{i,t}\beta_{i,t}/L
\end{aligned} \tag{29}$$

To summarize, we obtain equations similar to those used to train HMMs with the Baum-Welch algorithm. A forward linear recursion (equation 24) can be used to compute the likelihood (equation 25). During training, a backward linear recursion (equation 27) is performed, that is equivalent to back-propagating through time gradients of the likelihood with respect to the α's ($\beta_{i,t} = \frac{\partial L}{\partial \alpha_{i,t}}$, see also [35]). Notice that the sums in equations (24), (27) and (25) can be constrained by the transition graph underlying the model.

3.2.2 The Maximization Step

Each iteration of the EM algorithm requires to maximize $Q(\boldsymbol{\Theta};\boldsymbol{\Theta}^{(k)})$. As explained below, if the subnetworks are linear this can be done analytically (for example with symbolic inputs). In general, however, if the subnetworks have hidden sigmoidal units, or use a softmax function to constrain their outputs to sum to one, the maximum of Q cannot be found analytically. In these cases we can resort to a GEM algorithm, that simply produces an increase in Q, for example by gradient ascent. Although Theorem 1 guarantees the convergence of GEM algorithms to a local maximum of the likelihood, their convergence may be significantly slower compared to EM. However, the parameterization of transition probabilities through layers of neural units makes the learning algorithm smooth and suitable for use in on-line mode (i.e., updating the parameters after each sequence presentation, rather than accumulating parameter change information over the whole training set). This is a desirable property [36] and may often help to speed up learning. Indeed, in several experiments we noticed that convergence can be accelerated using stochastic (i.e., on-line) gradient ascent on the auxiliary function.

3.2.3 General form of the IOHMM training algorithm

In the most general form, IOHMM training can be summarized by the following algorithm.

Algorithm 1

1 **foreach** training sequence $(\boldsymbol{u}_1^T, \boldsymbol{y}_1^T)$ **do** ▷ *Estimation step*

1.1 **foreach** state $j \leftarrow 1 \ldots n$ **do**

compute $\varphi_{ij,t}$, $i \in \mathcal{S}_j$ and $\boldsymbol{\eta}_{j,t}$, by running forward the state and the output subnetworks $\mathcal{N}_j$ and $\mathcal{O}_j$;

1.2 **foreach** $i \leftarrow 1 \ldots n$ **do**

compute $\alpha_{i,t}$ and $\beta_{i,t}$ (forward backward recurrences (24) and (27)) using the current value $\hat{\boldsymbol{\Theta}}$ of the parameters;

compute the posterior probabilities $\hat{h}_{ij,t}$ (for each j such that $i \in \mathcal{S}_j$) and $\hat{g}_{i,t}$ (eqs. (28) and—(29));

2 **foreach** state $j \leftarrow 1 \ldots n$ **do** ▷ *Maximization step*

2.1 adjust the parameters $\boldsymbol{\theta}_j$ of state subnetwork $\mathcal{N}_j$ to maximize (or increase, for a GEM algorithm) the function

$$\sum_{p=1}^{P}\sum_{t=1}^{T}\sum_{i=1}^{n} \hat{h}_{ij,t} \log P(x_t = i \mid x_{t-1}=j, \boldsymbol{u}_t; \boldsymbol{\theta}_j);$$

2.2 adjust the parameters $\boldsymbol{\vartheta}_j$ to of output subnetwork $\mathcal{O}_j$ to maximize (or increase, for a GEM algorithm) the function

$$\sum_{p=1}^{P} \sum_{t=1}^{T} \hat{g}_{j,t} \log P(\boldsymbol{y}_t \mid x_t = j, \boldsymbol{u}_t; \boldsymbol{\vartheta}_j);$$

3 **let** $\hat{\boldsymbol{\Theta}} \leftarrow \boldsymbol{\Theta}$ and iterate using the updated parameters.

In general there are $m \leq n^2$ allowed transitions in the graph (for n states). Let p be the number of weights (or parameters) in the transition and output models. Therefore the time complexity for each time step is $O(m+p)$, as in ordinary HMMs (for which p is simply equal to the number of parameters in the output models). Therefore the total computation for a training epoch is $O((m+p)T)$, where T is the sum of the lengths of all the sequences in the training set. This is similar to the case of recurrent networks trained with backpropagation through time, $O(n_w T)$, where n_w is the number of weights.

3.3 Specializations of the training algorithm

Steps 2 and 3 of Algorithm 1 (corresponding to the Maximization step of EM) can be implemented in different forms, depending on the nature of the data and of the subnetworks that compose the architecture.

3.3.1 Lookup-table networks for symbolic data processing

We describe now a procedure that applies a true EM algorithm when the inputs are discrete and the subnetworks behave like lookup tables addressed by the input symbols. For simplicity, we restrict the following analysis to sequence classification tasks. Since we assume that the model will be able to discriminate the different classes of sequences, we will simplify the system by associating one final state to each of the classes, and by assuming that the last input is not necessary to perform the classification. Therefore, there will be no output except at the last time step, and at the last time step the probability distribution over final states $\mathrm{P}(x_T \mid \boldsymbol{u}_1^T)$ will directly give the probability distribution over output classes $\mathrm{P}(y \mid \boldsymbol{u}_1^T)$. Therefore, no output subnetworks need to be used in this particular application of the algorithm, since the output is directly read from the final state. During learning, the target class gives us a final target

state $x^\star(p)$ (for the p-th sequence). The likelihood function can then be simplified as follows:

$$L(\boldsymbol{\Theta}; \mathcal{D}) \stackrel{\text{def}}{=} \prod_{p=1}^{P} \mathrm{P}(x_T = x^\star(p) \mid \boldsymbol{u}_1^{T_p}(p); \boldsymbol{\Theta}). \tag{30}$$

Symbolic inputs can be encoded using index vectors. In particular, if $A = \{1, \ldots, m\}$ is the input alphabet, the symbol k is encoded by the vector $\boldsymbol{u}$ having a 1 in the k-th position and 0 elsewhere. Let us suppose that each state network has a single linear layer. Then, the weights of subnetwork $\mathcal{N}_j$ take the meaning of probabilities of transition from state j, conditional on the input symbol, i.e.

$$w_{ijk} = \mathrm{P}(x_t = i \mid x_{t-1} = j, u_{k,t} = 1) \tag{31}$$

In order to preserve consistency with the probabilistic interpretation of the model, such weights must be nonnegative and

$$\sum_{i \in \mathcal{S}_j} w_{ijk} = 1 \quad \forall j = 1, \ldots, n \quad \forall k = 1, \ldots, m. \tag{32}$$

This constraint can be easily incorporated in the maximization of the likelihood by introducing the new function

$$J(\boldsymbol{\Theta}, \hat{\boldsymbol{\Theta}}) \stackrel{\text{def}}{=} Q(\boldsymbol{\Theta}, \hat{\boldsymbol{\Theta}}) + \sum_{j=1}^{m} \sum_{k=1}^{m} \left(1 - \sum_{i \in \mathcal{S}_j} w_{ijk} \right) \lambda_{jk} \tag{33}$$

where λ_{jk} are Lagrange multipliers. Taking the derivatives of this function with respect to the weights we find

$$\frac{\partial J}{\partial w_{ijk}} = \sum_{p=1}^{P} \sum_{t:\sigma_t = k} \hat{h}_{ij,t} \frac{1}{w_{ijk}} - \lambda_{jk} \tag{34}$$

where the second sum is extended to all the time steps for which the input symbol σ_t takes on the value k. The above expression is zero if

$$w_{ijk} = \frac{\displaystyle\sum_{p=1}^{P} \sum_{t:\sigma_t = k} \hat{h}_{ij,t}}{\lambda_{jk}} \tag{35}$$

and then, imposing the constraint $\sum_{i \in \mathcal{S}_j} w_{ijk} = 1$ we obtain $\lambda_{jk} = \sum_{i \in \mathcal{S}_j} \sum_{p=1}^{P} \sum_{t:\sigma_t = k} \hat{h}_{ij,t}$.

3.3.2 Nonlinear Subnetworks

We consider here the general case of nonlinear (for example, multilayered) subnetworks. Since direct analytic maximization of Q is not possible, a GEM algorithm must be used. A very simple way of

producing an increase in Q is to use gradient ascent. The derivatives of Q with respect to the parameters can be easily computed as follows. Let θ_{jk} be a generic weight in the state subnetwork $\mathcal{N}_j$. From equation (20) we have

$$\frac{\partial Q(\boldsymbol{\Theta};\hat{\boldsymbol{\Theta}})}{\partial \theta_{jk}} = \sum_{p=1}^{P}\sum_{t=1}^{T_p}\sum_{i\in\mathcal{S}_j} \hat{h}_{ij,t}\frac{1}{\varphi_{ij,t}}\frac{\partial \varphi_{ij,t}}{\partial \theta_{jk}} \tag{36}$$

where the partial derivatives $\frac{\partial \varphi_{ij,t}}{\partial \theta_{jk}}$ can be computed using back-propagation.

Similarly, denoting with ϑ_{ik} a generic weight of the output subnetwork $\mathcal{O}_i$, we have:

$$\frac{\partial Q(\boldsymbol{\Theta};\hat{\boldsymbol{\Theta}})}{\partial \vartheta_{ik}} = \sum_{p=1}^{P}\sum_{t=1}^{T_p}\sum_{\ell=1}^{r} \hat{g}_{i,t}\frac{\partial}{\partial \eta_{i\ell,t}}\log P(\boldsymbol{y}_t \mid x_{t=}i, \boldsymbol{u}_t)\frac{\partial \eta_{i\ell,t}}{\partial \vartheta_{ik}} \tag{37}$$

where $\frac{\partial \eta_{i\ell,t}}{\partial \vartheta_{ik}}$ are also computed using back-propagation. Intuitively, the parameters are updated as if the estimation step of EM had provided soft targets for the outputs of the $2n$ subnetworks, for each time t.

4 Comparisons

4.1 Standard Hidden Markov Models

The model proposed here is a natural extension of HMMs [20, 21, 22]: the distribution of the output sequence is conditioned on an input sequence. Furthermore, we propose to parameterize the next-state and output distributions with complex modules such as artificial neural networks.

The most typical applications of standard HMMs are in automatic speech recognition [21, 37]. In these cases each lexical unit is associated to one model $\mathcal{M}_i$. During recognition one computes for each model the probability $\mathrm{P}(\boldsymbol{y}_1^T \mid \mathcal{M}_i)$ of having generated the observed acoustic sequence $\boldsymbol{y}_1^T$. During training the parameters are adjusted to maximize the probability that the correct model $\mathcal{M}_i$ generates the acoustic observations associated to instances of the i-th lexical unit. Training is therefore not discriminant: it does not try to learn how to decide which phoneme sequence is most likely, instead it learns (in an essentially unsupervised way) what is the distribution of observations associated to each class (e.g., phoneme). The likelihood of observations is maximized using the Baum-Welsh algorithm, which is an EM algorithm. Dynamic programming techniques may be used to decode the most likely sequence of states. This most likely state sequence can be also used during training (Viterbi algorithm) to approximate the estimation

step of EM.

The architecture proposed in this paper differs from standard HMMs in two respects: computing style and learning. With IOHMMs, sequences are processed similarly to recurrent networks, e.g., an input sequence can be synchronously transformed into an output sequence. This computing style is real-time and predictions of the outputs are available as the input sequence is being processed. This architecture thus allows us to model a transformation from an input sequence space to an output sequence space: in this way, all the fundamental sequence processing tasks such as *production*, *prediction*, and *classification* can be dealt with. Finally, standard HMMs are based on a *homogeneous* Markov chain, whereas in IOHMMs, transition probabilities are conditional on the input and thus depend on time, resulting in an *inhomogeneous* Markov chain. Consequently, the *dynamics* of the system (specified by the transition probabilities) are not fixed but are *adapted* in time depending on the input sequence.

The other fundamental difference is in the learning procedure. While interesting for their capabilities of modeling sequential phenomena, a weakness of standard HMMs is their poor discrimination power when trained by maximum likelihood estimation (MLE) [30]. Consider, for example, the application of HMMs to speech recognition. In the MLE framework, each lexical unit model (corresponding to a word or a phoneme) is trained to fit the distribution of that particular unit. Each model learns from positive examples only, without being informed by the teacher of what classes it will have to compete with. An approach that has been found useful to improve discrimination in HMMs is based on maximum mutual information (MMI) training [38]. When using MMI, the parameters for a given model are adjusted taking into account the likelihoods of *all* the models and not only the likelihood of the model for the correct class (as with the MLE criterion). It has been pointed out that supervised learning in neural networks and discriminant learning criteria like MMI are actually strictly related [35]. Unfortunately, MMI training of standard HMMs can only be done with gradient ascent. On the other hand, for IOHMMs, the parameter adjusting procedure is based on MLE and EM can be used. The variable $\boldsymbol{y}_1^T$ is used as a *desired output* in response to the input $\boldsymbol{u}_1^T$, resulting in more discriminant training.

Furthermore, as discussed in section 5, IOHMMs are better suited for learning to represent long-term context than HMMs (the argument hinges on the fact that IOHMMs are non-homogeneous).

Finally, it is worth mentioning that a number of hybrid approaches have been proposed to integrate connectionist approaches into the HMM framework. For example in [31] the observations used by the HMM are generated by a recurrent neural network. Bourlard et al. [29, 30] use a feedforward network to estimate state probabilities, conditioned on the acoustic sequence. Instead of deriving an exact EM or GEM algorithm, they apply Viterbi decoding in order to estimate the most likely state trajectory, thereafter used as a target sequence for the feedforward network. A common feature of these algorithms and the one proposed in this paper is that neural networks are used to extract temporally local information whereas a Markovian system integrates long-term constraints. Unlike most of these systems, IOHMMs represent a conditional distribution of a (desired) output sequence when an (observed) input sequence is given, rather than being a model of some observation sequence. Furthermore (for some classes of output and transition models), the EM algorithm can still be applied, even though IOHMMs represent a discriminant model (whereas other hybrids of neural networks with HMMs which are discriminant can't be trained with the EM algorithm).

4.2 First and Second Order Recurrent Networks

A first order fully recurrent network with sigmoidal nonlinearities evolves according to the nonlinear iterated map $\boldsymbol{x}_t = f(W\boldsymbol{x}_{t-1} + V\boldsymbol{u}_t)$, where $\boldsymbol{x}_t$ is a continuous state vector and W and V are weight matrices. The dynamics of an IOHMM, instead, are controlled by the recurrence $\boldsymbol{\zeta}_t = \sum_{j=1}^{n} \zeta_{j,t-1}\, \boldsymbol{\varphi}_j(\boldsymbol{u}_t)$, that updates the state distribution $\boldsymbol{\zeta}_t$ given the input sequence $\boldsymbol{u}_1^t$. In the IOHMM case, the dynamics are linear in the state variable (but nonlinear in the inputs), which may result in less general computational capabilities, compared to recurrent networks. In order to gain some intuition about the computational power of IOHMMs, it is useful to consider the limit case of transition probabilities that tend to 0 or 1. Such deterministic behavior (corresponding to equation 1) is obtained when the output units of the state networks are saturated [6]. In this case, each state network j partitions the input space into n regions Ω_{ij} such that a transition from state j at time $t-1$ to state i at time t occurs if and only if $\boldsymbol{u}_t \in \Omega_{ij}$. If multilayered state networks with enough hidden units are used, then, because of the

[6] In fact, the softmax function is used (eq. 2) and $\lim_{\gamma\to\infty} \frac{e^{\gamma a_i}}{\sum_\ell e^{\gamma a_\ell}}$ equals to 1 if $i = \mathrm{argmax}_\ell\, a_\ell$ and 0 otherwise.

universality results [39], the regions Ω_{ij} can be arbitrarily shaped. When the output units of the state networks are not saturated (i.e. transition probabilities are not exactly 0 or 1), we can obtain a similar interpretation, except that the regions Ω_{ij} have soft boundaries.

Because of the multiplicative links, there are some analogies between our architecture and second order recurrent networks that encode discrete states [40]. A second order network with n state units and m inputs evolves according to the equation

$$\boldsymbol{x}_t = f\left(\sum_{j=1}^{n} x_{j,t-1} W_j \boldsymbol{u}_t\right) \tag{38}$$

where $W_j, j = 1, \ldots n$ are n by m matrices of weights. An IOHMM that uses one-layered state subnetworks would evolve, instead, with the linear recurrence

$$\boldsymbol{\zeta}_t = \sum_{j=1}^{n} \zeta_{j,t-1} f\left(W_j \boldsymbol{u}_t\right). \tag{39}$$

Following [40], a second order network can represent discrete states by "one-hot" encoding: $x_{i,t} = 1$ if the state at time t is i, and $x_{i,t} = 0$ otherwise. If these encoding assumption are satisfied (again, this will happen if the state units are saturated), equations 39 and 38 are equivalent. In second order networks encoding discrete states, the previous state selects the weight matrix W_j to be used to predict the next state, given the input. Thus the saturated second order network behaves like a modular architecture, similar to the one we have described, in which distinct subnetworks are activated at time t depending on the discrete state at time $t-1$. A similar interpretation of second order networks, although limited to symbolic inputs, was proposed in [41].

4.3 Adaptive Mixtures of Experts

Adaptive mixtures of experts (ME) [27] and hierarchical mixtures of experts (HME) [42] have been introduced as a *divide and conquer* approach to supervised learning in static connectionist models. A mixture of experts is composed by a modular set of subnetworks (experts) that compete to gain responsibility in modeling outputs in a given region of input space. The system output $\boldsymbol{y}$ is obtained as a convex combination of the experts' outputs $\boldsymbol{y}_j$: $\boldsymbol{y} = \sum_j g_j \boldsymbol{y}_j$ where the weights g_j are computed as a parametric function

of the inputs by a separate subnetwork (*gating network*) that assigns responsibility to different experts for different regions of the input space.

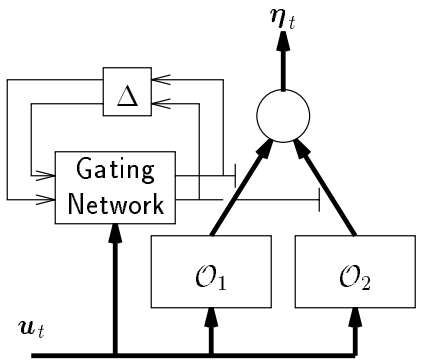


Figure 3: The mixture of controllers (MC) architecture (Cacciatore & Nowlan, 1994).

Recently, Cacciatore & Nowlan [28] have proposed a recurrent extension to the ME architecture, called *mixture of controllers* (MC), in which the gating network has feedback connections, thus taking temporal context into account. The MC architecture is shown in Figure 3. The IOHMM architecture (i.e., Figure 1) can clearly be interpreted as a special case of the MC architecture, in which the gating network has a modular structure and second order connections. In practice, Cacciatore & Nowlan used a one layer first-order gating net, resulting in a model with weaker capacity. However, the more significant difference lies in the organization of processing. In the MC architecture modularity is exploited at the output prediction level, whereas in IOHMMs modularity is exploited at the state prediction level as well. Another potentially important difference lies in the presence of a saturating non-linearity in the recurrence loop of the MC architecture, as in most recurrent networks. Instead, the recurrence loop of IOHMMs is purely linear. It has been shown that such a non-linearity in the loop makes very difficult the learning of long-term context [13] (see next section for a discussion of learning long-term dependencies).

Learning in the MC architecture uses approximated gradient ascent to optimize the likelihood, in contrast to the EM supervised learning algorithm proposed by Jordan & Jacobs (1994) for the HME. The approximation of gradient is based on one step truncated back-propagation through time (somehow similar to Elman's approach [43]) and allows online updating for continually running sequences, which is useful for control tasks. As shown earlier in this paper, the interpretation of the state sequences of IOHMMs as

missing data yields to maximization of the likelihood with an EM or a GEM algorithm.

5 Learning Temporal Dependencies with IOHMMs

Generally speaking, sequential data presents long-term dependencies if the data at a given time t is significantly affected by the past data at times $\tau \ll t$. Accurate modeling of such sequences is typically difficult. In the case of recurrent networks trained by gradient descent, credit assignment through time is represented by a sequence of gradients of the error function with respect to the state of the sigmoidal units. However, many researchers have found this procedure ineffective for assigning credit over long temporal spawns [13, 14, 15]. In the case of IOHMMs trained by the EM algorithm, credit assignment through time is represented by the sequences of posterior (i.e., after having observed the data) probabilities $P(x_{t=}i \mid \boldsymbol{u}_1^t, \boldsymbol{y}_1^t)$. In the following we summarize the main results on the problem of learning long-term dependencies with Markovian models, which include IOHMMs and HMMs. A formal analysis of this problem can be found in [44].

5.1 Temporal Credit Assignment

In previous work [13] we found theoretical reasons for the difficulty in training parametric non-linear dynamical systems to capture long-term dependencies. For such systems, the dynamical evolution is controlled by a non-linear iterated map $\boldsymbol{a}_t = M(\boldsymbol{a}_{t-1}, \boldsymbol{u}_t)$, with $\boldsymbol{a}_t$ a continuous state vector. Systems described by such an equation include most recurrent neural network architectures. The main result states that either long-term storing or gradient propagation is harmed, depending on whether $\|M'\|$, the norm of the Jacobian of the state transition function, is less or greater than one. If $\|M'\| < 1$ then the system is endowed with *robust* dynamical attractors that can be used to reliably store pieces of information for an arbitrary duration. However, gradients will vanish exponentially as they are propagated backward in time. If $\|M'\| > 1$ then gradients do not vanish, but information about past inputs is gradually lost and can be easily deleted by noisy events. For example, if element A of the system can be used to detect particular conjunctions of inputs and state, and element B can latch that information, the problem is that in order to train A one has to back-propagate through B, but gradients through B vanish over long time

periods.

In the present paper we have introduced a connectionist model whose dynamical behavior is controlled by the constrained *linear* recurrence equation $\boldsymbol{\zeta}_t = \boldsymbol{\Phi}(\boldsymbol{u}_t)\boldsymbol{\zeta}_{t-1}$, where $\boldsymbol{\zeta}_t$ is interpreted as a probability distribution defined over a set of discrete states, and $\boldsymbol{\Phi}(\boldsymbol{u}_t)$ corresponds to a matrix of transition probabilities. In this case, the norm of the Jacobian of the state transition function is constrained to be exactly one. Like in recurrent networks, learning in non-deterministic Markovian models generally becomes increasingly difficult as the span of the temporal dependencies increases [44]. However, a very important qualitative difference is that in Markovian models long-term storing and temporal credit assignment are not necessarily incompatible: they either both occur or are both impractical. They both occur in the very special case of an essentially deterministic model. The difficulty increases as the model becomes more non-deterministic and is worst when it is completely ergodic. Eigenvalues of $\boldsymbol{\Phi}(\boldsymbol{u}_t)$ which are less than 1 correspond to a loss of information about initial conditions, a *diffusion* of information through time. Conversely, credit assignment backwards through time is harmed by this phenomenon of diffusion during the backward phase (which is just the transpose of the forward phase). On the contrary, if most of the eigenvalues of $\boldsymbol{\Phi}(\boldsymbol{u}_t)$ are close to one (i.e., the models tend towards a deterministic behavior), then both storing and credit assignment are more effective.

In order to provide some intuitions about the practical difficulty in learning long-term dependencies with Markovian models, let us consider a sequence classification problem with a discrete target variable y for the last step of each training sequence. If the sequences are long and contain relevant classification information at their beginning, then clearly the task exhibits long-term dependencies. Key variables in the temporal credit assignment problem are the probabilities $\beta_{j,t}$ of state j at time t, given the observed target y. As shown in section 3, these variables are computed during the estimation step of EM. They actually correspond to the gradient of the likelihood with respect to the state probabilities $\alpha_{j,t}$, i.e., they indicate how the probabilities associated to each state at each time step should increase in order to increase the total likelihood. We have found that in most cases the $\beta_{j,t}$ tend to become independent of j for t very far away from the final supervision. Clearly, this reflects a situation of maximum uncertainty about the changes required to increase the likelihood. If all $\beta_{j,t}$ are the same for a given t, then all the states at this time

step are equally responsible for the final likelihood: no (small) change of parameters would increase the likelihood. This represents a serious difficulty for propagating backwards in time *effective* temporal credit information, and makes very difficult learning in the presence of long-term dependencies. However, when the transition probabilities are close to 1 or 0, long-term context can be propagated *and* credit assignment through time performed correctly. Such a situation can be found for example in problems of grammar inference in which the input/output data is essentially deterministic (as with the task studied in section 6). An analysis of this problem of credit assignment is presented in [44], in which we study the problem from a theoretical point of view, applying established mathematical results on Markov chains [45] to the problem of learning long term dependencies in homogeneous and non-homogeneous HMMs. Although the analysis is the same for both ordinary HMMs and IOHMMs, there is a very important difference in the simplest cure, which is to have transition probabilities near 0 or 1. An HMM with deterministic (or almost deterministic) transition probabilities is not very useful because it can only model simple cycles. On the other hand an IOHMM can perform a large class of interesting computations (such as grammar inference) with this same constraint, because the transition probabilities can vary at each time step depending on the input sequence. The analyses reported in [44] also suggest that fully connected transition graphs have the worst behavior from the point of view of temporal credit propagation. The transition graph can be constrained using some prior knowledge on the problem. There are two main benefits that can be gained by introducing prior knowledge into an adaptive model: improving generalization to new instances and simplifying learning [46, 47, 48]. Techniques for injecting prior knowledge into recurrent neural networks have been proposed by many researchers [49, 50, 51]. In these cases the domain knowledge is supposed to be available as a collection of transition rules for a finite automaton. A similar approach could be used to choose good topologies of the transition graph in Markovian models (e.g., HMMs or IOHMMs). For example, structured left-to-right HMMs have been introduced in speech recognition with a topology that is based on elementary considerations about speech production and the structure of language [52, 37].

5.2 Reducing Credit Diffusion by Penalized Likelihood

As outlined in [44], the undesired diffusion of temporal credit depends on the fast convergence of the rank of the product of n successive matrices of transition probabilities as n increases. The rate of rank lossage can be reduced by controlling the norm of the eigenvalues of the transition matrices $\boldsymbol{\Phi}_t$. The ideal condition for credit assignment is a 0–1 transition matrix, whose eigenvalues are on the unitary complex circle. Since the determinant of a matrix equals the product of its eigenvalues, a simple way to reduce the diffusion effect is to add a penalty term to the log likelihood, as follows:

$$l(\boldsymbol{\Theta}; \mathcal{D}) + \gamma \sum_{p=1}^{P} \sum_{t=1}^{T_p} |\det \boldsymbol{\Phi}_t| \tag{40}$$

where the constant γ weights the influence of the penalty term. In this case, the maximization step of EM will require gradient ascent (i.e., we need to use a GEM algorithm). The contribution of the penalty term to the gradient can be computed using the relationship

$$\frac{\partial |\det \boldsymbol{\Phi}_t|}{\partial \varphi_{ij,t}} = |\det \boldsymbol{\Phi}_t| \left(\boldsymbol{\Phi}_t^{-1} \right)_{ji} . \tag{41}$$

We have found this "trick" useful for some particularly nasty problems with very long sequences such as the parity problem (see next section).

5.3 Experimental Comparisons with Recurrent Networks

We present here results on two problems for which one can control the span of input/output dependencies: The 2-sequence problem and the Parity problem. These two simple benchmarks were used in [13] to compare the long-term learning capabilities of recurrent networks trained by back-propagation and five other alternative algorithms.

The 2-sequence problem is the following: classify a univariate input sequence, at the end of the sequence, in one of two types, when only the first N elements ($N = 3$ in our experiments) of this sequence carry information about the sequence class. The sequences are constructed artificially by choosing a different random initial pattern for each class. Only the final time step in the output sequence is considered for classifying the input sequence. Uniform noise is added to the input sequence. For the first 6 methods (see Tables 1–4) we used a fully connected recurrent network with 5 units (with 25 free parameters).

For the IOHMM, we used a 7-state system with a sparse connectivity matrix (an initial state, and two separate left-to-right sub-models of three states each to model the two types of sequences, as shown in Figure 4a). No output subnetworks are required in this case and supervision may be expressed in terms of desired final state, as explained in section 3.3.1. The resulting architecture is shown in Figure 4b. Other experiments with a full transition matrix yield much worse results, although improvements were obtained by introducing the penalty term (40).

The parity problem consists in producing the parity of an input sequence of 1's and -1's (i.e., a 1 should be produced at the final output if and only if the number of 1's in the input is odd). The target is only given at the end of the sequence. For the first 6 methods we used a minimal size network (1 input, 1 hidden, 1 output, 7 free parameters). For the IOHMM, we used a 2-state system with a full connectivity matrix. In this task we found that performance could be drastically improved by using stochastic gradient ascent in a way that helps the training algorithm get out of local optima. The learning rate is decreased when the likelihood improves but it is increased when the likelihood remains flat (the system is stuck in a plateau or local optimum).

For both tasks and each method, initial parameters were chosen randomly for each of 20 training trials. Noise added to the input sequence was also uniformly distributed and chosen independently for each training sequence. We considered two criteria: (1) the average classification error at the end of training, i.e., after a stopping criterion has been met (when either some allowed number of sequence presentations has been performed or the task has been learned), (2) the average number of function evaluations needed to reach the stopping criterion.

In the tables, "p-n" stands for pseudo-Newton [53]. Time-weighted pseudo-newton is a variation in which derivatives with respect to the instantiation of a parameter at a particular time step are weighted by the inverse of the corresponding second derivatives [13]. Multigrid is similar to simulated annealing with constant temperature 0. The discrete error propagation algorithm [13] attempts to propagate backwards discrete error information in a recurrent network with discrete units. Each column of the tables corresponds to a value of the maximum sequence length T for a given set of trials. The sequence length for a particular training sequence was picked randomly within $T/2$ and T. Numbers reported are averages over 20 or more

Table 1: 2-sequences problem: Final classification error with respect to the maximum sequence length.

	5	10	20	50	100
back-prop	58	56	43	53	50
pseudo-newton (p-n)	2	3	10	25	29
time-weighted p-n	0	0	9	34	14
multigrid	2	6	1	3	6
discrete error prop.	6	16	29	23	22
simulated annealing	6	0	7	4	11
IOHMMs	0	0	0	0	0

Table 2: 2-sequences problem: # sequence presentations with respect to the maximum sequence length.

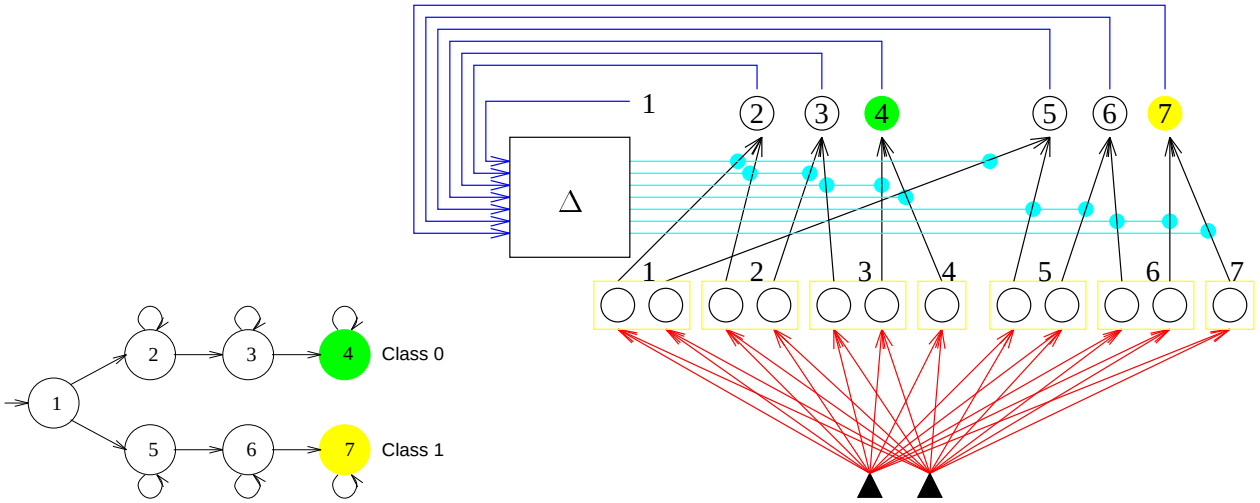
	5	10	20	50	100
back-prop	2.9e3	3.0e3	2.9e3	3.0e3	2.8e3
pseudo-newton (p-n)	5.1e2	1.1e3	1.9e3	2.6e3	2.5e3
time-weighted p-n	5.4e2	4.3e2	2.4e3	2.9e3	2.7e3
multigrid	4.1e3	5.8e3	2.5e3	3.9e3	6.4e3
discrete error prop.	6.6e2	1.3e3	2.1e3	2.1e3	2.1e3
simulated annealing	2.0e5	3.9e4	8.2e4	7.7e4	4.3e4
IOHMMs	3.2e3	4.0e3	2.9e3	3.2e3	2.9e3

Table 3: Parity problem: Final classification error with respect to the maximum sequence length.

	3	5	10	20	50	100	500
back-prop	2	20	41	38	43		
pseudo-newton (p-n)	3	25	41	44	40	47	
time-weighted p-n	26		39	43	44		
multigrid	15			44	45		
discrete error prop.	0		0	0	5		
simulated annealing	3			10	0		
IOHMMs		0	6	0	14	0	12

Table 4: Parity problem: # sequence presentations with respect to the maximum sequence length.

	3	5	9	20	50	100	500
back-prop	3.6e3	5.5e3	8.7e3	1.6e4	1.1e4		
pseudo-newton (p-n)	2.5e2	8.9e3	8.9e3	7.7e4	1.1e4	1.1e5	
time-weighted p-n	4.5e4		7.0e4	3.4e4	8.1e4		
multigrid	4.2e3			1.5e4	3.1e4		
discrete error prop.	5.0e3		7.9e3	1.5e4	5.4e4		
simulated annealing	5.1e5			1.2e6	8.1e5		
IOHMMs		2.3e3	1.5e3	1.3e3	3.2e3	2.6e3	3.4e3



(a) (b)

Figure 4: (a): Transition graph used in the two-sequences problem. (b): the corresponding recurrent architecture, which is not fully connected.

trials. The results in Tables 1–4 clearly show that IOHMMs can achieve better performance than those obtained with the other algorithms, except possibly for the discrete error propagation algorithm.

6 Regular Grammar Inference

In this section we describe an application of our architecture to the problem of 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, i.e., that can classify a new sequence of symbols as part of the language, or not part of the 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 [54]. Many researchers [55, 56, 57] have approached grammatical inference with recurrent networks. These studies demonstrate that second-order neural 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

Table 5: Definitions of the seven Tomita grammars

Grammar	Definition
1	$1^\star$
2	$(10)^\star$
3	string does not contain $1^{2n+1}0^{2m+1}$ as a substring
4	string does not contain 000 as a substring
5	string contains an even number of 01's and 10's
6	number of 0's - number of 1's is a multiple of 3
7	$0^\star1^\star0^\star1^\star$

automata rules from the trained network [55, 57]. 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 sub-symbolic 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 [58] or by integrating clustering techniques in the training procedure [59].

We report experimental results on the application of IOHMMs to a set of regular grammars introduced by Tomita [60] and afterwards used by other researchers as a benchmark to measure the accuracy of inference methods based on recurrent networks [59, 55, 58, 56, 57]. The grammars use the binary alphabet $\{0, 1\}$ and are reported in Table 5. 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. $P(y_T) = \eta_T^{y_T}(1-\eta_T)^{1-y_T}$, where η_T is the system final (expected) output and the target output $y_T = 0$ 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 section 5.3 — since there can be more than one accepting or rejecting

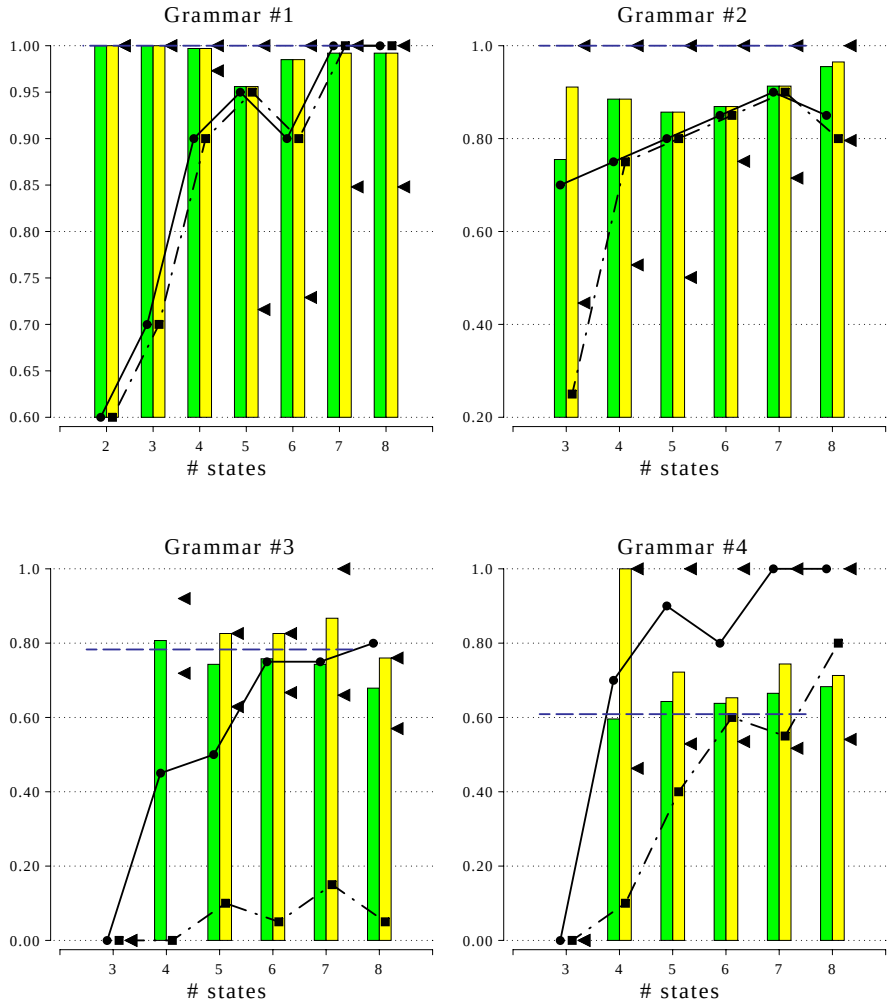


Figure 5: Convergence and generalization attained by varying the number of discrete states n in the model. Results are averaged over 20 trials. ■ Frequency of convergence to $e = 0$ classification errors on the training set. • Frequency of convergence to $e < 3$ classification errors on the training set. The two gray levels of the vertical bars show the corresponding accuracies on the test data. The triangles ($\triangleleft$) denote the generalization accuracy for the best and the worst trial. The horizontal dashed line represents the best result reported by Watrous & Kuhn. (continued ...).

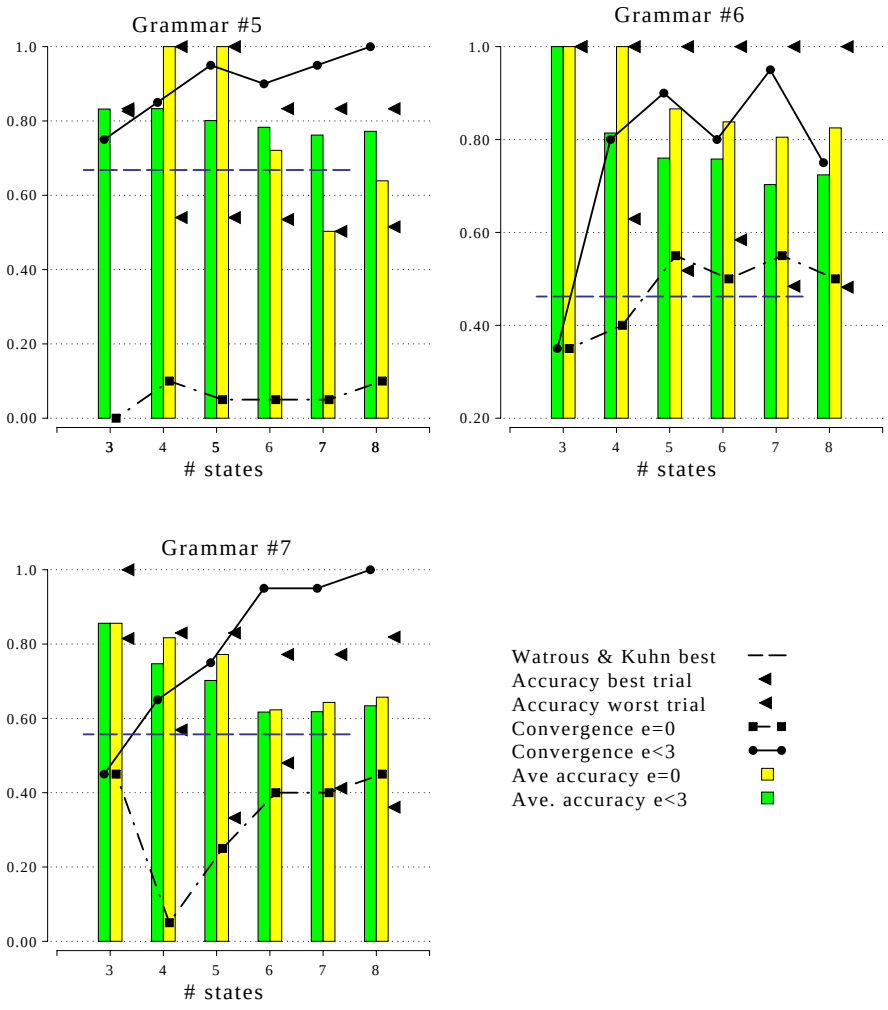


Figure 5: (... continuation).

Table 6: Summary of experimental results on the seven Tomita's grammars (see text for explanation).

Grammar	Sizes		Frequency of	Accuracies			
	$n^\star$	FSA min	Convergence	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

state. In [55] this problem is circumvented by appending a special "end" symbol to each string. However, in our case this would increase the number of parameters.

The task of accepting strings can be solved by a Moore finite state machine [61], in which the output is function of the state only (i.e., strings are accepted or rejected depending on what final state is reached). Hence, we did not apply external inputs to the output networks, that reduced to one unit fed by a bias input. In this way, each output network computes a constant function of the last state reached by the model. The system output is a combination of these, weighted by $\boldsymbol{\zeta}_t$, the state distribution at time t.

Given the absence of prior knowledge about plausible state paths, we used an *ergodic* transition graph in which all transitions are allowed. 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 sizes for the recurrent architecture. 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. Interestingly, except for grammars 2 and 3, the same $n^\star$ would have been obtained by choosing the smallest model successfully trained to correctly classify the learning set, as shown in Figure 6. This figure shows the generalization accuracy (triangles) and the frequency of convergence to zero errors on the training set (squares), for each of the grammars, with a comparison to the best result of 5 trials obtained by Watrous & Kuhn [57] with a second-order recurrent network (dashed horizontal line) on the same data. We see that most of the IOHMM trials performed better than the best of 5 recurrent network trials and that the best IOHMM trial always generalized perfectly (unlike the best recurrent network). For comparison, in Table 6 we also report for each grammar the number of states of the minimal recognizing FSA [60].

We tested the trained IOHMMs on a corpus of $2^{13} - 1$ binary strings of length $T \leq 12$. The final results are numerically summarized in Table 6. 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 training sets proposed by Tomita. For comparison, in the last column of Table 6 we reproduce the results reported by Watrous & Kuhn [57] in the best of five trials. Other researchers also obtained interesting results, although they are not directly comparable because of the use of larger[7] training sets [59, 55] or different experimental conditions [58]. 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 of 0 or 1 would require to develop infinite weights because of the softmax function). Of course this is consistent with the deterministic nature of the problem. It is however interesting to note that, apart from numerical precision problems, these trained models actually behave like finite automata, rendering trivial the extraction of the corresponding deterministic automaton. Indeed, for grammars 1,4,5, and 6, we found that the trained IOHMMs behave exactly like the minimal

[7] We used the training sets defined by Tomita [60].

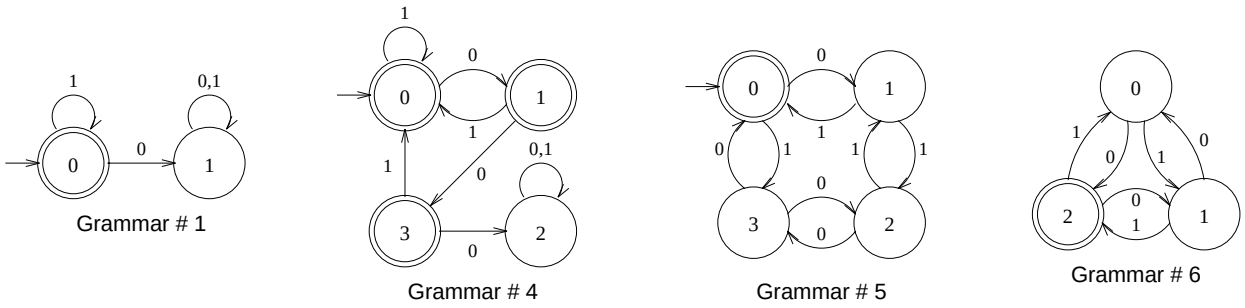


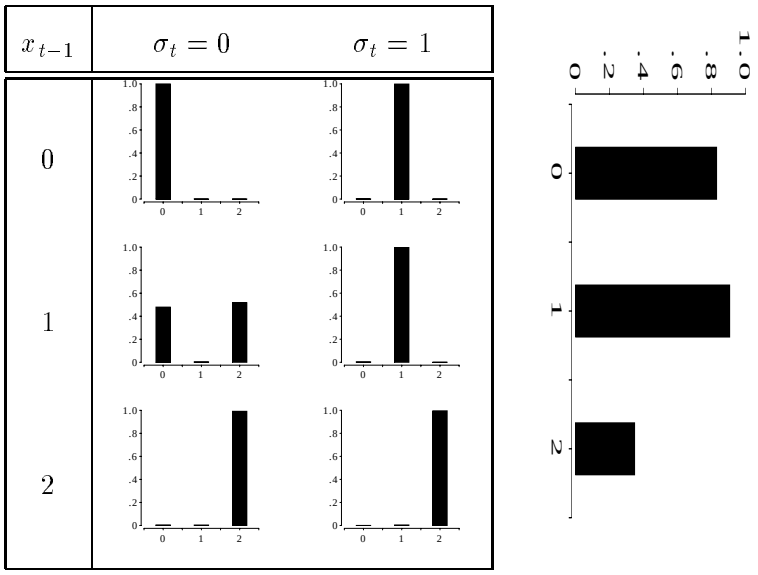
Figure 6: Finite automata equivalent to the IOHMMs trained on Tomita's grammars 1,4,5, and 6.

recognizing FSA (see Figure 6).

In some cases, however, the IOHMM 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 7(a), and the output probabilities in Figure 7(b). 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 IOHMM still made no errors.

A potential training problem is the presence of local maxima in the likelihood function. For example, the fraction of converged trials for grammars 3, 4, and 5 is 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 [18] 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.

To conclude, we have found IOHMMs to perform well on a task of grammar inference in comparison to recurrent networks. Furthermore, we have found that they can sometime find solutions involving less states than the minimum required if the system was completely deterministic.



(a) (b)

Figure 7: Learned transition probabilities for grammar # 7: (a): Transition probabilities; bar chart on row j and column k represents the discrete distribution $\mathrm{P}(x_t \,|\, x_{t-1} = j, u_t = k)$. (b): Probabilities of accepting the input string, $P(y_T = 1 \,|\, x_T)$. This network correctly classifies all the test strings.

7 Conclusions

We have presented a recurrent architecture suitable for modeling an input/output relationship in discrete state dynamical systems. The architecture has a probabilistic interpretation, called Input/Output Hidden Markov Model (IOHMM), and can be trained by an EM or GEM algorithm, using the state paths as missing data. It can be seen both as an extension of standard Hidden Markov Models (HMMs), with a conditioning input sequence, and as an extension of the mixture of experts (ME) model [27], with a constrained linear feedback loop and two sets of experts (for predicting the output and for predicting the next state).

On two test problems, in which the span of the temporal dependencies can be controlled, we have found that IOHMMs learn long-term dependencies more effectively than back-propagation and other alternative algorithms described in [13, 18]. An analysis of the problem of credit assignment through time in HMMs and IOHMMs [44] explains why they could solve this problem better than recurrent networks (which

have a non-linearity in the recurrence loop), and revealed that best results would be obtained when transition probabilities are near 0 or 1. Although this corresponds to uninteresting models in the case of HMMs, it corresponds to a large class of useful models in the case of IOHMMs, because the latter are non-homogeneous (transition probabilities change during the sequence). Furthermore, when HMMs are trained with the (more efficient than gradient ascent) EM algorithm they are trained in a basically non-discriminant way, whereas IOHMMs can be trained with the EM algorithm while using a discriminant training criterion.

The results obtained in recent experiments suggest that IOHMMs are appropriate for solving grammatical inference problems. In particular, for the benchmark problem proposed by Tomita [60], IOHMMs compare favorably to second order nets trained by gradient descent, in terms of generalization performance.

Future work will have to address extensions of this model in several directions. How well does the algorithm perform on larger scale tasks with a large state space? How can we use dynamic programming in applications such as speech recognition where we wish to assign a certain meaning to particular states or transitions? How well does the algorithm work on tasks of sequence prediction (e.g. multivariate time-series) and sequence production (e.g. control and robotics tasks)? Are there other ways to improve credit assignment through time when the data to be modeled is very non-deterministic? We are exploring a solution based on a hierarchical representation of the state. This can be achieved by introducing several sub-state variables whose Cartesian product corresponds to the system state. Each of these sub-state variables can operate at a different time scale, thus allowing credit to propagate over long temporal spans for some of these variables. A different option for simplifying the sequential learning task, is to exploit some form of prior knowledge, which can be used, not only to reduce the number of free parameters, but also, to reduce the difficulty in capturing long-term dependencies.

References

[1] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 328–339, 1989.

[2] D. Seidl and D. Lorenz, "A structure by which a recurrent neural network can approximate a nonlinear dynamic

system," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 709–714, July 1991.

[3] E. Sontag, "Systems combining linearity and saturations and relations to neural networks," Tech. Rep. SYCON–92–01, Rutgers Center for Systems and Control, 1992.

[4] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing* (D. Rumelhart and J. McClelland, eds.), vol. 1, ch. 8, pp. 318–362, Cambridge: MIT Press, 1986.

[5] B. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, vol. 1, pp. 263–269, 1989.

[6] G. Kuhn, "A first look at phonetic discrimination using connectionist models with recurrent links." CCRP – IDA SCIMP working paper No.4/87, Institute for Defense Analysis, Princeton, NJ, 1987.

[7] A. Robinson and F. Fallside, "Static and dynamic error propagation networks with application to speech coding," in *Neural Information Processing Systems* (D. Anderson, ed.), (Denver, CO), pp. 632–641, American Institute of Physics, New York, 1988.

[8] R. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

[9] M. Gori, Y. Bengio, and R. De Mori, "BPS: A learning algorithm for capturing the dynamical nature of speech," in *Proceedings of the International Joint Conference on Neural Networks*, (Washington D.C.), pp. 643–644, IEEE, New York, 1989.

[10] M. Mozer, "A focused back-propagation algorithm for temporal pattern recognition," *Complex Systems*, vol. 3, pp. 349–381, 1989.

[11] P. Frasconi, M. Gori, and G. Soda, "Local feedback multi-layered networks," *Neural Computation*, vol. 4, no. 1, pp. 120–130, 1992.

[12] A. Tsoi and A. Back, "Locally recurrent globally feedforward networks, a critical review of architectures," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 229—239, 1994.

[13] 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.

[14] M. C. Mozer, "The induction of multiscale temporal structure," in *Advances in Neural Information Processing Systems 4* (J. Moody, S. Hanson, and R. Lipmann, eds.), (San Mateo, CA), pp. 275–282, Morgan Kaufmann, 1992.

[15] R. Rohwer, "The time dimension of neural network models," *ACM Sigart Bulleting*, vol. 5, pp. 36–44, July 1994.

[16] Y. Bengio, P. Frasconi, and P. Simard, "The problem of learning long-term dependencies in recurrent networks," in *IEEE International Conference on Neural Networks*, (San Francisco), pp. 1183–1195, IEEE Press, 1993. (invited paper).

[17] R. Rohwer, "The "moving targets" training algorithm," in *Advances in Neural Information Processing Systems 2* (D. Touretzky, ed.), (Denver, CO), pp. 558–565, Morgan Kaufmann, San Mateo, 1990.

[18] Y. Bengio and P. Frasconi, "Credit assignment through time: Alternatives to backpropagation," in *Advances in Neural Information Processing Systems 6* (J. Cowan, G. Tesauro, and J. Alspector, eds.), Morgan Kaufmann, 1994.

[19] 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.

[20] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains," *Ann. Math. Statistic.*, vol. 41, pp. 164–171, 1970.

[21] S. Levinson, L. Rabiner, and M. Sondhi, "An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition," *Bell System Technical Journal*, vol. 64, no. 4, pp. 1035–1074, 1983.

[22] A. Poritz, "Hidden Markov models: a guided tour," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 7–13, 1988.

[23] A. Kehagias, "Stochastic recurrent networks: Prediction and classification of time series," tech. rep., Brown University. Division of Applied Mathematics, Providence, RI 02912, 1991.

[24] H. Leprieur and P. Haffner, "Discriminant learning with minimum memory loss for improved non-vocabulary rejection," in *EUROSPEECH'95*, (Madrid, Spain), 1995.

[25] Y. Bengio, Y. LeCun, and D. Henderson, "Globally trained handwritten word recognizer using spatial representation, space displacement neural networks and hidden Markov models," in *Advances in Neural Information Processing Systems 6* (J. Cowan, G. Tesauro, and J. Alspector, eds.), pp. 937–944, 1994.

[26] G. J. McLachlan and K. E. Basford, *Mixture models: Inference and applications to clustering.* Marcel Dekker, 1988.

[27] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixture of local experts," *Neural Computation*, vol. 3, pp. 79–87, 1991.

[28] T. W. Cacciatore and S. J. Nowlan, "Mixtures of controllers for jump linear and non-linear plants," in *Advances in Neural Information Processing Systems 6* (J. Cowan, G. Tesauro, and J. Alspector, eds.), (San Mateo, CA), Morgan Kaufmann, 1994.

[29] H. Bourlard and C. Wellekens, "Links between hidden Markov models and multilayer perceptrons," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 1167–1178, 1990.

[30] H. Bourlard and N. Morgan, *Connectionist Speech Recognition. A Hybrid Approach*, vol. 247 of *The Kluwer international series in engineering and computer science.* Boston: Kluwer Academic Publishers, 1993.

[31] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Global optimization of a neural network-hidden Markov model hybrid," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 252–259, 1992.

[32] E. Levin, "Word recognition using hidden control neural architecture," in *International Conference on Acoustics, Speech and Signal Processing*, (Albuquerque, NM), pp. 433–436, 1990.

[33] J. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neuro-computing: Algorithms, Architectures, and Applications* (F. Fogelman-Soulie and J. Hérault, eds.), New York: Springer-Verlag, 1989.

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

[35] J. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," in *Advances in Neural Information Processing Systems 2* (D. Touretzky, ed.), pp. 211–217, Morgan Kaufmann, 1990.

[36] P. Baldi and Y. Chauvin, "Smooth on-line learning algorithms for hidden Markov models," *Neural Computation*, vol. 6, no. 4, pp. 307–318, 1994.

[37] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[38] P. Brown, *The Acoustic-Modeling problem in Automatic Speech Recognition*. PhD thesis, Dept. of Computer Science, Carnegie-Mellon University, 1987.

[39] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[40] C. L. Giles and C. W. Omlin, "Inserting rules into recurrent neural networks," in *Neural Networks for Signal Processing II, Proceedings of the 1992 IEEE workshop* (Kung, Fallside, Sorenson, and Kamm, eds.), pp. 13–22, IEEE Press, 1992.

[41] G. Z. Sun, H. H. Chen, Y. C. Lee, and C. L. Giles, "Recurrent neural networks, hidden Markov models and stochastic grammars," in *Proc. Int. Joint Conference on Neural Networks*, vol. I, (San Diego CA), pp. 729–734, 1990.

[42] M. I. Jordan and R. A. Jacobs, "Hierarchies of adaptive experts," in *Advances in Neural Information Processing Systems 4* (J. Moody, S. Hanson, and R. Lipmann, eds.), (San Mateo, CA), pp. 985–992, Morgan Kaufmann, 1992.

[43] J. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, 1990.

[44] Y. Bengio and P. Frasconi, "Diffusion of credit in markovian models," in *Advances in Neural Information Processing Systems 7* (G. Tesauro, D. S. Touretzky, and J. Alspector, eds.), (San Mateo, CA), Morgan Kaufmann, 1995.

[45] E. Seneta, *Nonnegative Matrices and Markov Chains*. New York: Springer, 1981.

[46] Y. S. Abu-Mostafa, "Learning from hints in neural networks," *Journal of Complexity*, vol. 6, pp. 192–198, 1990.

[47] G. G. Towell, J. W. Shavlik, and M. O. Noordewier, "Refinement of approximate domain theories by knowledge-based neural networks," in *Proceedings of the Eighth National Conference on Artificial Intelligence*, (Boston, MA), pp. 861–866, 1990.

[48] V. Tresp, J. Hollatz, and S. Ahmad, "Network structuring and training using rule-based knowledge," in *Advances in Neural Information Processing Systems 5* (S. J. Hanson, J. D. Cowan, and C. L. Giles, eds.), San Mateo, CA: Morgan Kaufman Publishers, 1993.

[49] C. W. Omlin and C. L. Giles, "Training second-order recurrent neural networks using hints," in *Machine Learning: Proc. of the Ninth Int. Conference* (D. Sleeman and P. Edwards, eds.), (San Mateo CA), Morgan Kaufmann, 1992.

[50] R. Maclin and J. W. Shawlik, "Refining domain theories expressed as finite-state automata," in *Machine Learning: Proceedings of the Eighth International Workshop* (L. Birnbaum and G. Collins, eds.), (San Mateo CA), Morgan Kaufmann, 1991.

[51] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Unified integration of explicit rules and learning by example in recurrent networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 2, pp. 340–346, 1995. (in press).

[52] R. Bakis, "Continuous speech recognition via centisecond acoustic states," in *19st Meeting of the Acoustic Society of America*, April 1976.

[53] S. Becker and Y. LeCun, "Improving the convergence of back-propagation learning with second order methods," in *Proceedings of the 1988 Connectionist Models Summer School* (D. Touretzky, G. Hinton, and T. Sejnowski, eds.), (Pittsburg 1988), pp. 29–37, Morgan Kaufmann, San Mateo, 1989.

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

[55] 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.

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

[57] 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.

[58] 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).

[59] 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, G. Tesauro, and J. Alspector, eds.), Morgan Kaufmann, 1994.

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

[61] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley Publishing Company, Inc., 1979.