LETTER Communicated by C. Lee Giles

Stable Encoding of Finite-State Machines in Discrete-Time Recurrent Neural Nets with Sigmoid Units

Rafael C. Carrasco
Mikel L. Forcada
M. Ángeles Valdés-Muñoz
Departament de Llenguatges i Sistemes Informàtics, Universitat d'Alacant, E-03071 Alacant, Spain

Ramón P. Ñeco
Departament de Ciències Experimentals i Tecnologia, Universitat Miguel Hernàndez, E-03202 Elx, Spain

There has been a lot of interest in the use of discrete-time recurrent neural nets (DTRNN) to learn finite-state tasks, with interesting results regarding the induction of simple finite-state machines from input–output strings. Parallel work has studied the computational power of DTRNN in connection with finite-state computation. This article describes a simple strategy to devise stable encodings of finite-state machines in computationally capable discrete-time recurrent neural architectures with sigmoid units and gives a detailed presentation on how this strategy may be applied to encode a general class of finite-state machines in a variety of commonly used first- and second-order recurrent neural networks. Unlike previous work that either imposed some restrictions to state values or used a detailed analysis based on fixed-point attractors, our approach applies to any positive, bounded, strictly growing, continuous activation function and uses simple bounding criteria based on a study of the conditions under which a proposed encoding scheme guarantees that the DTRNN is actually behaving as a finite-state machine.

1 Introduction

The relationship between discrete-time recurrent neural networks (DTRNN) and finite-state machines (FSM) has been explored in a number of different ways by many researchers in the past 10 years, although this relationship has earlier roots (McCulloch & Pitts, 1943; Kleene, 1956; Minsky, 1967). All of these early papers refer to neural networks made up of threshold units (with steplike activation functions). More recently, Alon, Dewdney, and Ott (1991), Indyk (1995), and Horne and Hush (1996) have studied in more detail bounds on the number of threshold units necessary to implement an FSM of a given number of states. Also, in a recent work, Kremer (1995) has

shown that Elman's (1990) simple recurrent nets (a class of DTRNN) using threshold units can actually represent any deterministic finite automaton (a class of FSM).

The interest in the relationship between FSM and DTRNN is partly motivated by the fact that one can view DTRNN as state machines: a new state for the hidden units is computed from the previous state, and the currently available input in each cycle, and possibly an output, is computed in each cycle too. One can say that DTRNN using activation functions other than thresholds are state machines that are not restricted to be finite. The architectural similarities between FSM and DTRNN will be discussed in detail in the next section of this article.

Under this intuitive assumption that being nonfinite state machines themselves, DTRNN can emulate FSMs, a number of researchers set out to test whether DTRNN with real-valued, continuous sigmoid activation functions could learn FSM behavior from samples (Cleeremans, Servan-Schreiber, & McClelland, 1989; Pollack, 1991; Giles et al., 1992; Watrous & Kuhn, 1992; Maskara & Noetzel, 1992; Sanfeliu & Alquézar, 1994; Manolios & Fanelli, 1994; Forcada & Carrasco, 1995; Tiňo & Sajda, 1995; Ñeco & Forcada, 1996; Gori, Maggini, Martinelli, & Soda, 1998); and even compared the performance of different architectures (Miller & Giles, 1993; Horne & Giles, 1995). The use of sigmoid functions is motivated by the need to have error functions that are differentiable with respect to the weights so that gradient-descent algorithms may be used for learning.

The results of these works show that DTRNN indeed may learn finite-state-like behavior from samples of that behavior, but some problems persist. First, after learning, finite-state-like behavior is observed for short input strings (with lengths in the range of that of strings used for training); however, for longer input strings, the clusters observed for the values of the hidden state vector (usually interpreted as the states of the FSM learned) start to blur and finally merge, leading to incorrect state representations and, accordingly, incorrect output. This behavior is often referred to as *instability*, and hampers the ability of trained DTRNN to generalize the learned behavior (see Tiňo, Horne, Giles, & Colingwood, 1998, for a theoretical analysis of generalization loss). As a partial solution, some authors force learning to occur in such a way that state values form clusters (Das & Das, 1991; Zeng, Goodman, & Smythe, 1993, 1994; Das & Mozer, 1998). Second, when the FSM to be learned has long-term dependencies, that is, outputs that depend on inputs that have been presented very early during the processing of a string, gradient-descent algorithms suffer from the problem of vanishing gradients, which makes it difficult to relate late contributions to the error to small changes in the state of neurons in early stages of string processing (Bengio, Simard, & Frasconi, 1994).

Once the DTRNN has been trained, researchers use specialized algorithms to extract FSM from the dynamics of the DTRNN; some use a straightforward equipartition of neural state-space followed by a branch-and-bound

algorithm (Giles et al., 1992) or a clustering algorithm (Cleeremans et al., 1989; Manolios & Fanelli, 1994; Gori et al., 1998). Very often the finite-state automaton extracted behaves correctly, and then does so, obviously, for strings of any length. However automaton extraction algorithms have been criticized (Kolen & Pollack, 1995; Kolen, 1994) in the sense that the extraction of FSM may not reflect the computation actually being performed by the DTRNN. More recently, Casey (1996) has shown that DTRNN can indeed "organize their state space to mimic the states in the ... state machine that can perform the computation" and be trained or programmed to behave as FSM. Arai and Nakano (1996) stated that when in a DTRNN, the gain of the sigmoid function reaches a certain finite value, the DTRNN behaves as a stable FSM[1] and use this result to formulate a training method (Arai & Nakano, 1996, 1997); however, they fail to provide a rigorous proof and resort to an intuitive explanation. Also recently, Blair and Pollack (1997) showed that an increasing-precision dynamical analysis may identify, in the limit, those DTRNNs that have actually learned to behave like FSM.

Finally, some researchers have set out to define ways to program a sigmoid-based DTRNN so that it behaves like a given FSM, that is, sets of rules for choosing the weights and initial states of the DTRNN based on the transition function and the output function of the corresponding FSM.

Omlin and Giles (1996a, 1996b) have proposed an algorithm for encoding deterministic finite-state automata (a class of FSM) in second-order recurrent neural networks such as the ones used by Giles et al. (1992). The encoding is based on a study of the fixed points of the logistic sigmoid function. There are many similarities between their work and the one presented here: the use of worst-case studies; the definition of low, high, and forbidden state values; a similar scheme for choosing weights; and so on. One of the main differences is the level on which the conditions for stable encoding are expressed. We do not explicitly require our states to be structured around fixed points of the activation function. Finally, weight schemes, although very similar, are not completely identical.

Alquézar and Sanfeliu (1995) have shown that deterministic finite-state automata may be encoded in Elman (1990) nets provided that sigmoids taking and returning rational numbers are used (general real-valued sigmoids are not allowed).[2] Their construction will be generalized to real-valued sigmoids and a larger class of FSM in this article. Their proof relies on converting the deterministic finite-state automaton (DFA) into a new DFA in which all states are split in as many states as there are symbols in the input alphabet of the DFA (as Minsky, 1967, did).

[1] An infinite value of the gain would correspond to the use of a step function; the result in this case is well known (Minsky, 1967).

[2] Kremer's (1995) work used Elman nets with threshold functions, not sigmoids.

Kremer (1996) has recently shown that a single-layer first-order recurrent neural network with real sigmoid squashing functions returning values in [0, 1] can represent the state transition function of any finite-state automaton, provided that the states are split as in the previous work. In contrast to Alquézar and Sanfeliu's (1995) and Omlin and Giles's (1996a, 1996b) work and to the approach used in this article, Kremer's construction uses different weights for each state neuron.

Frasconi, Gori, and Maggini (1996) have shown how finite-state automata may be encoded in recurrent networks using radial basis functions instead of sigmoids.

Recently, Šíma (1997) has shown that the behavior of any DTRNN using threshold activation functions may be stably emulated by another DTRNN using activation functions in a very general class that includes the sigmoid functions considered in this article. Šíma provides constructive proof that contains a prescription to compute the new values of weights. In a more recent paper, Šíma and Wiedermann (1998) show that any regular expression of length l may be recognized by a DTRNN having $\Theta(l)$ threshold units.[3] Combining both results, one obtains a prescription to encode any regular expression into a sigmoid DTRNN so that it recognizes the corresponding language. As will be discussed in more detail in section 8, Šíma's (1997) result may also be combined with existing results on the simulation of FSM (Alon et al., 1991; Indyk, 1995; Horne & Hush, 1996) in threshold DTRNN to extend them to analog DTRNN. This article gives a detailed description on how to directly encode FSMs—including regular language recognizers—into a variety of commonly used first- and second-order sigmoid recurrent neural networks so that the weights for stable simulation are relatively small. Small weights are of interest if the method is used to inject partial a priori knowledge into the DTRNN before training it through gradient descent, where activation function saturation would be a serious problem.

This article aims at expanding the current results on stable encoding of FSM on DTRNN to a larger family of sigmoids, a larger variety of DTRNN, and a wider class of FSM architectures by establishing a simplified procedure to prove the stability of a devised encoding scheme. Some of the results presented have already been used by Kremer, Ñeco, and Forcada (1998) to constrain the training of DTRNN so that they assume an FSM-like behavior.

Section 2 defines the classes of FSM and DTRNN that will be studied and establishes architectural parallelisms between them; section 3 describes the conditions under which a DTRNN behaves as an FSM and defines a general encoding scheme based on these conditions and a set of sufficient conditions for the validity of the encoding; sections 4, 5, and 6 describe encoding schemes for Mealy FSM, Moore FSM, and DFA; section 7 shows experimental results that support the proposed encodings and illustrate the sufficiency

[3] We thank an anonymous referee for calling our attention to this work.

of the conditions defining each particular encoding; finally, concluding remarks and a table summarizing the encoding schemes (Table 6) may be found in section 8.

2 Definitions

2.1 Mealy and Moore Machines and Deterministic Finite Automata. Mealy machines (Hopcroft & Ullman, 1979) are finite-state machines that act as transducers or translators, taking a string on an input alphabet and producing a string of equal length on an output alphabet. Formally, a Mealy machine is a six-tuple

$$M = (Q, \Sigma, \Gamma, \delta, \lambda, q_I) \tag{2.1}$$

where

- $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$ is a finite set of states.
- $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\Sigma|}\}$ is a finite input alphabet.
- $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_{|\Gamma|}\}$ is a finite output alphabet.
- δ: $Q \times \Sigma \rightarrow Q$ is the next-state function, such that a machine in state q_j, after reading symbol σ_k, moves to state $\delta(q_j, \sigma_k) \in Q$.
- λ: $Q \times \Sigma \rightarrow \Gamma$ is the output function, such that a machine in state q_j, after reading symbol σ_k, writes symbol $\lambda(q_j, \sigma_k) \in \Gamma$.
- $q_I \in Q$ is the initial state in which the machine is found before the first symbol of the input string is processed.

A *Moore machine* (Hopcroft & Ullman, 1979) may be defined by a similar six-tuple, with the only difference that symbols are output after the transition to a new state is completed instead of during the transition itself, and the output symbol depends on only the state just reached, that is, λ: $Q \rightarrow \Gamma$. The classes of translations that may be performed by Mealy machines and Moore machines are identical. Indeed, given a Mealy machine, it is straightforward to construct the equivalent Moore machine, and vice versa (Hopcroft & Ullman, 1979).

Deterministic finite automata (DFA) (Hopcroft & Ullman, 1979) may be seen as a special case of Moore machines. A DFA is a five-tuple

$$M = (Q, \Sigma, \delta, q_I, F) \tag{2.2}$$

where Q, Σ, δ, and q_I have the same meaning as in Mealy and Moore machines and $F \subseteq Q$ is the set of accepting states. If the state reached by the DFA after reading a complete string in Σ^* (the set of all finite-length strings over Σ, including the empty string λ) is in F, then the string is accepted; if not, the string is not accepted (or it is rejected). This would be equivalent

to having a Moore machine whose output alphabet has only two symbols, Y (yes) and N (no), and looking only at the last output symbol (not at the whole output string) to decide whether the string is accepted or rejected.

2.2 Discrete-Time Recurrent Neural Networks. Discrete-time recurrent neural networks (DTRNN) (see Haykin, 1998; Hertz, Krogh, & Palmer, 1991; Hush & Horne, 1993; Tsoi & Back, 1997), may be viewed as neural state machines (NSM) and defined in a way that is parallel to the above definitions of Mealy and Moore machines.[4] A neural state machine N is a six-tuple

$$N = (X, U, Y, \mathbf{f}, \mathbf{h}, \mathbf{x}_0) \tag{2.3}$$

in which

- $X = [S_0, S_1]^{n_X}$ is the state-space of the NSM, with S_0 and S_1 the values defining the range of outputs of the activation functions and n_X the number of state units.
- $U = \mathcal{R}^{n_U}$ defines the set of possible input vectors, with $\mathcal{R}$ the set of real numbers and n_U the number of input lines.
- $Y = [S_0, S_1]^{n_Y}$ is the set of outputs of the NSM, with n_Y the number of output units (it is assumed that the activation function of output and state units is the same).
- $\mathbf{f}$: $X \times U \to X$ is the next-state function, a feedforward neural network that computes a new state $\mathbf{x}[t]$ from the previous state $\mathbf{x}[t-1]$ and the input just read $\mathbf{u}[t]$.
- $\mathbf{h}$ is the output function, which in the case of a Mealy NSM is $\mathbf{h}$: $X \times U \to Y$, that is, a feedforward neural network that computes a new output $\mathbf{y}[t]$ from the previous state $\mathbf{x}[t-1]$ and the input just read $\mathbf{u}[t]$, and in the case of a Moore NSM is $\mathbf{h}$: $X \to Y$, a feedforward neural network that computes a new output $\mathbf{y}[t]$ from the newly reached state $\mathbf{x}[t]$.
- $\mathbf{x}_0$ is the initial state of the NSM, that is, the value that will be used for $\mathbf{x}[0]$.

Most classical DTRNN architectures may be directly defined using the NSM scheme; the following section shows some examples (in all of them, weights and biases are assumed to be real numbers). In all of the following it will be assumed that $\mathbf{f}$ and $\mathbf{h}$ are defined for all possible values of their arguments. This is consistent with their implementation as feedforward neural networks.

[4] This parallelism is inspired in the relationship established by Pollack (1991) between DFA and a class of second-order DTRNN, under the name of *dynamical recognizers*.

2.2.1 Neural Mealy Machines. A commonly used second-order recurrent neural network (Giles et al, 1992; Watrous & Kuhn, 1992; Pollack, 1991; Forcada & Carrasco, 1995; Zeng et al., 1993) may be formulated as a Mealy NSM described by a next-state function whose ith coordinate ($i = 1, \ldots, n_X$) is

$$f_i(\mathbf{x}[t-1], \mathbf{u}[t]) = g\left(\sum_{j=1}^{n_X}\sum_{k=1}^{n_U} W_{ijk}^{xxu} x_j[t-1]u_k[t] + W_i^x\right), \tag{2.4}$$

where g: $\mathcal{R} \to [S_0, S_1]$ is a sigmoid function and an output function whose ith coordinate ($i = 1, \ldots, n_Y$) is

$$h_i(\mathbf{x}[t-1], \mathbf{u}[t]) = g\left(\sum_{j=1}^{n_X}\sum_{k=1}^{n_U} W_{ijk}^{yxu} x_j[t-1]u_k[t] + W_i^y\right). \tag{2.5}$$

Throughout the article, a homogeneous notation will be used for weights. Superscripts indicate the computation in which the weight is involved: the xxu in W_{ijk}^{xxu} indicates that the weight is used to compute a state (x) from a state and an input (xu); the y in W_i^y (a bias) indicates that it is used to compute an output. Subscripts designate, as usual, the particular units involved and run parallel to superscripts.

Another Mealy NSM is Robinson and Fallside's (1991) recurrent error propagation network, a first-order DTRNN that has a next state function whose ith coordinate is given by

$$f_i(\mathbf{x}[t-1], \mathbf{u}[t]) = g\left(\sum_{j=1}^{n_X} W_{ij}^{xx} x_j[t-1] + \sum_{j=1}^{n_U} W_{ij}^{xu} u_j[t] + W_i^x\right), \tag{2.6}$$

and an output function $\mathbf{h}(\mathbf{x}[t-1], \mathbf{u}[t])$ whose ith component ($i = 1, \ldots, n_Y$) is given by

$$h_i(\mathbf{x}[t-1], \mathbf{u}[t]) = g\left(\sum_{j=1}^{n_X} W_{ij}^{yx} x_j[t-1] + \sum_{j=1}^{n_U} W_{ij}^{yu} u_j[t] + W_i^y\right). \tag{2.7}$$

It is easy to show (using a suitable odd-parity counterexample in the way described by Goudreau, Giles, Chakradhar, & Chen, 1994) that Robinson and Hallside's (1991) networks cannot represent the output function of all Mealy machines unless a two-layer scheme like the following is used:

$$h_i(\mathbf{x}[t-1], \mathbf{u}[t]) = g\left(\sum_{j=1}^{n_Z} W_{ij}^{yz} z_j[t] + W_i^y\right) \tag{2.8}$$

with

$$z_i[t] = g\left(\sum_{j=1}^{n_X} W_{ij}^{zx} x_j[t-1] + \sum_{j=1}^{n_U} W_{ij}^{zu} u_j[t] + W_i^z\right) \tag{2.9}$$

($i = 1, \ldots, n_Z$) and n_Z the number of units in the layer before the actual output layer (the hidden layer of the output function). The new architecture will be named an *augmented Robinson-Fallside network* in this article.

2.2.2 Neural Moore Machines. Elman's (1990) simple recurrent net, a widely used Moore NSM, is described by a next-state function identical to the next-state function of Robinson and Fallside's (1991) net, equation 2.6, and an output function $\mathbf{h}(\mathbf{x}[t])$ whose ith component ($i = 1, \ldots, n_Y$) is given by

$$h_i(\mathbf{x}[t]) = g\left(\sum_{j=1}^{n_X} W_{ij}^{yx} x_j[t] + W_i^y\right). \tag{2.10}$$

The second-order counterpart of Elman's (1990) simple recurrent net has been used by Carrasco, Forcada, and Santamaría (1996) and Blair and Pollack (1997). In that case, the ith coordinate of the next-state function is identical to equation 2.4, and the output function is identical to equation 2.10.

2.2.3 Sigmoid Functions. When DTRNN are used to learn FSM behavior from samples (Cleeremans et al., 1989; Pollack, 1991; Giles et al., 1992; Watrous & Kuhn, 1992; Maskara & Noetzel, 1992; Sanfeliu & Alquézar, 1994; Manolios & Fanelli, 1994; Forcada & Carrasco, 1995; Ñeco & Forcada, 1996; Gori et al., 1998), a common choice is to use continuous, real-valued activation functions for neurons in the DTRNN; this allows the construction of gradient-descent-based learning algorithms. Most researchers have used a bounded sigmoid function called the *logistic function*:

$$g_L(x) = \frac{1}{1 + \exp(-x)}. \tag{2.11}$$

This function is strictly growing, defined for any real value, takes positive values, is continuous, and is bounded by 0 and 1; as a consequence, it has an inverse. We will consider a slightly more general class of sigmoid functions in which any element g has the same four properties as the logistic:

- Strictly growing ($g'(x) > 0\ \forall x \in \mathcal{R}$)
- Continuous ($\lim_{x \to c} g(x) = g(c),\ \forall c \in \mathcal{R}$)
- Positive ($g(x) \geq 0\ \forall x \in \mathcal{R}$)

- Bounded (between S_0 and S_1, with $S_1 > S_0$):

$$g: \mathcal{R} \to [S_0, S_1]. \tag{2.12}$$

2.2.4 High and Low Signals. Throughout this article, the output range of all units, $[S_0, S_1]$, will be partitioned in three subintervals (as in Omlin & Giles, 1996a)—high: $[\epsilon_1, S_1]$, low: $[S_0, \epsilon_0]$, and forbidden: $]\epsilon_0, \epsilon_1[$—with $S_0 < \epsilon_0 < \epsilon_1 < S_1$. The values of ϵ_0 and ϵ_1 will be determined later. Note that the forbidden interval may in principle be arbitrarily small; the only thing required by the above definition is that high and low values form disjoint sets and therefore cannot be confused.

Defining valid high and low signals and forbidden intervals is customary when designing devices with digital (discrete) behavior from analog equipment such as integrated circuits based on semiconductors, so that assemblies of these circuits still show the correct digital behavior (see, e.g., Floyd, 1996), and indeed, the goal of this article is similar: we want to ensure finite-state behavior in networks built from analog activation functions (real sigmoids).

3 Encoding

3.1 Conditions for a DTRNN to Behave Like an FSM. Recently paper, Casey (1996) (see also Casey, 1998) has shown that a DTRNN performing a robust DFA-like computation (a special case of FSM-like computation) must organize its state-space in mutually disjoint, closed sets with nonempty interiors corresponding to the states of the DFA. These states can be taken to be all of the points such that if the DTRNN is initialized with any of them, the DTRNN will produce the same output as the DFA initialized in the corresponding state. Casey (1996) uses *robust emulation* to mean correct emulation even under the injection of bounded additive noise into states; a similar result is described by Maass and Orponen (1998), who also prove, as does Šíma (1997), that any discrete neural net may be simulated by an analog net even in the presence of bounded noise. When noise is not bounded (as, for example, zero-mean gaussian noise), Maass and Sontag (1999) have shown that DTRNN are not even capable of recognizing all regular languages, but only a subset of them. Our article deals with the noiseless emulation of FSM by DTRNN, although the results presented here could easily be generalized to the case of bounded noise.

Casey's (1996) formulation, which is closely connected to that of Pollack's (1991) dynamical recognizer, has inspired the following definition. A DTRNN $N = (X, U, Y, \mathbf{f}, \mathbf{h}, \mathbf{x}_0)$ behaves like an FSM $M = (Q, \Sigma, \Gamma, \delta, \lambda, q_I)$ when two sets of conditions, one relating to representation and intepretation (R1–R4) and the other to the dynamics of the DTRNN (D1 and D2), are held.

R1. Representation of state. Each state $q_i \in Q$ is assigned a nonempty region $X_i \subseteq X$ such that the DTRNN N is said to be in state q_i at time t when $\mathbf{x}[t] \in X_i$. Accordingly, these regions must be disjoint: $X_i \cap X_j = \emptyset$ if $q_i \neq q_j$. There may be regions of X that are not assigned to any state. That is, there exists a mapping[5] $\mathcal{F}$: $Q \to 2^X$ from states in Q into subsets of X and a backward mapping $\mathcal{J}$: $X \to Q$, which assigns a state in Q to each valid state vector in X.

R2. Representation of input symbols. Each possible input symbol $\sigma_k \in \Sigma$ is assigned a different vector $\mathbf{u}_k \in U$ (it would also be possible to assign a nonempty region $U_k \subseteq U$ to each symbol). We may say that there exists a mapping $\mathcal{G}$: $\Sigma \to U$ from symbols in Σ into points in U.

R3. Interpretation of output. Each possible output symbol $\gamma_m \in \Gamma$ is assigned a nonempty region $Y_m \subseteq Y$ such that the DTRNN N is said to output symbol γ_m at time t when $\mathbf{y}[t] \in Y_m$. Analogously, these regions must be disjoint: $Y_m \cap Y_n = \emptyset$ if $\gamma_m \neq \gamma_n$. There may be regions of Y that are not assigned to any output symbol. This may be expressed by saying that there exists a mapping $\mathcal{H}$: $\Gamma \to 2^Y$ from symbols in Γ into subsets of Y, and a backward mapping $\mathcal{K}$: $Y \to \Gamma$, which assigns a symbol in Γ to each valid output vector in Y.

R4. Correctness of the initial state. The initial state of the DTRNN belongs to the region assigned to the initial state q_I, that is, $\mathbf{x}_0 \in X_I$. Equivalently, $\mathcal{J}(\mathbf{x}_0) = q_I$.

D1. Correctness of the next-state function. For any state q_j and symbol σ_k of M, the transitions performed by the DTRNN N from any point in the region of state-space X_j assigned to state q_j when symbol σ_k is presented to the network must lead to points that belong to the region X_i assigned to $q_i = \delta(q_j, \sigma_k)$; formally, this may be expressed as

$$\mathbf{f}_k(X_j) \subseteq X_i \quad \forall q_j \in Q, \sigma_k \in \Sigma\colon \delta(q_j, \sigma_k) = q_i, \tag{3.1}$$

where the shorthand notation $\mathbf{f}_k(A) = \{\mathbf{f}(\mathbf{x}, \mathbf{u}_k)\colon \mathbf{x} \in A\}$ has been used. Using the mappings just defined, this is equivalent to saying that $\delta(q_j, \sigma_k) = q_i \Leftrightarrow \mathcal{J}(\mathbf{f}(\mathcal{F}(q_j), \mathcal{G}(\sigma_k)) = q_i$.

D2. Correctness of output. In the case of Mealy NSM, for any state q_j and symbol σ_k of M, the output produced by the DTRNN N from any point in the region of state-space X_j assigned to state q_j when symbol σ_k is presented to the network belongs to the region Y_m assigned to $\gamma_m = \lambda(q_j, \sigma_k)$; formally, this may be expressed as

$$\mathbf{h}_k(X_j) \subseteq Y_m \quad \forall q_j \in Q, \sigma_k \in \Sigma\colon \lambda(q_j, \sigma_k) = \gamma_m, \tag{3.2}$$

[5] We thank one of the anonymous referees for having suggested the use of explicit mappings between the FSM and its representation in the DTRNN.

where the shorthand notation

$$\mathbf{h}_k(A) = \{\mathbf{h}(\mathbf{x}, \mathbf{u}_k): \mathbf{x} \in A\} \tag{3.3}$$

has been used. This is equivalent to saying that $\lambda(q_j, \sigma_k) = \gamma_m \Leftrightarrow \mathcal{K}(\mathbf{h}(\mathcal{F}(q_j), \mathcal{G}(\sigma_k)) = \gamma_m$. In the case of Moore NSM, for any state q_i, the output produced by the DTRNN N from any point in the region, the condition may be expressed as:

$$\mathbf{h}(X_j) \subseteq Y_m \quad \forall q_i \in Q: \lambda(q_i) = \gamma_m, \tag{3.4}$$

with $\mathbf{h}(A) = \{\mathbf{h}(\mathbf{x}): \mathbf{x} \in A\}$. Or, equivalently, $\lambda(q_i) = \gamma_m \Leftrightarrow \mathcal{K}(\mathbf{h}(\mathcal{F}(q_i)) = \gamma_m$.

The regions $X_i \subseteq X$, $i = 1, \ldots, |Q|$ and $Y_m \subseteq Y$, $m = 1, \ldots, |\Gamma|$ may have an uncountable number of points because of being subsets of $\mathcal{R}^n$ with nonempty interiors for some n. However, for a finite input alphabet Σ, only a countable number of points in the state (X) and output (Y) spaces are actually visited by the net for the set of all possible input strings over Σ, denoted Σ^*, which is also denumerable.

DFA represent a special case. As noted in section 2.1.3, deterministic finite automata may be seen as Moore or Mealy machines having an output alphabet $\Gamma = \{\mathcal{Y}, \mathcal{N}\}$ whose output is examined only after the last symbol of the input string is presented, and it is such that, for a Moore machine,

$$\lambda(q_i) = \begin{cases} \mathcal{Y} \text{ if } q_i \in F \\ \mathcal{N} \text{ otherwise,} \end{cases} \tag{3.5}$$

and for a Mealy machine,

$$\lambda(q_i, \sigma_k) = \begin{cases} \mathcal{Y} \text{ if } \delta(q_i, \sigma_k) \in F \\ \mathcal{N} \text{ otherwise.} \end{cases} \tag{3.6}$$

A region in output space Y would be assigned to each one of these two symbols: $Y_{\mathcal{Y}}$, $Y_{\mathcal{N}}$, such that $Y_{\mathcal{Y}} \cap Y_{\mathcal{N}} = \emptyset$, and the output of the NSM would be examined only after the whole string has been processed.

3.2 A General Encoding Scheme. To encode an FSM in a DTRNN, it suffices to find a way of assigning an initial state $\mathbf{x}_0$, the regions $X_i \subseteq X$ and $Y_m \subseteq Y$, the input vectors $\mathbf{u}_k$, and the functions $\mathbf{f}$ and $\mathbf{h}$ so that it fulfills all the conditions given in the previous section.

We propose a simple scheme to encode FSM in DTRNN that is general enough to be applied to a variety of FSM and DTRNN architectures (but admittedly neither the only possible nor the best one can design). This scheme is based in the customary "corner" or "exclusive" scheme in which each

FSM state q_i is encoded by DTRNN state vectors having coordinate i close to the highest possible output value of the sigmoid (S_1) and all other coordinates close to the lowest possible output value (S_0); each output symbol γ_m is encoded in a similar way, and input vectors $\mathbf{u}_k$ are unit vectors in $\mathcal{R}^{|\Sigma|}$ having all their coordinates equal to 0 except for the kth coordinate, which is taken to be equal to 1. As will be shown, it is not difficult to find architectures and weight schemes for the DTRNN to so behave.

If one would require DTRNN state values to be exactly S_0 and S_1, that would require some weights to be infinite, and the result would be similar to not using sigmoids at all, but using steplike functions instead. If sigmoids and finite weights are used, neuron outputs are never exactly S_0 and S_1, and this in turn "contaminates" further outputs due to feedback; this accumulative "contamination" may in principle be such that outputs get so far from S_0 and S_1 that the requirement of disjoint regions for each state may be violated (a behavior that some authors call *instability*), but this is not always necessarily the case. The encoding scheme proposed here establishes a way to assign regions of DTRNN state-space to FSM states (and thus a way to establish forbidden regions) and fixing weights to values such that DTRNN transitions departing from points in region X_j with symbol σ_k always end in X_i if $\delta(q_j, \sigma_k) = q_i$, that is, a scheme in which state "contamination" is bounded to tolerable levels, and outputs are also guaranteed to be correct (see the description of low, high, and forbidden intervals in section 2.2.4). This is what some authors (e.g., Omlin & Giles, 1996b) would call a *stable encoding*.

The "corner" scheme with tolerances is naturally related to the definition of low, high, and forbidden values for DTRNN states. The values of ϵ_0 and ϵ_1 will be chosen in each encoding to be as far as possible from S_0 and S_1, respectively, to allow the smallest possible values for weights. Moving away from saturating the sigmoids by using small weights and tolerance intervals around the limiting values S_0 and S_1 makes special sense in a gradient-descent learning setting, because saturated sigmoids lead to vanishing derivatives.

All of our sigmoid constructions use sets of weights that are either zero or simple multiples of a single positive parameter H (H, $-H$; $H/2$, $3H/2$), and are such that in the limit $H \to \infty$, they would behave exactly like classical step-function constructions using weights that are zero or simple rational numbers (1, -1, $1/2$, $3/2$, etc.), taking state vectors to the exact corners of state-space.[6]

The main assumption of all of our encoding schemes is the following: there exist sets of values of ϵ_0, ϵ_1, and H such that the corresponding

[6] It could also be said that our sigmoid constructions are obtained by using these simple rational weights and setting H, the gain of all sigmoids, to a small finite value that still guarantees FSM-like behavior of the DTRNN. This is related to the result by Arai and Nakano (1996) mentioned in section 1.

DTRNNs may be interpreted as behaving exactly like FSM. What follows is a description of the common details of all the encoding schemes:

R1. Representation of state. The number of state units, n_X, is taken to be equal to the number of states in the FSM,[7] $|Q|$. Each state $q_i \in Q$ is assigned a region $X_i(\epsilon_0, \epsilon_1) \subseteq X$ such that the DTRNN N is said to be in state q_i at time t when $\mathbf{x}[t] \in X_i(\epsilon_0, \epsilon_1)$. These regions are defined by

$$X_i(\epsilon_0, \epsilon_1) = \{\mathbf{x} \in X\colon\ x_i \in [\epsilon_1, S_1] \wedge x_j \in [S_0, \epsilon_0], \forall j \neq i\}, \tag{3.7}$$

that is, all the coordinates of vectors in X_i are low except for coordinate i, which is high, and the region stands at a corner of the hypercube $X = [S_0, S_1]^{n_X}$. These regions are disjoint because $\epsilon_0 < \epsilon_1$: $X_i(\epsilon_0, \epsilon_1) \cap X_j(\epsilon_0, \epsilon_1) = \emptyset$ if $q_i \neq q_j$. There are points in X that are not assigned to any region.

R2. Representation of input symbols. The number of inputs to the network n_U is chosen to be equal to the number of symbols in the input alphabet, $|\Sigma|$. Each possible input symbol $\sigma_k \in \Sigma$ is assigned a vector $\mathbf{u}_k \in U$ such that its jth coordinate, u_{jk}, equals δ_{jk}, that is, it is 1 if $j = k$ and zero otherwise. This is usually known as *exclusive*, *one-hot*, or *unary* encoding of inputs.

R3. Interpretation of output. The number of output units n_Y is chosen to be equal to the number of symbols in the output alphabet, $|\Gamma|$. Each possible output symbol $\gamma_m \in \Gamma$ is assigned a region $Y_m(\epsilon_0, \epsilon_1) \subseteq Y$ such that the DTRNN N is said to output symbol γ_m at time t when $\mathbf{y}[t] \in Y_m(\epsilon_0, \epsilon_1)$. The regions are defined as follows:

$$Y_m(\epsilon_0, \epsilon_1) = \{\mathbf{y} \in Y\colon\ y_m \in [\epsilon_1, S_1] \wedge y_n \in [S_0, \epsilon_0], \forall n \neq m\}, \tag{3.8}$$

that is, all the coordinates of output vectors in Y_m are low except for coordinate m, which is high, and the region stands at a corner of the hypercube $Y = [S_0, S_1]^{n_Y}$. The use of identical values of ϵ_0 and ϵ_1 for both state and output regions is not required but it is convenient because it simplifies the ensuing theoretical treatment. It is obvious that these regions are disjoint, $Y_m(\epsilon_0, \epsilon_1) \cap Y_n(\epsilon_0, \epsilon_1) = \emptyset$ if $\gamma_m \neq \gamma_n$, provided that $\epsilon_0 < \epsilon_1$. There are points in Y that are not assigned to any of the output regions Y_m.

As discussed in section 2.1.3 and later at the end of section 3.1, deterministic finite automata may be seen as Mealy and Moore machines having a two-symbol output alphabet $\Gamma = \{\mathcal{Y}, \mathcal{N}\}$ and whose output is examined only after the whole input string has been processed. In the general

[7] As will be shown, some DTRNN architectures are computationally incapable of representing all FSM, but in the cases studied, the original FSM M may be converted into a new FSM M' with a larger number of states that may itself be encoded in the DTRNN.

encoding scheme used in this section, two output neurons would have to be used to represent this computation. However, in the DFA case, it is convenient (and parallel to the work by Omlin & Giles, 1996a, 1996b, and Alquézar & Sanfeliu, 1995) to use a more distributed encoding in which a single output neuron is used, since the state of the other neuron would be its complement. In this case, $Y_{\mathcal{Y}} = [\epsilon_1, S_1]$ and $Y_{\mathcal{N}} = [S_0, \epsilon_0]$. Distributed output schemes would in principle be adequate for output alphabets of any size; our general encoding scheme uses an exclusive interpretation of output because this is the common choice when DTRNN are used to learn transductions of any kind, and also because it makes it easier to derive the conditions for the DTRNN to behave like the corresponding FSM.

R4. Correctness of the initial state. The initial state of the DTRNN N, is $\mathbf{x}_0 = \{S_0 + \delta_{il}(S_1 - S_0)\}_{i=1}^{n_X}$, which is in $X_I(\epsilon_0, \epsilon_1)$ regardless of the choice of ϵ_0 and ϵ_1 (any other state in $X_I(\epsilon_0, \epsilon_1)$ would have been equally valid but less convenient).

D1. Correctness of the next state function. The next-state function for all the DTRNN architectures discussed here will be chosen to have a single adjustable parameter $H > 0$. For any state q_j and symbol σ_k of M, the transitions performed by the DTRNN N from any point in the region of state-space $X_j(\epsilon_0, \epsilon_1)$ assigned to state q_j when symbol σ_k is presented to the network must lead to points that belong to the region $X_i(\epsilon_0, \epsilon_1)$ assigned to $q_i = \delta(q_j, \sigma_k)$. Formally, this may be expressed as

$$\mathbf{f}_k^{(H)}(X_j(\epsilon_0, \epsilon_1)) \subseteq X_i(\epsilon_0, \epsilon_1) \;\; \forall q_j, \sigma_k \colon \delta(q_j, \sigma_k) = q_i, \tag{3.9}$$

where the superscript (H) expresses the parametric dependence on H. This equation is derived from the general condition, equation 3.1, and the definition of the regions assigned to each FSM state, equation 3.7.

D2. Correctness of output. The output function for all the DTRNN architectures discussed here will be chosen to have a single adjustable parameter, $H > 0$. In the case of Mealy NSM, for any state q_j and symbol σ_k of M, the output produced by the DTRNN N from any point in the region of state-space $X_j(\epsilon_0, \epsilon_1)$ assigned to state q_j when symbol σ_k is presented to the network belongs to the region $Y_m(\epsilon_0, \epsilon_1)$ assigned to $\gamma_m = \lambda(q_j, \sigma_k)$. Formally, this may be expressed as

$$\mathbf{h}_k^{(H)}(X_j(\epsilon_0, \epsilon_1)) \subseteq Y_m(\epsilon_0, \epsilon_1) \;\; \forall q_j, \sigma_k \colon \lambda(q_j, \sigma_k) = \gamma_m, \tag{3.10}$$

where the superscript (H) expresses the parametric dependence on H. This equation is derived from the general condition, equation 3.2, and the definition of the regions assigned to each FSM state, equation 3.7, and each output symbol, equation 3.8. It would have been possible to use different values of ϵ_0, ϵ_1, and H for output and state; this would allow

for the choice of smaller weight values but would duplicate the number of parameters. In the case of Moore NSM, for any state q_j, the output produced by the DTRNN N from any point in the region of state-space $X_j(\epsilon_0, \epsilon_1)$ assigned to state q_j belongs to the region $Y_m(\epsilon_0, \epsilon_1)$ assigned to $\gamma_m = \lambda(q_j)$. The condition may be expressed as:

$$\mathbf{h}^{(H)}(X_j(\epsilon_0, \epsilon_1)) \subseteq Y_m(\epsilon_0, \epsilon_1) \; \forall q_j \colon \lambda(q_j) = \gamma_m. \tag{3.11}$$

This equation is derived from condition 3.4 and the definitions of the regions assigned to each FSM state (see equation 3.7) and to each output symbol (see equation 3.8).

Conditions R1 through R4 may be applied to all of the architectures and will be assumed to be implicit in all of the encodings proposed; therefore, we will focus on the dynamical conditions D1 and D2 to derive stability conditions for each encoding.

If values of ϵ_0, ϵ_1, and H exist such that the conditions $S_0 < \epsilon_0 < \epsilon_1 < S_1$, and the conditions derived for a particular DTRNN architecture from equations 3.9 and either 3.10 for a Mealy machine or 3.11 for a Moore machine are met, then the DTRNN is guaranteed to behave like an FSM. The properties of the class of sigmoid functions studied in this article, given in section 2.2.3, in particular their monotonous growth, make it very easy to obtain conditions on H, ϵ_0, and ϵ_1 in closed form.

These conditions are in general sufficient but not necessary. One of the reasons has been given in section 3.1: the set of all possible values of $\mathbf{x}$ and $\mathbf{y}$ reachable by the DTRNN after reading any string over the alphabet Σ is a denumerable set, whereas the above conditions work on nondenumerable regions of nondenumerable state and output spaces. For example, the conditions require even the worst (furthest from the corner) points of all regions $X_i(\epsilon_0, \epsilon_1)$ to map onto the corresponding valid regions of state-space$_i$, while it may be possible that those worst points are never reached from the initial state $\mathbf{x}_0$ with any string over Σ. As a result, smaller values of H, and even weight schemes in which all weights are not different (as in Kremer, 1996) may still guarantee FSM-like behavior.

4 Encoding of Mealy Machines in DTRNN

4.1 Mealy Machines in Second-Order DTRNN. To encode Mealy machines in second-order DTRNN such as the ones described by equations 2.4 and 2.5, we will use $n_X = |Q|$, $n_U = |\Sigma|$, and $n_Y = |\Gamma|$. We propose two possible constructions that correspond to two different versions of the next-state function $\mathbf{f}_k^{(H)}$ and the output function $\mathbf{h}_k^{(H)}$. The first construction uses two sparse weight matrices, $\{W_{ijk}^{xxu}\}$ and $\{W_{ijk}^{yxu}\}$, and a bias on each state and output unit. Weights can only take two values, either H or 0, and all biases are equal to $-H/2$. This construction is similar but not identical to the one

proposed by Omlin and Giles (1996a, 1996b).[8] The second construction uses two dense weight matrices, $\{W^{xxu}_{ijk}\}$ and $\{W^{yxu}_{ijk}\}$, but no biases. Weights can take only two values, H and $-H$.

4.1.1 Using a Sparse Weight Matrix. In this construction[9] all state and output units are biased to assume low values by using a bias of $-H/2$; for a state unit to assume a high value, it has to be excited by means of a connection of weight H from a unit in a high state. The DTRNN is said to be in state q_i and outputting symbol γ_m at time t when its state vector $\mathbf{x}[t]$ is in $X_i(\epsilon_0, \epsilon_1)$ and its output vector $\mathbf{y}[t]$ is in $Y_m(\epsilon_0, \epsilon_1)$, that is, when all state units are in a low state except for unit i and all output units are low except for unit m. If the previous state $\mathbf{x}[t-1]$ was such that the DTRNN might be said to be in state q_j, then $\mathbf{x}[t-1] \in X_j(\epsilon_0, \epsilon_1)$ (only state unit j was high and the rest were low). If the symbol presented at time t is σ_k, all the components of the input vector $\mathbf{u}[t]$ are 0 except for component k. Thus, if $\delta(q_j, \sigma_k) = q_i$ and $\lambda(q_j, \sigma_k) = \gamma_m$, we want weights W^{xxu}_{ijk} and W^{yxu}_{mjk} to be high enough to surmount the negative bias $-H/2$ and excite state unit i and output unit m and all other weights $W^{xxu}_{i'jk}$ and $W^{yxu}_{m'jk}$ $(i \neq i', m \neq m')$ to be zero so that the negative bias on all other state and output units brings them to a low state. The resulting choice of weights is:

$$W^{xxu}_{ijk} = \begin{cases} H & \text{if } \delta(q_j, \sigma_k) = q_i, \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

$$W^{yxu}_{ijk} = \begin{cases} H & \text{if } \lambda(q_j, \sigma_k) = \gamma_i, \\ 0 & \text{otherwise,} \end{cases} \tag{4.2}$$

$$W^{x}_{i} = -\frac{H}{2}, \tag{4.3}$$

and

$$W^{y}_{i} = -\frac{H}{2}. \tag{4.4}$$

Now we want to find conditions on H, ϵ_0, and ϵ_1 such that the next-state function and the output function of the DTRNN constructed in this way are correct, that is, such that they satisfy conditions 3.9 and 3.10 (these are conditions D1 and D2 in section 3.2). With this encoding, the dynamics of

[8] They set some weights to $-H$.

[9] This encoding has also been used by some of us to encode an extension of Mealy machines (Ñeco et al., 1999) and to directly encode *nondeterministic* finite automata without converting them first to DFA (Carrasco et al., 1999).

the second-order recurrent neural network, represented by equations 2.4 and 2.5, may be studied in four cases: high and low states and high and low outputs.

Condition D1: Correctness of the next state. For the particular next-state function defined by equations 4.1 and 4.3, we will study the transition $\delta(q_j, \sigma_k) = q_i$; that is, $\mathbf{x}[t-1]$ is any point in $X_j(\epsilon_0, \epsilon_1)$, $\mathbf{u}[t] = \mathbf{u}_k$, and find conditions on ϵ_0, ϵ_1, and H so that any resulting state $\mathbf{x}[t]$ is in $X_i(\epsilon_0, \epsilon_1)$.

The new state of neuron i is given by

$$x_i[t] = g\left(\sum_{l \in C_{ik}} H x_l[t-1] - \frac{H}{2}\right), \tag{4.5}$$

where

$$C_{ik} = \{l\colon \delta(q_l, \sigma_k) = q_i\}, \tag{4.6}$$

and, obviously, $j \in C_{ik}$. The equation must fulfill $x_i[t] \geq \epsilon_1$ for $\mathbf{x}[t] \in X_i(\epsilon_0, \epsilon_1)$ to hold. In the worst case, $x_j[t-1]$ contributes with the lowest possible high signal ($x_j[t-1] = \epsilon_1$), and the rest of the $x_l[t-1]$ contribute with the strongest possible low signal for all valid sigmoid functions, that is, $S_0 = 0$; therefore,

$$g\left(H\left(\epsilon_1 - \frac{1}{2}\right)\right) \geq \epsilon_1 \tag{4.7}$$

guarantees a high state value for $x_i[t]$. This worst case may indeed occur when $C_{ik} = \{j\}$. This will be important to explain some of the experimental results shown in section 7.

On the other hand, for all other state units, $i' \neq i$, the signal has to be low, $x_{i'}[t] \leq \epsilon_0$. But some of the low signals at time $t-1$ may be high (far from S_0 and closer to ϵ_0) and raise the value of $x_{i'}[t]$, since there may exist states q_l such that $\delta(q_l, \sigma_k) = q_{i'}$, which prescribe nonzero weight values $W^{xxu}_{i'lk} = H$; these are the states in $C_{i'k}$ (see equation 4.6). It is convenient to define

$$\chi_x = \max_{j,k} |C_{jk}|, \tag{4.8}$$

that is, the size of the biggest of such sets, which may be understood as the maximum fan-in of each state in the transition diagram of the Mealy machine. In the worst case, the χ_x low signals have the highest possible low value (ϵ_0) and contribute through a weight H to weaken (increase) the desired low signal for $x_{i'}[t]$. In that worst case, the equation defining the state of $x_{i'}[t]$ (according to equations 2.4, 4.1, and 4.3)

$$x_{i'}[t] = g\left(\sum_{l \in C_{i'k}} H x_l[t-1] - \frac{H}{2}\right) \tag{4.9}$$

reduces to

$$g\left(H\left(\chi_x\epsilon_0 - \frac{1}{2}\right)\right) \leq \epsilon_0, \tag{4.10}$$

an equation that may be used to guarantee that all $i' \neq i$ are low.

Condition D2: Correctness of the output. For the particular output function defined by equations 4.2 and 4.4 we will study the process $\lambda(q_j, \sigma_k) = \gamma_m$; that is, $\mathbf{x}[t-1]$ is any point in $X_j(\epsilon_0, \epsilon_1)$, $\mathbf{u}[t] = \mathbf{u}_k$, and find conditions on ϵ_0, ϵ_1, and H so that any resulting output $\mathbf{y}[t]$ is in $Y_m(\epsilon_0, \epsilon_1)$. If we proceed analogously to the next-state study above, we obtain two conditions that guarantee a high value of $y_m[t]$ and a low value for $y_{m'}[t]$—one identical to equation 4.7 and the other one given by

$$g\left(H\left(\chi_y\epsilon_0 - \frac{1}{2}\right)\right) \leq \epsilon_0, \tag{4.11}$$

with

$$\chi_y = \max_{m,k} |D_{mk}|, \tag{4.12}$$

the maximum fan-in of the output function and

$$D_{mk} = \{l: \lambda(q_l, \sigma_k) = \gamma_m\}. \tag{4.13}$$

Therefore, if one can find ϵ_0 and ϵ_1 such that $S_0 < \epsilon_0 < \epsilon_1 < S_1$ and such that conditions 4.7, 4.10, and 4.11 are met, then the second-order DTRNN with the dynamics defined by equations 2.4 and 2.5 and constructed according to equations 4.1 through 4.4 behaves like the corresponding Mealy machine. Conditions 4.10 and 4.11 may be ensured by a more stringent condition,

$$g\left(H\left(\chi\epsilon_0 - \frac{1}{2}\right)\right) \leq \epsilon_0, \tag{4.14}$$

with $\chi = \max(\chi_x, \chi_y)$ (note that always $\chi \leq n_X$). The conditions derived here are sufficient but not necessary—due to the use of worst cases that may not occur in general and to the merging of conditions into more stringent ones—and thus, smaller values of H, larger values of ϵ_0, and smaller values of ϵ_1 may still be adequate for the second-order DTRNN to behave like the corresponding Mealy machine.

Due to the form of equations 4.7 and 4.14, not only sets of values of H, ϵ_0, and ϵ_1 can be found for any $\chi > 1$ (for any Mealy machine), but it is also the case that a minimum positive value of H can be found for each fan-in

χ. Clearly, reducing H increases ϵ_0 and reduces ϵ_1. One can directly search for the minimum possible H and tabulate the results for each possible value of χ. Note that conditions for H, ϵ_0, and ϵ_1 are independent of the size of the input alphabet Σ. This will indeed be the case in all the constructions presented in this article.

These values are easily obtained by realizing that the most stringent condition is 4.14. One can turn it into an equation, solve for H, and take $\partial H/\partial\epsilon_0 = 0$. The resulting equation reads

$$\frac{g^{-1}(\epsilon_0)}{\epsilon_0 - \frac{1}{2\chi}} = \frac{1}{g'\left(g^{-1}(\epsilon_0)\right)}, \tag{4.15}$$

with $0 < \epsilon_0 < 1/2\chi$ and $g'(x)$ the derivative of the sigmoid function. After solving for ϵ_0 (for example, using an iterative formula), the resulting value may be used in equation 4.14 to obtain the minimum value for H and in equation 4.7 to obtain the corresponding value of ϵ_1.

In particular, g is the logistic function g_L; then $S_0 = 0$, $S_1 = 1$, and the resulting values of ϵ_0, ϵ_1, and minimum H for some values of χ are shown in Table 1, where the superscripts $+$ and $-$ on some numbers denote values infinitesimally smaller or larger than the one given. Taking into account that $1/g_L'(g_L^{-1}(\epsilon_0)) = \epsilon_0^{-1} - (1-\epsilon_0)^{-1}$, a possible iteration scheme is

$$\epsilon_0[t+1] = \left(\frac{g_L^{-1}(\epsilon_0[t])}{\epsilon_0[t] - \frac{1}{2\chi}} - \frac{1}{1-\epsilon_0[t]}\right)^{-1}. \tag{4.16}$$

A few iterations starting with an intermediate value in $(0, 1/2\chi)$ such as $\epsilon_0[0] = 1/4\chi$ are enough for any $\chi > 1$.

As expected, the minimum H is a growing function of χ, shown in Figure 1,

$$H = \eta(\chi), \tag{4.17}$$

that grows slower than $\log(\chi)$ (the ratio $\eta(\chi)/\log(\chi)$ is a decreasing function of χ). On the other hand, ϵ_0 is a decreasing function of χ, shown in Figure 2,

$$\epsilon_0 = \epsilon(\chi) \tag{4.18}$$

which vanishes slightly faster than $1/\chi$.

The forbidden interval (ϵ_0, ϵ_1) is rather wide, except in the case $\chi = 1$, where it is infinitesimally small. Section 7 shows experimental results that illustrate the theoretical results obtained here.

Equation 4.7 corresponds to a worst-case situation that may indeed occur when the only transition into an FSM state q_i when reading a symbol σ_k comes from a single state q_j. In particular, if these transitions form loops

Table 1: Minimum Values of Weights and Limiting Values for Encoding a Mealy Machine on a Second-Order DTRNN with Biases Using the Logistic Sigmoid Function.

χ	H	ϵ_0	ϵ_1
1	4^+	0.5^-	0.5^+
2	7.17799	0.0753324	0.965920
3	8.36131	0.0415964	0.982628
4	9.14211	0.0281377	0.988652
7	10.5829	0.0136861	0.994704
10	11.4664	0.00879853	0.996648

Note: The value of ϵ_0 is obtained through iteration of equation 4.16; the corresponding values of H and ϵ_1 are obtained by directly solving equations 4.14 and 4.7, respectively.

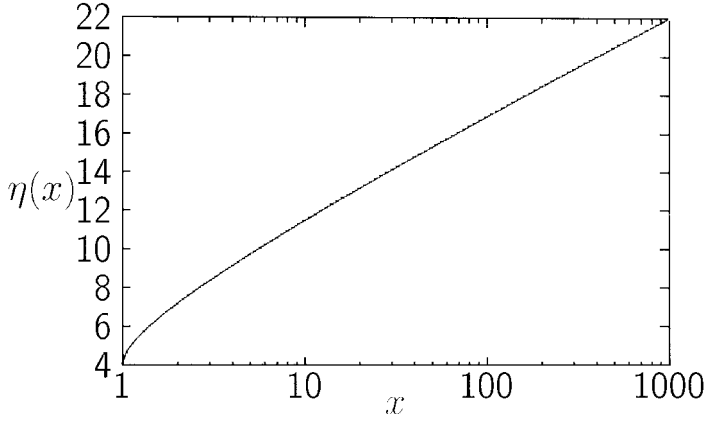

Figure 1: The function η.

such as $\delta(q_i, \sigma_k) = q_i$, the DTRNN happens to iterate function $g(Hx - H/2)$, which happens to have a fixed-point attractor exactly at $x^* = \epsilon_1$ if H is set to the theoretical value. For those FSM, if H is set to a value that is slightly lower than the theoretical value, then the attractor moves to a value $x^* < \epsilon_1$ and enters the forbidden interval, and after a few iterations, the DTRNN develops invalid representations of state. However, for other FSMs that do not show these loops, it may happen that a lower value of H may still be compatible with the validity of all possible representations of state. This has to be taken into account when interpreting some of the experimental results in section 7.

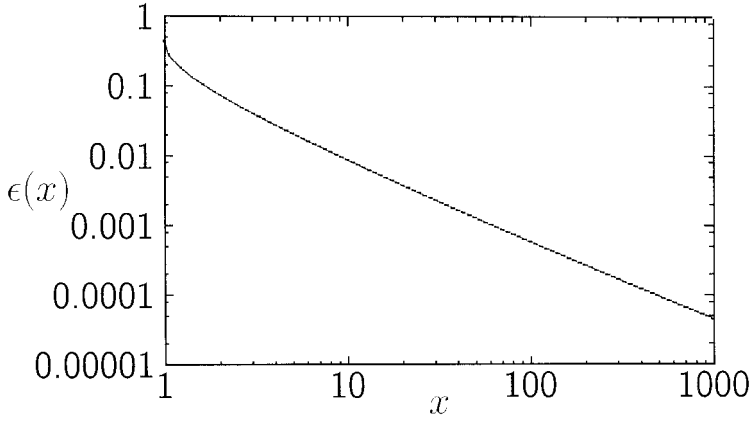

Figure 2: The function ϵ.

4.1.2 Using a Dense Weight Matrix. An alternative encoding[10] uses no biases, as in the original work by Giles et al. (1992). In this construction, weights of H are used as in the same way as in the previous construction, but weights of $-H$ are used instead of 0. The effect of the negative weights is to lower the value of a state or output unit when it has to be low, with a similar effect to that of a negative bias.

The values of weights are thus given by:

$$W_{ijk}^{xxu} = \begin{cases} H & \text{if } \delta(q_j, \sigma_k) = q_i \\ -H & \text{otherwise} \end{cases} \tag{4.19}$$

and

$$W_{ijk}^{yxu} = \begin{cases} H & \text{if } \lambda(q_j, \sigma_k) = \gamma_i \\ -H & \text{otherwise} \end{cases} . \tag{4.20}$$

The conditions for this construction to guarantee a correct next state and a correct output are derived in a similar way as that of the previous section, that is, from conditions D1 and D2 of section 3.2.

Condition D1: Correctness of the next state. When the Mealy machine moves reads symbol σ_k, state $x_i[t]$ is computed as follows:

$$x_i[t] = g\left(\sum_{l \in C_{ik}} H x_l[t-1] + \sum_{l \notin C_{ik}} (-H) x_l[t-1]\right). \tag{4.21}$$

[10] Which has already been preliminarily presented by two of us (Kremer et al., 1998).

If the 2ODTRNN may be said to be in state q_j at time $t-1$, then $\mathbf{x}[t-1]$ is in $X_j(\epsilon_0, \epsilon_1)$. If the input presented to it, $\mathbf{u}[t]$, is $\mathbf{u}_k$, representing σ_k, and $\delta(q_j, \sigma_k) = q_i$, then the lowest value of $x_i[t]$—which should be larger than ϵ_1 for $\mathbf{x}[t]$ to be in $X_i(\epsilon_0, \epsilon_1)$, that is, a valid neural representation of state q_i in the Mealy machine—is obtained when $C_{ik} = \{j\}$ (that is, $\delta(q_l, \sigma_k) \neq q_i$ for all $l \neq j$, a worst case that may never occur in a given automaton), $x_j[t-1] = \epsilon_1$, and all other $x_l[t-1]$, $l \notin C_{ik}$ are equal to ϵ_0. In this worst case, the following inequality results:

$$g\left(H\epsilon_1 - (|Q|-1)H\epsilon_0\right) \geq \epsilon_1. \tag{4.22}$$

Now let us consider the values of all $x_{i'}[t]$, $i' \neq i$ at time t, which have to be low (smaller than ϵ_0) for $\mathbf{x}[t]$ to be in $X_i(\epsilon_0, \epsilon_1)$, that is, a valid neural representation of q_i. The worst situation would be when $\delta(q_l, \sigma_k) = q_{i'}$ for all q_l except for q_j. In this case,

$$x_{i'}[t] = g\left(-Hx_j[t-1] + \sum_{l \neq j} Hx_l[t-1]\right). \tag{4.23}$$

The worst possible set of values for $\mathbf{x}[t-1]$ that is still in $X_j(\epsilon_0, \epsilon_1)$, that is, a valid neural representation of $q_j[t]$ (the set of values leading to the highest value for $x_{i'}[t]$), is obtained when $x_j[t-1]$ takes the lowest allowed value (ϵ_1), and all other $x_l[t-1]$, $l \neq j$ take their highest allowed value (ϵ_0). The resulting inequality is

$$g\left(-H\epsilon_1 + (|Q|-1)H\epsilon_0\right) \leq \epsilon_0. \tag{4.24}$$

Condition D2: Correctness of output. A parallel study of the output dynamics of this construction yields two conditions that are exactly equivalent to equations 4.22 and 4.24.

Values for ϵ_0, ϵ_1, and H such that $S_0 < \epsilon_0 < \epsilon_1 < S_1$ and that fulfill conditions 4.22 and 4.24 guarantee that the 2ODTRNN constructed according to equations 4.19 and 4.20 behaves exactly like the corresponding Mealy machine.

To find the minimum value of H satisfying these equations for a given $|Q|$, one may use a direct search in (ϵ_0, ϵ_1) space. If g is the logistic function g_L, the solution is found to have symmetric low and high intervals,[11] that

[11] It is easy to prove why the solution for the logistic function g_L satisfies $\epsilon_0 + \epsilon_1 = 1$: let us suppose that we have $\epsilon_0 > 1 - \epsilon_1$ and H satisfying both inequalities 4.22 and 4.24, and define $\epsilon_0' = 1 - \epsilon_1$; clearly $\epsilon_0' < \epsilon_0$ and therefore equation 4.22 is also satisfied by the pair $(\epsilon_0', \epsilon_1)$. Applying the transformation $g_L(-x) = 1 - g_L(x)$ yields the inequality $g_L(-H\epsilon_1 + (|Q|-1)H\epsilon_0') \leq 1 - \epsilon_1 = \epsilon_0'$ which shows that the pair $(\epsilon_0', \epsilon_1)$ also satisfies equation 4.24.

Table 2: Values of Weights and Limiting Values for Encoding a DFA on a Second-Order DTRNN Without Biases.

$\|Q\|$	H	ϵ_0
2	2^+	0.5^-
3	3.11297	0.121951
4	3.58899	0.0753324
5	3.92227	0.0538956
6	4.18066	0.0415964
7	4.39206	0.0336391
10	4.86330	0.0210033
30	6.22427	0.00538437

Note: $\epsilon_1 = 1 - \epsilon_0$.

is, $\epsilon_0 + \epsilon_1 = 1$. In this case, equations 4.22 and 4.24 both reduce to

$$g_L\left(-H + |Q|H\epsilon_0\right) \leq \epsilon_0, \tag{4.25}$$

which may be solved similarly to equation 4.14 by realizing that it may be rewritten as

$$g_L\left(2H\left(\frac{|Q|}{2}\epsilon_0 - \frac{1}{2}\right)\right) \leq \epsilon_0. \tag{4.26}$$

Solutions are, accordingly, $H = (1/2)\eta(|Q|/2)$ and $\epsilon_0 = \epsilon(|Q|/2)$; that is, values of H obtained for this construction are smaller than (less than half of) those obtained for the construction in the previous section, and the forbidden interval is narrower.[12] Some values are shown in Table 2. Experimental results supporting the validity of this construction are given in section 7.

4.2 Mealy Machines in First-Order DTRNN. First-order, single-layer Mealy neural state machines, also known as Robinson and Fallside's (1991) recurrent error propagation networks, defined by equations 2.6 and 2.7, cannot perform all possible Mealy machine computation if an exclusive interpretation of output (one neuron per output symbol) is used. This section shows how to encode any Mealy machine in the modified architecture whose next-state function is computed by a single-layer feedforward

[12] It is interesting to note here that for the case $|Q| = 2$, if instead of defining a forbidden interval that finally ends up being of zero measure, one simply divides the range $(0, 1)$ of the logistic function in two halves representing low and high signals, it may be proved that any weight $H > 0$ still guarantees the correct behavior. This will be apparent in the results shown in section 7.

network of the form given in equation 2.6 and whose output function is computed by a two-layer feedforward neural network of the form given in equations 2.8 and 2.9.

The encoding will use $n_U = |\Sigma|$, $n_Y = |\Gamma|$, $n_X = |Q||\Sigma|$, and $n_Z = |\Gamma||\Sigma|$. The reasons for the last two choices will be clear in the following.

The construction defines a new Mealy machine $M' = (Q', \Sigma, \Gamma', \delta', \lambda', q'_I)$ (the split-state, split-output Mealy machine) as follows:

- The new set of states is $Q' = Q \times \Sigma$, that is, new states are pairs (q_i, σ_k).
- The new set of outputs is $\Gamma' = \Gamma \times \Sigma$, that is, new outputs are pairs (γ_m, σ_k).
- The new next-state function δ': $Q' \times \Sigma \to Q'$ is defined as follows:

$$\delta'((q_i, \sigma_k), \sigma_l) = (\delta(q_i, \sigma_l), \sigma_l), \tag{4.27}$$

  that is, the second component of each state in Q' represents the symbol that was read before reaching it.
- The new output function λ': $Q' \times \Sigma \to \Gamma'$ is defined as follows:

$$\lambda'((q_i, \sigma_k), \sigma_l) = (\lambda(q_i, \sigma_l), \sigma_l). \tag{4.28}$$

- The new initial state q'_I is any of the states (q_I, σ_k), $\sigma_k \in \Sigma$.

It is not at all difficult to show that the split-state, split-output Mealy machine M' performs the same finite-state computation (transduction) as M, if all the symbols in each of the sets $\{(\gamma_m, \sigma_k)\colon \sigma_k \in \Sigma\}$ output by M' are interpreted as γ_m. Note that after splitting, some of the states in Q' and some of the outputs in Γ' may be useless (states may be inaccessible from the initial state or outputs may never be produced) and could be eliminated before proceeding with the construction.

The proposed scheme encodes the split-state, split-output Mealy machine M', hence the need for $n_X = |Q'| = |Q||\Sigma|$ state units and $n_Y = |\Gamma'| = |\Gamma||\Sigma|$ hidden units in the output function. Each state unit (resp., each hidden unit in the output function) is made to correspond to (and numbered as) a state in Q' (resp., an output in Γ'). The values of weights and biases (as generalized from Alquézar & Sanfeliu's, 1995, construction of DFAs on Elman, 1990, nets) for the next-state function are given by:

$$W^{xx}_{ij} = \begin{cases} H & \text{if } \delta'(q'_j, \sigma_k) = q'_i \text{ for some } \sigma_k \in \Sigma \\ 0 & \text{otherwise,} \end{cases} \tag{4.29}$$

$$W^{xu}_{ik} = \begin{cases} H & \text{if } \delta'(q'_j, \sigma_k) = q'_i \text{ for some } q'_j \in Q' \\ 0 & \text{otherwise,} \end{cases} \tag{4.30}$$

$$W^{x}_{i} = -\frac{3H}{2}, \ \forall i = 1, \ldots, n_X. \tag{4.31}$$

The construction works as follows: a state neuron i will be excited to a high state only if both a state neuron j such that $\delta'(q'_j, \sigma_l) = q'_i$ for some σ_l is high *and* an input k such that $\delta'(q'_l, \sigma_k) = q'_i$ for some q'_l is also high, because a bias of $-3H/2$ has to be surmounted (the bias of $-3H/2$ is necessary because we want the output of a unit to be high only when at least two inputs—one from a state unit and another one from an input signal—contribute with a value around $+H$). The split-state, split-output Mealy machines are such that this is possible only if $\delta'(q'_j, \sigma_k) = q'_i$; otherwise, the bias will bring the state unit i to a low state.[13]

The output function is defined by the following weight scheme:

$$W_{ij}^{zx} = \begin{cases} H & \text{if } \lambda'(q'_j, \sigma_k) = \gamma'_i \text{ for some } \sigma_k \in \Sigma \\ 0 & \text{otherwise,} \end{cases} \tag{4.32}$$

$$W_{ik}^{zu} = \begin{cases} H & \text{if } \lambda'(q'_j, \sigma_k) = \gamma'_i \text{ for some } q'_j \in Q' \\ 0 & \text{otherwise,} \end{cases} \tag{4.33}$$

$$W_i^z = -\frac{3H}{2} \; \forall i = 1, \ldots, n_Z. \tag{4.34}$$

The construction works similarly to the next-state function: a hidden neuron in the output section of the net, i, will be excited to a high state only if both a state neuron j such that $\lambda'(q'_j, \sigma_l) = \gamma'_i$ for some σ_l is high *and* an input k such that $\lambda'(q'_l, \sigma_k) = \gamma'_i$ for some q'_l is also high, because a bias of $-3H/2$ has to be surmounted. The split-state, split-output Mealy machines are such that this is possible only if $\lambda'(q'_j, \sigma_k) = \gamma'_i$; otherwise, the bias will bring the hidden unit i to a low state.

Finally, the output layer collects the activations of hidden output nodes to produce output that may be interpreted in an exclusive fashion. To achieve this, each output unit m is biased with $-H/2$ to remain in a low state unless it receives, through a weight of H, a high signal from one of the $|\Sigma|$ hidden units corresponding to symbol γ_m. The construction is as follows:

$$W_{ij}^{yz} = \begin{cases} H & \text{if } \gamma'_j = (\gamma_i, \sigma_k) \text{ for some } \sigma_k \in \Sigma \\ 0 & \text{otherwise,} \end{cases} \tag{4.35}$$

$$W_i^y = -\frac{H}{2} \; \forall i = 1, \ldots, n_Y. \tag{4.36}$$

[13] This next-state function construction differs from the one proposed by Kremer (1996) in that the same parameter H is used for all weights. Kremer used different values for each state unit, and the output interval of the logistic function $[0, 1]$ was divided in low, forbidden, and high interval in a symmetric fashion ($\epsilon_0 + \epsilon_1 = 1$).

In contrast to Alquézar & Sanfeliu's (1995) construction, the next-state weights and biases are given in terms of a single parameter (H) to simplify the construction, but it would not be difficult to use different parameters and define conditions to determine their values; the conditions for stability are derived from conditions D1 and D2 in section 3.2.

Condition D1: Correctness of the next state. Again, we want to find conditions to ensure that whenever the DTRNN is in any state $\mathbf{x}[t-1]$ in the region $X_j(\epsilon_0, \epsilon_1)$ corresponding to state q'_j and reads as input $\mathbf{u}[t]$ the vector $\mathbf{u}_k$ corresponding to symbol σ_k, it always produces a next state in region $X_i(\epsilon_0, \epsilon_1)$ if $\delta'(q'_j, \sigma_k) = q'_i$, according to equation 3.9.

Taking into account the weight and bias prescriptions—equations 4.29 through 4.31—and the special characteristics of the split-state transition function, equation 2.6 computes $x_i[t]$ as follows:

$$x_i[t] = g\left(\sum_{l \in C_{ik}} H x_l[t-1] - \frac{H}{2}\right), \tag{4.37}$$

identical to equation 4.5, where

$$C_{ik} = \{l\colon \delta'(q'_l, \sigma_k) = q'_i\} \tag{4.38}$$

is the set of the indices of states that may contribute to the activation of unit i (note that, due to the splitting of states, if state q'_i is reachable through transitions involving symbol σ_k, it cannot be reached through transitions involving other symbols). Obviously, $q'_j \in C_{ik}$. For $\mathbf{x}[t]$ to be in $X_i(\epsilon_0, \epsilon_1)$, equation 4.37 must fulfill $x_i[t] \geq \epsilon_1$. The worst case is, as in section 4.1.1, when $x_j[t-1]$ contributes the lowest possible high signal, ϵ_1 and all other signals are the lowest possible low signal, that is, $S_0 = 0$ (for any sigmoid function in the class considered). In that case, we recover equation 4.7. The conditions that ensure that all other $x_{i'}[t]$, $i' \neq i$ are low, that is, $x_{i'}[t] \leq \epsilon_0$, are identical to those discussed in section 4.1.1, and lead to a worst-case condition identical to condition 4.10.

Therefore, if conditions 4.7 and 4.10 and $S_0 < \epsilon_0 < \epsilon_1 < S_1$ are met, it is guaranteed that the augmented Robinson and Fallside network will always exhibit the next-state behavior of the corresponding Mealy machine.

Condition D2: Correctness of the output. To ensure that the output is correct, we have to ensure, first, that the units in the hidden layer represent the split output of the split-state, split-output Mealy machine M' correctly and, second, that the output layer collects these representations correctly into an output that may be interpreted in an exclusive fashion.

Let us call $Z = [S_0, S_1]^{n_Z}$ the set of possible values of the hidden output layer. A region $Z_i(\epsilon_0, \epsilon_1) \in Z$ will be assigned to each symbol γ'_i in Γ in the

same way as regions $Y_i(\epsilon_0, \epsilon_1) \in Y = [S_0, S_1]$ are assigned to each symbol γ_i in Γ (see equation 3.8).

First, we want to find conditions to ensure that whenever the DTRNN is in any state $\mathbf{x}[t-1]$ in the region $X_j(\epsilon_0, \epsilon_1)$ corresponding to state q'_j and reads as input $\mathbf{u}[t]$ the vector $\mathbf{u}_k$ corresponding to symbol σ_k, it always produces a hidden output $\mathbf{z}[t]$ in region $Z_i(\epsilon_0, \epsilon_1)$ if $\lambda'(q'_j, \sigma_k) = \gamma'_i$, according to equation 4.9.

Taking into account the weight and bias prescriptions—equations 4.32 through 4.34—equation 2.9 computes $z_i[t]$ as follows:

$$z_i[t] = g\left(\sum_{l \in D_{ik}} H x_l[t-1] - \frac{H}{2}\right), \tag{4.39}$$

very similar to equation 4.37, where

$$D_{ik} = \{l: \lambda'(q'_l, \sigma_k) = \gamma'_i\}. \tag{4.40}$$

Obviously, $q'_j \in C_{ik}$. For $\mathbf{z}[t]$ to be in $Z_i(\epsilon_0, \epsilon_1)$ equation 4.39 must fulfill $z_i[t] \geq \epsilon_1$. The worst case is, as in section 4.1.1, when $x_j[t-1]$ contributes the lowest possible high signal, ϵ_1 and all other signals are the lowest possible low signal, that is, $S_0 = 0$ (for any sigmoid function in the class considered). In that case, we recover equation 4.7. The conditions that ensure that all other $z_{i'}[t]$, $i' \neq i$ are low, that is, $z_{i'}[t] \leq \epsilon_0$, are identical to those discussed in section 4.1.1, and lead to a worst-case condition identical to condition 4.11 with

$$\chi_y = \max_{m,k} |D_{mk}|. \tag{4.41}$$

That is, if conditions 4.7 and 4.11 and $S_0 < \epsilon_0 < \epsilon_1 < S_1$ are met, then the DTRNN is guaranteed to produce a correct representation of output in the hidden output layer, provided that the state representations are correct.

Second, we have to ensure that the output layer collects the hidden output representations in a way that it produces a correct output. We have to find conditions ensuring that for any $\gamma'_j \in \Gamma'$, any $\mathbf{z}[t] \in Z_j(\epsilon_0, \epsilon_1)$ is mapped into an output $\mathbf{y}[t] \in Y_i$ if $\gamma'_j = (\gamma_i, \sigma_k)$ for some $\sigma_k \in \Sigma$.

Taking into account the weight and bias prescriptions (see equations 4.35 and 4.36), equation 2.8 computes the mth component of the output vector according to

$$y_m[t] = g\left(\sum_{\gamma'_l = (\gamma_m, \sigma_k)} H z_l[t] - \frac{H}{2}\right). \tag{4.42}$$

If $m = i$, the worst case for this equation (lowest possible value of $y_i[t]$) is when all $z_l[t]$ are 0 ($S_0 = 0$ is the worst case for all sigmoids) except for $z_j[t]$,

which is exactly ϵ_1. Since we want $y_i[t]$ to be high, that is, $y_i[t] \geq \epsilon_1$, we obtain the condition

$$g\left(H\left(\epsilon_1 - \frac{1}{2}\right)\right) \geq \epsilon_1, \tag{4.43}$$

which is identical to condition 4.7. If $m \neq i$, then we want $y_m[t]$ to be low. The worst case for equation 4.42 (highest value of $y_m[t]$) occurs when all $z_l[t]$ have the highest valid value, ϵ_0. Since we want $y_m[t]$ to be low, that is, $y_m[t] \leq \epsilon_0$, we obtain the condition

$$g\left(H\left(n_U\epsilon_0 - \frac{1}{2}\right)\right) \leq \epsilon_0. \tag{4.44}$$

As in section 4.1.1, the single condition, 4.14, ensures that all three conditions—4.10, 4.11, and 4.44—are simultaneously fulfilled, but in this case

$$\chi = \max(\chi_x, \chi_y, n_U). \tag{4.45}$$

Therefore, if H, ϵ_0 and ϵ_1 are chosen according to conditions 4.7, 4.14, and 4.45, and $S_0 < \epsilon_0 < \epsilon_1 < S_1$, then the augmented Robinson and Fallside network behaves like the corresponding Mealy machine. In the case of the logistic function, $g(x) = g_L(x)$, the results obtained for H are identical to those discussed in section 4.1.1.

5 Encoding of Moore Machines in DTRNN

5.1 Moore Machines in First-Order DTRNN. We will show that Moore machines can be encoded in Elman's (1990) simple recurrent nets, a class of first-order neural Moore machines whose dynamics is defined by equations 2.6 and 2.10. Elman nets have been proved to be computationally equivalent to any DFA if a step function is used (Kremer, 1995). The need for a specialized output layer was explained earlier by Goudreau et al. (1994), who showed that there are DFA whose computation cannot be represented on a first-order DTRNN without the additional layer. Alquézar and Sanfeliu (1995) adapted Minsky's (1967) construction and showed that an Elman network with sigmoid functions taking and returning rational numbers could stably represent a DFA. Here the construction is extended to real-valued sigmoid functions and Moore machines.

The split-state Moore machine M' equivalent to a Moore machine M is defined analogously to the split-state, split-output Mealy machine of section 4.2; the only difference is the output function, λ': $Q' \to \Gamma$, which is defined as follows:

$$\lambda'((q_i, \sigma_k)) = \lambda(q_i). \tag{5.1}$$

Accordingly, the encoding uses $n_U = |\Sigma|$ input lines as usual, $n_Y = |\Gamma|$ output lines, and $n_X = |\Sigma||Q|$ state units. Condition D1 in section 3 leads to the same conditions as in section 4.2, because the next-state function is constructed identically: equations 4.7 and 4.10 and $S_0 < \epsilon_0 < \epsilon_1 < S_1$. Therefore it suffices to consider condition D2 (correctness of output).

The output function is defined by the following weight scheme:

$$W_{ij}^{yx} = \begin{cases} H & \text{if } \lambda(q_j') = \gamma_i \\ 0 & \text{otherwise,} \end{cases} \tag{5.2}$$

$$W_i^y = -\frac{H}{2} \; i = 1, \ldots, n_Y, \tag{5.3}$$

so that an output unit m is biased to be in a low state unless one of the state units i such that $\lambda(q_i') = \gamma_m$ is high enough to surmount that bias.

We want to find conditions to ensure that, whenever the DTRNN has reached any state $\mathbf{x}[t]$ in the region $X_i(\epsilon_0, \epsilon_1)$ corresponding to state q_i', it will always produce an output vector $\mathbf{y}[t]$ in region $Y_m(\epsilon_0, \epsilon_1)$ if $\lambda'(q_i', \sigma_k) = \gamma_m$, according to equation 3.11.

Taking into account the weight and bias prescriptions (see equations 5.2 and 5.3), equation 2.10 computes the jth component of the output vector according to

$$y_j[t] = g\left(\sum_{l:\, \lambda'(q_l')=\gamma_j} H x_l[t] - \frac{H}{2}\right). \tag{5.4}$$

If $j = m$, the worst case for this equation (lowest possible value of $y_m[t]$) is when all $x_l[t]$ are 0 ($S_0 = 0$ is the worst case for all sigmoids) except for $x_i[t]$, which is exactly ϵ_1. Since we want $y_m[t]$ to be high, that is $y_m[t] \geq \epsilon_1$, we obtain a condition identical to condition 4.7. If $j \neq m$ then we want $y_j[t]$ to be low. The worst case for equation 5.4 (highest value of $y_j[t]$) occurs when all $x_l[t]$ have the highest valid value, ϵ_0. Since we want $y_j[t]$ to be low, that is, $y_j[t] \leq \epsilon_0$, we recover condition 4.11, with the maximum output fan-in χ_y defined as follows:

$$\chi_y = \max_m |D_m|, \tag{5.5}$$

where

$$D_m = \{l: \lambda(q_l') = \gamma_m\}. \tag{5.6}$$

As in section 4.1.1, a single condition, 4.14, ensures that both conditions 4.10 and 4.11 are simultaneously fulfilled. In the case of the logistic function, $g(x) = g_L(x)$, the results obtained for H are identical to those discussed in section 4.1.1.

5.2 Moore Machines in Second-Order DTRNN. A Moore machine may be straightforwardly encoded in a second-order Moore NSM (Blair & Pollack, 1997; Carrasco et al., 1996) with a next-state function defined by equation 2.4 and an output function defined by equation 2.10 using $n_U = |\Sigma|$, $n_X = |Q|$, $n_Y = |\Gamma|$:

- If the construction with biases defined in section 4.1.1 is used, then the conditions to be fulfilled are $S_0 < \epsilon_0 < \epsilon_1 < S_1$, equation 4.7 and equation 4.14, with $\chi = \max(\chi_x, \chi_y)$, χ_x defined by equation 4.8 and χ_y defined by equation 5.5 (but dropping the primes in equation 5.6). The values of H for the logistic sigmoid function $g_L(x)$ are identical to those obtained in section 4.1.1.
- If the construction without biases defined in section 4.1.2 is used, then the conditions to be fulfilled are $S_0 < \epsilon_0 < \epsilon_1 < S_1$, equation 4.22, equation 4.24 and equation 4.11, with χ_y defined as in equation 5.5. Note that conditions 4.22 and 4.24 define H as a function of the number of states $|Q|$, whereas condition 4.10 defines H as a function of the maximum output fan-in χ_y. To look for the minimum value of H ensuring both conditions for a given FSM, one may look for the minimum H in both sets of equations and then take the larger one. Note that even if $\chi_y \leq |Q|$, the values of H defined by conditions 4.22 and 4.24 are much lower, as discussed at the end of section 4.1.2, and therefore χ_y is likely to define H in many cases.

6 Encoding of Deterministic Finite Automata in DTRNN

DFA will be encoded in DTRNN as a special case of Mealy and Moore machines, with a single output neuron, as described in section 3, even when DFA may actually be seen as outputting two different symbols, $\mathcal{Y}$ (input string accepted) and $\mathcal{N}$ (input string not accepted). Any of the constructions described for Mealy and Moore machines are valid, but here the output neuron devoted to the rejection symbol $\mathcal{N}$ is eliminated. When using first-order DTRNN, Elman nets will be used as Moore NSM, and the DFA will be encoded as in section 5.1. When using second-order DTRNN, the most convenient constructions are those described in sections 4.1.1 and 4.1.2, where the DFA would be implemented as a Mealy machine. This section will detail only the weight and bias prescriptions for output, because the next-state functions are identical to those described in the sections mentioned.

6.1 Encoding DFA in Elman Nets. Encoding a DFA in an Elman net is accomplished in the same way as for a Moore machine, as described in section 5.1. The construction is therefore very similar to that used by Alquézar and Sanfeliu (1995): a new split DFA $M' = (Q', \Sigma, \delta', q'_I, F')$ is built from $M = (Q, \Sigma, \delta, q_i, F)$ (see section 2.1.3) by splitting states (Q', δ', and q'_I are defined as in section 4.2 and $F' = \{(q_i, \sigma_k)\colon q_i \in F\}$; then the next-

state function is encoded according to equations 4.29 through 4.31, and the output function is constructed as follows:

$$W_{1j}^{yx} = \begin{cases} H & \text{if } q_j' \in F' \\ 0 & \text{otherwise,} \end{cases} \tag{6.1}$$

and

$$W_1^y = -\frac{H}{2}. \tag{6.2}$$

The definition of χ_y used here is $\chi_y = |F'|$, instead of equation 5.5. The smallest values of H guaranteeing this construction with the logistic sigmoid $g_L(x)$ are therefore identical to those obtained in section 4.1.1.

6.2 Encoding DFA in Second-Order DTRNN. Encoding a DFA in a second-order net with biases is accomplished in the same way as for a Mealy machine, as described in section 4.1.1; the construction is therefore very similar to that used by Omlin and Giles (1996a, 1996b) but not identical. The next-state function is encoded according to equations 4.1 and 4.3, and the output function is constructed as follows:

$$W_{1jk}^{yxu} = \begin{cases} H & \text{if } \delta(q_j, \sigma_k) \in F \\ 0 & \text{otherwise,} \end{cases} \tag{6.3}$$

$$W_1^y = -\frac{H}{2}. \tag{6.4}$$

The definition of χ_y used here is $\chi_y = |F|$, instead of equation 4.12. The smallest values of H guaranteeing this construction with the logistic sigmoid $g_L(x)$ are therefore identical to those obtained in section 4.1.1.

Encoding a DFA in a second-order net without biases is accomplished in the same way as for a Mealy machine, as described in section 4.1.2. The next-state function is encoded according to equation 4.19, and the output function is constructed as follows:

$$W_{1jk}^{yxu} = \begin{cases} H & \text{if } \delta(q_j, \sigma_k) \in F \\ -H & \text{otherwise.} \end{cases} \tag{6.5}$$

The smallest values of H guaranteeing this construction with the logistic sigmoid $g_L(x)$ are those given in Table 2.

7 Experimental Results

In this section we show results of experiments performed to estimate the minimum value of H needed to ensure correct encodings of randomly generated Mealy and Moore machines. For each machine, we looked experimentally for this value by searching for $H(L)$, the minimum value of H needed to ensure the correct behavior of the network for strings of length L.

The experimental results shown in this section are not intended to prove the validity of the constructions described in the previous sections, but instead aim at illustrating it and also at showing that the upper bounds given by the theoretical treatment are in many cases very high, due to the imposition of necessary conditions based on worst-case situations that are not expected to occur for most FSM and DTRNN.

Experimental results will be shown for the following constructions:

- Mealy machines in second-order Mealy NSM, with and without bias (described in sections 4.1.1 and 4.1.2)
- Mealy machines in augmented Robinson and Fallside networks (first-order Mealy NSM, described in section 4.2)
- Moore machines in Elman nets (described in section 5.1)
- Moore machines in second-order Moore NSM with biases (described in section 5.2)

We have decided not to show any results for deterministic finite automata because they may be easily treated as special cases of Mealy and Moore machines, as shown in section 6.

The results of three kinds of experiments will be shown:

- Experiments to estimate a value of H for small FSM by running all input strings up to a given length and depicting the asymptotic behavior of H (section 7.1)
- Experiments to estimate a value of H for small FSM from random sets of very long strings
- Experiments to show the correct behavior of H for large FSM

All of the experiments will be performed on binary input and output alphabets in view of the result that the values of H do not depend on the size of the alphabet.

7.1 Asymptotical Values of H for Small FSM. This section describes the results of experiments performed to estimate the minimum value of H needed to ensure a correct encoding of a finite-state machine in each of the encodings studied in this article. This value can only be estimated, since there is just one obvious way of performing the experiments: directly searching (to a given precision, 0.01 in our calculations) for the minimum value of H needed to ensure correct behavior of the network for strings of increasing length (this value is $H(L)$). The network is taken to be behaving correctly for a given length L when the DTRNN produces correct output for all strings of length L. In our experiments in this section, $L = 1, \ldots, 20$.

We use two criteria to define a correct output of the network: a weak criterion, using a forbidden interval of zero measure ($\epsilon_0 = \epsilon_1 = 0.5$), and a strong criterion, using the theoretical forbidden interval computed for each

network ($S_0 < \epsilon_0 < \epsilon_1 < S_1$). As will be shown, the value of $H(L)$, increases with L, is always smaller than or equal to the theoretical upper bound H_{th} estimated for the corresponding construction for all the lengths tested (up to length 20), and seems to tend to stabilize around a value

$$H_{exp} = \max_{L>0} H(L) \tag{7.1}$$

(the overall shape of $H(L)$ vaguely resembles a noisy hyperbolic tangent in most cases).

For each of the encoding schemes studied, 10 FSM (5 Mealy and 5 Moore machines) with $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1\}$ with a given number of states $|Q|$ or with a given fan-in χ (depending on the encoding) were randomly generated, and the values of $H(L)$ for $L = 1, 2, \ldots, 20$ were computed to a precision of 0.01 for each one of the FSM. The number of states (resp., fan-in) of the generated automata was $|Q|$ (resp., χ) $= 2, 3, 4, 7, 10$. We have found the same asymptotical behavior for all the size parameter values used in the experiments.

We have found that the values of H obtained in our simulations are in general very insensitive to the precision with which the calculations of the states of each unit are performed (as tested using a parameter to control the number of significant digits returned by the sigmoid $g_L(x)$). The only problematic case occurred when the minimum $H(L)$ appeared to be a positive infinitesimal, that is, almost zero, for any length (this happens for two-state Mealy machines in the dense-weight version of the 2ODTRNN encoding, section 4.1.1, even though the theory prescribes an upper bound of 2); in this last case, the results were strongly dependent on precision. We found indeed that $H(L)$ was a decreasing function of precision, and that it could be made as small as desired by increasing the number of significant figures used in the calculations.

In Figures 3 to 7 we show the results obtained for the FSM of size 10 (number of states $|Q|$ or fan-in χ, depending on the construction). In these figures we represent normalized values of $H(L)$ (i.e., $H(L)/H_{\text{th}}$, where $H(L)$ is the experimental value of H obtained for each string length and H_{th} is the theoretical value of H obtained for each encoding). We show the results using error bars with the minimum, average, and maximum value of the normalized experimental H obtained for each encoding and string length L, $L = 1, 2, \ldots, 20$. The results for the other FSM of sizes 2, 3, 4, and 7 are very similar; Table 3 summarizes the results for all these FSM, normalized as $H(20)/H_{\text{th}}$.

It is important to comment on the fact that when the forbidden interval is enforced, some constructions, all of them based in condition 4.7 (first-order constructions and second-order constructions with biases), show experimental values of H that are equal to the theoretical value. As discussed at the end of section 4.1.1, this happens for some kinds of FSM with loops and is due to the fact that for the theoretical values of H and ϵ_1, a fixed-point

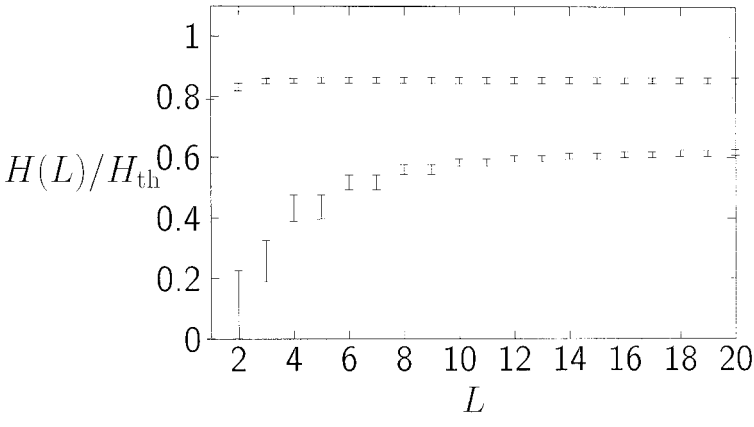

Figure 3: Minimum and maximum values of $H(L)/H_{\text{th}}$ for five randomly generated Mealy machines of $|Q| = 10$ encoded in second-order Mealy NSM without bias. Upper lines: theoretical ϵ_0 (see equation 4.25). Lower lines: $\epsilon_0 = 0.5$.

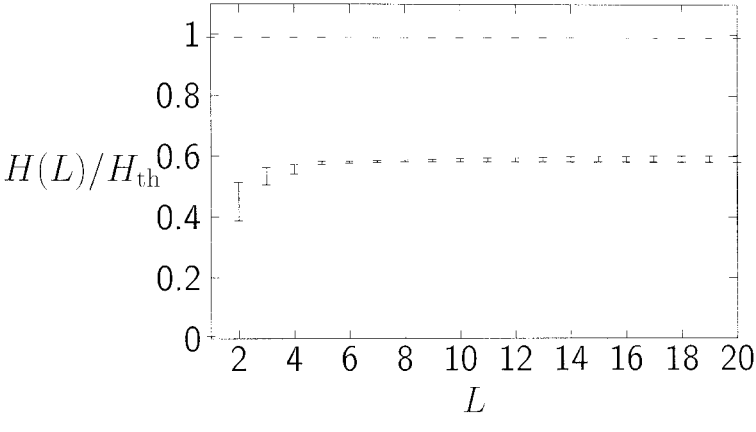

Figure 4: Minimum and maximum values of $H(L)/H_{\text{th}}$ for five randomly generated Mealy machines with $\chi = 10$ encoded in second-order Mealy NSM with bias. Upper lines: theoretical ϵ_0 and ϵ_1 (see equations 4.7 and 4.14). Lower lines: $\epsilon_0 = \epsilon_1 = 0.5$.

attractor exists at the very edge of the forbidden interval and any value of H smaller than the theoretical value moves this attractor into the forbidden interval.

7.2 Experiments with Long Strings. In this section we show experiments to test the stability of the encodings described when long strings are

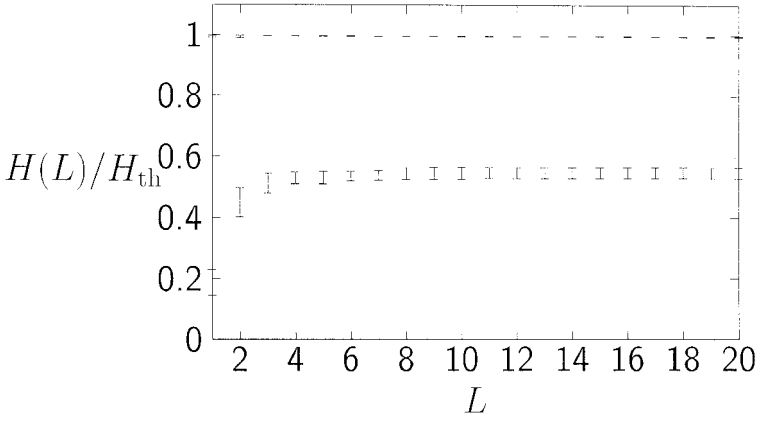

Figure 5: Minimum and maximum values of $H(L)/H_{\rm th}$ for five randomly generated Mealy machines with $\chi = 10$ encoded in first-order Mealy NSM. Upper lines: theoretical ϵ_0 and ϵ_1 (see equations 4.7 and 4.14). Lower lines: $\epsilon_0 = \epsilon_1 = 0.5$.

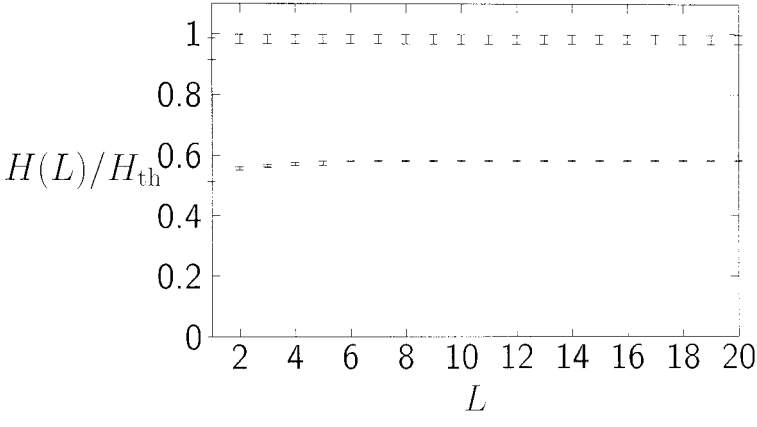

Figure 6: Minimum and maximum values of $H_{\rm exp}$ for five randomly generated Moore machines with $\chi = 10$ encoded in second-order Moore NSM with bias. Upper lines: theoretical ϵ_0 and ϵ_1 (see equations 4.7 and 4.14). Lower lines: $\epsilon_0 = \epsilon_1 = 0.5$.

Table 3: Minimum, Maximum, and Average $H(20)/H_{\text{th}}$ Values for 25 Randomly Generated Mealy and Moore Machines of sizes 2, 3, 4, 7, and 10 and All Strings Up to Length 20.

Architecture	Moore machines					
	$H(20)/H_{\text{th}}$ for $\epsilon_0 = \epsilon_1 = 0.5$			$H(20)/H_{\text{th}}$ for Theoretical ϵ_0 and ϵ_1		
	Minimum	Average	Maximum	Minimum	Average	Maximum
First order	0.0025	0.4307	0.6912	0.6982	0.9557	1.0000
Second order with bias	0.0050	0.5307	0.7636	0.9137	0.9742	1.0000
Architecture	Mealy machines					
	$H(20)/H_{\text{th}}$ for $\epsilon_0 = \epsilon_1 = 0.5$			$H(20)/H_{\text{th}}$ for Theoretical ϵ_0 and ϵ_1		
	Minimum	Average	Maximum	Minimum	Average	Maximum
First order	0.1456	0.5320	0.7872	0.9318	0.9931	1.0000
Second order with bias	0.0078	0.5360	0.7622	0.9319	0.9829	1.0000
Second order without bias	0.0000	0.3748	0.7710	0.0250	0.7285	0.8995

Note: The left and right parts of the table differ in what "correctly classified" means: correct side of [0, 1] with no forbidden interval ($\epsilon_0 = \epsilon_1 = 0.5$) on the left; inside predicted high and low intervals for stable behavior on the right.

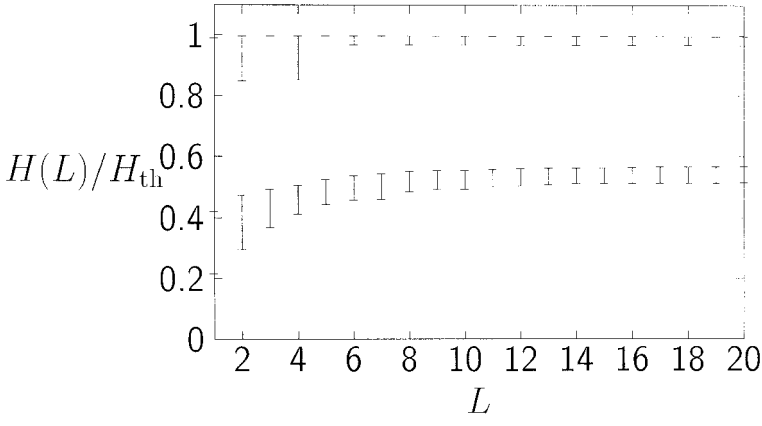

Figure 7: Minimum and maximum values of H_{exp} for five randomly generated Moore machines with $\chi = 10$ encoded in Elman nets (first-order NSM). Upper lines: theoretical ϵ_0 and ϵ_1 (see equations 4.7 and 4.14). Lower lines: $\epsilon_0 = \epsilon_1 = 0.5$.

presented to the network. We have used the same FSM as in the previous section and looked for the minimum H needed to process correctly 200 randomly generated strings of length 2000 (the same set of strings for all the experiments). We call this value H_{2000}.

The values of H obtained in these experiments are the same values obtained in the previous section or slightly larger. The maximum deviation from the experimental H obtained for $L = 20$ is 0.02 for all the encodings. This illustrates that the value of $H(L)$ seems to stabilize around an asymptotical value that is very close to the value obtained for $L = 20$.

In Table 4 we summarize these results: normalized minimum, average, and maximum experimental values of H_{2000} for all the FSM and DTRNN encodings described, using the same FSM as in section 7.1. As in this section, we used two criteria to define a correct output of the network: a weak criterion (using a forbidden interval of zero measure) and a strong criterion (using a forbidden interval with the theoretical ϵ_0 and ϵ_1 obtained in the previous sections). As in the experiments in section 7.1, we can see that the values of H_{2000}/H_{th} obtained with the weak criterion are lower than the values of H_{2000}/H_{th} obtained with the strong criterion. For the same encodings as in section 7.1 (first order and second order with bias encodings), we found that the maximum value of H_{2000} is exactly the theoretical upper limit H_{th} obtained for the encoding, and the average H_{2000}/H_{th} is very close to 1.0. Most of the minimum values of H_{2000}/H_{th} are very close to zero because the second-order constructions without biases for FSM of size 2 need a very low value of H (see note 12).

Table 4: Minimum, Maximum, and Average Values of H_{2000}/H_{th} for 5 Randomly Generated Mealy and Moore Machines for 200 Strings of Length 2000.

Architecture	Moore Machines					
	H_{2000}/H_{th} for $\epsilon_0 = \epsilon_1 = 0.5$			H_{2000}/H_{th} for Theoretical ϵ_0 and ϵ_1		
	Minimum	Average	Maximum	Minimum	Average	Maximum
First order	0.0014	0.4900	0.7218	0.6982	0.9582	1.0000
Second order with bias	0.0014	0.5698	0.7344	0.9319	0.9763	1.0000
Architecture	Mealy Machines					
	H_{2000}/H_{th} for $\epsilon_0 = \epsilon_1 = 0.5$			H_{2000}/H_{th} for Theoretical ϵ_0 and ϵ_1		
	Minimum	Average	Maximum	Minimum	Average	Maximum
First order	0.0113	0.5137	0.8149	0.9319	0.9938	1.0000
Second order with bias	0.4604	0.6115	0.7191	0.9374	0.9869	1.0000
Second order without bias	0.0161	0.5748	1.0000	0.0250	0.7740	1.0000

7.3 Experiments with Large FSM. We have made experiments to test the stability of the encodings with large FSM. We generated randomly 10 automata (5 Mealy and 5 Moore machines) with $|Q| = 200$ and, eliminating the useless states, we obtained automata of sizes in the range 81–165. Using the same long strings as in section 7.2, we obtained the minimum H_{2000} needed to obtain the correct outputs, using the weak and the strong criterion. The results obtained are shown in Table 5. In this table we show the size parameters for each construction (minimum, average, and maximum) and the normalized H_{2000} with the weak and strong criterion (minimum, average, and maximum).

In all the experiments we can see that the experimental values of H_{2000} obtained are always lower than the theoretical H computed in previous sections for each architecture (i.e., $H_{2000}/H_{\text{th}} \leq 1$). The main difference with the results reported in the previous section is that automata having very low fan-in and therefore leading to very small values of H_{th} are now very unlikely to appear. The results reported in this section illustrate again that the upper bounds given by the theory are sufficient to ensure a stable behavior in the network even when the network size and string length are large.

8 Conclusion

We have completed the results by Omlin and Giles (1996a, 1996b) on stable encoding of FSM on DTRNN to a larger family of sigmoids (in particular, to any positive, strictly growing, continuous, bounded activation function), a larger variety of DTRNN architectures (including various first-order and second-order networks), and a wider class of FSM architectures (Mealy and Moore machines), by establishing a simplified procedure to prove the stability of encoding schemes based in the classical one-hot representation of states in the hidden units of DTRNN. A summary of the encodings described in the paper is presented in Table 6.

Our formulation is based on ideas put forward by Casey (1996) and earlier by Pollack (1991). To justify the constructions, we have found it very natural to speak about DTRNN in terms of Mealy and Moore neural state machines. These constructions are based on bounding criteria, in contrast to the one by Omlin and Giles (1996a), which relies on a detailed analysis of fixed-point attractors (although, as one of the referees pointed out, the only way for the activations of a neural unit to stay near one of the two saturation values is through the existence of a near-saturation attractive fixed point (Li, 1992)).

We have performed extensive numerical experiments for the standard logistic function that illustrate the validity of the results and show that for some constructions, the weights prescribed by the theory are larger than necessary, due to the fact that the worst cases used to derive them seldom occur.

We plan to expand the work to study distributed—as opposed to one-hot—encodings of state. Some of the results presented here have already

Table 5: Minimum, Average, and Maximum Values of H_{2000}/H_{th} Obtained for Five Randomly Generated Mealy and Moore Machines with $|Q|$ up to 200 and Eliminating Useless States.

Moore Machines									
Architecture	Size			H_{2000}/H_{th} for $\epsilon_0 = \epsilon_1 = 0.5$			H_{2000}/H_{th} for Theoretical ϵ_0 and ϵ_1		
	Minimum	Average	Maximum	Minimum	Average	Maximum	Minimum	Average	Maximum
First order	104	118	136	0.5442	0.5640	0.5725	0.9180	0.9231	0.9257
Second order with bias	78	87	97	0.6285	0.6313	0.6343	0.9452	0.9458	0.9464
MEALY MACHINES									
Architecture	Size			H_{2000}/H_{th} for $\epsilon_0 = \epsilon_1 = 0.5$			H_{2000}/H_{th} for Theoretical ϵ_0 and ϵ_1		
	Minimum	Average	Maximum	Minimum	Average	Maximum	Minimum	Average	Maximum
First order	100	110.4	116	0.5910	0.5956	0.6030	0.8850	0.8862	0.8869
Second order with bias	81	86	94	0.6294	0.6316	0.6346	0.9451	0.9457	0.9462
Second order without bias	149	157.4	165	0.7242	0.7473	0.7796	0.8776	0.8802	0.8860

Note: We used 200 randomly generated strings of length 2000.

Table 6: Summary of the Encodings Proposed in the Article.

FSM Class	DTRNN Architecture	Number of Hidden Units	Low Weight	High Weight	Bias	H determined by	Section
Mealy	Second order	$\|Q\|$	0	H	$-H/2$	fan-in	4.1.1
Mealy	Second order	$\|Q\|$	$-H$	H	0	$\|Q\|$	4.1.2
Mealy	Robinson-Fallside	$(\|Q\| + \|\Gamma\|)\|\Sigma\|$	0	H	$-3H/2$ or $-H/2$	Fan-in	4.2
Moore	Elman	$\|Q\|\|\Sigma\|$	0	H	$-3H/2$ or $-H/2$	Fan-in	5.1
Moore	Second order	$\|Q\|$	0	H	$-H/2$	Fan-in	5.2
Moore	Second order	$\|Q\|$	$-H$ or 0	H	0 or $-H/2$	Fan-in and $\|Q\|$	5.2

been used by some of us (Kremer et al., 1998) to constrain the training of DTRNN so that they learn to behave as FSM from samples, to encode nondeterministic finite automata into second-order DTRNN (Carrasco et al., 1999), and to encode an extension of Mealy machines into second-order DTRNN (Ñeco, Forcada, Carrasco, & Valdés-Muñoz, 1999).

The results we have presented are related to those obtained by Šíma (1997), who proves that the behavior of any DTRNN using threshold activation functions may be stably emulated by another DTRNN using activation functions in a very general class that includes the sigmoid functions considered in this article. As one of the referees has pointed out, this result may be combined with existing results concerning the encoding of finite-state machines in threshold DTRNN:

- A result by Šíma and Wiedermann (1998) shows that any regular expression of length l may be recognized by a threshold DTRNN having $\Theta(l)$ units. The result is that a regular expression may be very efficiently encoded in analog DTRNN having activation functions more general than ours. The main difference between this result and ours is that we are interested in encoding finite-state machines directly.
- Alon et al. (1991), Indyk (1995), and Horne and Hush (1996) show that finite-state machines with n states may be encoded in threshold DTRNN having a number of units that grows sublinearly with n. Combining this with Šíma's (1997) result, we obtain that finite-state machines may be sublinearly enncoded in analog DTRNN. However, Alon et al.'s (1991) and Horne and Hush's (1996) results, though more efficient, are not as directly applicable to a given FSM as ours.
- All of the constructions presented in this article work may also be straightforwardly applied to threshold DTRNN using $H = 1$. Šíma's (1997) result may convert these threshold DTRNN into sigmoid DTRNN. The resulting weights are basically twice as large as the ones obtained in this article. This is due to the fact that Šíma's construction is valid for general threshold DTRNN, whereas our result takes advantage of the fact that states and inputs are encoded in simple one-hot fashion.

We provide a detailed description on how to construct stable encodings of a variety of FSM into a variety of commonly used first- and second-order DTRNN and a way to choose weights that are relatively small, which may be of interest when the construction will be followed by gradient-descent learning. Weights are still rather large for gradient-descent learning, but as section 7 shows, smaller values of H may still guarantee correct behavior.

Acknowledgments

This work has been supported by the Spanish Comision Interministerial de Ciencia y Tecnología through grant TIC97-0941. We also thank C. Lee

Giles, Stefan C. Kremer, Christian W. Omlin, José Oncina, and all of the anonymous referees for their critical comments and ideas.

References

Alon, N., Dewdney, A. K., & Ott, T. J. (1991). Efficient simulation of finite automata by neural nets. *Journal of the Association of Computing Machinery, 38*, 495–514.

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

Arai, K.-I., & Nakano, R. (1996). Annealing RNN learning of finite state automata. In *Proceedings of ICANN'96* (pp. 519–524).

Arai, K.-I., & Nakano, R. (1997). Adaptive β-scheduling learning method of finite state automata by recurrent neural networks. In *Progress in connectionist-based information systems: Proceedings of the 1997 International Conference on Neural Information Processing and Intelligent Information Systems* (Vol. 1, pp. 351–354). Singapore: Springer-Verlag.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks, 5*(2), 157–166.

Blair, A., & Pollack, J. B. (1997). Analysis of dynamical recognizers. *Neural Computation, 9*(5), 1127–1142.

Carrasco, R. C., Forcada, M. L., & Santamaría, L. (1996). Inferring stochastic regular grammars with recurrent neural networks. In L. Miclet & C. de la Higuera (Eds.), *Grammatical inference: Learning syntax from sentences* (pp. 274–281). Berlin: Springer-Verlag.

Carrasco, R. C., Oncina, J., & Forcada, M. L. (1999). Efficient encodings of finite automata in discrete-time recurrent neural networks. In *Proceedings of ICANN'99, International Conference on Artificial Neural Networks* (Vol. 2, pp. 673–677).

Casey, M. (1996). The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation, 8*(6), 1135–1178.

Casey, M. (1998). Correction to proof that recurrent neural networks can robustly recognize only regular languages. *Neural Computation, 10*(5), 1067–1069.

Cleeremans, A., Servan-Schreiber, D., & McClelland, J. L. (1989). Finite state automata and simple recurrent networks. *Neural Computation, 1*(3), 372–381.

Das, S., & Das, R. (1991). Induction of discrete state-machine by stabilizing a continuous recurrent network using clustering. *Computer Science and Informatics, 21*(2), 35–40.

Das, S., & Mozer, M. (1998). Dynamic on-line clustering and state extraction: An approach to symbolic learning. *Neural Networks, 11*(1), 53–64.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*, 179–211.

Floyd, T. L. (1996). *Digital fundamentals* (6th ed.). Englewood Cliffs, NJ: Prentice-Hall.

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

Frasconi, P., Gori, M., Maggini, M., & Soda, G. (1996). Representation of finite-state automata in recurrent radial basis function networks. *Machine Learning, 23*, 5–32.

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

Gori, M., Maggini, M., Martinelli, E., & Soda, G. (1998). Inductive inference from noisy examples using the hybrid finite state filter. *IEEE Transactions on Neural Networks, 9*(3), 571–575.

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

Haykin, S. (1998). *Neural networks—A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). 1991. *Introduction to the theory of neural computation*. Redwood City, CA: Addison-Wesley.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages, and computation*. Reading, MA: Addison–Wesley.

Horne, B. G., & Giles, C. L. (1995). An experimental comparison of recurrent neural networks. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems, 7* (pp. 697–704). Cambridge, MA: MIT Press.

Horne, B. G., & Hush, D. R. (1996). Bounds on the complexity of recurrent neural network implementations of finite state machines. *Neural Networks, 9*(2), 243–252.

Hush, D., & Horne, B. (1993). Progress in supervised neural networks. *IEEE Signal Processing Magazine, 10*(1), 8–39.

Indyk, P. (1995). Optimal simulation of automata by neural nets. In *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science* (pp. 337–348). Berlin: Springer-Verlag.

Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In C. E. Shannon & J. McCarthy (Eds.), *Automata studies* Princeton, NJ: Princeton University Press.

Kolen, J. F. (1994). Fool's gold: Extracting finite state machines from recurrent network dynamics. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems, 6* (pp. 501–508). San Mateo, CA: Morgan Kaufmann.

Kolen, J. F., & Pollack, J. B. (1995). The observer's paradox: Apparent computational complexity in physical systems. *Journal of Experimental and Theoretical Artificial Intelligence, 7*, 253–277.

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

Kremer, S. (1996). *A theory of grammatical induction in the donnectionist paradigm*. Unpublished Ph.D. dissertation, University of Alberta, Edmonton, Alberta.

Kremer, S., Ñeco, R. P., & Forcada, M. L. (1998). Constrained second-order recurrent networks for finite-state automata induction. In L. Niklasson, M. Bodén, & T. Ziemke (Eds.), *Proceedings of the 8th International Conference on Artificial Neural Networks ICANN'98* (Vol. 2, pp. 529–534). London: Springer.

Li, L.K. (1992). Fixed point analysis for discrete-time recurrent neural networks. In *Proceedings of IJCNN* (Vol. 4, pp. 134–139).

Maass, W., & Orponen, P. (1998). On the effect of analog noise in discrete-time analog computations. *Neural Computation, 10*(5), 1071–1095.

Maass, W., & Sontag, E. D. (1999). Analog neural nets with gaussian or other common noise distribution cannot recognize arbitrary regular languages. *Neural Computation, 11*(3), 771–782.

Manolios, P., & Fanelli, R. (1994). First order recurrent neural networks and deterministic finite state automata. *Neural Computation, 6*(6), 1154–1172.

Maskara, A., & Noetzel, A. (1992). Forcing simple recurrent neural networks to encode context. In *Proceedings of the 1992 Long Island Conference on Artificial Intelligence and Computer Graphics*.

McCulloch, W. S., & Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115–133.

Miller, C. B., & Giles, C. L. (1993). Experimental comparison of the effect of order in recurrent neural networks. *International Journal of Pattern Recognition and Artificial Intelligence, 7*(4), 849–872.

Minsky, M. L. (1967). *Computation: Finite and infinite machines*. Englewood Cliffs, NJ: Prentice-Hall.

Ñeco, R. P., & Forcada, M. L. (1996). Beyond Mealy machines: Learning translators with recurrent neural networks. In *Proceedings of the World Conference on Neural Networks '96* (pp. 408–411). San Diego.

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

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

Omlin, C. W., & Giles, C. L. (1996b). Stable encoding of large finite-state automata in recurrent neural networks with sigmoid discriminants. *Neural Computation, 8*, 675–696.

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning, 7*, 227–252.

Robinson, T., & Fallside, F. (1991). A recurrent error propagation network speech recognition system. *Computer Speech and Language, 5*, 259–274.

Sanfeliu, A., & Alquézar, R. (1994). Active grammatical inference: A new learning methodology. In D. Dori & A. Bruckstein (Eds.), *Shape and structure in pattern recognition*. Singapore: World Scientific.

Šíma, J. (1997). Analog stable simulation of discrete neural networks. *Neural Network World, 7*, 679–686.

Šíma, J., & Wiedermann, J. (1998). Theory of neuromata. *Journal of the ACM, 45*(1), 155–178.

Tiňo, P., Horne, B. G., Giles, C. L., & Colingwood, P. C. (1998). Finite state machines and recurrent neural networks—automata and dynamical systems approaches. In J. E. Dayhoff & O. Omidvar (Eds.), *Neural networks and pattern recognition* (pp. 171–220). San Diego, CA: Academic.

Tiňo, P., & Sajda, J. (1995). Learning and extracting initial Mealy automata with a modular neural network model. *Neural Computation, 7*(4).

Tsoi, A. C., & Back, A. (1997). Discrete time recurrent neural network architectures: A unifying review. *Neurocomputing, 15*, 183–223.

Watrous, R. L., & Kuhn, G. M. (1992). Induction of finite-state languages using second-order recurrent networks. *Neural Computation, 4*(3), 406–414.

Zeng, Z., Goodman, R. M., & Smyth, P. (1994). Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks, 5*(2), 320–330.

Zeng, Z., Goodman, R. M., & Smyth, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Computation, 5*(6), 976–990.

Received November 16, 1998; accepted September 23, 1999.