# An Algebraic Framework to Represent Finite State Machines in Single-Layer Recurrent Neural Networks

R. Alquézar and A. Sanfeliu

Institut de Cibernètica (UPC - CSIC)

Diagonal 647, 2a, 08028 Barcelona (Spain)

**Abstract**

In this paper we present an algebraic framework to represent finite state machines (FSMs) in single-layer recurrent neural networks (SLRNNs), which unifies and generalizes some of the previous proposals. This framework is based on the formulation of both the state transition function and the output function of a FSM as a linear system of equations, and it permits an analytical explanation of the representational capabilities of first-order and higher-order SLRNNs. The framework can be used to insert symbolic knowledge in RNNs prior to learning from examples and to keep this knowledge while training the network. This approach is valid for a wide range of activation functions, whenever some stability conditions are met. The framework has already been used in practice in a hybrid method for grammatical inference reported elsewhere (Sanfeliu and Alquezar, 1994).

## 1 Introduction

The representation of finite-state machines (FSMs) in recurrent neural networks (RNNs) has attracted the attention of researchers for several reasons, ranging from the pursuit of hardware implementations to the integration (and improvement) of symbolic and connectionist approaches to grammatical inference and recognition. Some previous works (Minsky 1967; Alon *et al.* 1991; Goudreau *et al.* 1994) have shown how to build different RNN models, using hard-limiting activation functions, that perfectly simulate a given finite state machine. None of these approaches yields the minimum size RNN which is required.

Minsky's method (1967) uses a recurrent layer of McCulloch-Pitts' units to implement the state transition function, and a second layer of OR gates to cope with the output function; the recurrent layer has $n \times m$ units, where $n$ is the number of states and $m$ is the number of input symbols. The method by Alon *et al.* (1991) uses a three-layer recurrent network which needs a number of threshold cells of order $n^{3/4} \times m$. Recently, Goudreau *et al.* (1994) have proven that, while second-order single-layer RNNs (SLRNNs) can easily implement any $n$-state automaton using $n$ recurrent units (and a total number of $n^2 \times m$ weights), first-order SLRNNs have limited representational capabilities. Other studies have been devoted to the design of methods for incorporating symbolic knowledge into RNNs made up of sigmoid activation units, both for first-order (Frasconi *et al.* 1991) and second-order (Omlin and Giles 1992) RNNs. This may yield faster learning and better generalization performance, as it permits a partial substitution of training data by symbolic rules, when compared with full inductive approaches that infer finite-state automata from examples (Pollack 1991; Giles *et al.* 1992).

This paper introduces a linear model for FSM representation in SLRNNs (Section 2), which improves the models reported elsewhere (Sanfeliu and Alquezar 1992; Alquezar and Sanfeliu 1993). A study of the analytical conditions that are needed to ensure the stability of the state representation is included. This new model unifies and generalizes some of the previous proposals (Minsky 1967; Goudreau *et al.* 1994) and explains the limitations of first-order SLRNNs (Section 3). A related method for inserting symbolic knowledge prior to learning in RNNs, which is valid for a wide class of activation functions, is described (Section 4). This method has been used in a hybrid approach for grammatical inference reported recently (Sanfeliu and Alquezar 1994).

# 2 The FS-SLRNN Linear Model of Finite State Machine Representation in Single-Layer Recurrent Neural Networks

## 2.1 Basic definitions and background

In the following we consider Single-Layer Recurrent Neural Networks (SLRNNs)[1], such as the one shown in Figure 1. An SLRNN has $M$ inputs, which are labelled $x_1, x_2, ..., x_M$, and a

[1]We approximately follow the notation that is described by Goudreau *et al.* (1994).

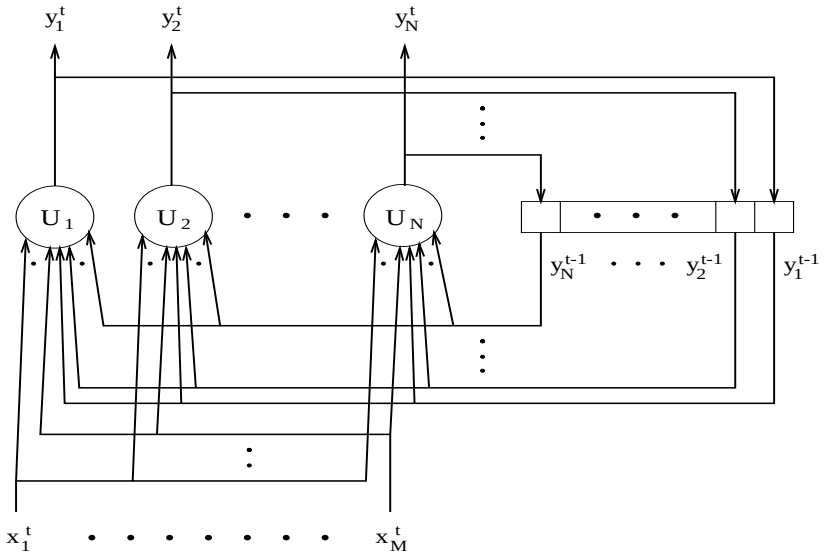


**Fig. 1** *Single-Layer Recurrent Neural Network (SLRNN) architecture.*

single-layer of $N$ units (or neurons) $U_1, U_2, ..., U_N$, whose output (or activation values) are labelled $y_1, y_2, ..., y_N$. The values at time $t$ of inputs $x_i$ $(1 \leq i \leq M)$ and unit outputs $y_j$ $(1 \leq j \leq N)$ are denoted by $x_i^t$ and $y_j^t$ respectively. The activation values of the neurons represent collectively the state of the SLRNN, which is stored in a bank of latches. Each unit computes its output value based on the current state vector $\mathbf{S}^t = [y_1^{t-1}, y_2^{t-1}, ..., y_N^{t-1}]^T$ and the input vector $\mathbf{I}^t = [x_1^t, x_2^t, ..., x_M^t]^T$, so the network is fully-connected. Some number $P$ of the neurons $(1 \leq P \leq N)$ can be used to supply an output vector $\mathbf{O}^t = [y_1^t, y_2^t, ..., y_P^t]^T$ in order to accomplish a given task. Those neurons that are not involved in the output function of the SLRNN are usually called hidden units.

The equations that describe the dynamic behavior of the SLRNN are:

$$\sigma_k^t \;=\; f\;(\mathbf{W}_k, \mathbf{I}^t, \mathbf{S}^t) \qquad \text{for } 1 \leq k \leq N \tag{1}$$

$$y_k^t \;=\; g\;(\sigma_k^t) \qquad \text{for } 1 \leq k \leq N, \tag{2}$$

where $f$ is a weighted sum of terms that combines inputs and received activations to give the net input values $\sigma_k$, $g$ is usually a non-linear function, and $\mathbf{W}_k$ is a vector of weights that are associated with the incoming connections of unit $U_k$. The SLRNNs can be classified according to the types of their $f$ and $g$ functions, which will be referred to as the aggregation and activation functions of the SLRNN, respectively.

The usual choices for function $f$ characterize the SLRNN either as first-order type:

$$f\ (\mathbf{W}_k, \mathbf{I}^t, \mathbf{S}^t)\ =\ \sum_{i=1}^{M} w_{ki} x_i^t\ +\ \sum_{j=1}^{N} w_{k(M+j)} y_j^{t-1} \tag{3}$$

or second-order type (Giles *et al.* 1992)[2]:

$$f\ (\mathbf{W}_k, \mathbf{I}^t, \mathbf{S}^t)\ =\ \sum_{i=1}^{M} \sum_{j=1}^{N} w_{kij}\ x_i^t\ y_j^{t-1} \tag{4}$$

The most common choices for the activation function $g$ are the hard-limiting threshold function

$$g_h\ (\sigma)\ =\ \begin{cases} 1 & \text{if } \sigma \geq 0 \\ 0 & \text{if } \sigma < 0 \end{cases} \tag{5}$$

and the sigmoid function

$$g_s\ (\sigma)\ =\ \frac{1}{1+e^{-\sigma}} \quad . \tag{6}$$

The former has been used mainly for implementation purposes (Minsky 1967) and to perform analytical studies of computability (Alon *et al.* 1991; Goudreau *et al.* 1994). The latter allows the use of gradient descent methods, such as the RTRL algorithm (Williams and Zipser 1989), to train the SLRNN to learn a sequential task (e.g. symbol prediction (Cleeremans *et al.* 1989) or string classification (Giles *et al.* 1992)).

A Mealy machine (Booth 1967) is an FSM defined as a quintuple $(I, O, S, \delta, \eta)$, where $I$ is a set of $m$ input symbols, $O$ is a set of $p$ output symbols, $S$ is a set of $n$ states, $\delta : I \times S \rightarrow S$ is the state transition function, and $\eta : I \times S \rightarrow O$ is the output function. In a Moore machine (Booth 1967), the output function depends only on the states (i.e. $\eta : S \rightarrow O$).

## 2.2 Construction of the linear model of FSM representation in SLRNNs

We now show that the representation of a finite-state machine (FSM) in an SLRNN can be modelled as two linear systems of equations. We refer to this algebraic representation as an FS-SLRNN (Finite State Single-Layer Recurrent Neural Network) model.

[2]In the first-order case, the SLRNN has $N \times (M+N)$ weights, while in the second-order case the number of weights rises to $N^2 \times M$.

First, we concentrate our attention on the state transition function $\delta$ of a FSM and assume that the SLRNN is just concerned with its implementation. Later on, we will discuss the representation of the output function $\eta$.

When an SLRNN is running at a discrete time step $t$, one can think of the $M$ input signals $x_i^t$ as encoding a symbol $a \in I$ of the machine input alphabet, the feedback of recurrent units $y_j^{t-1}$ as representing the current state $q \in S$ reached after the sequence of previous symbols, and the output of the $N$ neurons $y_j^t$ as standing for the destination state $q'$ that results from the current transition. Thus, the set of $N$ unit equations can be seen as implementing the state transition $\delta(a, q) = q'$ that occurs at time $t$. Since an FSM has a finite number $D$ of possible transitions (at most $D = mn$), at any given time step $t$, the network dynamic equations should implement one of them. Without loss of generality, let us assume that $\delta$ is defined for all the pairs $(a \in I, q \in S)$. Hence, if we number all the FSM state transitions $\delta(a, q)$ with an index $d$ $(1 \leq d \leq\ mn)$, the finite set of equations (7) should be satisfied by the weights of an equivalent SLRNN at any arbitrary time $t$.

$$y_{dk}^t \ = \ g \ ( \ f \ (\mathbf{W}_k, \mathbf{I}_d^t, \mathbf{S}_d^t) \ ) \quad \text{for } 1 \leq d \leq mn \quad \text{for } 1 \leq k \leq N \tag{7}$$

where $\mathbf{I}_d^t$ and $\mathbf{S}_d^t$ refer to the input and state vectors that encode the input symbol $a$ and current state $q$ of the $d$-th transition $\delta(a, q)$, respectively, and the activation values $y_{dk}^t$ $(1 \leq k \leq N)$ are related to the code of its destination state.[3]

The system of nonlinear equations (7) describes a complete deterministic state transition function $\delta$ where only the weights $\mathbf{W}_k$ $(1 \leq k \leq N)$ of the SLRNN are unknown. Due to the difficulty of studying nonlinear systems analytically, it is often desirable to convert them into linear systems if possible. In this case, we can transform equations (7) into a manageable linear system by performing the following steps:

1. Drop the time variable, i.e. convert the dynamic equations into static ones.
2. Use the inverse of the nonlinear activation function $g$.

The first step is justified by the fact that the set of equations (7) must be fulfilled for arbitrary time steps $t$. However, this simplification can be made as long as the stability of the state codes is

[3]It should be noted that the ordering defined by the index $d$ is static and arbitrary. In particular, we can use the following ordered list: $\delta(a_1, q_1)$, $\delta(a_1, q_2)$, .., $\delta(a_1, q_n)$, $\delta(a_2, q_1)$, ..., $\delta(a_m, q_n)$.

guaranteed (see next subsection). In addition, the SLRNN must be initialized to reproduce the code of the start state on the unit activation values before running any sequence of transitions.

Concerning the use of the inverse function $g^{-1}$, it must be satisfied that for all points $y$ which can appear in the state codes, either a unique $g^{-1}(y)$ exists, or if the inverse is not unique (as for the hard-limiter $g_h$) then there exists an established criterion for selecting the proper value in any case[4].

Therefore, the preceding transformations convert the nonlinear system of equations (7) into the linear system[5]:

$$\sigma_{dk} \;=\; g^{-1}(y_{dk}) \;=\; f\ (\mathbf{W}_k, \mathbf{I}_d, \mathbf{S}_d) \qquad \text{for } 1 \le d \le mn \quad \text{for } 1 \le k \le N \tag{8}$$

which can be described in matrix representation as

$$AW = B \tag{9}$$

where $A$ $(D \times E)$ is the array of the neuron inputs, $W$ $(E \times N)$ is the (transposed) weight array, $B$ $(D \times N)$ is the array of the neuron linear outputs, $D = mn$ is the number of transitions in the FSM, and $E$ is the number of inputs to each neuron.

For a first-order SLRNN, $E = M + N$, and equation (9) takes the following form[6]:

$$\begin{pmatrix} I_{11} & \cdot\cdot & I_{1M} & S_{11} & \cdot\cdot & S_{1N} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ I_{d1} & \cdot\cdot & I_{dM} & S_{d1} & \cdot\cdot & S_{dN} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ I_{D1} & \cdot\cdot & I_{DM} & S_{D1} & \cdot\cdot & S_{DN} \end{pmatrix} \begin{pmatrix} w_{11} & \cdot\cdot & w_{1M} & w_{1(M+1)} & \cdot\cdot & w_{1(M+N)} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ w_{k1} & \cdot\cdot & w_{kM} & w_{k(M+1)} & \cdot\cdot & w_{k(M+N)} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ w_{N1} & \cdot\cdot & w_{NM} & w_{N(M+1)} & \cdot\cdot & w_{N(M+N)} \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} \sigma_{11} & \cdot\cdot & \sigma_{1N} \\ \cdot & \cdot & \cdot \\ \sigma_{d1} & \cdot\cdot & \sigma_{dN} \\ \cdot & \cdot & \cdot \\ \sigma_{D1} & \cdot\cdot & \sigma_{DN} \end{pmatrix}$$

where $I_{di}$ and $S_{di}$ refer to the $i$-th element of the vectors that respectively encode the input symbol and state of the $d$-th transition of the FSM.

For a second-order SLRNN, $E = MN$, and equation (9) takes the following form:

---

[4]Such a criterion will be applied in the representation of Minsky's general solution under our FS-SLRNN model

[5]Caution: this does not mean that a linear sequential machine (Booth 1967) is involved. A linear machine would require a linear function $g$.

[6]If a bias term is included (corresponding to a weighted input signal 1 for each unit) $M$ is increased by one.

$$\begin{pmatrix} I_{11}S_{11} & \cdot\cdot & I_{11}S_{1N} & \cdot\cdot & I_{1M}S_{11} & \cdot\cdot & I_{1M}S_{1N} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ I_{d1}S_{d1} & \cdot\cdot & I_{d1}S_{dN} & \cdot\cdot & I_{dM}S_{d1} & \cdot\cdot & I_{dM}S_{dN} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ I_{D1}S_{D1} & \cdot\cdot & I_{D1}S_{DN} & \cdot\cdot & I_{DM}S_{D1} & \cdot\cdot & I_{DM}S_{DN} \end{pmatrix} \begin{pmatrix} w_{111} & \cdot\cdot & w_{1MN} \\ \cdot & \cdot & \cdot \\ w_{k11} & \cdot\cdot & w_{kMN} \\ \cdot & \cdot & \cdot \\ w_{N11} & \cdot\cdot & w_{NMN} \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} \sigma_{11} & \cdot\cdot & \sigma_{1N} \\ \cdot & \cdot & \cdot \\ \sigma_{d1} & \cdot\cdot & \sigma_{dN} \\ \cdot & \cdot & \cdot \\ \sigma_{D1} & \cdot\cdot & \sigma_{DN} \end{pmatrix}$$

The above construction will be illustrated with an example. Consider the odd-parity recognizer shown in Figure 2. The coefficient arrays $A$ and $B$ that are obtained for a first-order and for a second-order SLRNN (with sigmoid activation function), by using local encoding and applying the procedure explained so far, are shown in Figures 3 and 4, respectively, where each row is labelled by the associated transition. Note that the first-order system is not solvable. Furthermore, note that the use of local encoding for both symbols and states in a second-order SLRNN implies an identity diagonal matrix $A$, and therefore a solvable system.

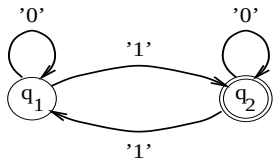


**Fig. 2** *Odd parity recognizer*

$$\begin{matrix} \delta('0',q_1) \\ \delta('0',q_2) \\ \delta('1',q_1) \\ \delta('1',q_2) \end{matrix} \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} W = \begin{pmatrix} C & -C \\ -C & C \\ -C & C \\ C & -C \end{pmatrix} \begin{matrix} q_1 \\ q_2 \\ q_2 \\ q_1 \end{matrix}$$

**Fig. 3** *Representation in a first-order FS-SLRNN model of the state transition function for the odd parity recognizer. The value $C$ is chosen such that $g_s(C) \simeq 1$ and $g_s(-C) \simeq 0$.*

$$\begin{matrix} \delta('0',q_1) \\ \delta('0',q_2) \\ \delta('1',q_1) \\ \delta('1',q_2) \end{matrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} W = \begin{pmatrix} C & -C \\ -C & C \\ -C & C \\ C & -C \end{pmatrix} \begin{matrix} q_1 \\ q_2 \\ q_2 \\ q_1 \end{matrix}$$

**Fig. 4** *Representation in a second-order FS-SLRNN model of the state transition function for the odd parity recognizer. The value $C$ is chosen such that $g_s(C) \simeq 1$ and $g_s(-C) \simeq 0$.*

With respect to the output function $\eta : (I \times S) \rightarrow O$ of the Mealy machine, two approaches can be taken to represent it using the FS-SLRNN model. If, as introduced in the beginning of this section, the output vector $\mathbf{O}$ is considered as part of the state vector $\mathbf{S}$, then the representation

of the output function just corresponds to some part of the linear system $AW = B$ of eq.(9) (e.g. the first $P$ columns in arrays $W$ and $B$), that is:

$$\sigma_{dk} \;=\; g^{-1}(y_{dk}) \;=\; f\ (\mathbf{W}_k, \mathbf{I}_d, \mathbf{S}_d) \qquad \text{for } 1 \leq d \leq mn \quad \text{for } 1 \leq k \leq P \tag{10}$$

Goudreau *et al.* (1994) have demonstrated that second-order SLRNNs can implement any output function $\eta$ for all the FSMs following this scheme, but first-order SLRNNs need to be augmented to implement some $\eta$ functions, either by adding a feed-forward layer of output units (Minsky 1967, Elman 1990), or by allowing a one-time-step delay before reading the output (Goudreau and Giles 1993).

Therefore, for (first-order) augmented SLRNNs, the appropriate approach is to separate the representation of the output function $\eta$ from that of the state transition function $\delta$. To this end, the $N$ recurrent units are preserved for state encoding, and $P$ new non-recurrent units $O_l$ are added to provide the output vector $\mathbf{O}^t = [o_1^{t+1}, ..., o_P^{t+1}]$, which is given by

$$o_l^{t+1} \;=\; g\ (\ f\ (\mathbf{W}_l, \mathbf{S}^{t+1})\ )\ \;=\; g\ (w_{l1} + \sum_{j=1}^{N} w_{l(1+j)} y_j^t\ ) \qquad \text{for } 1 \leq l \leq P \tag{11}$$

where $o_l^{t+1}$ refers to the activation value of output unit $O_l$ at time $t+1$, and $\mathbf{W}_l$ is the vector of weights of unit $O_l$.

In this case, after a "linearization" process for the output units, an additional linear system is yielded to represent $\eta$ :

$$g^{-1}(o_{il}) \;=\; w_{l1} + \sum_{j=1}^{N} w_{l(1+j)} y_{ij} \qquad \text{for } 1 \leq i \leq n \quad \text{for } 1 \leq l \leq P \tag{12}$$

where the values $o_{il}$ $(1 \leq l \leq P)$ are related to the code of the output corresponding to the $i$-th state of the FSM. Eq.(12) can be expressed in matrix form as

$$A_O W_O = B_O \tag{13}$$

where $A_O$ $(n \;\times\; (N+1))$ is the array of the output-unit inputs, $W_O$ $((N+1) \times P)$ is the (transposed) array of the output-unit weights, $B_O$ $(n \times P)$ is the array of the output-unit linear outputs, and $n$ is the number of states in the FSM. Note that the augmented SLRNN actually represents a Moore machine $(\eta : S \rightarrow O)$, unless the state encoding of $\mathbf{S}^{t+1}$ in some way incorporates the information of the previous input $\mathbf{I}^t$ (this occurs in a split-state representation of Mealy machines (Minsky, 1967)).

In order to clarify the notation, we will refer to the linear system that only represents the state transition function $\delta$ as $A_S W_S = B_S$, instead of $AW = B$, which will be reserved for the case where the output values are part of the state representation vector, so $AW = B$ includes both $\delta$ and $\eta$ functions.

### 2.3 Stability of state representation in the FS-SLRNN model

In the preceding derivation of the FS-SLRNN model, we have assumed that the state codes yielded by the recurrent unit activations are stable regardless of the length of the input strings. This assumption is true only if some conditions on the input and state encodings, the activation function $g$, and the network implementation, which are stated below, are met. Otherwise, the linear model is just an approximation, and the state stability cannot be guaranteed when long strings are fed into the net.

Let $X$ and $Y$ be the sets of numbers used in the input and state encodings respectively, and let $\Sigma = \{\sigma \mid \exists y \in Y,\ \sigma = g^{-1}(y)\}$ be the set of numbers that are obtained by applying the inverse of a given activation function to each member of $Y$.

The first requirement concerns the exactness of a linear system solution. It is well known that, if $AW = B$ is solvable, the solution $W$ will be exact, if and only if, the coefficients of the $A$ and $B$ arrays are rational numbers and integer arithmetic is used to compute the elements of $W$ (which, consequently, are also rational). In our case, the values contained in array $A$ are either members of $X$ or members of $Y$ (for first-order SLRNNs), or the product of members of $X$ and $Y$ (for second-order SLRNNs); and the values contained in array $B$ are members of $\Sigma$. Therefore, the following conditions are necessary:

(c1) All the values used in the input encoding ($x \in X$) must be rational numbers.

(c2) The activation function $g$ and the set $Y$ of state encoding activation values must satisfy the condition that $\forall y \in Y,\ \exists \sigma,\ \sigma = g^{-1}(y)$ and both $y$ and $\sigma$ are rational numbers.

In practice, one pair of values is enough both for $X$ and $Y$. $X = \{0, 1\}$ is a common choice for input encoding that meets (c1). $Y = \{0, 1\}$ is adequate for the hard-limiter $g_h$ (given two selected rational inverse values, e.g. $\Sigma = \{-1, 1\}$), but not for the sigmoid $g_s$, for which 0 and

1 can only be approximated by taking $\Sigma = \{-C, C\}$ and a large $C$ (Figs.3-4). However, there are other sigmoid-like functions for which a set $Y$ satisfying (c2) can be found; for example, $g_{s3}(\sigma) = 1 / (1 + 3^{-\sigma})$, and $Y = \{0.1, 0.9\}$, for which $\Sigma = \{-2, 2\}$.

If a continuous function $g$, such as $g_{s3}$, is used in the SLRNN, a third condition must be included to guarantee an exact emulation:

(c3a) The SLRNN must operate with integer arithmetic and integer-based representation for rational weights and activation values.

If a discrete function $g$, such as $g_h$, or a discretized approximation of a continuous function, is used in the SLRNN, the strong condition (c3a) can be replaced by

(c3b) There exists an error bound $|\epsilon|$ in the computation by the SLRNN of the values $\sigma \in \Sigma$ from weights in $W$ and values in $A$, such that $\forall y \in Y,\ \exists \sigma,\ \forall s \in [\sigma - \epsilon, \sigma + \epsilon]\ \ g(s) = y$.

Furthermore, in the case of a discrete $g$, if the error bound $|\epsilon|$ takes into account the error caused by solving a real-valued linear system (which depends on the system matrix condition number), then conditions (c1) and (c2) can be removed since they collapse into (c3b).

# 3 Implementation of Finite State Machines in SLRNNs using the FS-SLRNN model

In this section, our algebraic model is used to explain the representational limitations of first-order SLRNNs and to analyze the methods of FSM implementation in SLRNNs with hard-limiters proposed by Minsky (1967) (for augmented first-order SLRNNs) and Goudreau *et al.* (1994) (for second-order SLRNNs). Moreover, these can be included as particular cases in the general representation scheme described, and we show that many other solutions exist for FSM implementation, even with rather arbitrary activation functions, whenever the stability requirements mentioned in subsection 2.3 are met.

### 3.1 Implementation of an FSM with first-order SLRNNs

Let us first concentrate on the implementation of the state transition function $\delta : I \times S \rightarrow S$ of an FSM, which is represented by the linear system $A_S W_S = B_S$. It is assumed that for all the $m$ input symbols and for all the $n$ states of the FSM, there is only one corresponding code, given, respectively, by the $M$ input signals and the $N$ recurrent units of the SLRNN (i.e. both the input and state encodings are uniquely-defined). Recall that the number of $A_S$ rows is $mn$ (for a completely-defined $\delta$), since each row is associated with a different pair $(a_i \in I, q_j \in S)$. The following theorem establishes an upper bound on the rank of $A_S$ for first-order SLRNNs, which (in general) renders the solution of the system $A_S W_S = B_S$ unfeasible.

**Theorem 1.** *The rank of matrix $A_S$ in a first-order FS-SLRNN model is at most $m + n - 1$ for all the possible (uniquely-defined) encodings of the $m$ inputs and $n$ states of an FSM.*

*Proof.* Let $\mathbf{r}_1$ be the first row of $A_S$, associated with the pair $(a_1, q_1)$; let $\mathbf{r}_j$ be the row of the pair $(a_1, q_j)$; let $\mathbf{r}_{(i-1)n\,+\,1}$ be the row of the pair $(a_i, q_1)$; and let $\mathbf{r}_{(i-1)n\,+\,j}$ be the row of the pair $(a_i, q_j)$. Regardless of the encoding chosen, the following $(m-1)(n-1)$ linear relations among $A_S$ rows always hold:

$$\mathbf{r}_{(i-1)n\,+\,j} = \mathbf{r}_{(i-1)n\,+\,1} + \mathbf{r}_j - \mathbf{r}_1 \quad \text{for } 2 \leq i \leq m \text{ and for } 2 \leq j \leq n$$

Hence, there are at most $mn - (m-1)(n-1) = m + n - 1$ linearly independent rows in $A_S$. □

The rank of $A_S$ is equal to the upper limit $m+n-1$ if this is the number of linearly independent columns; for example, this occurs if a local orthogonal encoding is used for both inputs and states, with $M = m$ and $N = n$.

In order to design an SLRNN capable of implementing any $\delta$ function of an FSM, there are two possible alternatives for overcoming the above restriction. One is to increase the order of the SLRNN (see next subsection), and the other consists of representing an equivalent machine with "split" states keeping the first-order architecture. In this second approach, the goal is to find a model in which the linear relations among the rows of $A_S$ are also satisfied by the rows of $B_S$ for any $\delta$, thus allowing a solution $W_S$. We will show that this apparently strict condition is met by Minsky's method (Minsky 1967).

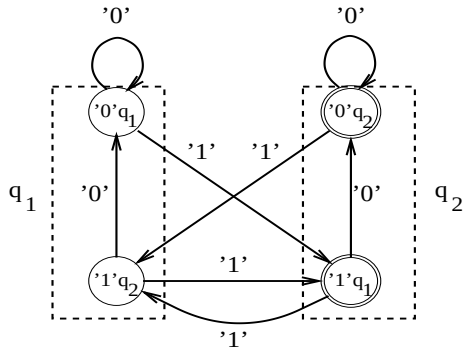


**Fig. 5** *Maximally-split odd-parity recognizer*

$$
\begin{array}{l}
\delta('0','0'q_1) \\ \delta('0','1'q_2) \\ \delta('0','0'q_2) \\ \delta('0','1'q_1) \\ \delta('1','0'q_1) \\ \delta('1','1'q_2) \\ \delta('1','0'q_2) \\ \delta('1','1'q_1)
\end{array}
\begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0
\end{pmatrix}
W_S =
\begin{pmatrix}
2\omega+\theta & \omega+\theta & \omega+\theta & \theta \\
2\omega+\theta & \omega+\theta & \omega+\theta & \theta \\
\omega+\theta & 2\omega+\theta & \theta & \omega+\theta \\
\omega+\theta & 2\omega+\theta & \theta & \omega+\theta \\
\omega+\theta & \theta & 2\omega+\theta & \omega+\theta \\
\omega+\theta & \theta & 2\omega+\theta & \omega+\theta \\
\theta & \omega+\theta & \omega+\theta & 2\omega+\theta \\
\theta & \omega+\theta & \omega+\theta & 2\omega+\theta
\end{pmatrix}
\begin{array}{l}
'0'q_1 \\ '0'q_1 \\ '0'q_2 \\ '0'q_2 \\ '1'q_1 \\ '1'q_1 \\ '1'q_2 \\ '1'q_2
\end{array}
$$

$$
W_S =
\begin{pmatrix}
\theta & \omega & 0 & \omega & 0 & 0 & \omega \\
\theta & \omega & 0 & 0 & \omega & \omega & 0 \\
\theta & 0 & \omega & \omega & 0 & 0 & \omega \\
\theta & 0 & \omega & 0 & \omega & \omega & 0
\end{pmatrix}^{\mathrm{T}}
$$

**Fig. 6** *Linear model representation of a first-order SLRNN implementing the state transition function of the (maximally-split) odd-parity recognizer.*

The key point is that instead of representing the original automaton of $n$ states $F = (I, S, \delta)$, an equivalent automaton of $mn$ states $F' = (I, S', \delta')$ is implemented, where for each original state $q_k \in S$ there are as many states $q_{kij} \in S'$ as the number of incoming transitions to $q_k$, each one described by a pair $(a_i \in I, q_j \in S)$ such that $\delta(a_i, q_j) = q_k$; the new transition function $\delta'$ is built accordingly. We will refer to the latter as the maximally-split equivalent automaton. For example, Figure 5 displays the (maximally-split) 4-state automaton equivalent to the 2-state odd parity recognizer in Fig.2, and Figure 6 shows the linear model of its $\delta$ function for the solution given by the next theorem, which generalizes Minsky's method for $\delta$ implementation.

**Theorem 2.** *The sufficient conditions on a first-order SLRNN to implement a given state transition function $\delta$ of a FSM with $m$ inputs and $n$ states are the following:*

(i) The first-order SLRNN has $mn$ state units, each one standing for one of the $mn$ states of

the maximally-split equivalent automaton, and $m$ input signals.

(ii) An orthogonal encoding with 1 and 0 values is followed for both inputs ($X = \{0, 1\}$) and states ($Y = \{0, 1\}$).

(iii) The activation function $g$ must meet that, there exist three rational numbers $\sigma_1, \sigma_{01}, \sigma_{02}$ such that $\sigma_1 = 2(\sigma_{01} - \sigma_{02}) + \sigma_{02}$ and $g(\sigma_1) = 1,\ g(\sigma_{01}) = 0,\ g(\sigma_{02}) = 0$.

(iv) Either the stability condition (c3a) applies to the first-order SLRNN or a discrete activation function $g$ is used that satisfies the stability condition (c3b) for $\Sigma = \{\sigma_1, \sigma_{01}, \sigma_{02}\}$.

(v) Let $\omega = \sigma_{01} - \sigma_{02}$ and $\theta = \sigma_{02}$. Let the pair $(u, v)$ be the label of the unit associated with the transition $\delta(u, v)$. A solution weight matrix $W_S$ is given by the assignment:

$$w_{(uv)l} = \begin{cases} \theta & \text{if } l = 0 \text{ (this is the bias weight)} \\ \omega & \text{if } l = u \text{ (i.e. } l \text{ stands for the connection from input } u) \\ \omega & \text{if } l \text{ stands for the connection from any of the units that code the states } q_{vij} \text{ (split from } q_v) \\ 0 & \text{otherwise} \end{cases}$$

*Proof.* See Appendix 1.

**Corollary 1.** Minsky's proposal for the implementation of the state transition function of any Mealy machine in networks of McCulloch-Pitts neurons is a particular case of the above solution, in which the activation function is the hard-limiter $g_h$, $\omega = 1$ and $\theta = -2$ (i.e. $\Sigma = \{-2, -1, 0\}$). For the same activation function $g_h$, there is a solution for any positive rational number $\omega > 2\epsilon$, where the rational threshold $-\theta$ can be chosen in the interval $(\omega + \epsilon, 2\omega - \epsilon]$, for a SLRNN with error bound $|\epsilon|$ in the computation of $\sigma \in \Sigma$.

Concerning the output function $\eta : I \times S \to O$ of the FSM, Goudreau *et al.* (1994) proved that an augmented first-order SLRNN architecture is needed to include all the possible output mappings. Therefore, the linear system $A_O W_O = B_O$ (Eq.13), which results from the incorporation of an output layer of $P$ units, must be solved. For this purpose, it is enough to use an orthogonal state encoding to obtain a full-rank matrix $A_O$, which guarantees a weight solution $W_O$ for any possible output encoding represented in matrix $B_O$. Moreover, the only requirement on the activation function of the output layer is that its range must include the values used in the codes of the output symbols.

In summary, in order to implement all FSMs, first-order SLRNNs need both to be augmented (to implement all $\eta$ functions) and to use state splitting (to implement all $\delta$ functions).

### 3.2 Implementation of a FSM with high-order SLRNNs

Here the goal is that the rank of matrix $A_S$ becomes $mn$, and so, $W_S$ can always be found for any $B_S$ (i.e., for any state transition function $\delta$). To this end, some terms of second or higher order are needed as neuron inputs to provide the required number of linearly independent $A_S$ columns. There are several solutions of this kind, which can be proven to yield $rank(A_S) = mn$. For example, the following two approaches:

- Use a second-order SLRNN of the type determined by equation (4), with activation function $g_h$, and select a local orthogonal encoding for both inputs and states (this is the solution described by Goudreau *et al.* (1994)). In this case, it is easy to show that $A_S$ is always an identity diagonal matrix (e.g. Fig.4).
- Use a high-order SLRNN of just one recurrent unit $U_1$, with multiple rational values in the range of its activation function $g$ (e.g. a discrete approximation of a sigmoid), and just one input signal $x_1$, for coding the states and inputs, respectively; and let the neuron input terms be given by a family of high-order functions, indexed by $c$ ($1 \leq c \leq mn$),

$$f_c(x_1, y_1) = x_1^{u_c} y_1^{v_c} \tag{14}$$

  where $u_c$ and $v_c$ are positive integers such that $u_c = ((c-1) mod\ m) + 1$ and $v_c = ((c-1) div\ m) + 1$. This SLRNN can be regarded as an extreme case in which the required linear independence is achieved through the variable-order nonlinear connections[7].

The following theorem permits the use of quite arbitrary activation functions in second or higher order SLRNNs to implement any $\delta$ of an FSM.

**Theorem 3.** *If the aggregation function $f$ of the SLRNN and the input and state encodings are selected such that the rank of the matrix $A_S$ in the FS-SLRNN model is equal to the number of transitions of the given FSM, then, in order to implement the state transition function $\delta$,*

[7] The operators *div* and *mod* refer to integer division and module operations.

*the only requirement is the satisfaction of the stability conditions (c1), (c2), and (c3a) or (c3b) (depending on whether the activation function g of the SLRNN is continuous or discrete).*

*Proof.* Let the given FSM have $m$ inputs and $n$ states. Let $A_S$ be the matrix that represents, in the given encoding, the pairs of $m$ inputs and $n$ states for which the state transition function $\delta$ of the given FSM is defined. The activation function $g$ and the chosen encoding determine the contents of matrix $B_S$ for the given $\delta$. However, since the rank of $A_S$ is equal to the number of rows, then for any possible $B_S$ there exists a corresponding weight matrix $W_S$ that solves $A_S W_S = B_S$. Therefore, it is only required that matrix $B_S$ can be constructed given the predetermined state encoding (condition (c2)) and that the state codes be stable during network operation (conditions (c1),(c2) and (c3a) or (c3b)). □

The output function ($\eta : I \times S \rightarrow O$) of the FSM can easily be implemented by extending any of the configurations of high-order SLRNN and data encoding characterized by a full rank of $A_S$ (e.g. the second-order SLRNN with local orthogonal encoding for both states and inputs). As introduced in Section 2, the system $A_S W_S = B_S$ can be extended to a larger system $AW = B$, by taking $P$ new recurrent units to encode the $p$ output symbols of the FSM. This causes $P$ new columns in $A$, $W$ and $B$, and also $P$ new rows in $W$, but the number of rows is the same in $A_S$ and $A$, and furthermore, $rank(A) = rank(A_S)$. Consequently, all the possible stable encodings of the $p$ output symbols in the $P$ output units, using rational values in the range of the selected activation function $g$, are feasible, since there is always a weight solution $W$ for any $B$, due to the full rank of $A$.

## 4 Insertion of FSMs in SLRNNs for subsequent learning

The FS-SLRNN model can be employed to insert symbolic knowledge into discrete-time SLRNNs prior to learning by examples and to preserve this knowledge while learning proceeds (Sanfeliu and Alquezar, 1994). This is useful when the neural inductive process is to be guided according to *a-priori* knowledge of the problem or from a partial symbolic solution. The SLRNNs can be made up of any activation function $g$ that supports a neural learning scheme, provided that the aforementioned stability conditions are fulfilled. Note that even for discrete $g$ such as $g_h$, learning algorithms are available (e.g. the pseudo-gradient technique described in (Zeng *et al.*, 1993)).

The key point is that, given a sufficient number of hidden units in the SLRNN[8], underdetermined systems $A_S W_S = B_S$ and $A_O W_O = B_O$ can be built, in which some of the network weights are free parameters in the solutions $W_S$ and $W_O$ and the rest are determined by a linear combination of the former. Hence, the learning algorithm can be adapted to search for error minima in a linear subspace (with the free weights as variables), for which all the corresponding networks implement at least the inserted FSM.

The FSM insertion method consists of two steps:

1. Establish underdetermined linear systems $A_S W_S = B_S$ and $A_O W_O = B_O$, that represent the $\delta$ and $\eta$ functions of the inserted FSM, using an architecture and a data encoding suitable for FSM implementation, but including more units than required to solve the systems (e.g. second-order SLRNN with orthogonal encoding and $N > n$).
2. Initialize the weights of the hidden and output units to any of the solutions $W_S$ and $W_O$ that result from solving the above linear systems[9].

Usually, a neural learning scheme updates *all* the network weights to minimize an error function. In such a case, the inserted rules may be degraded and eventually forgotten as the weights are modified to cope with the training data. Although this behavior allows for rule refinement and it may be valid to learn a given task, an alternative approach that preserved the inserted FSM would be preferable if this were known to be part of a task solution.

To that end, a *constrained neural learning* procedure must be followed. For example, if an on-line gradient-descent algorithm such as RTRL (Williams and Zipser, 1989) is used, then the free weights of a recurrent unit $k$ should be changed according to:

$$\triangle w_{kl}(t) = -\alpha \left( \frac{\delta E(t)}{\delta w_{kl}} + \sum_{w_{ka} \in D(W_k)} \frac{\delta E(t)}{\delta w_{ka}} \frac{\delta w_{ka}}{\delta w_{kl}} \right) \tag{15}$$

where $\alpha$ is the learning rate, $E(t)$ is the overall network error at time $t$, $D(W_k)$ denotes the subset of determined weights in unit $k$, and the partial derivatives $\frac{\delta w_{ka}}{\delta w_{kl}}$ are known constants given by the linear relations among weights. The RTRL algorithm itself can be employed to

[8] augmented SLRNN for first-order type

[9] The preferred initialization is that in which the sum of squared weights is minimal and the weights are non-zero.

compute both the partial derivatives $\frac{\delta E(t)}{\delta w_{kl}}$ and $\frac{\delta E(t)}{\delta w_{ka}}$. On the other hand, the weights $w_{ka}$ in $D(W_k)$ should be changed at each step, after updating all the free weights $w_{kl}$, to keep the linear constraints specified in the underdetermined solution of the system.

## 5 Conclusions

An algebraic linear framework to represent finite state machines (FSMs) in discrete-time single-layer recurrent neural networks (SLRNNs), which has been termed an FS-SLRNN model, has been presented. The scheme is based on the transformation of the nonlinear constraints imposed on the network dynamics by the given FSM and data encoding into a set of static linear equations to be satisfied by the network weights. This transformation leads to an exact emulation of the FSM when some stability conditions, which have been stated, are met; otherwise the linear model is just a static approximation of the network dynamics.

It has been proved, using the FS-SLRNN model, that first-order SLRNNs have some limitations in their representation capability, which are caused by the existence of some linear relations (that always hold) among the equations associated with the state transitions. In order to overcome this problem, a first-order SLRNN may need to represent a larger equivalent machine with split states. Furthermore, first-order SLRNNs need to be augmented (e.g. with an output layer) to be able to represent every output mapping. According to these requirements, the method for FSM implementation in augmented first-order SLRNNs by Minsky (1967) has been generalized.

On the other hand, second-order (or higher-order) SLRNNs can easily implement all the FSMs, since their corresponding FS-SLRNN models, given an orthogonal data encoding, are characterized by the full rank of the system matrix. The method for FSM implementation in second-order SLRNNs by Goudreau *et al.* (1994) can be seen as a particular case of this class of solutions. The actual requirements on the network activation function have been determined, and these have been shown to be quite weak (i.e. a large spectrum of activation functions can be used for FSM implementation in SLRNNs).

The framework proposed can be used to insert symbolic knowledge into discrete-time SLRNNs prior to neural learning from examples. This can be done by initializing the network weights to any of the possible solutions of an underdetermined linear system representing the inserted

(partial) FSM with an excess of recurrent units. In comparison with other published methods (Frasconi *et al.* 1991; Omlin and Giles 1992) that insert FSMs into RNNs, the method proposed is more general, since it can be applied to a wide variety of both activation and aggregation functions. Moreover, a new distinguishing feature of our insertion method is that it allows the inserted rules to be kept during subsequent learning, by training a subset of free weights and updating the others to force the satisfaction of the linear constraints in the system solution.

The ultimate aim of this paper is to establish a well-defined link between symbolic and connectionist sequential machines. Linear algebra has been used as a suitable tool to aid to the study of representational issues. Further research is needed to fully exploit the proposed model for other applications, such as the determination of the smallest SLRNN that simulates a given FSM, and the development of improved learning techniques for grammatical inference (Sanfeliu and Alquezar, 1994).

## Acknowledgements

We thank the reviewers for their comments and suggestions, which have helped us to improve the contents and presentation of the work.

## References

Alon, N., Dewdney, A.K., Ott, T.J. 1991. Efficient simulation of finite automata by neural nets. *Journal of the Association for Computing Machinery* **38** (2), 495-514.

Alquezar, R., and Sanfeliu, A. 1993. Representation and recognition of regular grammars by means of second-order recurrent neural networks. In *New Trends in Neural Computation, Proc. of the Int. Workshop on Artificial Neural Networks IWANN'93, Sitges, Spain, June 1993.* J.Mira, J.Cabestany, A.Prieto, Eds., Springer-Verlag, 143-148.

Booth, T.L. 1967. *Sequential Machines and Automata Theory.* John Wiley, New York.

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

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

Frasconi, P., Gori, M., Maggini, M., Soda, G. 1991. A unified approach for integrating explicit knowledge and learning by example in recurrent networks. In *Proc. Int. Joint Conf. on Neural Networks*, Vol.1, IEEE Press, Piscataway NJ, 811-816.

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

Goudreau, M.W., and Giles, C.L. 1993. On recurrent neural networks and representing finite state recognizers. In *Proc. Third Int. Conf. on Artificial Neural Networks*, IEE, London, UK.

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

Minsky, M. 1967. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs NJ, Chap.3.

Omlin, C.W., and Giles, C.L. 1992. Training second-order recurrent neural networks using hints. In *Proc. Ninth Int. Conf. on Machine Learning*, D. Sleeman and P. Edwards, Eds., Morgan Kaufmann Pub., San Mateo CA, 363-368.

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

Sanfeliu, A., and Alquezar, R. 1992. Understanding neural networks for grammatical inference and recognition. In *Advances in Structural and Syntactic Pattern Recognition, Proc. of the IAPR Int. Workshop on SSPR, Bern, Switzerland, August 1992*, H.Bunke, Ed., World Scientific, 75-98.

Sanfeliu, A., and Alquezar, R. 1994. Active grammatical inference: a new learning methodology. In *Proc. of the IAPR Int. Workshop on Structural and Syntactic Pattern Recognition, SSPR'94, Nahariya, Israel, October 1994.*

Williams, R.J., and Zipser, D. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* **1** (2), 270-280.

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

# Appendix 1.

Let $A_S W_S = B_S$ be the linear model that represents the state transition function $\delta'$ of the maximally-split automaton (with $m$ inputs and $mn$ states) equivalent to the original automaton ($m$ inputs, $n$ states, and transition function $\delta$) of the given FSM, using a first-order SLRNN and an orthogonal data encoding (with values $\{0,1\}$) for both states and inputs.

$A_S$ is a matrix of $m^2 n$ rows (for a complete $\delta$) and $(m + mn + 1)$ columns; whereas $B_S$ has also $m^2 n$ rows but just $mn$ columns. The rows of both $A_S$ and $B_S$ can be organized into $m$ blocks of $mn$ rows, where each block ($R_i^A$ and $R_i^B$) corresponds to the transitions with the input symbol $a_i$. Let $\mathbf{r}_{ij}^A$ and $\mathbf{r}_{ij}^B$ denote the $j$-th row of blocks $R_i^A$ and $R_i^B$, respectively (which are associated with transition $\delta'(a_i, q'_j)$ of the maximally-split automaton).

Each block of rows is divided into $n$ sub-blocks (as many as states in the original FSM), and each sub-block ($R_{ik}^A$ and $R_{ik}^B$) has as many rows as states resulting from split the state $q_k$. The rows in any sub-block $R_{ik}^B$ of matrix $B_S$ are identical since they code the same destination state (identified by the pair $(a_i, q_k)$). Let $K(j)$ be a function that indicates the number of sub-block $k$ to which the $j$-th row of any block belongs.

The columns of $B_S$ (denoted as $\mathbf{c}_{uv}^B$) are labelled by two sub-indexes, $u = 1, ..., m$ and $v = 1, ..., n$, which associate $\mathbf{c}_{uv}^B$ with the unit that flags the state of the split automaton characterized by "being the destination of a transition with the $u$-th input symbol from a state equivalent to the $v$-th state of the original automaton". Finally, let $\sigma_{(ij)(uv)}$ be the element in the row $\mathbf{r}_{ij}^B$ and in the column $\mathbf{c}_{uv}^B$, and let $y_{(ij)(uv)} = g(\sigma_{(ij)(uv)})$ be the corresponding activation value.

Since we deal with a first-order SLRNN and an orthogonal encoding is followed for both inputs and states, Theorem 1 establishes that the rank of $A_S$ is $mn + m - 1$ and there exist $(mn - 1)(m - 1)$ linear relations among the rows of $A_S$, these being the following:

$$\mathbf{r}_{ij}^A = \mathbf{r}_{i1}^A + \mathbf{r}_{1j}^A - \mathbf{r}_{11}^A \quad 2 \leq i \leq m \quad 2 \leq j \leq mn \tag{16}$$

In order to prove the theorem we need to deduce that for all the $m^2 n$ transitions of $\delta'$, the orthogonal code of the destination state is obtained in the unit activation values, given the selected weight assignment and the properties of the activation function $g$. In addition, it will be shown that $B_S$ satisfies the same

linear relations among rows than $A_S$, that is

$$\mathbf{r}_{ij}^B = \mathbf{r}_{i1}^B + \mathbf{r}_{1j}^B - \mathbf{r}_{11}^B \quad 2 \leq i \leq m \quad 2 \leq j \leq mn \tag{17}$$

By multiplying matrix $A_S$ by the matrix $W_S$ built up with the former weight assignment, it follows that the linear output of the network state units can be expressed as follows:

$$\sigma_{(ij)(uv)} = \begin{cases} 2\omega + \theta & \text{if } u = i \ \wedge \ v = K(j) \\ \omega + \theta & \text{if } u = i \ \oplus \ v = K(j) \\ \theta & \text{if } u \neq i \ \wedge \ v \neq K(j) \end{cases} \tag{18}$$

Due to the orthogonal encoding which is followed, the required activation values are:

$$y_{(ij)(uv)} = \begin{cases} 1 & \text{if } u = i \ \wedge \ v = K(j) \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

Therefore, since $g(2\omega + \theta) = 1,\ g(\omega + \theta) = 0,$ and $g(\theta) = 0$ , the desired code for the destination state is obtained for all the $m^2 n$ transitions of $\delta'$ (i.e. for all the $m^2 n$ rows of $B_S$).

The proof that the assignment of $B_S$ elements given by Eq.(18) implies

$$\sigma_{(ij)(uv)} = \sigma_{(i1)(uv)} + \sigma_{(1j)(uv)} - \sigma_{(11)(uv)} \quad 2 \leq i \leq m \quad 2 \leq j \leq mn \quad 1 \leq u \leq m \quad 1 \leq v \leq n \tag{20}$$

can be performed on a case-by-case basis.

Firstly, when $K(j) = K(1)$ it follows immediately that $\sigma_{(ij)(uv)} = \sigma_{(i1)(uv)}$ and $\sigma_{(1j)(uv)} = \sigma_{(11)(uv)}$, since all the rows $\mathbf{r}_{ij}^B$ in the same subblock $R_{iK(j)}^B$ are identical because they code the same destination state.

When $K(j) \neq K(1)$, only the next four cases may occur, where we replace $\sigma_{(ij)(uv)}$, $\sigma_{(i1)(uv)}$, $\sigma_{(1j)(uv)}$ and $\sigma_{(11)(uv)}$ in Eq.(20) by the values given in Eq.(18):

**a)** $u = i \ \wedge \ v = K(j)$

$$2\omega + \theta \ = \ (\omega + \theta) \ + \ (\omega + \theta) \ - \ \theta$$

**b)** $u = i \ \wedge \ v \neq K(j)$

$$\begin{cases} \omega + \theta \ = \ (2\omega + \theta) \ + \ \theta \ - \ (\omega + \theta) & \text{if } v = K(1) \\ \omega + \theta \ = \ (\omega + \theta) \ + \ \theta \ - \ \theta & \text{if } v \neq K(1) \end{cases}$$

**c)** $u \neq i \ \wedge \ v = K(j)$

$$\begin{cases} \omega + \theta \ = \ \theta \ + \ (2\omega + \theta) \ - \ (\omega + \theta) & \text{if } u = 1 \\ \omega + \theta \ = \ \theta \ + \ (\omega + \theta) \ - \ \theta & \text{if } u \neq 1 \end{cases}$$

**d)** $u \neq i \ \wedge \ v \neq K(j)$

$$\begin{cases} \theta \ = \ (\omega + \theta) + \ (\omega + \theta) \ - \ (2\omega + \theta) & \text{if } u = 1 \ \wedge \ v = K(1) \\ \theta \ = \ \theta \ + \ (\omega + \theta) \ - \ (\omega + \theta) & \text{if } u = 1 \ \oplus \ v = K(1) \\ \theta \ = \ \theta \ + \ \theta \ - \ \theta & \text{if } u \neq 1 \ \wedge \ v \neq K(1) \end{cases}$$

□