UNIVERSITAT POLITÈCNICA DE CATALUNYA

Departament de Llenguatge i Sistemes Informàtics
Ph.D. Programme: Artificial Intelligence

SYMBOLIC AND CONNECTIONIST LEARNING TECHNIQUES FOR GRAMMATICAL INFERENCE

Autor: René Alquézar Mancho
Director: Alberto Sanfeliu Cortés

March 1997

Chapter 7

Representation of finite-state machines in recurrent neural networks and the active grammatical inference methodology

In this chapter, an algebraic framework to represent finite state machines (FSMs) in SLRNNs and two-layer ASLRNNs is presented, which unifies and generalizes some of the previous proposals for encoding FSMs in RNNs [Mins:67, SaAl:92, AlSa:93, GoGC:94]. This theoretical work has been published in the *Neural Computation* journal [AlSa:95b]. The algebraic framework is based on the formulation of both the state transition function and the output function of an FSM as a linear system of equations, and it permits an analytical explanation of the different representational capabilities of first-order and higher-order SLRNNs. Moreover, two methods based on this linear model are given for the implementation of FSMs in first-order 2L-ASLRNNs and second-order SLRNNs (or 2L-ASLRNNs), respectively, which cover the implementation of DFAs and stochastic FSAs as particular cases. However, these methods do not yield in general the network of minimum size capable of implementing the given FSM. On the other hand, it is explained how the proposed framework can help in the search of a solution with fewer units.

In addition, the presented linear model can be used to insert FSMs (symbolic knowledge) in RNNs, through linear system solving, prior to learning from examples and to force the RNN to keep this knowledge while it is being trained, through the maintenance of some linear relations among the network weights. In contrast with other reported approaches [FrGM:91,OmGi:96], the insertion method proposed is valid for a wide range of activation functions, whenever some stability conditions are met.

In the last part of the chapter, a hybrid connectionist-symbolic methodology for RGI, called *active grammatical inference* (AGI) [SaAl:95], is described which uses the insertion and learning techniques derived from the preceding algebraic model, together with the connectionist RGI and UFSA extraction methods described in Chapter 6. This new methodology allows a variety of heuristic methods (with the common feature that RNNs are used as the basic learning tool) to infer UFSAs or FSAs from string examples and a-priori knowledge, if available. Both fully-automated and semi-automated RGI approaches can be devised within the AGI methodology. In the latter approaches, a teacher may guide the learning process by introducing positive and/or negative rules and validating partial results in several steps (cycles). Hence, the AGI methodology is based on a combination of neural and symbolic techniques and representations, with the interesting feature that the inductive inference performed by the RNN may be guided by means of the inserted symbolic rules. In the most general case, the whole RGI process is conceived as a sequence of learning cycles, each one including the steps of automaton insertion, neural training, automaton extraction, symbolic manipulation and validation.

7.1 Antecedents and related work

Some complex neural networks with rational weights and a certain type of sigmoidal units have been demonstrated to be as powerful in computational ability as Turing machines [SiSo:91,SiSo:92], though learning techniques for these networks are not known yet. On the other hand, the representation of finite-state machines (FSMs) in recurrent neural networks (RNNs) has attracted the attention of researchers for a variety of reasons, ranging from the pursuit of hardware implementations to the integration (and improvement) of symbolic and connectionist approaches to grammatical inference and recognition. Several researchers have discussed the problem of representing FSMs or FSAs in different types of RNN architectures and have proposed some encoding techniques:

1) the Minsky's method for first-order two-layer ASLRNNs with threshold units [Mins:67];
2) the method by Alon *et al.* for a type of three-layer RNNs with threshold cells [AlDO:91];
3) the method by Goudreau *et al.* for second-order SLRNNs with threshold activation function [GoGC:94];
4) the method by Frasconi *et al.* for a type of first-order SLRNNs with bistable elements based on a squash activation function (the hyperbolic tangent) [FrGM:91];
5) the method by Omlin and Giles for second-order SLRNNs with sigmoid activation function [OmGi:92,OmGi:96].

The former three works listed [Mins:67, AlDO:91, GoGC:94] have shown how to build different RNN models with a hard-limiting activation function that perfectly simulate a given FSM. Nevertheless, none of these approaches yields the minimum size RNN which is required.

Minsky's method [Mins:67] 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, and the second layer has as many units as the number of output symbols. In terms of the RNN architectures presented in Chapter 4, the network proposed by Minsky is a first-order 2L-ASLRNN. The much more recent method by Alon *et al.* [AlDO:91] uses a three-layer recurrent network which needs a number of threshold cells of order $n^{3/4} \times m$. In the same work, some lower bounds were established for the minimum size network capable of simulating any n-state machine.

More recently, Goudreau *et al.* [GoGC:94] proved that, while second-order SLRNNs can easily implement any Mealy machine, first-order SLRNNs can not. In order to implement an FSM with n states, m input and p output symbols, a second-order SLRNN with $n+p$ recurrent units and a total number of $(n+p)^2 \times m$ weights was shown to be enough. On the contrary, a first-order SLRNN may require to be *augmented* to implement some FSMs (e.g. the odd parity recognizer), either by using an additional output layer, i.e. a 2L-ASLRNN [GoGC:94] as in Minsky's approach, or by allowing a one-time-step lag before reading the output values [GoGi:93]. Moreover, it was also shown that first-order ASLRNNs may need to simulate non-minimal FSMs that are equivalent to the optimal size FSMs [GoGC:94]; again, the odd parity recognizer was provided as an example where state splitting is required. This property exhibited by first-order SLRNNs was considered "intriguing" by Goudreau *et al.* In this chapter, a theorem is demonstrated that explains the causes of this limitation.

Other studies have been devoted to the design of methods for incorporating symbolic knowledge into trainable 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 FSAs from examples [Poll:91, GiMC:92, WaKu:92]. The last two methods in the previously displayed list [FrGM:91,OmGi:92] allow the insertion of an FSA into an RNN with sigmoidal units for subsequent learning from examples.

Frasconi *et al.* [FrGM:91] proposed a method to insert the transitions of a DFA into a type of first-order SLRNNs with hyperbolic tangent activation function, in which the units are seen as bistable elements. The dynamics of such a network is not the same than the dynamics of the usual SLRNNs described in Chapter 4: once an input is entered, each unit evolves freely a certain number of time steps (with all incoming connections disabled except self feedback) up to determine whether the sign of the unit's activation changes or not. In this way, the sigmoidal units can be forced to implement bistable elements. Given a DFA and an encoding for inputs and states, a set of inequalities can be established, so that the weights of the recurrent units must satisfy these inequalities in order to simulate the transitions of the DFA, and a region of admissible solutions may be found by using linear programming techniques [FrGM:91]. However, Frasconi *et al.* did not demonstrate that any transition function could be implemented in this manner. In addition, they proposed an RNN architecture to integrate the explicit inserted rules and learning by examples, which consists of two SLRNN subnetworks with common inputs, one subnet with the constrained weights implementing the inserted rules and another subnet with free weights, that are augmented by a shared output layer. Using this type of ASLRNN architecture, the inserted transition rules are not destroyed during learning.

Omlin and Giles proposed a method to insert DFAs in second-order SLRNNs with sigmoid activation function g_s and a bias weight in each unit [OmGi:92], and recently, these authors reported a variation of their method that allows a stable encoding of any DFA in the mentioned architecture [OmGi:96]. To implement a DFA with n states and m input symbols, a second-order SLRNN with $n+1$ units and m input signals is constructed, using a local one-hot encoding for states and symbols, where the first unit S_0 is consulted to accept or reject strings. For each transition $\delta(a_k, q_j) = q_i$ of a DFA, at most three weights of the network $(w_{ikj}, w_{jkj}, w_{0kj},)$ have to be programmed to a value, which is either H (a large positive constant called *strength*) or $-H$; in addition, all the bias weights are set to $-H/2$ and the rest of network weights are set to zero to ensure that all neurons which do not correspond to the previous or current state have a low output. To guarantee a stable behavior, the strength H scales with the network size and thus with the number of states of the given DFA, although in most cases a value of $H \simeq 6$ is enough in practice [OmGi:96].

In order to insert a partial DFA for subsequent learning from examples in the preceding second-order SLRNNs, an excess of units is recommended (with respect to the size of the inserted DFA) and all the weights that are not programmed to $-H$, $-H/2$ or H, are initialized to small random values [OmGi:92]. Moreover, all the network weights including the programmed ones are adaptable by the learning algorithm, thus allowing rule refinement. Giles and Omlin reported some experimental results showing that, for small values of H, the larger the number of inserted rules, the shorter the convergence time to learn a training set, whereas the generalization performances of networks trained with and without prior knowledge were comparable [GiOm:93].

In the following section, a linear model for FSM representation in SLRNNs and 2L-ASLRNNs is described, which improves the models reported in [SaAl:92, AlSa:93]. A study of the conditions that are needed to ensure the stability of the state representation is included. This model explains the limitations of first-order SLRNNs and serves to unify and generalize the FSM implementation methods by Minsky [Mins:67] and Goudreau *et al.* [GoGC:94] (Section 7.3). A related technique for injecting symbolic knowledge (a partial FSM) in SLRNNs and 2L-ASLRNNs prior to learning, which is valid for a wide class of activation functions, is described in Section 7.4. Finally, a methodology for regular grammatical inference that uses the FSM insertion and learning techniques derived from the model is presented in Section 7.5.

7.2 The FS-SLRNN linear model of FSM representation in RNNs

In the following let us consider single-layer recurrent neural networks (SLRNNs), such as the one shown in Figure 4.1, and let us recall the definitions and notation given in Section 4.1.1. To refresh them, an excerpt is repeated here.

An SLRNN has M inputs, with values $x_i(t)$, $1 \leq i \leq M$, at time t, and a recurrent fully-connected single-layer of N units, with activation values $y_j(t)$, $1 \leq j \leq N$, computed at time t. Let $\mathbf{S}^t = [y_1(t-1), ..., y_N(t-1)]^T$ and $\mathbf{I}^t = [x_1(t), ..., x_M(t)]^T$ denote the current state vector and the input vector, respectively, just before updating the activations of the network units at time t. In principle, let us assume that the first P neurons, $1 \leq P \leq N$, are output units, so that $\mathbf{O}^t = [y_1(t), ..., y_P(t)]^T$ is the output vector at time t. Note that the time counter t is increased by one after updating the activations and before reading the next input values. Hence, a general equation that describes the dynamics of an SLRNN is

$$y_k(t) = g\ (\ f\ (\mathbf{W}_k, \mathbf{I}^t, \mathbf{S}^t)\ ) \quad \text{for } 1 \leq k \leq N, \tag{7.1}$$

where $\mathbf{W}_k$ is the weight vector of the k-th unit, g is an activation function (of one variable), and f is an aggregation function of inputs and activation values, which is given by Eq.(4.3) for first-order SLRNNs and by Eq.(4.4) for second-order SLRNNs.

On the other hand, let us recall the definition of the two types of FSMs: a Mealy machine is a sixtuple $(I, O, S, q_0, \delta, \eta)$, where I is a set of input symbols, O is a set of output symbols, S is a set of states, q_0 is the start state, $\delta : I \times S \rightarrow S$ is a state transition function, and $\eta : I \times S \rightarrow O$ is an output function; whereas in a Moore machine, all is as in a Mealy machine except that the output function depends only on the states, i.e. $\eta : S \rightarrow O$. It is well-known that for every Mealy machine, there is an equivalent Moore machine, and viceversa [Booth:67].

It is shown next that the representation of an FSM in an SLRNN can be modelled as two linear systems of equations, one for the state transition function δ and another one for the output function η. This algebraic representation is referred to as an FS-SLRNN model.

7.2.1 The representation of the state transition function

Let $\mathcal{A} = (I, O, S, q_0, \delta, \eta)$ be a Mealy machine with m input symbols, n states and p output symbols. Let $\mathcal{N}$ be an SLRNN with M input signals, N units and a subset of P output units, that is wanted to emulate $\mathcal{A}$. First, it is shown how to construct a linear system for the state transition function δ, which involves the weights, inputs and activation values of $\mathcal{N}$. The representation of the output function η will be discussed later.

When $\mathcal{N}$ is running at a discrete time step t, one can think of the 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 activation values $y_j(t)$ as standing for the destination state q' that results from the current transition. Thus, the set of N units can be seen as implementing the state transition $\delta(a, q) = q'$ that occurs at time t.

Let D be the number of transitions (the number of arcs in the state transition diagram) of the FSM $\mathcal{A}$, where $D \leq mn$, and let d be an index of these transitions. Without loss of generality, let us say that δ is complete, i.e. it is defined for all the pairs $(a \in I, q \in S)$, and thus $D = mn$. At every time step t, the network $\mathcal{N}$ should implement one of the D transitions. Therefore, the weights of $\mathcal{N}$ should satisfy, at any arbitrary time step t, the set of nonlinear equations

$$y^t_{dk} = g\left(f\left(\mathbf{W}_k, \mathbf{I}^t_d, \mathbf{S}^t_d\right)\right) \quad \text{for } 1 \leq d \leq D, \quad \text{for } 1 \leq k \leq N, \tag{7.2}$$

where, if the d-th transition is $\delta(a, q) = q'$, then $\mathbf{I}^t_d$ and $\mathbf{S}^t_d$ refer to the input and state vectors that encode the input symbol a and current state q, respectively, and the activation values y^t_{dk} $(1 \leq k \leq N)$ are related to the code of the destination state q'. It should be noted that the ordering defined by the index d is static and arbitrary; for instance, we may use the following ordering: $\delta(a_1, q_1)$, $\delta(a_1, q_2)$, .., $\delta(a_1, q_n)$, $\delta(a_2, q_1)$, ..., $\delta(a_m, q_n)$. In the case of a partial function δ, where $D < mn$, it will be considered that the dynamics of the network $\mathcal{N}$ are irrelevant from the moment when an input a is entered in state q and $\delta(a, q)$ is not defined.

The nonlinear system of equations (7.2) describes a deterministic state transition function δ where, once an encoding scheme is selected for the input symbols and the states, only the weights $\mathbf{W}_k$, $1 \leq k \leq N$, of the SLRNN $\mathcal{N}$ 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 (7.2) into a manageable linear system by performing the two 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 (7.2) must be fulfilled for arbitrary time steps t. However, this simplification can be made as long as the stability of the state codes is guaranteed (see next subsection). In addition, the SLRNN must be initialized to reproduce the code of the start state q_0 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 a threshold function) then there exists an established criterion for selecting a proper value in every case[1].

Therefore, the preceding transformations convert the nonlinear system of equations (7.2) into the static linear system[2]:

$$\sigma_{dk} = g^{-1}(y_{dk}) = f\left(\mathbf{W}_k, \mathbf{I}_d, \mathbf{S}_d\right) \quad \text{for } 1 \leq d \leq D, \quad \text{for } 1 \leq k \leq N, \tag{7.3}$$

which can be described in matrix representation as

$$AW = B \tag{7.4}$$

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 net inputs, and E is the number of inputs to each neuron.

For a first-order SLRNN, $E = M + N$, and equation (7.4) takes the following form:

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

[1]Such a criterion will be applied in the representation of Minsky's general solution using the FS-SLRNN model.

[2]Caution: this does not mean that a linear sequential machine [Booth:67] is involved. A linear machine would require a linear activation function g.

where I_{di} and S_{di} refer to the i-th element of the vectors that respectively encode the input symbol a and the state q of the d-th transition of the FSM, $\delta(a,q)$. The dimension M is increased by one if a bias term, corresponding to a weighted fixed input 1, is included for each unit.

For a second-order SLRNN, $E = MN$, and equation (7.4) takes the following form:

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

The above construction will be illustrated with an example. Consider the state transition function δ of the odd-parity recognizer shown in Figure 7.1 (where the start state is q_1). The coefficient arrays A and B that are obtained for a first-order SLRNN and for a second-order SLRNN, both with the sigmoid activation function g_s of Eq.(4.5), by using local encoding and applying the procedure explained so far, are shown in Figures 7.2 and 7.3, respectively. In these figures, each row of the linear system is labeled by the associated transition $\delta(a,q) = q'$, and the positive value H is chosen such that $g_s(H) \simeq 1$ and $g_s(-H) \simeq 0$. Note that the system displayed in Fig. 7.2 for a first-order SLRNN is not solvable. On the other hand, the use of local encoding for both symbols and states in a second-order SLRNN implies an identity diagonal matrix A (see Fig. 7.3), and therefore a solvable system.

7.2.2 Stability of state representation

In the preceding construction of the linear model for the state transition function, it has been assumed implicitly 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. Then, three conditions are sufficient to guarantee the stability of states:

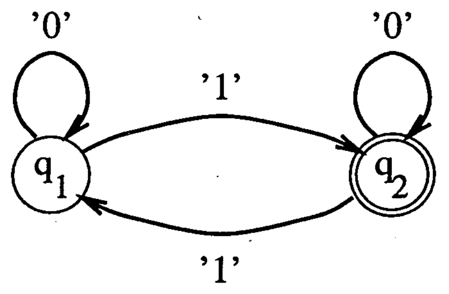

Fig. 7.1 *Odd parity recognizer of minimal size.*

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

Fig. 7.2 *Representation in a first-order FS-SLRNN model of the state transition function for the odd parity recognizer.*

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

Fig. 7.3 *Representation in a second-order FS-SLRNN model of the state transition function for the odd parity recognizer.*

(c1) all the values used in the input encoding ($x \in X$) are rational numbers;

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

(c3) there exists an error bound $\epsilon \geq 0$ in the computation by the SLRNN of the values $\sigma \in \Sigma$ from weights in W and values in A, such that $\forall y \in Y$, $\exists \sigma$, $\forall s \in [\sigma - \epsilon, \sigma + \epsilon]$ $g(s) = y$.

The two former conditions above are related to the exactness of a linear system solution, whereas the third condition is related to the precision in the network computation.

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 the case of the linear system of Eq.(7.4), 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 Σ. Therefore, the conditions (c1) and (c2) are necessary to be able to compute an exact solution, if some solution exists, of the linear system of Eq.(7.4).

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-limiting threshold function

$$g_h(\sigma) = \begin{cases} 1 & \text{if } \sigma \geq 0 \\ 0 & \text{if } \sigma < 0 \end{cases} \tag{7.5}$$

since two rational inverse values can be selected, e.g. $\Sigma = \{-1, 1\}$, but not for the sigmoid g_s, for which 0 and 1 can only be approximated by taking $\Sigma = \{-H, H\}$ and a large H, as in Figs. 7.2 and 7.3. However, there are other sigmoidal functions for which a set Y satisfying (c2) can be found; for example,

$$g_{s3}(\sigma) = \frac{1}{1 + 3^{-\sigma}} \tag{7.6}$$

and $Y = \{0.1, 0.9\}$, for which $\Sigma = \{-2, 2\}$.

Concerning the condition (c3), if certain continuous and differentiable activation functions g are used in the SLRNN, then an exact emulation of the transition function based on the FS-SLRNN model may only be guaranteed if the network is able to achieve $\epsilon = 0$, i.e. full precision. Activation functions of this kind are the sigmoid functions like g_s, g_{s3} (above), or the hyperbolic tangent, the sinusoidal function g_{sin}, the Gaussian function g_G, the antisymmetric logarithm g_{al}, and the simple linear function g_{lin}. All of these functions share the property that, for $\epsilon > 0$, a set Y of at least two values cannot be found such that $\forall y \in Y$, $\exists \sigma$, $\forall s \in [\sigma - \epsilon, \sigma + \epsilon]$ $g(s) = y$. Note that to achieve full precision ($\epsilon = 0$), the SLRNN should operate with integer arithmetic and integer-based representation for rational weights and activation values.

Nevertheless, continuous and differentiable activation functions g can be devised that allow a stable state encoding using the FS-SLRNN model without the previous strong requirement. For instance, let us define a sinusoidal-based sigmoid function g_{sinsgm} as

$$g_{sinsgm}(\sigma) = \begin{cases} 0 & \text{if } a\sigma < -\pi/2 \\ \frac{1}{2}(1 + sin(a\sigma)) & \text{if } a\sigma \in [-\pi/2, \pi/2] \\ 1 & \text{if } a\sigma > \pi/2 \end{cases} \tag{7.7}$$

where a is a positive constant, which parameterizes a family of similar functions with a different interval of non-saturation. It is clear that, even for a large error bound ϵ, we can select many sets of two rational values $\Sigma = \{\sigma_0, \sigma_1\}$, such that $g_{sinsgm}(\sigma_0) = 0$, $g_{sinsgm}(\sigma_1) = 1$, $\sigma_0 < -\epsilon - \pi/2$ and $\sigma_1 > \epsilon + \pi/2$, which allow a stable state encoding with values $Y = \{0, 1\}$.

If a discrete activation function g is used in the SLRNN, such as the hard-limiter g_h, or a discretized approximation of one of the continuous activation functions aforementioned, then it is also easy to select sets Y and Σ that meet the condition (c3) for any reasonable error bound ϵ. Hence, any of these discrete functions g allows a stable state encoding. It is worthwhile to remember that SLRNNs with discrete activation functions can be trained using some neural learning algorithms, like the pseudo-gradient technique proposed by Zeng *et al.* [ZeGS:93].

Furthermore, in the case of a discrete g or a saturated continuous function like g_{sinsgm}, if the error bound ϵ takes into account the error caused by solving a real-valued linear system, which depends on the system matrix condition number, then the stability conditions (c1) and (c2) may be removed since they collapse into (c3).

7.2.3 The representation of the output function

With respect to the output function $\eta : I \times S \rightarrow O$ of a Mealy machine, two different representations are possible using the FS-SLRNN model, one that applies to SLRNNs and another one that applies to 2L-ASLRNNs.

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.(7.4); more precisely, it corresponds to the first P columns ($P < N$) in arrays W and B, that is:

$$\sigma_{dk} = g^{-1}(y_{dk}) = f(\mathbf{W}_k, \mathbf{I}_d, \mathbf{S}_d) \quad \text{for } 1 \leq d \leq D, \quad \text{for } 1 \leq k \leq P. \qquad (7.8)$$

Goudreau *et al.* [GoGC:94] demonstrated that second-order SLRNNs can implement any output function η for all the FSMs following this scheme, whereas first-order SLRNNs need to be augmented to implement some η functions, either by adding a feed-forward layer of output units, as in Minsky's and Elman's networks [Mins:67, Elman:90], or by allowing a one-time-step delay before reading the output [GoGi:93].

Therefore, an alternative approach is to separate the representation of the output function η from that of the state transition function δ by means of using a 2L-ASLRNN architecture. This can be done for both first- and second-order networks, and it is the appropriate choice if first-order RNNs are used to emulate FSMs. It may also be preferable in the case of second-order RNNs, since the total number of network weights is reduced and the output encoding is not involved in the state transition function.

Thus, in a 2L-ASLRNN, the N recurrent units are preserved for state encoding, and an output layer of P non-recurrent units O_l (with a bias weight in each unit) is added to provide an output vector $\mathbf{O}^t = [o_1(t), ..., o_P(t)]$, which is given by

$$o_l(t) \;=\; g\;(w_{l1} + \sum_{j=1}^{N} w_{l(1+j)} y_j(t)\;) \quad \text{for } 1 \le l \le P, \tag{7.9}$$

where g is not necessarily the same activation function of the recurrent layer, $o_l(t)$ refers to the activation value of output unit O_l at time t, w_{l1} is the bias weight of unit O_l, and $w_{l(1+j)}$, $1 \le j \le N$, are the weights associated with the connections from the recurrent units U_j to the output unit O_l.

In this case, after a "linearization" process similar to the one explained in subsection 7.2.1, a second linear system is obtained that represents η and affects to the weights of the output units:

$$g^{-1}(o_{il}) \;=\; w_{l1} + \sum_{j=1}^{N} w_{l(1+j)} y_{ij} \quad \text{for } 1 \le i \le n, \quad \text{for } 1 \le l \le P, \tag{7.10}$$

where the values o_{il} $(1 \le l \le P)$ are related to the code of the output symbol corresponding to the i-th state of the FSM. The preceding linear system of equations can be expressed in matrix form as

$$A_O W_O = B_O \tag{7.11}$$

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 net inputs, and n is the number of states in the FSM.

Note that a 2L-ASLRNN actually represents a Moore machine, where $\eta : S \rightarrow O$, unless the state encoding in some way incorporates the information of the previous input, as occurs in a split-state representation of Mealy machines [Mins:67].

In order to clarify the notation, from now on, we will refer to the linear system that only represents the state transition function δ 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 (i.e. non-augmented SLRNNs) and $AW = B$ includes both δ and η functions.

7.3 Implementation of FSMs in RNNs using the FS-SLRNN model

The FS-SLRNN model is used here to explain the representational limitations of first-order SLRNNs and to show that the FSM encoding methods by Minsky [Mins:67] and Goudreau *et al.* [GoGC:94] (for RNNs with a threshold activation function g_h) can be generalized, so that many other solutions exist for FSM implementation in SLRNNs and 2L-ASLRNNs, even with rather arbitrary activation functions, whenever the stability requirements mentioned in the previous section are met. The simulations of DFAs and stochastic DFAs in these architectures, based on the FS-SLRNN model, are discussed as well.

The generalized methods proposed for FSM implementation in first-order ASLRNNs and second-order SLRNNs (or ASLRNNs) do not necessarily yield the minimal size networks, but it is explained how the FS-SLRNN model may also be used to search a solution with less units. With this aim, a variation of the FS-SLRNN model that specifically deals with RNNs built with the threshold activation function g_h of Eq.(7.5) is described.

7.3.1 Implementation of FSMs in first-order ASLRNNs

Let $\mathcal{A} = (I, O, S, q_0, \delta, \eta)$ be an FSM. Let us first concentrate on the implementation of the state transition function $\delta : I \times S \rightarrow S$ in a first-order SLRNN, which is represented by a linear system $A_S W_S = B_S$ using the FS-SLRNN model. It is assumed that for all the m input symbols and for all the n states of the FSM $\mathcal{A}$, 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 rows in A_S is mn for a completely defined δ, 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 7.1. *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:

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

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 $N = n$. Since the number of rows (equations) in A_S is mn for a complete δ, the system $A_S W_S = B_S$ is in general unsolvable for first-order SLRNNs.

In order to design an SLRNN capable of implementing any δ 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 the state transition function of 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 δ, thus allowing a solution W_S. It is shown hereinafter that this apparently strict condition is met by Minsky's method [Mins:67].

The key point is that instead of representing the original FSM $\mathcal{A} = (I, O, S, q_0, \delta, \eta)$ with n states, the maximally-split equivalent FSM $\mathcal{A}' = (I, O, S', q_0', \delta', \eta')$ with at most $mn + 1$ states is represented, 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 in $\mathcal{A}$, each one associated with a transition $\delta(a_i, q_j) = q_k$; in the case that q_0 had no incoming transition, another state would be needed in S' to be assigned as the start state q_0'. The δ' and η' functions of $\mathcal{A}'$ are built accordingly to keep the same input/output behavior. For example, Figure 7.4 displays the (maximally-split) 4-state automaton equivalent to the 2-state odd parity recognizer shown in Fig. 7.1, where the start state can be either the state labeled $'0'q_1$ or the state labeled $'1'q_2$.

The next theorem generalizes Minsky's method for the implementation of any transition function δ in the recurrent layer of a first-order 2L-ASLRNN (this is, in fact, a first-order SLRNN). To illustrate the construction given in the theorem, Figure 7.5 displays the linear model, corresponding to the described solution, for the state transition function of the maximally-split odd parity recognizer shown in Fig. 7.4.

Theorem 7.2. *A set of sufficient conditions on a first-order SLRNN to implement the state transition function δ of an FSM $\mathcal{A} = (I, O, S, q_0, \delta, \eta)$ with m inputs and n states is the following:*

(i) *The first-order SLRNN has m input signals and as many units as states (at most $mn+1$) in the maximally-split equivalent FSM $\mathcal{A}' = (I, O, S', q_0', \delta', \eta')$.*

(ii) *An orthogonal encoding with 0 and 1 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 $\sigma_{01}, \sigma_{02}, \sigma_1$ such that $\sigma_1 = 2(\sigma_{01} - \sigma_{02}) + \sigma_{02}$, $g(\sigma_{01}) = 0$, $g(\sigma_{02}) = 0$ and $g(\sigma_1) = 1$.*

(iv) *The activation function g and the precision of the first-order SLRNN must satisfy the stability condition (c3) for $Y = \{0,1\}$ and $\Sigma = \{\sigma_{01}, \sigma_{02}, \sigma_1\}$.*

(v) *Let $\omega = \sigma_{01} - \sigma_{02}$ and $\theta = \sigma_{02}$. Let the pair (u,v), $u > 0$, be the label of the unit associated with the transition $\delta(a_u, q_v)$, and let $(0,0)$ be the label of the unit associated with a possible start state q_0 with no incoming transition, if any. A solution weight matrix W_S is given by the assignment:*

$$w_{(uv)l} = \begin{cases} \theta & \textit{if } l = 0 \ \ \textit{(this is the bias weight)} \\ \omega & \textit{if } l \textit{ stands for the connection from input } a_u \in I,\ u > 0 \\ \omega & \textit{if } l \textit{ stands for the connection from any of the units that code the} \\ & \textit{states in } S' \textit{ equivalent to (and split from) } q_v \in S \\ 0 & \textit{otherwise} \end{cases}$$

Proof.

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

Without loss of generality, let us assume that δ is complete. Then, the number of units in the first-order SLRNN (and states in S') will be $N = mn$ if $\exists a \in I$, $\exists q \in S$, $\delta(a,q) = q_0$, and $N = mn+1$ otherwise. A_S is a matrix of mN rows and $(m+N+1)$ columns (including a fixed input 1); whereas B_S has also mN rows but just N columns. The rows of both A_S and B_S can be organized into m blocks of N rows, where each block (R_i^A and R_i^B, respectively) 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 FSM. Each block of rows is divided into n sub-blocks (as many as states in the original FSM $\mathcal{A}$), and each sub-block (R_{ik}^A and R_{ik}^B, respectively) 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, which is associated with 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.

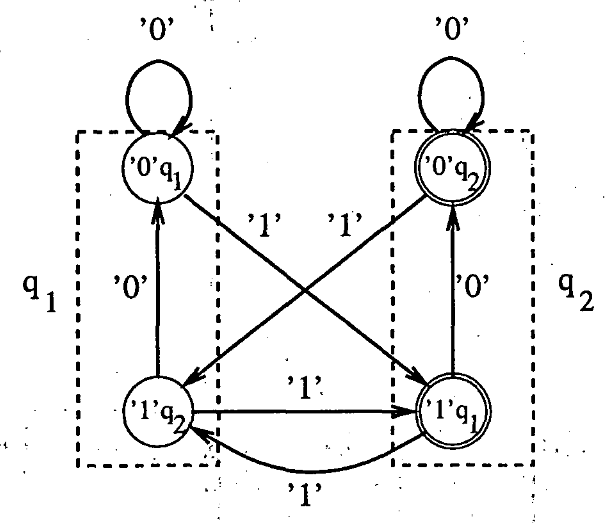

Fig. 7.4 *Maximally-split odd-parity recognizer*

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

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

Fig. 7.5 *Linear model representation of a first-order SLRNN that implements the state transition function of the (maximally-split) odd-parity recognizer.*

The columns of B_S, denoted as $\mathbf{c}_{uv}^B$, are labeled by two sub-indexes, u, v, which associate $\mathbf{c}_{uv}^B$ with the unit that flags the state of the maximally-split FSM characterized by "being the destination of a transition with the input symbol a_u from a state equivalent to the state q_v of the original FSM", if $u > 0$, or by "being a start state with no incoming transition", if $u = 0$.

Finally, let $\sigma_{(ij)(uv)}$ be the element of B_S in the row $\mathbf{r}^B_{ij}$ and in the column $\mathbf{c}^B_{uv}$, 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 7.1 establishes that the rank of A_S is $N + m - 1$ and there exist $(N-1)(m-1)$ linear relations among the rows of A_S, these being the following:

$$\mathbf{r}^A_{ij} = \mathbf{r}^A_{i1} + \mathbf{r}^A_{1j} - \mathbf{r}^A_{11} \quad 2 \le i \le m, \quad 2 \le j \le N. \tag{7.12}$$

In order to prove the theorem we need to demonstrate that for all the mN transitions of δ', 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

$$\mathbf{r}^B_{ij} = \mathbf{r}^B_{i1} + \mathbf{r}^B_{1j} - \mathbf{r}^B_{11} \quad 2 \le i \le m, \quad 2 \le j \le N. \tag{7.13}$$

By multiplying matrix A_S by the matrix W_S built up with the proposed weight assignment, it results that the net inputs of the network units can be expressed as follows:

$$\sigma_{(ij)(uv)} = \begin{cases} 2\omega + \theta & \text{if } u = i \ \wedge \ v = K(j) \\ \omega + \theta & \text{if } u = i \ \oplus \ v = K(j) \\ \theta & \text{if } u \ne i \ \wedge \ v \ne K(j) \end{cases} \tag{7.14}$$

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

$$y_{(ij)(uv)} = \begin{cases} 1 & \text{if } u = i \ \wedge \ v = K(j) \\ 0 & \text{otherwise} \end{cases} \tag{7.15}$$

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 mN transitions of δ'. This proves the theorem.

In order to show that the assignment of B_S elements given by Eq.(7.14) implies the satisfaction of the linear relationships

$$\sigma_{(ij)(uv)} = \sigma_{(i1)(uv)} + \sigma_{(1j)(uv)} - \sigma_{(11)(uv)} \quad 2 \le i \le m, \ 2 \le j \le N, \ 1 \le u \le m, \ 1 \le v \le n, \tag{7.16}$$

a case-by-case analysis can be performed as follows.

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 $\mathbf{r}^B_{ij}$ in the same subblock $R^B_{iK(j)}$ 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.(7.16) by the values given by Eq.(7.14):

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

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

□

Minsky's proposal for the implementation of the state transition function of any Mealy machine in networks of McCulloch-Pitts neurons [Mins:67] is a particular case of the solution established in Theorem 7.2, in which the activation function is the hard-limiter g_h of Eq.(7.5), $\omega = 1$ and $\theta = -2$ (i.e. $\Sigma = \{-2,-1,0\}$). For a first-order SLRNN with error bound ϵ, and using the same activation function g_h, there is a stable solution for any positive number $\omega > 2\epsilon$, where a negative number θ satisfying $-\theta \in (\omega+\epsilon, 2\omega-\epsilon]$ can be chosen. Other solutions with different activation functions, such as g_{sinsgm} of Eq.(7.7), can be imagined.

Concerning the output function $\eta : I \times S \rightarrow O$ of the FSM, Goudreau *et al.* [GoGC:94] proved that first-order SLRNNs cannot implement all the possible output mappings. Therefore, the linear system $A_O W_O = B_O$ of Eq. (7.11), 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 $P \geq 1$ and any possible matrix B_O. Moreover, the only requirement on the activation function g of the output layer is that the range of g must include the values used in the encoding of the output symbols.

In summary, in order to implement all FSMs, first-order SLRNNs need both to use state splitting (to implement all δ functions) and to be augmented (to implement all η functions), i.e. first-order 2L-ASLRNNs are required[3].

7.3.2 Implementation of FSMs in second- and higher-order SLRNNs

In order to represent any state transition function δ of an FSM with m inputs and n states in an SLRNN without the need of state splitting, the rank of matrix A_S in the FS-SLRNN model must become mn, so that network weights W_S can always be found for any B_S. To this end, some terms of second or higher order are needed as neuron inputs to provide the required number of linearly independent columns in A_S. 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 with activation function g_h, and select an orthogonal one-hot encoding for both inputs and states (this is the solution described by Goudreau *et al.* [GoGC:94]). In this case, it is easy to show that A_S is always an identity diagonal matrix (see, for instance, Fig. 7.3).
- Use a high-order SLRNN of just one recurrent unit 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 ($1 \leq c \leq mn$),
$$f_c(x_1, y_1) = x_1^{u_c} y_1^{v_c} \tag{7.17}$$
where u_c and v_c are positive integers such that $u_c = ((c-1) mod\ m) + 1$ and $v_c = ((c-1) div\ m) + 1$, and the operators *div* and *mod* refer to integer division and module operations. This SLRNN can be regarded as an extreme case in which the required linear independence is achieved through the variable-order nonlinear connections.

[3]Equivalently, the output units might be put with zeroed outgoing connections in a first-order SLRNN and read with one-time-step delay [GoGi:93].

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

Theorem 7.3. *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 a given FSM $A = (I, O, S, q_0, \delta, \eta)$, then, in order to implement the state transition function δ, the only requirement is the satisfaction of the stability conditions (c1), (c2) and (c3).*

Proof. The rows of matrix A_S represent, under the given encoding, the pairs (input,state) for which the state transition function δ is defined. The activation function g and the chosen encoding determine the contents of matrix B_S for the given δ. 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 (c3)). □

On the other hand, the output function $\eta : I \times S \rightarrow O$ of any Mealy machine can easily be implemented by extending any of the configurations of high-order SLRNN and data encoding characterized by a full rank of matrix A_S. For example, second-order SLRNNs with local orthogonal encoding for both states and inputs may be used. As commented in Section 7.2, the system $A_S W_S = B_S$ can be extended to a larger system $AW = B$ that also includes the output function η, in which P recurrent units are added 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 in the extended matrices A and B are the same than in the matrices A_S and B_S. Hence, if the rank of A_S is equal to the number of rows, then *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.

Finally, the output function $\eta : S \rightarrow O$ of any Moore machine with n states may be implemented as well in the output layer of a second- or higher-order 2L-ASLRNN, subject to the same conditions mentioned for the case of first-order 2L-ASLRNNs. This is, to use a state encoding (with a number of recurrent units $N \geq n$) such that *rank*$(A_O) = n$, in order to solve the linear system $A_O W_O = B_O$, which represents η in the output layer, for any possible output encoding.

7.3.3 Implementation of DFAs and stochastic DFAs

Let $\mathcal{A}_D = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting some regular language. It is clear that $\mathcal{A}_D$ is equivalent to a Moore machine $(I, O, S, q_0, \delta, \eta)$, where $I = \Sigma$, $O = \{0, 1\}$, $S = Q$, $F \subseteq S$, and the output function $\eta : S \to O$ is defined as

$$\eta(q) = \begin{cases} 1 & \text{if } q \in F \text{ (i.e. } q \text{ is a final state)} \\ 0 & \text{otherwise} \end{cases}$$

Hence, since $\mathcal{A}_D$ is a Moore machine, it can be implemented in an RNN using one of the approaches based on the FS-SLRNN model that have been explained previously. In all cases, only a single output unit is enough ($P = 1$) to encode the two output symbols of the FSM, associated with the "accept" and "reject" actions of the DFA, using a pair of different values in the range of the activation function g. Let us say that $\mathcal{A}_D$ contains n states and $|\Sigma| = m$. Then, $\mathcal{A}_D$ may be implemented, for instance, using a local orthogonal encoding for inputs and states and a proper activation function g (recall the stability conditions), in a first-order 2L-ASLRNN with mn recurrent units and 1 output unit, in a second-order SLRNN with $n + 1$ units, or in a second-order 2L-ASLRNN with n recurrent units and 1 output unit.

The case of deterministic UFSAs (DUFAs) can be handled similarly. Let $\mathcal{U}_D = (Q, \Sigma, \delta, q_0, F_P, F_N)$ be a DUFA. Then, $\mathcal{U}_D$ is equivalent to a Moore machine with output set $O = \{-1, 0, 1\}$ and output function

$$\eta(q) = \begin{cases} 1 & \text{if } q \in F_P \text{ (i.e. } q \text{ is a final positive state)} \\ -1 & \text{if } q \in F_N \text{ (i.e. } q \text{ is a final negative state)} \\ 0 & \text{otherwise} \end{cases}$$

A DUFA $\mathcal{U}_D$ may be implemented, for example, in the same RNN configurations than a DFA $\mathcal{A}_D$ of the same size, except that the activation function of the output unit should provide three different values, not necessarily $\{-1, 0, 1\}$, to encode the three output symbols. Note also that the *complete* DUFAs, where all states are either final positive or final negative, can be implemented as DFAs.

The FS-SLRNN model can also be adapted to represent and implement stochastic DFAs in 2L-ASLRNNs, as is shown next, whenever the activation function of the output units is selected to include the interval $[0, 1]$ in its range. Thus, activation functions like g_{lin} (6.3), g_{al} (6.18), g_{sin} (6.24) or g_{sinsgm} (7.7) may be used in the output units; a sigmoid function like g_s (4.5) or g_{s3} (7.6) could in principle be used too, taking into account that the output values 0 and 1 could only be approximated.

Let $\mathcal{A}_{SD} = (Q, \Sigma', \delta, q_0, p)$ be a stochastic DFA accepting some stochastic regular language, where $\Sigma' = \Sigma \cup \{\$\}$ is an extended input alphabet that includes a special symbol \$ (which is used as mark of the end of a string), $\delta : \Sigma \times Q \rightarrow Q$ is a deterministic state transition function, and $p : \Sigma' \times Q \rightarrow [0,1]$ is an associated probability function such that

$$\begin{aligned} \forall q_j \in Q : \quad & \sum_{a_i \in \Sigma'} p(a_i, q_j) = 1; \\ \forall a_i \in \Sigma, \ \forall q_j \in Q : \quad & p(a_i, q_j) > 0 \iff \exists q_k \in Q, \ \delta(a_i, q_j) = q_k; \end{aligned}$$

$p(\$, q_j)$ is the conditional probability of reading (or generating) the end mark of a grammatical string[4] in state q_j (and accepting the string read in so far); and $p(a_i, q_j)$, $a_i \in \Sigma$, is the conditional probability of reading (or generating) the input symbol a_i in state q_j as the next symbol of a grammatical string (and moving to the state given by $\delta(a_i, q_j)$). All the states $q_f \in Q$ such that $p(\$, q_f) > 0$ are considered as final states. Each string $s = c_1c_2...c_{|s|}$ in the stochastic regular language represented by $\mathcal{A}_{SD}$ has an associated probability p_s, which is computed as the product of probabilities $p(c_1, q_0)\ p(c_2, \delta(c_1, q_0))\ ...\ p(\$, q_f)$, where q_f is the final state reached after reading (or generating) the last symbol $c_{|s|}$.

Although a stochastic DFA is not exactly an FSM, it can be represented using the FS-SLRNN model by defining an equivalent *pseudo* Moore machine with a probabilistic output function η. Thus, a *pseudo* Moore machine associated with a stochastic DFA $\mathcal{A}_{SD} = (Q, \Sigma \cup \{\$\}, \delta, q_0, p)$ can be defined as the sixtuple $(I, O, S, q_0, \delta, \eta)$, where $I = \Sigma$, $O = \Sigma \cup \{\$\}$, $S = Q$, and the probabilistic output function $\eta : S \times O \rightarrow [0,1]$ is simply given by the next-symbol probabilities of each state, $\eta(q, a) = p(a, q)$.

Hence, we can use a 2L-ASLRNN to implement the state transition function δ of a stochastic DFA (of n states and m input symbols) in the recurrent layer, and take $P = m + 1$ output units, one for each input symbol plus one for the end-symbol \$, to implement the probabilistic η in the output layer. To this end, each of the activation values o_{il}, $1 \leq i \leq n$, $1 \leq l \leq P$, in Eq.(7.10) must correspond to the probability $\eta(q_i, a_l) = p(a_l, q_i)$ and the inverse values $g^{-1}(o_{il})$ must exist (or be chosen) for the activation function g of the output layer, in order to build matrix B_O and solve the linear system $A_O W_O = B_O$.

It can be seen that this representation of stochastic DFAs is similar to the one that can be inferred from positive examples of a stochastic regular language by training a 2L-ASLRNN to learn the next-symbol prediction task (recall Sections 4.2.1 and 6.2.2). Moreover, the 2L-ASLRNN built from the stochastic DFA can be used to accept the grammatical strings and reject the ungrammatical ones, as explained in Section 4.2.1, provided that a prediction threshold u is selected in the proper range given by Eq.(4.95).

[4]A grammatical string is a string in the stochastic regular language accepted (and generated) by the stochastic DFA.

To the contrary of the learning case, where it may be difficult to select u, there is no problem to choose u when the stochastic DFA is known.

Finally, a consequence that can be extracted from the work by Goudreau *et al.* [GoGC:94] is that first-order SLRNNs may not be able to represent any possible probabilistic output mapping of a stochastic DFA, and therefore, they need to be augmented to implement all the stochastic DFAs. On the other hand, second-order SLRNNs can implement all the stochastic DFAs, whenever a stable encoding of the required probability values associated to each state is guaranteed[5].

7.3.4 The problem of searching minimal size networks that implement a given FSM

Although some methods to implement FSMs or FSAs in first- and second-order RNNs have been proposed, both here and elsewhere [Mins:67, AlDO:91, GoGC:94, OmGi:96], none of them returns the smallest neural network (of first- or second-order type) that is able to emulate a given FSM or FSA. It would be very interesting to design a method which built the smallest network required. However, it is generally believed that the problem of finding the minimal size second-order SLRNN implementing a given FSM or FSA is an NP-complete problem [SiSG:92, OmGi:96] (and a similar belief applies to the case of first-order 2L-ASLRNNs).

In a work related to this topic, Siegelmann *et al.* [SiSG:92] showed that the number of neurons required in second-order SLRNNs with linear activation function to accept a regular language (*l-complexity*) is an upper bound for second-order SLRNNs with sigmoidal and saturated activation functions. Moreover, they reported a method to compute an upper bound of the *l-complexity* of a regular language L, the so-called $\mathcal{H}$*-complexity*, such that $\forall L : \mathcal{H}\text{-}complexity(L) \leq n$, where n is the number of states of the minimal DFA accepting L (i.e. the canonical automaton for L). Note that in the methods by Omlin and Giles [OmGi:96], Goudreau *et al.* [GoGC:94], and the generalization of the latter given in 7.3.2, a second-order SLRNN of size $n+1$ is constructed to implement a DFA of n states. Indeed, Siegelmann *et al.* claimed that a linear-activation network with $\mathcal{H}$*-complexity*(L) size can be built from a regular expression R describing L in time polynomial in the size of R, using techniques from the theory of rational power series in noncommuting variables [SiSG:92].

In the following, it is discussed how the FS-SLRNN model presented might help in

[5]A possible way to achieve this is by setting to zero the weights corresponding to the connections from the output units to all the units of the SLRNN.

the search of smaller solutions than those provided by the methods described in the previous subsections which used a unary state encoding.

Let us first consider the problem of implementing a given Mealy machine $\mathcal{A} = (I, O, S, q_0, \delta, \eta)$ with m inputs, n states and p outputs in a second-order SLRNN using a denser representation of states. Assume that a dense state encoding is selected that uses less than n units, and that a certain output encoding is chosen, where the number of output units is $P \leq p$, so that the total number of units is $N < n + p$. In addition, assume that a fixed encoding is taken for the m input symbols through the M input signals of the SLRNN. Once these encodings are known, the matrix A of a linear system $AW = B$ representing both δ and η of $\mathcal{A}$ can be written. Let Y be the set of values used in the state and output codes, e.g. $Y = \{0, 1\}$ in a binary representation. If, thanks to the P columns corresponding to the output values, it results that $rank(A) = mn$ (the number of rows), then many solutions can be easily obtained with activation functions that include Y in their range. But, normally for dense representations, the rank of A will be less than mn and a number of $mn - rank(A)$ linear relations among the rows of A will hold.

If the activation function g of the SLRNN is such that a single inverse value $g^{-1}(y)$ exists for each of the values $y \in Y$, forming a unique possible set Σ of inverse values associated with Y, then a unique linear system $AW = B$ representing $\mathcal{A}$ under the given encodings can be established. In such a case, it suffices to test the solvability of this system to know whether the selected encoding scheme allows the implementation of the given FSM; to this end, if $rank(A) < mn$, then the rows of B must satisfy the same linear relations. However, if two or more inverse values can be chosen for some of the values in Y for the activation function g, then an exponential number of possible systems $AW = B$, all with same A but different B, should be checked in principle. In some cases, as for the threshold activation function g_h of Eq.(7.5), the number of possible systems to test would be even infinite.

To avoid this shortcoming, an extension of the FS-SLRNN model is presented next that enables to know whether a selected encoding of a given FSM $\mathcal{A}$ is implementable by an SLRNN with the threshold activation function g_h and a bias weight in each unit, by means of solving at most N linear programming problems and one linear system of equations. Once an aggregation function f is chosen, that determines whether a first-, second-, or higher-order SLRNN is involved, and some binary encoding is selected for state and output representation, a nonlinear system of equations

$$y_{ij} = g_h\left(f\left(\mathbf{W}_j, \mathbf{I}_i, \mathbf{S}_i\right) \right) \quad \text{for } 1 \leq i \leq mn, \quad \text{for } 1 \leq j \leq N, \tag{7.18}$$

can be established that represents δ and η of $\mathcal{A}$, where $\mathbf{W}_j$ is the vector of weights of the j-th unit, the vectors $\mathbf{I}_i$, $\mathbf{S}_i$ encode the i-th (input,state) pair, and $y_{ij} \in \{0, 1\}$.

Now, instead of forcing some specific inverse values in the domain of g_h for the 0 and 1 values, we relax each equation in the preceding system to a constraint

$$\sigma_{ij} \geq 0 \text{ or } \sigma_{ij} < 0 \quad \text{for } 1 \leq i \leq mn, \quad \text{for } 1 \leq j \leq N, \tag{7.19}$$

depending on whether $y_{ij} = 1$ or $y_{ij} = 0$, where $\sigma_{ij} = f(\mathbf{W}_j, \mathbf{I}_i, \mathbf{S}_i)$, and the availability of bias weights allows setting the threshold in the constraints to zero without loss of generality. We still can adopt a flexible matrix representation

$$AW = B_H \tag{7.20}$$

for the above linear system of constraints, which means that the elements σ_{ij} of the product of matrices AW, where W is unknown, must satisfy either $\sigma_{ij} \geq 0$ or $\sigma_{ij} < 0$ depending on the selected encoding of the FSM, as given by the matrix of constraints B_H.

Again, if all the rows of A are linearly independent, i.e. $rank(A) = mn$, then a solution W is easily determined by first setting all the variables σ_{ij} to arbitrary values in the allowed ranges and then solving an actual linear system $AW = B$. However, when $rank(A) < mn$, the method described in Algorithm 7.1 may be used to check the existence of a solution and to obtain a weight matrix W satisfying the constraints, if possible. This method tries to build a matrix B, column by column, such that it meets the inequalities expressed by B_H and, at the same time, satisfies the same row linear relations than A. If such a matrix B can be constructed, then a solution W is found by solving $AW = B$.

Now, the problem of finding a minimal-size second-order SLRNN with the threshold activation function g_h that implements a given DFA $\mathcal{A}_D = (Q, \Sigma, \delta, q_0, F)$ with n states can be transformed into the problem of finding a state (and output) encoding $\gamma : Q \rightarrow [0, 2^N - 1]$ with minimum N for which there exists a weight matrix W satisfying the set of constraints $AW = B_H$ associated with the representation of $\mathcal{A}_D$ using γ (and a local orthogonal input encoding). Unfortunately, this seems to be a hard problem. Only to illustrate the combinatorial complexity of the task, Algorithm 7.2 describes an exponential-time method, which would (theoretically) solve the problem by exhaustive search. In the worst case, the solution derived from Theorem 7.3 with $N = n + 1$ and a local one-hot state encoding would be returned. Algorithm 7.2 has a worst-case time complexity of $O(n2^{n^2})$. Even though a lot of computations, which are redundant in the exhaustive search, might be avoided by structuring the search space in some intelligent manner that pruned the encodings to test, it is probably impossible to devise a polynomial-time algorithm that found the minimal network due to the combinatorial nature of the problem. As far as I know, this remains an open question for research.

ALGORITHM 7.1: *Determines the existence of a weight solution to implement an FSM in a threshold-based SLRNN with a given binary encoding for states and outputs, and finds one such solution if any.*

```
Inputs:
    $N$      is the number of units in the SLRNN (including output units);
    $A$      is a matrix of values such that $rank(A) < number\_of\_rows(A) = mn$;
    $B_H$    is a matrix of constraints such that $number\_of\_rows(B) = number\_of\_rows(A)$,
             and every element of the matrix is either $\sigma_{ij} \geq 0$ or $\sigma_{ij} < 0$;
Outputs:
    existence   is a boolean variable which tells whether or not there exists some weight
                matrix $W$ for which the product $AW$ satisfies the constraints $B_H$;
    $B$         when existence=TRUE, it contains a matrix of values that satisfies $B_H$;
    $W$         when existence=TRUE, it contains the solution of $AW = B$;
begin_algorithm
    Let $r_i^A$ denote a real-valued variable associated with the $i$-th row of $A$, $1 \leq i \leq mn$;
    Establish an homogeneous system of equations $Z\mathbf{R}^A = \mathbf{0}$ that represents the linear
        relations found among the rows of $A$, where $\mathbf{R}^A = (r_1^A, ..., r_{mn}^A)^T$ and $Z$ is the
        matrix of coefficients given by the linear relations;
    Solve $Z\mathbf{R}^A = \mathbf{0}$ in a parametric way (because the system is underdetermined), thus
        obtaining a parameterized vector $\mathbf{R}^A$ which stands for a linear subspace including
        the origin in the space $\Re^{mn}$;
    existence := TRUE;
    for $j := 1$ to $N$ do
        Let $\mathbf{B}_{Hj}$ be the $j$-th column of the matrix of constraints $B_H$;
            { $\mathbf{B}_{Hj}$ represents a cone of $\Re^{mn}$ which corresponds approximately to a $2^{mn}$-
            orthant of the space }
        if $(\mathbf{R}^A \cap \mathbf{B}_{Hj}) \neq \emptyset$
            { the Simplex algorithm can be used to test the feasibility of the intersection
            space, since this is a linear programming problem }
        then
            Select some arbitrary point $\mathbf{P} = (r_1, ..., r_{mn}) \in (\mathbf{R}^A \cap \mathbf{B}_{Hj})$ to set the values
                $\sigma_{ij}$ $(1 \leq i \leq mn)$ in the $j$-th column of matrix $B$;
        else
            existence := FALSE;
            break { the loop $j$ };
        end_if
    end_for
    if existence then
        Solve $AW = B$, since both $A$ and $B$ have the same linear relations among rows.
    end_if
end_algorithm
```

ALGORITHM 7.2: *Finds a threshold-based second-order SLRNN of minimal size that implements a given DFA by exhaustive search.*

Inputs:

$\mathcal{A}_D$	is a given DFA $(Q, \Sigma, \delta, q_0, F)$ with n states;

Outputs:

N_{min}	is the minimal number of units that are needed to implement $\mathcal{A}_D$ in a threshold-based second-order SLRNN;
W_{min}	contains a solution of weights for a second-order SLRNN with N_{min} units;
γ_{min}	is the state (and output) binary encoding $\gamma_{min} : Q \rightarrow [0, 2^{N_{min}} - 1]$ used in the solution;

```
begin_algorithm
  $N_{min} := n + 1$;
  for $N := \lceil \log_2 n \rceil + 1$ to $n$ do
    for $i := 1$ to $\binom{2^{N-1}}{n}$ {different code sets of $n$ elements with $N-1$ bits}
      do
        for $j := 1$ to $n!$ {different permutations of $n$ elements within a code set}
          do
            for $k := 1$ to 2 {possible output encodings with 1 bit} do
              Generate a new encoding table $\gamma : Q \rightarrow [0, 2^N - 1]$;
              Write matrix $A$ and constraints $B_H$ of Eq.(7.20) associated with
                the representation of $\mathcal{A}_D$ using $\gamma$;
              $<existence, B, W> :=$ Run_Algorithm_7.1 $(A, B_H)$;
              if existence then
                $N_{min} := N$;   $W_{min} := W$;   $\gamma_{min} := \gamma$;
                break { the loops $k, j, i, N$ };
              end_if
            end_for { $k$ }
          end_for { $j$ }
      end_for { $i$ }
  end_for { $N$ }
  if $N_{min} = n + 1$ then
    Store the orthogonal one-hot encoding in $\gamma_{min}$;
    Write matrix $A$ and constraints $B_H$ of Eq.(7.20) associated with
        the representation of $\mathcal{A}_D$ using $\gamma_{min}$;
    Choose an arbitrary matrix $B$ satisfying $B_H$;
    Solve $AW_{min} = B$;
  end_if
end_algorithm
```

It must be remarked that the general problem of finding a minimal-size second-order SLRNN with the threshold activation function g_h that implements a given FSM $\mathcal{A} = (I, O, S, q_0, \delta, \eta)$ with n states and p output symbols is still more difficult than the previous one, since p is simply 2 for a DFA. Note that if the exhaustive search method described by Algorithm 7.2 were extended to the case of an FSM with p output symbols, a number exponential in p of possible output encodings should be tested in the innermost loop instead of just the two output encodings that were tested for the case of a DFA. Moreover, if one deals with first-order ASLRNNs instead of second-order SLRNNs, the situation is even worse, since in order to implement an FSM with n states and m inputs, more than n units may be needed (up to $N = mn$, recall Theorem 7.2), and therefore, an exhaustive search procedure for first-order ASLRNNS would have to check not only the given FSM but also other equivalent FSMs resulting from splitting up to n states, which would further increase the time complexity by a factor of 2^n.

7.4 Insertion of FSMs in RNNs for subsequent learning

The FS-SLRNN model can be employed to insert symbolic knowledge, in the form of an incomplete FSM, into 2L-ASLRNNs or second-order SLRNNs and to preserve this knowledge (i.e. the emulation of the inserted FSM) while the network is trained from examples to complete an input/output behavior that minimizes the error in a given task. This is useful when an RNN-based inductive process is to be guided according to *a-priori* knowledge of the problem or from a symbolic partial solution. To be applied in such a guided learning approach, an RNN of one of the above types can be made up of any activation function g that supports a neural learning scheme, provided that the stability conditions mentioned in Section 7.2.2 are fulfilled. Remember that learning algorithms are available even for RNNs with a discrete g such as the hard-limiter g_h (e.g. the pseudo-gradient technique described by Zeng *et al.* in [ZeGS:93]).

We have seen that two linear systems, $A_S W_S = B_S$ and $A_O W_O = B_O$, are constructed in the FS-SLRNN model to represent the transition and output functions, respectively, of an FSM in a 2L-ASLRNN from a selected encoding of inputs, states and outputs. In general, these linear systems can be overdetermined, determined or underdetermined, depending on the given FSM, the data encodings (which determine the number of input, state and output units in the net), and the specific architecture (usually first- or second-order). The key point is that, given a sufficient number of recurrent units in the network, 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, a 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.

A similar argument applies to the case of second-order SLRNNs, where an underdetermined system $AW = B$ can be obtained, given sufficient recurrent units, that represents both δ and η functions of an FSM. On the other hand, it has been demonstrated that first-order SLRNNs are not able to represent all FSMs, whatever the number of units, and therefore, they are not appropriate for FSM insertion.

In the following, only the insertion of Moore machines in 2L-ASLRNNs is described, where the emulation of the transition and output functions is clearly separated: the recurrent layer takes care of δ and the output layer takes care of η. However, a similar method to the one explained next can be easily derived to insert Mealy machines in second-order SLRNNs.

Let $\mathcal{A} = (I, O, S, q_0, \delta, \eta)$ be a Moore machine with m inputs, n states and p outputs. It is assumed that $\mathcal{A}$ corresponds to a part of an unknown target machine $\mathcal{A}^T = (I, O, S^T, q_0, \delta^T, \eta^T)$ to be learned by the 2L-ASLRNN, where $S^T \supseteq S$, δ is a partial definition of δ^T, and η is a partial definition of η^T. At first, the number of states n^T in the target FSM is considered to be unknown, though $n^T \geq n$. The FSM insertion method consists of two steps, to be performed prior to the learning stage:

1. Establish underdetermined linear systems $A_S W_S = B_S$ and $A_O W_O = B_O$, that represent the δ and η functions of $\mathcal{A}$, using a 2L-ASLRNN architecture and a data encoding suitable for FSM implementation, but including more units than those required to solve the systems (e.g. use a second-order 2L-ASLRNN with orthogonal encoding for inputs and states and $N > n$).
2. Initialize the weights of the hidden and output units to any of the solutions W_S and W_O, respectively, that result from solving the underdetermined linear systems built in the previous step.

The problem of fixing the number of additional units is similar to the problem of fixing N when the network starts from scratch (no a-priori knowledge) to learn a task from examples. The selection of the number of additional units is easier if there is an estimate of n^T, $\tilde{n}^T$, since in that case, an upper bound is given by $\tilde{n}^T - n$.

Concerning the selection of a particular initialization for the network weights, this should help to provide a good starting point for the learning algorithm to be applied afterwards. For gradient-based learning algorithms, a preferred initialization may be one in which the sum of squared weights is minimal (or near minimal) among the possible solutions and all the network weights are non-zero. Indeed, it is easy to show that the assignment of the free weights that produces the minimal sum of squared weights can always be found by solving, for each unit, a linear system with the free weights in that unit as variables. Likewise, it must be noted that every dependent weight is a linear combination of free weights within the same neuron, since for every column of the solutions W_S and W_O there is a set of linear relations involving only the weights in that column.

Obviously, if all the network weights are freely modified by the neural learning algorithm, then the emulation of the inserted FSM is no longer guaranteed. Thus, if any of the true gradient-descent algorithms described in Section 4.1.2 for training ASLRNNs, which update *all* the network weights, were used after FSM insertion, then the injected knowledge would probably be degraded and eventually forgotten. Although this behavior allows for knowledge refinement and it might 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 keep the inserted FSM, a *constrained neural learning* procedure is proposed. It basically consists of minimizing the error function with respect to the free weights, taking into account their influence on the rest of weights of the net, and updating the latter to maintain the relationships. In this way, just a linear subspace of the whole weight space is explored to minimize the error, where all the points in this subspace correspond to networks that emulate the inserted FSM.

For example, if a gradient-descent algorithm based on BPTT is used to train the 2L-ASLRNN, then the free weights of a recurrent hidden unit U_k, $1 \leq k \leq N$, should be changed according to

$$\Delta w_{kl} = -\alpha \left(\frac{\delta E_{total}(1,t_f)}{\delta w_{kl}} + \sum_{w_{ka} \in D(W_k)} \frac{\delta E_{total}(1,t_f)}{\delta w_{ka}} \frac{\delta w_{ka}}{\delta w_{kl}} \right) \tag{7.21}$$

where α is the learning rate, $E_{total}(1,t_f)$ is the overall network error in a training sequence, $D(W_k)$ denotes the subset of dependent weights in unit U_k, and the partial derivatives $\frac{\delta w_{ka}}{\delta w_{kl}}$ are known constants given by the linear relations among weights. The same algorithm described in Section 4.1.2 can be employed to compute both the partial derivatives $\frac{\delta E_{total}(1,t_f)}{\delta w_{kl}}$ and $\frac{\delta E_{total}(1,t_f)}{\delta w_{ka}}$. On the other hand, the weights w_{ka} in $D(W_k)$ should be changed after updating all the free weights w_{kl}, to meet again the linear constraints specified in the underdetermined solution of the system $A_S W_S = B_S$. A similar approach should be used to modify the free and dependent weights, respectively, of each of the output units O_i, $1 \leq i \leq P$, subject to the linear constraints in the solution of $A_O W_O = B_O$.

The gradient-descent methods for training 2L-ASLRNNs based on RTRL (for on-line learning) and Schmidhuber's algorithm can also be adapted easily to follow this constrained neural learning procedure. For instance, the counterpart of Eq.(7.21) in the case of using RTRL for the recurrent layer is

$$\Delta w_{kl}(t) = -\alpha \left(\frac{\delta E(t)}{\delta w_{kl}} + \sum_{w_{ka} \in D(W_k)} \frac{\delta E(t)}{\delta w_{ka}} \frac{\delta w_{ka}}{\delta w_{kl}} \right) \tag{7.22}$$

where $E(t)$ is the overall network error at time step t, and the partial derivatives $\frac{\delta E(t)}{\delta w_{kl}}$ and $\frac{\delta E(t)}{\delta w_{ka}}$ are calculated forward in time at each step t.

Since DFAs, DUFAs, and stochastic DFAs can all be represented as Moore machines, as shown in Section 7.3.3, the insertion and constrained learning methods that have been described here are perfectly applicable when the symbolic knowledge to be injected into the network takes the form of one of these types of automata[6]. This opens the door to the possibility of incorporating a priori knowledge in the connectionist approaches to regular grammatical inference, as is discussed in the next section. This knowledge will typically be given as a (partial) DUFA in the case of learning the classification task from positive and negative examples, and as a (partial) stochastic DFA in the case of learning the next-symbol prediction task from only positive strings.

7.5 The active grammatical inference (AGI) methodology

We call *active grammatical inference* (or AGI, for short) to a methodology for the inference of DFAs or DUFAs from examples and optional a-priori knowledge, in which RNNs are used as the inductive inference engine [SaAl:95]. The AGI methodology encompasses a large variety of heuristic connectionist and hybrid RGI approaches, with the novel feature that the learning process performed by the neural network can be guided. In the most general case, the whole process is conceived as a sequence of learning cycles, where a cycle includes the steps of automaton insertion, (possibly constrained) neural training, automaton extraction, symbolic manipulation, and validation. Hence, the neural training step in each cycle can be biased by inserting some symbolic knowledge prior to learning from examples.

If the learning examples are restricted to only positive strings, the RNN is trained for a prediction task (as explained in Section 4.2.1), whereas if both positive and

[6]Actually, stochastic DFAs can only be associated with pseudo-Moore machines having a probabilistic output function (see Section 7.3.3), but this extended class of FSMs can also be inserted using the proposed approach.

negative strings are supplied, the RNN is trained for a classification task (as explained in Section 4.2.2). Second-order RNNs are preferred to first-order RNNs, since in the latter the insertion of a DFA may require the insertion of its equivalent maximally split automaton, and consequently, a much larger number of neurons. Also, 2L-ASLRNNs are preferred to SLRNNs, because they allow a clean separation between the representation of the state transition function δ and the output function η, which is related to final state information or symbol prediction probabilities. As already mentioned, the recurrent layer in a 2L-ASLRNN is just concerned with learning or simulating δ, while the output layer is just concerned with η. To sum up, second-order 2L-ASLRNNs are recommended for AGI.

The global general procedure of the AGI approach is as follows (see also Figure 7.6):

Step 1: *Initialization of RNN and learning mode.*
The weights of the RNN are initialized by the FSM insertion procedure if some a-priori knowledge is available, or randomly otherwise; the learning mode is selected respectively as *constrained* or *free*.

Repeat

> **Step 2**: *Neural run.*
> In *free learning* mode, all the weights of the RNN are trained using a gradient-descent learning algorithm either to predict or to classify the given training strings. In *constrained learning* mode, only some of the weights of the RNN are trained, and the rest are updated to keep the inserted rules. The neural run ends when a stop criterion is satisfied that depends on the trained task.
>
> **Step 3**: *FSA extraction from RNN.*
> A DFA or a DUFA is extracted from the RNN dynamics, which is always consistent with the given examples.
>
> **Step 4**: *Validation.*
> If the extracted automaton passes a validation test, typically related to generalization performance, then **Exit** loop and **Stop**;
> otherwise **Continue**.
>
> **Step 5**: *Symbolic manipulation.*
> The extracted automaton is transformed (by statistical filtering, deletion or addition of states and transitions, ...) to improve or simplify the current hypothesis, according to prior knowledge of the problem, general heuristics, or from the results of the validation test.
>
> **Step 6**: *FSA insertion into RNN.*
> The transformed automaton is inserted into a new RNN, through (underdetermined) linear system solving, using a chosen encoding to represent the states. The learning mode of the next neural run is selected.

End_repeat

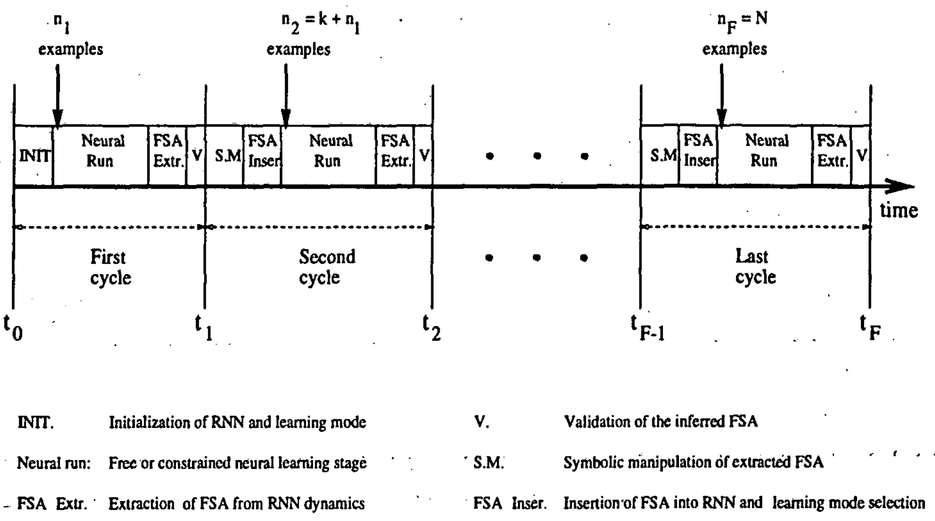
n_1 examples
$n_2 = k + n_1$ examples
$n_F = N$ examples
INIT
Neural Run
FSA Extr.
V
S.M
FSA Inser
time
First cycle
Second cycle
Last cycle
t_0
t_1
t_2
t_{F-1}
t_F
INIT. Initialization of RNN and learning mode
Neural run: Free or constrained neural learning stage
FSA Extr. Extraction of FSA from RNN dynamics
V. Validation of the inferred FSA
S.M. Symbolic manipulation of extracted FSA
FSA Inser. Insertion of FSA into RNN and learning mode selection

Fig. 7.6 *Time diagram of an Active Grammatical Inference process*

Most of the steps in the AGI general procedure can be carried out using techniques that have been described in previous chapters and sections. In the neural run (Step 2), the true gradient-descent algorithms explained in Sections 4.1.1 and 4.1.2 can be used for free learning, when no rule has been inserted or it is not wanted to fix the injected knowledge; the learning method given in Section 7.4 must be used for constrained learning, when the emulation of the inserted automaton is wanted to be preserved. Depending on whether a prediction or a classification task has been trained in the neural run, the corresponding extraction method described in Section 6.1 can be applied in Step 3. Likewise, depending on the trained task, prediction or classification, a stochastic DFA or a DUFA, respectively, may be inserted in Step 6 (and Step 1), using in both cases the FSM insertion procedure proposed in Section 7.4. The validation test (Step 4) is considered as a user-defined procedure, that may be adapted to different requirements, although typically, it will be related to the generalization performance of the current hypothesis on a test sample. Remember that the consistency of the hypothesis with the training examples is ensured by the UFSA extraction algorithm.

The symbolic manipulation (Step 5) is viewed also as an open process, since in this step, an operator (*teacher*) may intervene to select the next automaton to be inserted, according to some problem-dependent criteria. Nevertheless, some automatic procedures may be applied in Step 5, which are related to general heuristics. For example, the states and transitions in an extracted DUFA that have been seldom used in classifying the training examples may be removed (*statistical filtering*), where the heuristic assumption is that infrequently visited paths are associated with poorly generalized data and do not probably belong to the target automaton. Moreover, the states and transitions to be deleted may be determined automatically from the analysis of the validation test errors. In some cases, it might be decided not to keep anything of the current hypothesis and simply run a new learning cycle in which some of the strings erroneously classified in the test were incorporated to the training set.

Hence, if the learning process takes several cycles, the available examples may be supplied incrementally during some cycles until completely learned (as in Fig. 7.6), or they may also be used integrally as training set in all runs, if desired. It is important to note that the AGI general procedure includes as particular cases the connectionist RGI methods tested in Chapter 6, which consist of just a single learning cycle with no prior knowledge. Running several cycles makes sense mainly when an active teacher is available to guide the inference. However, even if this is not the case, fully automated methods can be devised that take several cycles, where the strategy for supplying the examples and the validation and symbolic manipulation steps must be completely specified in advance. An example of application of such a method to a particular RGI problem, that is illustrated in Fig. 7.7, is reported next.

The complete learning sample $S_2 = (S^+, S_2^-)$ consisted of 16 positive and 48 negative examples, that represented the types of contours shown in the top of Fig. 7.7. This complete sample was given as input both to the symbolic RPNI method (Oncina's algorithm) and to a second-order 2L-ASLRNN with one output unit (Fig. 6.6) trained to classify the examples. In addition, a DUFA was extracted from the second-order ASLRNN. The trained net, the extracted automaton, and the automaton inferred by the RPNI method, all classified correctly the sample but failed to generalize properly on a test set, yielding the error rates displayed in the figure. On the other hand, the same network was trained using all the positive examples S^+ but only a subset S_1^- of the negative sample in a first learning cycle. In this case, the extracted DUFA contained the basic structure of the target automaton, and moreover, this could be selected by automatic statistical filtering. Then, the resulting partial DUFA was inserted in a new network of the same type, and the complete sample was given in a second cycle, using the constrained learning method to train the net. Finally, the target DUFA[7] was extracted in the second cycle (see Fig. 7.7).

Obviously, the above example only serves to demonstrate that the AGI approach is feasible, and that in some cases, an incremental learning strategy may improve the performance of a connectionist RGI method. It is not claimed that the heuristic methods that follow the AGI methodology are better than other symbolic or connectionist RGI methods. AGI is merely a very flexible tool that extends the possibilities of RNNs for grammatical inference by permitting the insertion, extraction and refinement of symbolic knowledge.

[7]From this automaton, a description of a target context-sensitive language representing the positive examples can be inferred using a learning technique described in Chapter 9.

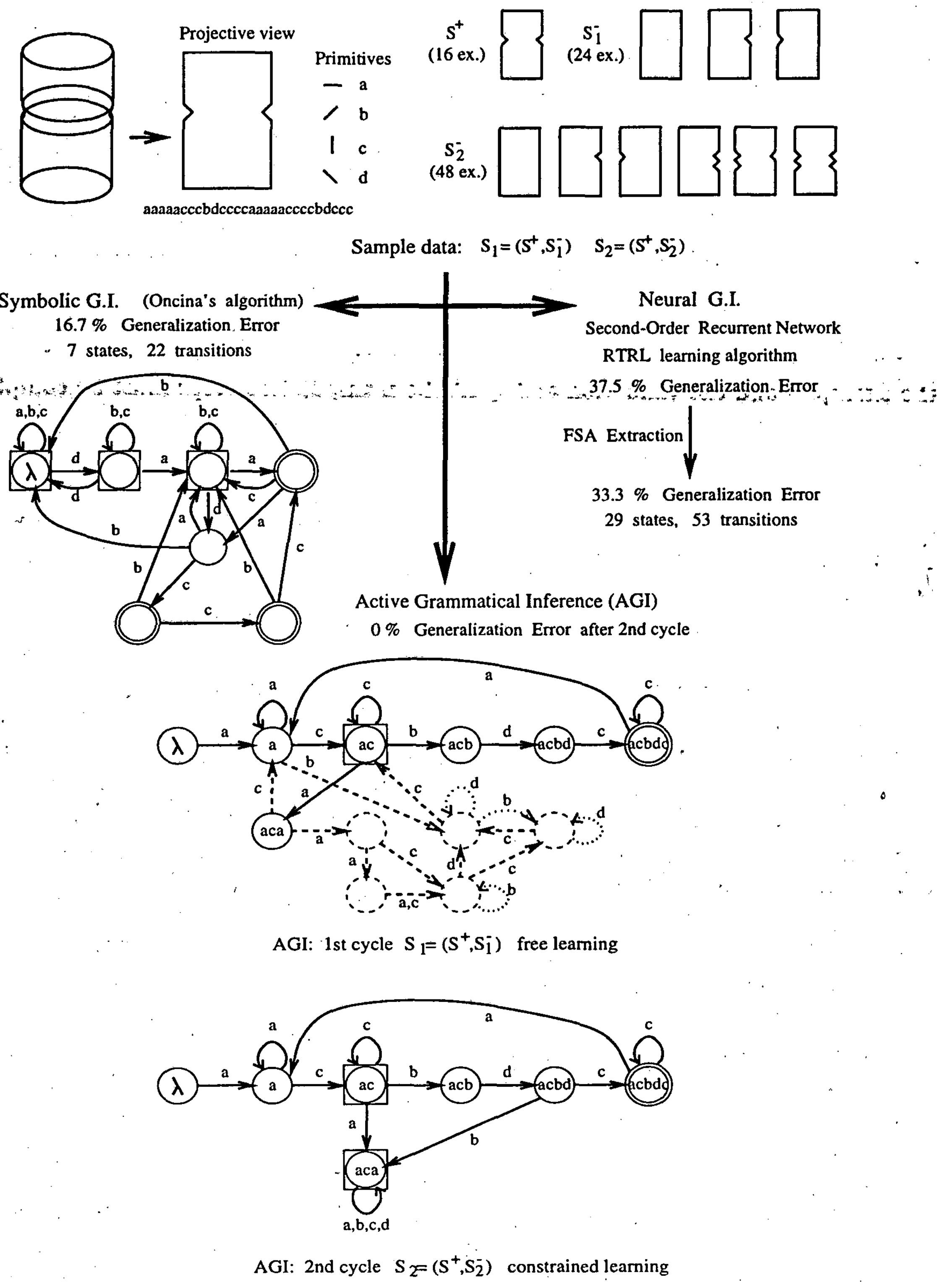
Projective view
Primitives
a
b
c
d
aaaaacccbdccccaaaaaccccbdccc
S^+ (16 ex.)
S_1^- (24 ex.)
S_2^- (48 ex.)
Sample data: $S_1 = (S^+, S_1^-)$ $S_2 = (S^+, S_2^-)$
Symbolic G.I. (Oncina's algorithm)
16.7 % Generalization Error
7 states, 22 transitions
Neural G.I.
Second-Order Recurrent Network
RTRL learning algorithm
37.5 % Generalization Error
FSA Extraction
33.3 % Generalization Error
29 states, 53 transitions
Active Grammatical Inference (AGI)
0 % Generalization Error after 2nd cycle
AGI: 1st cycle $S_1 = (S^+, S_1^-)$ free learning
AGI: 2nd cycle $S_2 = (S^+, S_2^-)$ constrained learning
λ
a
ac
acb
acbd
acbdc
aca
a,b,c,d
a,b,c
b,c
a,c

Fig. 7.7 *An example of application of Active Grammatical Inference.*