Extracting Symbolic Knowledge from Recurrent Neural Networks – A Fuzzy Logic Approach[*]

Eyal Kolman and Michael Margaliot[†]

July 11, 2007

Abstract

Considerable research has been devoted to the integration of fuzzy logic (FL) tools with classic artificial intelligence (AI) paradigms. One reason for this is that FL provides powerful mechanisms for handling and processing symbolic information stated using natural language. In this respect, fuzzy rule-based systems are white-boxes, as they process information in a form that is easy to understand, verify and, if necessary, refine.

The synergy between artificial neural networks (ANNs), which are notorious for their black-box character, and FL proved to be particularly successful. Such a synergy allows combining the powerful learning-from-examples capability of ANNs with the high-level symbolic information processing of FL systems.

In this paper, we present a new approach for extracting symbolic information from recurrent neural networks (RNNs). The approach is based on the mathematical equivalence between a specific fuzzy rule-base and functions composed of sums of sigmoids. We show that this equivalence can be used to provide a comprehensible explanation of the RNN functioning. We demonstrate the applicability of our approach by using it to extract the knowledge embedded within an RNN trained to recognize a formal language.

[*]This work was partially supported by the Adams Super Center for Brain Research.

[†]Corresponding author: Dr. Michael Margaliot, School of Electrical Engineering-Systems, Tel Aviv University, Israel 69978. Tel: +972-3-640 7768; Fax: +972-3-640 5027; Homepage: www.eng.tau.ac.il/∼michaelm Email: `michaelm@eng.tau.ac.il`

Keywords: Recurrent neural networks, formal language, regular grammar, knowledge extraction, rule extraction, rule generation, hybrid intelligent systems, neuro-fuzzy systems, knowledge-based neurocomputing, all permutations fuzzy rule-base.

1 Introduction

In 1959, Arthur Samuel [52] defined the main challenge for the emerging field of AI:

> *"How can computers be made to do what needs to be done, without being told exactly how to do it?"*

It is natural to address this question by an attempt to mimic the human reasoning process. Unlike computers, humans can *learn* what needs to be done, and how to do it. The human brain information-processing ability is thought to emerge primarily from the interactions of networks of neurons. Some of the earliest AI work aimed to imitate this structure using *connectionist models* and ANNs [38]. The development of suitable training algorithms provided ANNs with the ability to learn and generalize from examples. ANNs proved to be a very successful distributed computation paradigm, and are used in numerous real-world applications where exact algorithmic approaches are either unknown or too difficult to implement. Examples include tasks such as classification, pattern recognition, function approximation, and also the modeling and analysis of biological neural networks.

The knowledge that an ANN learns during its training process is distributed in the weights of the different neurons and it is very difficult to comprehend exactly what it is computing. In this respect, ANNs process information on a “black-box”, subsymbolic level. The problem of extracting the knowledge learned by the network, and representing it in a *comprehensible* form, has received a great deal of attention in the literature [1, 9, 57].

The knowledge extraction problem is highly relevant for both feedforward and recurrent neural networks. Recurrent architectures are more powerful and more difficult to understand due to their feedback connections [20]. RNNs are widely applied in various domains, such as financial forecasting [14, 32], control [39], speech recognition [49], visual pattern recognition [33], and more. However, their black-box character hampers their more widespread application. Knowledge extraction may help in explaining how the trained network functions, and thus increase the usefulness and acceptance of RNNs [46].

A closely related problem is *knowledge insertion*, i.e., using initial knowledge, concerning a problem domain, in order to design a suitable ANN. The knowledge can be used to determine the initial architecture or parameters [13, 59], and thus reduce training times and improve various features of the ANN (*e.g.*, generalization capability) [12, 53, 59].

Fuzzy rule-based systems process information in a very different form. The systems knowledge is represented as a set of If-Then rules stated using *natural language*. This makes it possible to comprehend, verify, and, if necessary, refine the knowledge embedded within the system [10]. Indeed, one of the major advantages of fuzzy systems lies in their ability to process perceptions, stated using natural language, rather than equations [17, 60, 63, 64, 65]. ¿From an AI perspective, many of the most successful applications of FL are *fuzzy expert systems*. These combine the classic expert systems of AI with FL tools, that provide efficient mechanisms for addressing the fuzziness, vagueness, and imprecision of knowledge stated using natural language [43].

A natural step is then to combine the learning capability of ANNs with the comprehensibility of fuzzy rule bases (FRBs). Two famous examples of such a synergy are: (1) the *adaptive network-based fuzzy inference system* (ANFIS) [22], which is a feedforward network representation of the fuzzy reasoning process; and (2) the *fuzzy-MLP* [42, 48], which is a feedforward network with fuzzified inputs. Numerous neuro-fuzzy systems are reviewed in [41].

In a recent paper [29], we showed that the input-output mapping of a specific FRB, referred to as the *all-permutations fuzzy rule-base* (APFRB), is a linear sum of sigmoid functions. Conversely, every such sum can be represented as the result of inferring a suitable APFRB. This mathematical equivalence provides a synergy between: (1) ANNs with sigmoid activation functions; and (2) symbolic FRBs. This approach was used to extract and insert symbolic information into *feedforward* ANNs [29, 30].

In this paper, we use the APFRB to develop a new approach for knowledge-based computing in RNNs. We focus on extracting symbolic knowledge, stated as a suitable FRB, from trained RNNs, leaving the issue of knowledge insertion to a companion paper [28]. We demonstrate the usefulness of our approach by applying it to provide a comprehensible description of the functioning of an RNN trained to recognize a formal language.

The rest of this paper is organized as follows. The next section briefly reviews existing approaches for extracting knowledge from trained RNNs. Section 3 recalls the definition of the APFRB and how it can be used for knowledge-based neurocomputing in *feedforward* ANNs. Section 4 presents our new approach for knowledge-based computing in RNNs using the APFRB. To demonstrate the new approach, section 5 considers an RNN trained to solve a classical language recognition problem. Symbolic rules that describe the network functioning are extracted in section 6. The final section concludes.

2 Knowledge Extraction from RNNs

The common technique for rule extraction from RNNs is based on transforming the RNN into an equivalent deterministic finite-state automaton (DFA). This is carried out using four steps: (1) quantization of the continuous state space of the RNN, resulting in a discrete set of states; (2) state and output generation by feeding the RNN with input patterns; (3) construction of the corresponding DFA, based on the observed transitions; and (4) minimization of the DFA [20].

Some variants include: quantization using equipartitioning of the state space [15, 45] and using vector quantization [11, 16, 66]; generating the state and output of the DFA by sampling the state space [34, 61]; extracting stochastic state machines [55, 56]; extracting fuzzy state machines [5], and more [20].

This methods for knowledge extraction suffers from several drawbacks. First, RNNs are continuous-valued and it is not at all clear whether they can be suitably modeled using discrete-valued mechanisms such as DFAs [26, 27]. Second, the resulting DFA crucially depends on the quantization level. Coarse quantization may cause large inconsistencies between the RNN and the extracted DFA, while fine quantization may result in an overly large and complicated DFA. Finally, the comprehensibility of the extracted DFA is questionable. This is particularly true for DFAs with many states, as the meaning of every state/state-transition is not necessarily clear. These disadvantages encourage the development of alternative techniques for knowledge extraction from RNNs [50, 51].

3 The All Permutations Fuzzy Rule-Base

The APFRB is a special Mamdani-type FRB [29]. Inferring the APFRB, using standard tools from FL theory, yields an input-output relationship that is mathematically equivalent to that of an ANN. More precisely, there exists an invertible mapping T such that

$$T(\text{ANN}) = \text{APFRB} \;\text{ and }\; T^{-1}(\text{APFRB}) = \text{ANN}. \tag{1}$$

The application of (1) for inserting and extracting symbolic knowledge from *feedforward* ANNs was demonstrated [29, 30] (for other approaches relating feedforward ANNs and FRBs, see [3, 6, 21, 35]).

To motivate the definition of the APFRB, we consider a simple example adapted from [37] (see also [36]).

Example 1 Consider the following two-rule FRB:

R_1: If x *equals* k, Then $f = a_0 + a_1$,

R_2: If x *equals* $-k$, Then $f = a_0 - a_1$,

where $x \in \mathbb{R}$ is the input, $f \in \mathbb{R}$ is the output, $a_0, a_1, k \in \mathbb{R}$, with $k > 0$. Suppose that the linguistic terms *equals* k and *equals* $-k$ are defined using the Gaussian membership functions:

$$\mu_{=k}(y) = \exp\left(-\frac{(y-k)^2}{2k}\right), \quad \text{and} \quad \mu_{=-k}(y) = \exp\left(-\frac{(y+k)^2}{2k}\right).$$

Applying the singleton fuzzifier and the center of gravity (COG) defuzzi-

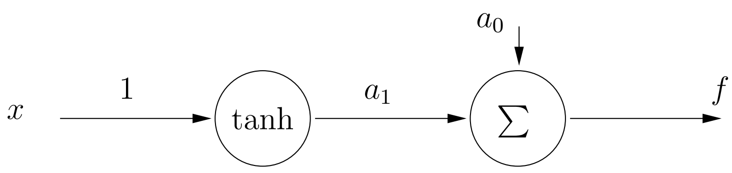

Figure 1: Graphical representation of the FRB input-output mapping.

fier [54] to this rule-base yields

$$\begin{aligned}
f(x) &= \frac{(a_0 + a_1)\mu_{=k}(x) + (a_0 - a_1)\mu_{=-k}(x)}{\mu_{=k}(x) + \mu_{=-k}(x)} \\
&= a_0 + \frac{a_1 \exp(-(x-k)^2/(2k)) - a_1 \exp(-(x+k)^2/(2k))}{\exp(-(x-k)^2/(2k)) + \exp(-(x+k)^2/(2k))} \\
&= a_0 + a_1 \tanh(x).
\end{aligned}$$

Thus, this FRB defines a mapping $x \to f(x)$ that is mathematically equivalent to that of a feedforward ANN with one hidden neuron whose activation function is $\tanh(\cdot)$ (see Fig. 1). Conversely, the ANN depicted in Fig. 1 is mathematically equivalent to the aforementioned FRB. $\square$

This example motivates the search for an FRB whose input-output mapping is a linear combination of sigmoid functions, as this is the mapping of an ANN with a single hidden layer.

A fuzzy rule-base with inputs $x_1, \dots, x_m$ and output $f \in \mathbb{R}$ is called an APFRB if the following conditions hold.[1] First, every input variable x_i is characterized by two linguistic terms: $term^i_-$ and $term^i_+$. These terms are

[1]For the sake of simplicity, we consider the case of a one-dimensional output; the generalization to the case $\mathbf{f} \in \mathbb{R}^n$ is straightforward [30].

modeled using two membership functions $\mu_-^i(\cdot)$ and $\mu_+^i(\cdot)$ that satisfy the following constraint: there exists a $v_i \in \mathbb{R}$ such that

$$\frac{\mu_+^i(y) - \mu_-^i(y)}{\mu_+^i(y) + \mu_-^i(y)} = \tanh(y - v_i), \qquad \forall\ y \in \mathbb{R}. \tag{2}$$

Second, the rule-base contains 2^m Mamdani-type rules spanning, in their If-part, all the possible linguistic assignments for the m input variables.

Third, there exist constants a_i, $i = 0, 1, \dots, m$, such that the Then-part of each rule is a combination of these constants. Specifically, the rules are:

R_1 : If (x_1 *is term*$_-^1$) and (x_2 *is term*$_-^2$) and ...and (x_m *is term*$_-^m$)

Then $f = a_0 - a_1 - a_2 - \cdots - a_m$.

R_2 : If (x_1 *is term*$_-^1$) and (x_2 *is term*$_-^2$) and ...and (x_m *is term*$_+^m$)

Then $f = a_0 - a_1 - a_2 - \cdots + a_m$.

$\vdots$

R_{2^m} : If (x_1 *is term*$_+^1$) and (x_2 *is term*$_+^2$) and ...and (x_m *is term*$_+^m$)

Then $f = a_0 + a_1 + a_2 + \cdots + a_m$.

Note that the signs in the Then-part are determined in the following manner: if the term characterizing x_i in the If-part is *term*$_+^i$, then in the Then-part, a_i is preceded by a plus sign; otherwise, a_i is preceded by a minus sign.

It is important to note that the constraint (2) is satisfied by several commonly used fuzzy membership functions. For example, the pair of Gaussian membership functions

$$\mu_{=k_1}(y;k_2) = \exp\left(-\frac{(y-k_1)^2}{k_1-k_2}\right), \quad \text{and} \quad \mu_{=k_2}(y;k_1) = \exp\left(-\frac{(y-k_2)^2}{k_1-k_2}\right) \tag{3}$$

with $k_1 > k_2$, satisfy (2) with $v = (k_1 + k_2)/2$. The logistic functions

$$\mu_{>k}(y) = \frac{1}{1+\exp\left(-2(y-k)\right)}, \quad \text{and} \quad \mu_{<k}(y) = \frac{1}{1+\exp\left(2(y-k)\right)}, \tag{4}$$

satisfy (2) with $v = k$. Thus, the pair of linguistic terms $\{$*equals* k_1, *equals* $k_2\}$ (modeled using Gaussians), and the pair $\{$*larger than* k, *smaller than* $k\}$ (modeled using logistic functions) can be used in an APFRB.

Summarizing, the APFRB is a standard Mamadani-type FRB satisfying four constraints: each input variable is characterized by two verbal terms; the terms are modeled using membership functions that satisfy (2); the rule-base contains exactly 2^m rules; and the values in the Then-part of the rules are not independent, but rather they are the sum of $m+1$ constants with alternating signs.

Example 2 For $m = 2$, $term^1_- =$ *smaller than* 5, $term^1_+ =$ *larger than* 5, $term^2_- =$ *equals* 1, $term^2_+ =$ *equals* 7, $a_0 = 1$, $a_1 = 1/3$, and $a_2 = 2/5$, the definition above yields the following APFRB:

R_1: If x_1 is *smaller than* 5 and x_2 *equals* 1, Then $f = 4/15$,

R_2: If x_1 is *larger than* 5 and x_2 *equals* 1, Then $f = 14/15$,

R_3: If x_1 is *smaller than* 5 and x_2 *equals* 7, Then $f = 16/15$,

R_4: If x_1 is *larger than* 5 and x_2 *equals* 7, Then $f = 26/15$.

Modeling the fuzzy terms *larger than 5* and *smaller than 5* using the membership functions (4), the terms *equals 7* and *equals 1* using (3), and inferring this APFRB yields

$$f(x_1, x_2) = 1 + \tanh(x_1 - 5)/3 + 2\tanh(x_2 - 4)/5.$$

Just as in Example 1, it is clear that the mapping $(x_1, x_2) \to f(x_1, x_2)$ can be realized using a suitable feedforward ANN. $\square$

3.1 ANNs and the APFRB

The next results shows that the APFRB is an ANN in disguise. The proof is similar to the proof of Theorem 1 in [29] and is, therefore, omitted.

Theorem 1 [29] *Applying the product-inference rule, singleton fuzzifier, and the COG defuzzifier to an APFRB yields*

$$f(x_1, \dots, x_m) = \sum_{i=1}^{m} a_i g(x_i - v_i), \tag{5}$$

where

$$g(z) := \frac{a_0}{\sum_{i=1}^{m} a_i} + \tanh(z). \tag{6}$$

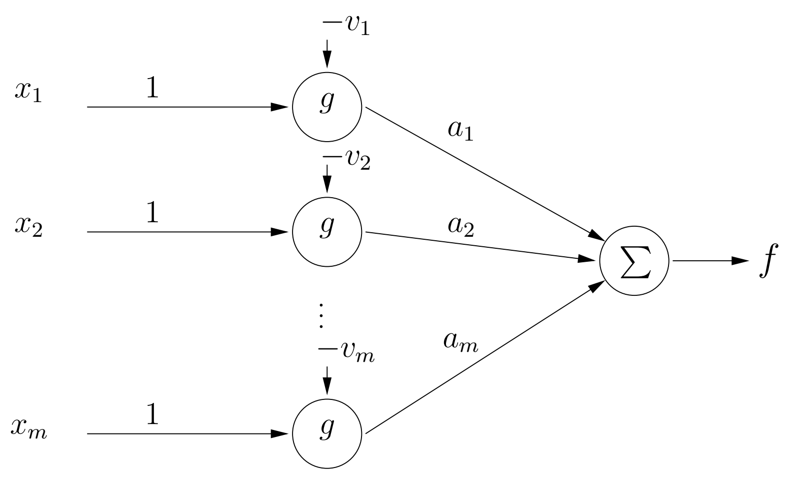

Figure 2: Graphical representation of the APFRB input-output mapping.

In other words, the output f of the APFRB can be obtained by first feeding the biased inputs $x_i - v_i$ to a hidden layer computing the activation function $g(\cdot)$, and then computing a weighted sum of the hidden layer outputs (see Fig. 2).

Note that for $a_0 = 0$, (6) yields the activation function $g(z) = \tanh(z)$, whereas setting $a_0 = \sum_{i=1}^{m} a_i$, yields $g(z) = 1 + \tanh(z) = 2/(1 + \exp(-2z))$. These are the standard activation functions in numerous ANNs, so in these cases (5) is just the mapping realized by an ANN with a single hidden layer of neurons.

Conversely, consider the single hidden-layer feedforward ANN depicted in Fig 3. The ANN output is:

$$f = \beta + \sum_{j=1}^{m} c_j h(y_j + b_j), \tag{7}$$

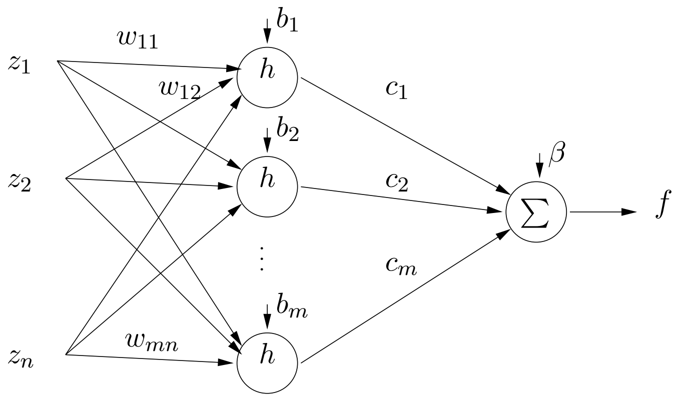

Figure 3: A feedforward ANN with a single hidden-layer.

where $y_j := \sum_{i=1}^{n} w_{ji} z_i$, $j = 1, \ldots, m$, and $h(\cdot)$ is the activation function. Standard choices include the hyperbolic tangent function $h(z) = \tanh(z)$ and the logistic function $h(z) = \sigma(z) := 1/(1 + \exp(-z))$. Comparing (7) with (5) yields the following result.

Corollary 1 *Consider the ANN depicted in Fig. 3. If the activation function in the ANN is $h(z) = \tanh(z)$, then* (7) *is the output of an APFRB with inputs $x_i = y_i$ and parameters*

$$a_0 = \beta,\ a_i = c_i, \quad \textit{and } v_i = -b_i, \quad i = 1, \ldots, m.$$

If $h(z) = \sigma(z)$, then (7) *is the output of an APFRB with inputs $x_i = y_i/2$ and parameters*

$$a_0 = \beta + \frac{1}{2} \sum_{i=1}^{m} c_i,\ a_i = c_i/2, \quad \textit{and } v_i = -b_i/2, \quad i = 1, \ldots, m.$$

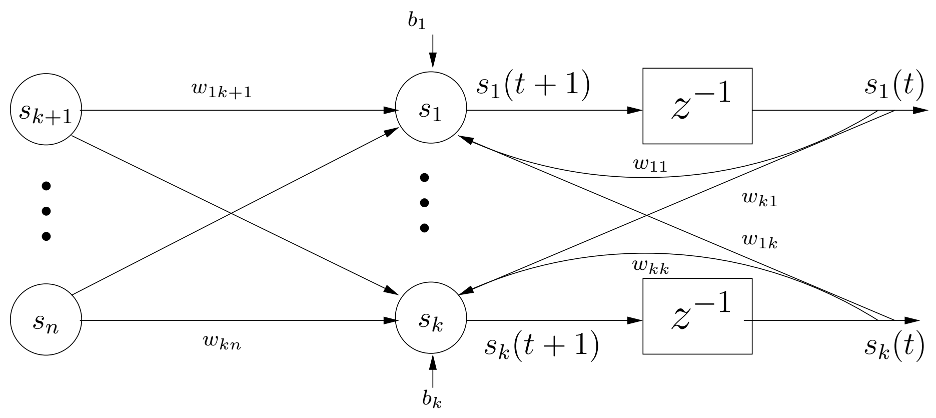

Figure 4: A first-order recurrent neural network.

Theorem 1 and Corollary 1 provide the explicit transformation T described in (1). In previous papers [29, 30], we demonstrated the implementation of (1) for extracting and inserting knowledge into *feedforward* neural networks. Here, we use (1) to develop a new approach to knowledge extraction into RNNs.

4 RNNs and the APFRB

Consider a first-order RNN with hidden neurons $s_1, \ldots, s_k$, input neurons $s_{k+1}, \ldots, s_n$, and weights w_{ij} (see Fig. 4). Denoting, for convenience, $s_0(t) \equiv 1$, and $w_{i0} = b_i$ yields

$$s_i(t+1) = h\left(\sum_{j=0}^{n} w_{ij} s_j(t)\right), \quad i = 1, \ldots, k, \tag{8}$$

where h is the activation function. Denoting $y_i(t) := \sum_{j=1}^{n} w_{ij}s_j(t)$ yields

$$s_i(t+1) = h\left(y_i(t) + w_{i0}\right), \quad i = 1, \ldots, k. \tag{9}$$

The next result follows immediately from Theorem 1.

Corollary 2 *If the activation function in the RNN* (9) *is* $h(z) = \tanh(z)$*, then* $s_i(t+1)$ *is the output of a two-rule APFRB with input* $x = y_i(t)$ *and parameters* $a_0 = 0,\ a_1 = 1,\ \text{ and } v_1 = -w_{i0}$.
If $h(z) = \sigma(z)$*, then* $s_i(t+1)$ *is the output of a two-rule APFRB with input* $x = y_i(t)/2$ *and parameters* $a_0 = 1/2,\ a_1 = 1/2,\ \text{ and } v_1 = -w_{i0}/2$.

Example 3 Using Corollary 2 to extract an APFRB from (9), with $h(z) = \sigma(z)$, yields

R_1: If $y_i(t)/2$ is *larger than* $-w_{i0}/2$, Then $s_i(t+1) = 1$,

R_2: If $y_i(t)/2$ is *smaller than* $-w_{i0}/2$, Then $s_i(t+1) = 0$,

where the fuzzy terms {*larger than*, *smaller than*} are modeled using the membership functions (4). □

A more descriptive set of rules can be extracted by restating (8) as

$$\begin{aligned} h^{-1}\left(s_i(t+1)\right) &= \sum_{j=0}^{n} w_{ij}s_j(t) \\ &= \sum_{j=0}^{n} w_{ij} h\left(\sum_{p=0}^{n} w_{jp}s_p(t-1)\right), \quad i = 1, \ldots, k, \end{aligned}$$

where $h^{-1}(\cdot)$ is the inverse function of $h(\cdot)$. Denoting

$$\tilde{y}_i(t) = \begin{cases} \sum\limits_{p=1}^{n} w_{ip} s_p(t-1), & i = 1, \dots, k, \\ h^{-1}\left(s_i(t)\right), & \text{otherwise}, \end{cases}$$

and recalling that $w_{j0} = 0$ for $j = 0, k+1, k+2, \dots, n$, yields

$$h^{-1}(s_i(t+1)) = \sum_{j=0}^{n} w_{ij} h(\tilde{y}_j(t) + w_{j0}), \quad i = 1, \dots, k. \tag{10}$$

Note that if h is a sigmoid function, then (10) implies that $h^{-1}(s_i(t+1))$ is a linear sum of sigmoid functions, and Theorem 1 yields the following result.

Corollary 3 *If the activation function in the RNN* (8) *is* $h(z) = \tanh(z)$, *then* $\tanh^{-1}(s_i(t+1))$ *is the output of an APFRB with inputs* $x_1 = \tilde{y}_1(t), \dots, x_n = \tilde{y}_n(t)$ *and parameters* $a_0 = w_{i0}$, $a_j = w_{ij}$, and $v_j = -w_{j0}$, $\quad j = 1, \dots, n$.
If $h(z) = \sigma(z)$, *then* $\sigma^{-1}(s_i(t+1))$ *is the output of an APFRB with inputs* $x_1 = \tilde{y}_1(t)/2, \dots, x_n = \tilde{y}_n(t)/2$ *and parameters* $a_0 = w_{i0} + \frac{1}{2}\sum_{j=1}^{n} w_{ij}$, $a_j = w_{ij}/2$, and $v_j = -w_{j0}/2$, $\quad j = 1, \dots, n$.

4.1 Knowledge Extraction

Corollaries 2 and 3 provide a symbolic representation of the RNN functioning in terms of a set of fuzzy rules. This mathematical equivalence between APFRBs and RNNs has several advantages: (1) it enables bidirectional flow of information between the RNN and the corresponding APFRB; (2) it enables the application of tools from the theory of RNNs to APFRBs and vice versa; (3) it is applicable to any standard RNN regardless of its specific architecture or parameters; (4) the rule extraction process is straightforward, as the APFRB structure and parameters are calculated directly from the RNN architecture and weights.

If the resulting APFRB contains a large set of rules then several simplifications can be carried out in order to increase its comprehensibility. For every simplification step, it is possible to bound the error it incurs (i.e., the difference between the mappings corresponding to the original and simplified FRBs).

We now list four possible simplifications.

(1) Before converting the function $\sum\limits_i c_i \sigma(x_i)$ into an APFRB, simplify it by setting coefficients c_j, with $|c_j|$ relatively small, to zero. The resulting error is smaller than $|c_j|$ [29].

(2) Change the inferencing mechanism from center of gravity (COG) to mean of maxima (MOM) [23, 54]. Specifically, let $d_i(\mathbf{x})$ denote the degree of firing (DOF) of the i^{th} rule for some input $\mathbf{x} \in D$, where D is the input space. Let $p(\mathbf{x}) := \arg\max_i\{d_i(\mathbf{x})\}$, that is, the index of the rule with the largest DOF. Then MOM inferencing yields $f(\mathbf{x}) = f_p$, where f_p is the value in the Then-part of rule p. Denoting $f_{COG}(\mathbf{x})$ as the APFRB output using the COG inferencing, the resulting error is bounded by

$$\max_{\mathbf{x}\in D} |f_{COG}(\mathbf{x}) - f_p(\mathbf{x})|. \tag{11}$$

Due to the mutually exclusive nature of the rules in the APFRB, (11) is usually small.

(3) Restate the rules in terms of *crisp* functions of the inputs. Note that the definition of the MOM defuzzifier implies that as long as $p(\mathbf{x})$ and $f_p(\mathbf{x})$ remain unchanged for all $\mathbf{x} \in D$, the rules can be modified, with no approximation error.

(4) Group together rules with similar outputs, and define one cluster of outputs as the default cluster [31]. Let $\hat{f}_i(\mathbf{x})$ denote the modified output of the i^{th} rule. Then, the added error is bounded by

$$\max_{\mathbf{x}\in D} |\hat{f}_p(\mathbf{x}) - f_p(\mathbf{x})|.$$

In the following sections, we demonstrate how the RNN–APFRB equivalence can be used for knowledge extraction using an RNN that is trained to solve a language recognition problem.

5 Language Recognition using RNNs

For the sake of completeness, we briefly review some ideas from the field of of formal languages.

5.1 Formal Languages

Let Σ denote a set of symbols (e.g., $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b, c, \ldots, z\}$). A *string* is a finite-length sequence of symbols from Σ. Let * denote the Kleene closure operator [24], so Σ^* is the (infinite) set of all the strings constructed over Σ. A set of strings $L \subseteq \Sigma^*$ is called a *formal language* [18].

Definition 1 *A* formal grammar *is a quadruple $G = \langle S, N, T, P \rangle$, where S is the start symbol, N and T are non-terminal and terminal symbols, respectively, and P is a set of productions of the form $u \rightarrow v$, where $u, v \in (N \cup T)^*$, and u contains at least one non-terminal symbol.*

Note that repeatedly applying the production rules generates a specific set of strings, that is, a language. This language is denoted $L(G)$.

Chomsky and Schützenberger [7, 8] divided formal languages into four classes: *recursive*, *context-sensitive*, *context-free* and *regular*. Each class is strictly contained in its predecessor (e.g., the set of regular languages is strictly contained in the set of context-free languages). The classes are defined by the type of production rules allowed in the corresponding grammars. Regular languages, generated by regular grammars, represent the smallest class in the hierarchy of formal languages.

Definition 2 [47] *A regular grammar G is a formal grammar with productions of the form $A \rightarrow a$ or $A \rightarrow aB$, where $A, B \in N$ and $a \in T$.*

Example 4 Consider the regular grammar defined by:

$$\begin{aligned}&S \text{ is the start symbol}, N = \{S, A, B\}, T = \{0, 1\}, \\ &P = \{S \to 1S, S \to 0A, A \to 1S, A \to 0B, B \to 1S, S \to \varepsilon, A \to \varepsilon, B \to \varepsilon\},\end{aligned}$$

where ϵ denotes the empty string. This grammar, known as *Tomita's* 4^{th} *grammar* [58], generates a regular language, denoted L_4, which is the set of all binary strings that do *not* include '000' as a substring. □

It is natural to associate a regular grammar with a deterministic finite-state automaton (DFA).

Definition 3 *A DFA is a 5-tuple $M = \langle \Sigma, Q, R, F, \delta \rangle$, where Σ is the alphabet of the language L, $Q = \{s_1, \ldots, s_M\}$ is a set of states, $R \in Q$ is the start state, $F \subseteq Q$ is a set of accepting states, and $\delta : Q \times \Sigma \to Q$ defines the state transitions.*

We say that a string x is *accepted* by a DFA M iff $s(x)$, the state that is reached after x has been read by M, is an accepting state [45]. For each regular language $L(G)$ there is an associated DFA M, such that a string x is accepted by M iff $x \in L(G)$.

5.2 Formal Languages and RNNs

RNNs are often used for recognizing formal languages. The training is performed using a set containing pairs of the form (string, label), where label

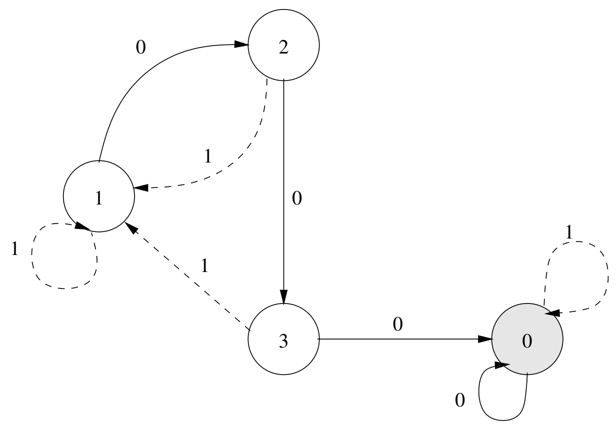

Figure 5: A DFA that recognizes L_4. State 1 is the starting state, states 1-3 are accepting states, and state 0 is a rejecting state. Arrows indicate state transitions.

indicates whether the string belongs to the language or not. Demonstration of knowledge extraction techniques is commonly done by using RNNs that have learned to recognize formal languages [2, 15, 40].

Usually, the knowledge embedded in the RNN is extracted in the form of a DFA, that represents the underlying regular grammar. For example, Giles *et al.* [15, 44] trained RNNs with 3 to 5 hidden neurons to recognize L_4. Fig. 5 depicts a four-state DFA extracted from one of those networks [44]. It is easy to see that this DFA indeed correctly recognizes L_4.

To demonstrate the efficiency of our approach, we trained an RNN to recognize L_4, and then extracted the knowledge embedded within the RNN as an APFRB.

5.3 The trained RNN

We used an RNN with three hidden neurons, one input neuron and one bias neuron. The network is thus described by

$$s_i(t) = \sigma(\sum_{j=0}^{4} w_{ij} s_j(t-1)), \qquad i = 1, 2, 3, \tag{12}$$

and the input neuron

$$s_4(t) = \begin{cases} 1, & \text{if } I(t) = 0, \\ -1, & \text{if } I(t) = 1, \end{cases} \tag{13}$$

where $I(t)$ is the value of the input at time t.

This RNN was trained using *real time recurrent learning* (RTRL) [62], with: learning rate $\eta = 0.1$, momentum $\gamma = 0.05$, and an added regularization term $10^{-3} \sum_{i,j} w_{ij}^2$ [19].

We generated 2000 RNNs, with different initial weights. The initial weights and biases were drawn from a uniform probability distribution over [-2,2] and [-1,1], respectively. Each RNN was trained for 700 epochs, where each epoch consisted of presenting the full training set to the RNN, and changing the weights after each presented string.

Our main focus is on extracting the knowledge from the trained network, and not on finding the optimal RNN. Therefore, we used the simple holdout approach rather than more sophisticated and time consuming approaches, such as bootstrap or cross-validation [4, 25]. The data set contained all the 8184 binary strings with length $3 \leq l \leq 12$ (of which 54% do not belong to the language L, i.e., have a '000' substring), and was split into training and test sets. The training set included all the 248 binary strings with length $3 \leq l \leq 7$ (of which 32% do not belong to the language L), and the test set included the 7936 binary strings with length $8 \leq l \leq 12$ (of which 55% do not belong to L).[2]

After training, the RNN with the best performance had the parameters:[3]

$$W = \begin{bmatrix} -7.6 & 15.2 & 8.4 & 0.2 & 3 \\ -4.7 & 0.2 & -0.2 & 4.5 & 1.5 \\ -0.2 & 0 & 0 & 0 & 4 \end{bmatrix}. \tag{14}$$

This RNN correctly classifies all the strings in the data set. Furthermore, the following result establishes the correctness of the RNN.

Proposition 1 *The RNN defined by* (12),(13), *and* (14) *correctly classifies any given binary string according to* L_4.

Proof. See the Appendix.

Although this RNN functions quite well, it is very difficult to understand exactly what it is doing. To gain more understanding, we transform this

[2]Since the RTRL algorithm performance deteriorates for longer strings, it is common to use shorter strings for training and then test the generalization capability of the trained network on longer strings.

[3]The numerical values were rounded to one decimal digit, without affecting the classification accuracy.

RNN into an equivalent APFRB.

6 Knowledge Extraction Using the APFRB

6.1 Extracting rules for $s_3(t)$

The dynamic equation for s_3 is

$$s_3(t) = \sigma\left(4s_4(t-1) - 0.2\right). \tag{15}$$

By Corollary 2, this is equivalent to the APFRB

R_1: If $2s_4(t-1)$ is *larger than* 0.1, Then $s_3(t) = 1$,

R_2: If $2s_4(t-1)$ is *smaller than* 0.1, Then $s_3(t) = 0$,

where the fuzzy terms *larger than* and *smaller than* are modeled (from here on) using the logistic functions (4).

Recall that $s_4(t-1) \in \{-1, 1\}$. Calculating the DOFs for these values shows that when $s_4(t-1) = -1$: $d_1 = 0.01$, $d_2 = 0.99$, and the output is $s_3(t) = (d_1 \cdot 1 + d_2 \cdot 0)/(d_1 + d_2) = 0.01$. For $s_4(t-1) = 1$: $d_1 = 0.98$, $d_2 = 0.02$ and the output is $s_3(t) = 0.98$. Thus, the error incurred by computing $s_3(t)$ using the MOM defuzzifier instead of the COG defuzzifier is bounded by 0.02, and we replace the COG defuzzifier with the MOM one. Using the MOM defuzzifier yields the *crisp* rules

R_1: If $s_4(t-1) = 1$, Then $s_3(t) = 1$,

R_2: If $s_4(t-1) = -1$, Then $s_3(t) = 0$.

Restating the rules in terms of the input yields

R_1: If $I(t-1) = 0$, Then $s_3(t) = 1$,

R_2: If $I(t-1) = 1$, Then $s_3(t) = 0$.

Finally, treating the case $s_3(t) = 0$ as the default class yields

$$\text{If } I(t-1) = 0, \text{ Then } s_3(t) = 1; \text{ Else, } s_3(t) = 0.$$

This rule provides a clear interpretation of the functioning of this neuron, namely, $s_3(t)$ is ON (i.e., equals 1) iff the last input bit was '0'.

6.2 Extracting rules for $s_2(t)$

The dynamic equation for $s_2(t)$ is

$$s_2(t) = \sigma\left(0.2s_1(t-1) - 0.2s_2(t-1) + 4.5s_3(t-1) + 1.5s_4(t-1) - 4.7\right). \tag{16}$$

Ignoring the relatively small weights yields

$$s_2(t) = \sigma\left(4.5s_3(t-1) + 1.5s_4(t-1) - 4.7\right) + e_1\left(s_1(t-1), s_2(t-1)\right), \tag{17}$$

with $e_1\left(s_1(t-1), s_2(t-1)\right) \leq 0.04$. This approximation has no effect on the network classification results on the test set. That is, the simplified RNN correctly classifies all the examples in the test set. Substituting (15) in (17) yields

$$\sigma^{-1}(s_2(t)) = 4.5\sigma(4s_4(t-2) - 0.2) + 1.5s_4(t-1) - 4.7. \tag{18}$$

The right hand side of this equation is not a sum of sigmoids. Since $s_4(t) \in \{-1, 1\}$, we can use the linear approximation

$$s_4(t-1) = 2\sigma(5s_4(t-1)) - 1 + e_2, \tag{19}$$

with $|e_2| \leq 0.013$. Substituting this in (18) and ignoring e_2 yields

$$\sigma^{-1}(s_2(t)) = 4.5\sigma(4s_4(t-2) - 0.2) + 3\sigma(5s_4(t-1)) - 6.2.$$

By Corollary 3, this is equivalent to the APFRB

R_1: If $2s_4(t-2)$ is *larger than* 0.1 and $2.5s_4(t-1)$ is *larger than* 0,
Then $\sigma^{-1}(s_2(t)) = 1.3$,

R_2: If $2s_4(t-2)$ is *larger than* 0.1 and $2.5s_4(t-1)$ is *smaller than* 0,
Then $\sigma^{-1}(s_2(t)) = -1.7$,

R_3: If $2s_4(t-2)$ is *smaller than* 0.1 and $2.5s_4(t-1)$ is *larger than* 0,
Then $\sigma^{-1}(s_2(t)) = -3.2$,

R_4: If $2s_4(t-2)$ is *smaller than* 0.1 and $2.5s_4(t-1)$ is *smaller than* 0,
Then $\sigma^{-1}(s_2(t)) = -6.2$.

Calculating for the four possible values of $(s_4(t-1), s_4(t-2))$ yields

$$\left|\sigma^{-1}\left(s_2\left(s_4(t-1), s_4(t-2)\right)\right) - f_p\left(s_4(t-1), s_4(t-2)\right)\right| \leq 0.12,$$

so we switch to the MOM defuzzifier, and we can restate the rules as crisp rules

R_1: If $I(t-2) = 0$ and $I(t-1) = 0$, Then $s_2(t) = \sigma(1.3) = 0.8$,

R_2: If $I(t-2) = 0$ and $I(t-1) = 1$, Then $s_2(t) = \sigma(-1.7) = 0.15$,

R_3: If $I(t-2) = 1$ and $I(t-1) = 0$, Then $s_2(t) = \sigma(-3.2) = 0$,

R_4: If $I(t-2) = 1$ and $I(t-1) = 1$, Then $s_2(t) = \sigma(-6.2) = 0$,

Defining the output $s_2(t) = 0$ as the default value, and approximating the second rule output by $s_2(t) = 0$, yields

If $I(t-2) = 0$ and $I(t-1) = 0$, Then $s_2(t) = 0.8$; Else, $s_2(t) = 0$.

The total error due to simplification of the rules is bounded by 0.17. It is now easy to understand the functioning of this neuron, namely, $s_2(t)$ is ON iff the last two input bits were '00'.

6.3 Extracting rules for $s_1(t)$

The dynamic equation for s_1 is

$$s_1(t) = \sigma\left(15.2 s_1(t-1) + 8.4 s_2(t-1) + 0.2 s_3(t-1) + 3 s_4(t-1) - 7.6\right). \tag{20}$$

There is one term with a relatively small weight: $0.2 s_3(t-1)$. Using (15) yields

$$s_3(t-1) = \sigma\left(4 s_4(t-2) - 0.2\right) = \begin{cases} 0.98, & \text{if } s_4(t-2) = 1, \\ 0.01, & \text{if } s_4(t-2) = -1. \end{cases}$$

Assuming a symmetrical distribution $P\left(s_4(t-2) = 1\right) = P\left(s_4(t-2) = -1\right) = 1/2$ yields $E\{0.2 s_3(t-1)\} = 0.1$, so we simplify (20) to

$$s_1(t) = \sigma\left(15.2 s_1(t-1) + 8.4 s_2(t-1) + 3 s_4(t-1) - 7.5\right), \tag{21}$$

The above approximation has no effect on the network classification results on the test set. Combining (21) with (15),(17), and (19) yields

$$\begin{aligned} \sigma^{-1}\left(s_1(t)\right) &= 15.2 s_1(t-1) + 8.4 s_2(t-1) + 3 s_4(t-1) - 7.5 \\ &= 15.2 s_1(t-1) + 8.4\sigma\left(4.5\sigma\left(4 s_4(t-3) - 0.2\right) + 1.5 s_4(t-2) - 4.7\right) \\ &\quad + 6\sigma\left(5 s_4(t-1)\right) - 10.5. \end{aligned}$$

Since

$$\sigma\left(4s_4(t-3)-0.2\right)=0.48s_4(t-3)+0.5+e_3\left(s_4(t-3)\right),$$

with $|e_3\left(s_4(t-3)\right)|<0.005$,

$$\begin{aligned}\sigma^{-1}\left(s_1(t)\right) &= 8.4\sigma\left(2.16s_4(t-3)+1.5s_4(t-2)-2.45\right)+6\sigma\left(5s_4(t-1)\right)\\ &\quad +15.2s_1(t-1)-10.5.\end{aligned} \tag{22}$$

Applying Corollary 3 to (22) yields the APFRB

R_1: If $1.08s_4(t-3)+0.75s_4(t-2)$ is *lt* 1.23 and $2.5s_4(t-1)$ is *lt* 0,
Then $\sigma^{-1}(s_1(t))=3.9+15.2s_1(t-1)$,

R_2: If $1.08s_4(t-3)+0.75s_4(t-2)$ is *lt* 1.23 and $2.5s_4(t-1)$ is *st* 0,
Then $\sigma^{-1}(s_1(t))=-2.1+15.2s_1(t-1)$,

R_3: If $1.08s_4(t-3)+0.75s_4(t-2)$ is *st* 1.23 and $2.5s_4(t-1)$ is *lt* 0,
Then $\sigma^{-1}(s_1(t))=-4.5+15.2s_1(t-1)$,

R_4: If $1.08s_4(t-3)+0.75s_4(t-2)$ is *st* 1.23 and $2.5s_4(t-1)$ is *st* 0,
Then $\sigma^{-1}(s_1(t))=-10.5+15.2s_1(t-1)$,

where *lt* stands for *larger than* and *st* stands for *smaller than*, modeled as in (4).

Calculating the error induced by using the MOM defuzzifier shows that it is actually rather large, so the MOM inferencing approximation cannot be used.

Using the relation $s_4(t) = -2I(t) + 1$ (see (13)), we can restate the fuzzy rules as

R_1: If $2.16I(t-3) + 1.5I(t-2)$ is *st* 0.6 and $5I(t-1)$ is *st* 2.5,
Then $\sigma^{-1}(s_1(t)) = f_1$,

R_2: If $2.16I(t-3) + 1.5I(t-2)$ is *st* 0.6 and $5I(t-1)$ is *lt* 2.5,
Then $\sigma^{-1}(s_1(t)) = f_2$,

R_3: If $2.16I(t-3) + 1.5I(t-2)$ is *lt* 0.6 and $5I(t-1)$ is *st* 2.5,
Then $\sigma^{-1}(s_1(t)) = f_3$,

R_4: If $2.16I(t-3) + 1.5I(t-2)$ is *lt* 0.6 and $5I(t-1)$ is *lt* 2.5,
Then $\sigma^{-1}(s_1(t)) = f_4$,

where $\mathbf{f} = (f_1, f_2, f_3, f_4) = (3.9 + 15.2s_1(t-1), -2.1 + 15.2s_1(t-1), -4.5 + 15.2s_1(t-1), -10.5 + 15.2s_1(t-1))$. To better understand this FRB we consider two extreme cases: $s_1(t-1) = 0.9$ and $s_1(t-1) = 0.1$.

- For $s_1(t-1) = 0.9$, the output is $\mathbf{f} = (17.6, 11.6, 9.2, 3.2)$. Since the inferencing yields a weighted sum of the f_is, $\sigma^{-1}(s_1(t)) \geq 3.2$, that is, $s_1(t) \geq 0.96$. Roughly speaking, this implies that if $s_1(t-1)$ is ON, then $s_1(t)$ is also ON, regardless of the value of the input.

- For $s_1(t-1) = 0.1$, the output is $\mathbf{f} = (5.4, -0.6, -3, -9)$, and $\sigma(\mathbf{f}) = (1, 0.35, 0.05, 0)$. This implies that $\sigma(\mathbf{f}) > 0.5$ iff the first rule has a high

DOF. Examining the If-part of the first rule, we see that this happens only if $I(t-1) = I(t-2) = I(t-3) = 0$. In other words, $s_1(t)$ will be ON only if the last three consecutive inputs were zero.

Recalling that the network rejects a string iff f_{out} – the value of s_1 at the end of the string – is larger than 0.5, we can now easily explain the entire RNN functioning as follows. The value of neuron s_1 is initialized to OFF. It switches to ON whenever three consecutive zeros are encountered. Once it is ON, it remains ON, regardless of the input. Thus, the RNN recognizes strings containing a '000' substring.

Summarizing, using the APFRB–RNN equivalence, fuzzy rules that describe the RNN behavior were extracted. Simplifying those symbolic rules offers a comprehensible explanation of the RNN internal features and performance.

7 Conclusions

The ability of ANNs to learn and generalize from examples makes them a suitable tool for numerous applications. However, ANNs learn and process knowledge in a form that is very difficult to comprehend. This subsymbolic, black-box character is a major drawback and hampers the more widespread use of ANNs. This problem is especially relevant for RNNs because of their intricate feedback connections.

We presented a new approach for extracting symbolic knowledge from

RNNs which is based on the mathematical equivalence between sums of sigmoids and a specific Mamdani-type FRB – the APFRB.

We demonstrated our approach using the well-known problem of designing an RNN that recognizes a specific formal language. An RNN was trained using RTRL to correctly classify strings generated by a regular grammar. Using the equivalence (1) provided a symbolic representation, in terms of a suitable APFRB, of the functioning of each neuron in the RNN. Simplifying this APFRB led to an easy to understand explanation of the RNN functioning.

An interesting topic for further research is the analysis of the dynamic behavior of various training algorithms. It is possible to represent the RNN *modus operandi*, as a set of symbolic rules, *at any point during the training process.* By understanding the RNN after each iteration of the learning algorithm, it may be possible to gain more insight not only into the final network, but into the learning process itself, as well.

In particular, note that the RNN described in section 5 is composed of several *interdependent* neurons (e.g., the correct functioning of a neuron that detects a '000' substring depends on the correct functioning of the neuron that detects a '00' substring). An interesting problem is the order in which the functioning of the different neurons develops during training. Do they develop in parallel or do the simple functions evolve first and are then used later on by other neurons? Extracting the information embedded in each neuron during different stages of the training algorithm may shed some light on this problem.

Appendix: Proof of Proposition 1

We require the following result.

Lemma 1 *Consider the RNN defined by* (12) *and* (13)*. Suppose that there exist* $\epsilon_1, \epsilon_2 \in [0, 0.5)$ *such that the following conditions hold:*

1. *If* $s_1(t-1) \geq 1 - \epsilon_1$ *then* $s_1(t) \geq 1 - \epsilon_1$.
2. *If* $s_4(t-1) = s_4(t-2) = s_4(t-3) = 1$ *then* $s_1(t) \geq 1 - \epsilon_1$.
3. *Else, if* $s_1(t-1) \leq \epsilon_2$ *then* $s_1(t) \leq \epsilon_2$.

Then, the RNN correctly classifies any binary string according to L_4.

Proof. Consider an arbitrary input string. Denote its length by l. We now consider two cases.
Case 1: The string does not include the substring '000'. In this case, the If-part in Condition 2 is never satisfied. Since $s_1(1) = 0$, Condition 3 implies that $s_1(t) \leq \epsilon_2$, for $t = 1, 2, 3 \ldots$, hence, $s_1(l+1) \leq \epsilon_2$. Recalling that the network output is $f_{out} = s_1(l+1)$, yields $f_{out} \leq \epsilon_2$.
Case 2: The string contains a '000' substring, say, $I(m-2)I(m-1)I(m)$ ='000', for some $m \leq l$. Then, according to Condition 2, $s_1(m+1) \geq 1-\epsilon_1$. Condition 1 implies that $s_1(t) \geq 1-\epsilon_1$ for $t = m+1, m+2, \ldots$, so $f_{out} \geq 1-\epsilon_1$.

Summarizing, if the input string includes a '000' substring, then $f_{out} \geq 1 - \epsilon_1 > 0.5$, otherwise, $f_{out} \leq \epsilon_2 < 0.5$, so the RNN accepts (rejects) all the strings that do (not) belong to the language. $\square$

We can now prove Proposition 1 by showing that the RNN with parameters (14) indeed satisfies the three conditions in Lemma 1. Using (14) yields

$$s_1(t) = \sigma\left(15.2 s_1(t-1) + 8.4 s_2(t-1) + 0.2 s_3(t-1) + 3 s_4(t-1) - 7.6\right). \tag{23}$$

Substituting (13) in (15) and (16) yields

$$s_4(t) \in \{-1, 1\}, s_3(t) \in \{0.015, 0.98\}, \text{ and } s_2(t) \in [0, 0.8]. \tag{24}$$

Now, suppose that

$$s_1(t-1) \geq 1 - \epsilon_1. \tag{25}$$

Since $\sigma(\cdot)$ is a monotonically increasing function, we can lower bound $s_1(t)$ by substituting the minimal value for the expression in the brackets in (23). In this case, (23), (24), and (25) yield

$$\begin{aligned} s_1(t) &\geq \sigma\big(15.2(1-\epsilon_1) + 8.4 \cdot 0 + 0.2 \cdot 0.015 - 3 - 7.6\big) \\ &= \sigma(-15.2\epsilon_1 + 4.6). \end{aligned}$$

Thus, Condition 1 in Lemma 1 holds if $\sigma(-15.2\epsilon_1 + 4.6) \geq 1 - \epsilon_1$. It is easy to verify that this indeed holds for any $0.01 < \epsilon_1 < 0.219$.

To analyze the second condition in Lemma 1, suppose that $s_4(t-1) = s_4(t-2) = s_4(t-3) = 1$. It follows from (12) and (14) that $s_3(t-1) = \sigma(3.8) = 0.98$, and $s_2(t-1) \geq \sigma\left(-0.2 \cdot 0.8 + 4.5\sigma(3.8) + 1.5 - 4.7\right) = 0.73$. Substituting these values in (23) yields

$$\begin{aligned}
s_1(t) &= \sigma\big(15.2 s_1(t-1) + 8.4 s_1(t-1) + 0.2 \cdot 0.98 + 3 - 7.6\big) \\
&\geq \sigma\big(15.2 s_1(t-1) + 8.4 \cdot 0.73 + 0.2 \cdot 0.98 + 3 - 7.6\big) \\
&\geq \sigma(1.72),
\end{aligned}$$

where the third line follows from the fact that $s_1(t-1)$, being the output of the Logistic function, is non-negative. Thus, Condition 2 in Lemma 1 will hold if $\sigma(1.72) \geq 1 - \epsilon_1$, or $\epsilon_1 \geq 0.152$.

To analyze the third condition in Lemma 1, suppose that $s_1(t-1) \leq \epsilon_2$. Then (23) yields

$$s_1(t) \leq \sigma\big(15.2\epsilon_2 + 8.4 s_2(t-1) + 0.2 s_3(t-1) + 3 s_4(t-1) - 7.6\big).$$

We can upper bound this by substituting the maximal values for the expression in the brackets. Note, however, that Condition 3 of the Lemma does not apply when $s_4(t-1) = s_4(t-2) = s_4(t-3) = 1$ (this case is covered by

Condition 2). Under this constraint, applying (24) yields

$$\begin{aligned} s_1(t) &\le \sigma\big(15.2\epsilon_2 + 8.4 \cdot 0.8 + 0.2 \cdot 0.98 - 3 - 7.6\big) \\ &= \sigma(15.2\epsilon_2 - 3.684). \end{aligned}$$

Thus, Condition 3 of Lemma 1 will hold if $\sigma(15.2\epsilon_2 - 3.684) \le \epsilon_2$ and it is easy to verify that this indeed holds for $0.06 < \epsilon_2 < 0.09$.

Summarizing, for $\epsilon_1 \in [0.152, 0.219)$ and $\epsilon_2 \in (0.06, 0.09)$, the trained RNN satisfies all the conditions of Lemma 1. This completes the proof of Proposition 1. $\square$

Acknowledgments

We are grateful to the anonymous reviewers for their detailed and constructive comments.

References

[1] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-Based Systems*, vol. 8, pp. 373–389, 1995.

[2] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *Neural Computation*, vol. 5, pp. 54–65, 1994.

[3] J. M. Benitez, J. L. Castro, and I. Requena, "Are artificial neural networks black boxes?" *IEEE Trans. Neural Networks*, vol. 8, pp. 1156–1164, 1997.

[4] C. M. Bishop, *Neural Networks for Pattern Recognition.* Oxford, U.K.: Oxford University Press, 1995.

[5] A. Blanco, M. Delgado, and M. C. Pegalajar, "Fuzzy automaton induction using neural network," *Int. J. Approximate Reasoning*, vol. 27, pp. 1–26, 2001.

[6] J. L. Castro, C. J. Mantas, and J. M. Benitez, "Interpretation of artificial neural networks by means of fuzzy rules," *IEEE Trans. Neural Networks*, vol. 13, pp. 101–116, 2002.

[7] N. Chomsky, "Three models for the description of language," *IRE Trans. Information Theory*, vol. IT-2, pp. 113–124, 1956.

[8] N. Chomsky and M. P. Schützenberger, "The algebraic theory of context-free languages," in *Computer Programming and Formal Systems*, P. Bradford and D. Hirschberg, Eds. North-Holland, 1963, pp. 118–161.

[9] I. Cloete and J. M. Zurada, Eds., *Knowledge-Based Neurocomputing.* MIT Press, 2000.

[10] D. Dubois, H. T. Nguyen, H. Prade, and M. Sugeno, "Introduction: the real contribution of fuzzy systems," in *Fuzzy Systems: Modeling and Control*, H. T. Nguyen and M. Sugeno, Eds. Kluwer, 1998, pp. 1–17.

[11] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Representation of finite-state automata in recurrent radial basis function networks," *Machine Learning*, vol. 23, pp. 5–32, 1996.

[12] L. M. Fu, "Learning capacity and sample complexity on expert networks," *IEEE Trans. Neural Networks*, vol. 7, pp. 1517 –1520, 1996.

[13] S. I. Gallant, "Connectionist expert systems," *Comm. ACM*, vol. 31, pp. 152–169, 1988.

[14] C. L. Giles, S. Lawrence, and A. C. Tsoi, "Noisy time series prediction using recurrent neural networks and grammatical inference," *Machine Learning*, vol. 44, pp. 161–183, 2001.

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

[16] M. Gori, M. Maggini, E. Martinelli, and G. Soda, "Inductive inference from noisy examples using the hybrid finite state filter," *IEEE Trans. Neural Networks*, vol. 9, pp. 571–575, 1998.

[17] S. Guillaume, "Designing fuzzy inference systems from data: an interpretability-oriented review," *IEEE Trans. Fuzzy Systems*, vol. 9, no. 3, pp. 426–443, 2001.

[18] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, 2nd ed. Addison-Wesley, 2001.

[19] M. Ishikawa, "Structural learning and rule discovery," in *Knowledge-Based Neurocomputing*, I. Cloete and J. M. Zurada, Eds. Kluwer, 2000, pp. 153–206.

[20] H. Jacobsson, "Rule extraction from recurrent neural networks: a taxonomy and review," *Neural Computation*, vol. 17, pp. 1223–1263, 2005.

[21] J.-S. R. Jang and C.-T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Trans. Neural Networks*, vol. 4, pp. 156–159, 1993.

[22] J.-S. R. Jang, C.-T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence.* Prentice-Hall, 1997.

[23] Y. Jin, W. Von Seelen, and B. Sendhoff, "On generating FC3 fuzzy rule systems from data using evolution strategies," *IEEE Trans. Systems, Man and Cybernetics*, vol. 29, pp. 829–845, 1999.

[24] S. C. Kleene, "Representation of events in nerve nets and finite automata," in *Automata Studies*, C. E. Shannon and J. McCarthy, Eds. Princeton University Press, 1956, vol. 34, pp. 3–41.

[25] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artificial Intelligence (IJCAI)*, 1995, pp. 1137–1145.

[26] J. F. Kolen, "Fool's gold: extracting finite state machines from recurrent network dynamics," in *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. Morgan Kaufmann, 1994, pp. 501–508.

[27] J. F. Kolen and J. B. Pollack, "The observer's paradox: apparent computational complexity in physical systems," *J. Experimental and Theoretical Artificial Intelligence*, vol. 7, pp. 253–277, 1995.

[28] E. Kolman and M. Margaliot, "A new approach to knowledge-based design of recurrent neural networks," *IEEE Trans. Neural Networks*, submitted.

[29] ——, "Are artificial neural networks white boxes?" *IEEE Trans. Neural Networks*, vol. 16, pp. 844–852, 2005.

[30] ——, "Knowledge extraction from neural networks using the all-permutations fuzzy rule base: The LED display recognition problem," *IEEE Trans. Neural Networks*, vol. 18, pp. 925–931, 2007.

[31] V. Kreinovich, C. Langrand, and H. T. Nguyen, "A statistical analysis for rule base reduction," in *Proc. 2nd Int. Conf. Intelligent Technologies (InTech'2001)*, Bangkok, Thailand, 2001, pp. 47–52.

[32] C.-M. Kuan and T. Liu, "Forecasting exchange rates using feedforward and recurrent neural networks," *J. Applied Econometrics*, vol. 10, pp. 347–364, 1995.

[33] S.-W. Lee and H.-H. Song, "A new recurrent neural network architecture for visual pattern recognition," *IEEE Trans. Neural Networks*, vol. 8, pp. 331–340, 1997.

[34] P. Manolios and R. Fanelly, "First order recurrent neural networks and deterministic finite state automata," *Neural Computation*, vol. 6, pp. 1155–1173, 1994.

[35] C. J. Mantas, J. M. Puche, and J. M. Mantas, "Extraction of similarity based fuzzy rules from artificial neural networks," *Int. J. Approximate Reasoning*, vol. 43, pp. 202–221, 2006.

[36] M. Margaliot and G. Langholz, "Hyperbolic optimal control and fuzzy control," *IEEE Trans. Systems, Man and Cybernetics*, vol. 29, pp. 1–10, 1999.

[37] ——, *New Approaches to Fuzzy Modeling and Control - Design and Analysis.* World Scientific, 2000.

[38] W. S. McCulloch and W. H. Pitts, "A logical calculus of the ideas imminent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–137, 1943.

[39] L. R. Medsker and L. C. Jain, Eds., *Recurrent Neural Networks: Design and Applications.* CRC Press, 1999.

[40] C. B. Miller and C. L. Giles, "Experimental comparison of the effect of order in recurrent neural networks," *Int. J. Pattern Recognition and Artificial Intelligence*, vol. 7, pp. 849–872, 1993.

[41] S. Mitra and Y. Hayashi, "Neuro-fuzzy rule generation: survey in soft computing framework," *IEEE Trans. Neural Networks*, vol. 11, pp. 748–768, 2000.

[42] S. Mitra and S. Pal, "Fuzzy multi-layer perceptron, inferencing and rule generation," *IEEE Trans. Neural Networks*, vol. 6, pp. 51–63, 1995.

[43] V. Novak, "Are fuzzy sets a reasonable tool for modeling vague phenomena?" *Fuzzy Sets and Systems*, vol. 156, pp. 341–348, 2005.

[44] C. W. Omlin and C. L. Giles, "Extraction and insertion of symbolic information in recurrent neural networks," in *Artificial Intelligence and Neural Networks: Steps toward Principled Integration*, V. Honavar and L. Uhr, Eds. Academic Press, 1994, pp. 271–299.

[45] ——, "Extraction of rules from discrete-time recurrent neural networks," *Neural Networks*, vol. 9, pp. 41–52, 1996.

[46] ——, "Symbolic knowledge representation in recurrent neural networks: insights from theoretical models of computation," in *Knowledge-Based Neurocomputing*, I. Cloete and J. M. Zurada, Eds. Kluwer, 2000, pp. 63–115.

[47] C. W. Omlin, K. K. Thornber, and C. L. Giles, "Fuzzy finite-state automata can be deterministically encoded into recurrent neural networks," *IEEE Trans. Fuzzy Systems*, vol. 6, pp. 76–89, 1998.

[48] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Trans. Neural Networks*, vol. 3, pp. 683–697, 1992.

[49] T. Robinson, M. Hochberg, and S. Renals, "The use of recurrent networks in continuous speech recognition," in *Automatic Speech and Speaker Recognition: Advanced Topics*, C. H. Lee, F. K. Soong, and K. K. Paliwal, Eds. Kluwer Academic, 1996, pp. 233–258.

[50] P. Rodriguez, "Simple recurrent networks learn context-free and context-sensitive languages by counting," *Neural Computation*, vol. 13, pp. 2093–2118, 2001.

[51] P. Rodriguez, J. Wiles, and J. L. Elman, "A recurrent neural network that learns to count," *Connection Science*, vol. 11, pp. 5–40, 1999.

[52] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Research and Development*, vol. 3, pp. 211–229, 1959.

[53] S. Snyders and C. W. Omlin, "What inductive bias gives good neural network training performance?" in *Proc. Int. Joint Conf. Neural Networks (IJCNN'00)*, Como, Italy, 2000, pp. 445–450.

[54] J. W. C. Sousa and U. Kaymak, *Fuzzy Decision Making in Modeling and Control*. World Scientific, 2002.

[55] P. Tiňo and M. Köteles, "Extracting finite-state representation from recurrent neural networks trained on chaotic symbolic sequences," *Neural Computation*, vol. 10, pp. 284–302, 1999.

[56] P. Tiňo and V. Vojtec, "Extracting stochastic machines from recurrent neural networks trained on complex symbolic sequences," *Neural Network World*, vol. 8, pp. 517–530, 1998.

[57] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich, "The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks," *IEEE Trans. Neural Networks*, vol. 9, pp. 1057–1068, 1998.

[58] M. Tomita, "Dynamic construction of finite-state automata from examples using hill-climbing," in *Proc. 4th Annual Conf. Cognitive Science*, 1982, pp. 105–108.

[59] G. Towell, J. Shavlik, and M. Noordewier, "Refinement of approximate domain theories by knowledge based neural network," in *Proc. 8th National Conf. Artificial Intelligence*, 1990, pp. 861–866.

[60] E. Tron and M. Margaliot, "Mathematical modeling of observed natural behavior: a fuzzy logic approach," *Fuzzy Sets and Systems*, vol. 146, pp. 437–450, 2004.

[61] R. L. Watrous and G. M. Kuhn, "Induction of finite-state languages using second-order recurrent networks," *Neural Computation*, vol. 4, pp. 406–414, 1992.

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

[63] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

[64] ——, "The concept of a linguistic variable and its application to approximate reasoning," *Information Sciences*, vol. 30, pp. 199–249, 1975.

[65] ——, “Fuzzy logic = computing with words,” *IEEE Trans. Fuzzy Systems*, vol. 4, pp. 103–111, 1996.

[66] Z. Zeng, R. M. Goodman, and P. Smyth, “Learning finite state machines with self-clustering recurrent networks,” *Neural Computation*, vol. 5, pp. 976–990, 1993.