the excellent performance of the another two our algorithms in the underdetermined BSS problem, for separation of artificially created signals with sufficient level of sparseness.

REFERENCES

[1] F. Abrard, Y. Deville, and P. White, "From blind source separation to blind source cancellation in the underdetermined case: A new approach based on time-frequency analysis," in *Proc. 3rd Int. Conf. Independent Component Analysis and Signal Separation (ICA'2001)*, San Diego, CA, Dec. 9–13, 2001, pp. 734–739.

[2] A. Cichocki and S. Amari, *Adaptive Blind Signal and Image Processing*. New York: Wiley, 2002.

[3] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 33–61, 1998.

[4] D. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via l^1 minimization," *Proc. Nat. Acad. Sci.*, vol. 100, no. 5, pp. 2197–2202, 2003.

[5] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis*. New York: Wiley, 2001.

[6] A. Jourjine, S. Rickard, and O. Yilmaz, "Blind separation of disjoint orthogonal signals: Demixing N sources from 2 mixtures," in *Proc. 2000 IEEE Conf. Acoustics, Speech, and Signal Processing (ICASSP'00)*, vol. 5, Istanbul, Turkey, Jun. 2000, pp. 2985–2988.

[7] T.-W. Lee, M. S. Lewicki, M. Girolami, and T. J. Sejnowski, "Blind sourse separation of more sources than mixtures using overcomplete representaitons," *IEEE Signal Process. Lett.*, vol. 6, no. 4, pp. 87–90, 1999.

[8] D. D. Lee and H. S. Seung, "Learning the parts of objects by nonnegative matrix factorization," *Nature*, vol. 40, pp. 788–791, 1999.

[9] F. J. Theis, E. W. Lang, and C. G. Puntonet, "A geometric algorithm for overcomplete linear ICA," *Neurocomput.*, vol. 56, pp. 381–398, 2004.

[10] K. Waheed and F. Salem, "Algebraic overcomplete independent component analysis," in *Proc. Int. Conf. Independent Component Analysis (ICA'03)*, Nara, Japan, pp. 1077–1082.

[11] M. Zibulevsky and B. A. Pearlmutter, "Blind source separation by sparse decomposition in a signal dictionary," *Neural Comput.*, vol. 13, no. 4, pp. 863–882, 2001.

Equivalence Between RAM-Based Neural Networks and Probabilistic Automata

Marcilio C. P. de Souto, Teresa B. Ludermir, and Wilson R. de Oliveira

***Abstract*—In this letter, the computational power of a class of random access memory (RAM)-based neural networks, called general single-layer sequential weightless neural networks (GSSWNNs), is analyzed. The theoretical results presented, besides helping the understanding of the temporal behavior of these networks, could also provide useful insights for the developing of new learning algorithms.**

***Index Terms*—Automata theory, computability, p random access memory (RAM) node, probabilistic automata, RAM-based neural networks, weightless neural networks (WNNs).**

I. INTRODUCTION

The neuron model used in the great majority of work involving neural networks is related to variations of the McCulloch–Pitts neuron, which will be called the *weighted neuron*. A typical weighted neuron can be described by a linear weighted sum of the inputs, followed by some nonlinear transfer function [1], [2]. In this letter, however, the neural computing models studied are based on artificial neurons which often have binary inputs and outputs, and no adjustable weight between nodes. Neuron functions are stored in lookup tables, which can be implemented using commercially available random access memories (RAMs). These systems and the nodes that they are composed of will be described, respectively, as weightless neural networks (WNNs) and weightless nodes [1], [2]. They differ from other models, such as the weighted neural networks, whose training is accomplished by means of adjustments of weights. In the literature, the terms "RAM-based" and "N-tuple based" have been used to refer to WNNs.

In this letter, the computability (computational power) of a class of WNNs, called general single-layer sequential weightless neural networks (GSSWNNs), is investigated. Such a class is an important representative of the research on temporal pattern processing in (WNNs) [3]–[8]. As one of the contributions, an algorithm (constructive proof) to map any probabilistic automaton (PA) into a GSSWNN is presented. In fact, the proposed method not only allows the construction of any PA, but also increases the class of functions that can be computed by such networks. For instance, at a theoretical level, these networks are not restricted to finite-state languages (regular languages) and can now deal with some context-free languages. Practical motivations for investigating probabilistic automata and GSSWNNs are found in their possible application to, among others things, syntactic pattern recognition, multimodal search, and learning control [9].

Manuscript received July 6, 2003; revised October 26, 2004.

M. C. P. de Souto is with the Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Campus Universitario, Natal-RN 59072-970, Brazil (e-mail: marcilio@dimap.ufrn.br).

T. B. Ludermir is with the Center of Informatics, Federal University of Pernambuco, Recife 50732-970, Brazil.

W. R. de Oliveira is with the Department of Physics and Mathematics, Rural Federal University of Pernambuco, Recife 52171-900, Brazil.

Digital Object Identifier 10.1109/TNN.2005.849838

II. DEFINITIONS

A. Probabilistic Automata

Probabilistic automata are a generalization of ordinary deterministic finite state automata (DFA) for which an input symbol could take the automaton into any of its states with a certain probability [9].

Definition 2.1: A PA is a 5-tuple $\mathbf{A}_P = (\Sigma, Q, H, q_I, F)$, where

- $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\Sigma|}\}$ is a finite set of ordered symbols called the *input alphabet*;
- $Q = \{q_0, q_2, \ldots, q_{|Q|}\}$ is a finite set of states;
- H is a mapping of $Q \times \Sigma$ into the set of $n \times n$ stochastic state transition matrices (where n is the number of states in Q). The interpretation of $H(a_m), a_m \in \Sigma$, can be stated as follows. $H(a_m) = [p_{ij}(a_m)]$, where $p_{ij}(a_m) \geq 0$ is the probability of entering state q_j from state q_i under input a_m, and $\sum_{j=1}^{n} p_{ij} = 1$, for all $i = 1, \ldots, n$. The domain of H can be extended from Σ to Σ^* by defining the following:
  1) $H(\epsilon) = I_n$, where ϵ is the empty string and I_n is an $n \times n$ identity matrix;
  2) $H(a_{m_1}, a_{m_2}, \ldots, a_{m_k}) = H(a_{m_1})H(a_{m_2}), \ldots, H(a_{m_k})$, where $k \geq 2$ and $a_{m_j} \in \Sigma, j = 1, \ldots, k$.
- $q_I \in Q$ is the initial state in which the machine is found before the first symbol of the input string is processed;
- F is the set of final states ($F \subseteq Q$).

The language accepted by a PA $\mathbf{A}_P$ is $T(\mathbf{A}_P) = \{(\omega, p(\omega)) | \omega \in \Sigma^*, p(\omega) = \pi_0 H(\omega)\pi_F > 0\}$ where: 1) π_0 is a n-dimensional row vector, in which the ith component is equal to one if $q_i = q_I$, and 0 otherwise and 2) π_F is an n-dimensional column vector, in which the jth component is equal to 1 if $q_j \in F$ and 0 otherwise.

The language accepted by $\mathbf{A}_P$ with *cut-point(threshold)* λ, such that $0 \leq \lambda < 1$, is $L(\mathbf{A}_P, \lambda) = \{\omega | \omega \in \Sigma^* \text{ and } \pi_0 H(\omega)\pi_F > \lambda\}$. ■

Probabilistic automata recognize exactly the class of weighted regular languages (WRLs) [9]. Such a class of languages includes properly

all regular languages and also relates to all other language classes in Chomsky's hierarchy. More specifically, there exist WRLs which are not regular languages. For example, [9] presented a weighted regular grammar that generates a context-free language that is not regular. In order to simplify the Proof of Theorem 3.1 presented in Section III, and without loss of generality, the PAs considered in this letter will be constrained to have at most two transitions under a certain symbol leaving a given state [10].

B. GSSWNNs and PAs

The model analyzed in this letter will consist of a variation of the single-layer sequential WNNs in [3]–[8] in that hidden neurons (state neurons) and a single output neuron (a feedforward node) are considered. This version will be called a GSSWNN. Such a structure, without loss of generality, will be regarded as a language acceptor. Furthermore, since the purpose is to have the GSSWNNs simulating the behavior of the PAs, these networks too will be constrained to have, at each time step, at most two possible current states. In order to do so, the set of possible storable values for the states nodes (hidden nodes) will be restricted to six different values $\{p_1, \ldots, p_6\}$, where $p_i \in [0,1], i = 1, \ldots, 6$. These values will have a one-to-one relationship to the following ordered set of binary vectors $\{(0),(1),(0,0),(0,1),(1,0),(1,1)\}$. With this encoding scheme, a set of at most two binary vectors is associated with the values stored in the state neurons, which will represent the actual network current state(s). Hereafter, this encoding scheme will be referred to as the output encoding scheme.

Definition 2.2: A GSSWNN is a seven-tuple $\mathbf{N} = (X, U, y, \mathbf{f}, \mathbf{p}, \mathbf{x}_0, R)$ where

- $X = \{0,1\}^{n_X}$ is the state–space of the GSSWNN, which is represented by a layer of n_X hidden weightless neurons (state neurons).
- $U = \{0,1\}^{n_U}$ defines the set of possible input vectors, with n_U the number of input lines for the network.
- $y = [0,1]$ is the output of GSSWNN, which represents the transition probability of the network.
- $\mathbf{f} : X \times U \rightarrow \{X \cup (X \times X)\}$ is the transition function, which computes a set of current states from the previous state $\mathbf{x}[t-1]$ and the current input $\mathbf{u}[t]$. This functions is implemented as a sequential WNN with only one layer of hidden units (state nodes). The set of possible storable values for these nodes is restricted to six different values $\{p_1, \ldots, p_6\}$, where $p_i \in [0,1], i = 1, \ldots, 6$. Also, each hidden node has a decoder associated with it. The decoder transforms a real-valued node output $p_i \in [0,1]$ into a vector with either one bit or two binary bits, in accordance with the output encoding scheme.
- $\mathbf{p} : X \times U \rightarrow [0,1]$ is the transition probability function, which computes the current transition probability $y[t]$ of the network from the previous state $\mathbf{x}[t-1]$ and current input $\mathbf{u}[t]$. This function is implemented as a feedforward network with a single weightless node.
- $\mathbf{x}_0$ is the initial state of the GSSWNNs, that is, the value that will be used for $\mathbf{x}[0]$.
- $R \subseteq X$ is the set of accepting states of the network.

In order to deal with the set of previous and current states, two first-in first-out (FIFO) queues are associated with the GSSWNN. The main FIFO queue (main FIFO) contains the set of previous states of the network. At each time step, one of the previous states is retrieved from the queue and fed to the network. The second queue, which will be called the auxiliary FIFO queue (auxiliary FIFO), stores the set of current states of the network, after they have been transformed in binary vectors. ■

Since U and y have been defined, respectively, in a binary and continuous spaces, pRAM nodes with binary inputs and real-valued outputs [11] are a straightforward type of weightless node that can be used to implement GSSWNNs. Hereafter, GSSWNNs in this letter will be assumed to be implemented with pRAM nodes with binary inputs and real-valued outputs. The functions $\mathbf{f}$ and $\mathbf{p}$ could be implemented by a coupled single-layer of an output neuron and state neurons. It is assumed that these networks are fully connected. In other words, for every node j in the network, the input terminals $\mathbf{i}_j$ of such a node are completely linked to the elements in the input layer, that is, $\mathbf{i}_j = (u_1, u_2, \ldots, u_{n_U}, x_1, x_2, \ldots, x_{n_X})$.

C. Probabilistic Recognition Algorithm

The special recognition algorithm introduced in this section, which will be called the probabilistic recognition algorithm, makes a GSSWNN function as a PA [10]. In order to deal with the set of previous and current states and to be able to follow all distinct paths, the probabilistic recognition algorithm uses the two FIFO queues associated with the GSSWNNs. Also, without loss of generality, it is assumed that all strings submitted to the network will have an initial symbol ε and final symbol ψ. The symbol ε tells the algorithm that a new string is being submitted to the network. The ε symbol will also be used when an ϵ-transition must be accomplished. Similarly, ψ tells the algorithm that the end of the sentence has been reached.

Based on the previous assumptions, the probabilistic recognition algorithm works as follows.

Step 1) When the network is fed with the first symbol from a sentence ω', the auxiliary FIFO will be set to store the initial state $\mathbf{x}_0$ of the network, and the main FIFO will be empty.

Step 2) If the end of the string has not been reached, the state(s) in the auxiliary FIFO is transferred to main FIFO and the next symbol in the string is read.

Step 3) This symbol together with the first state in the main FIFO are used to compute the current state(s) of the network. Such state(s) are decoded.

a) If the current states are labeled as rejected states, recognition stops and the string is said to be rejected by the network.

b) If any of the current states is labeled as containing ϵ-transitions, a recursive subroutine is called. The aim of such a subroutine is to follow all the ϵ-transitions associated with a given current state. Moreover, the current symbol being fed to network is temporally replaced by the additional symbol ε—recall that in terms of automata an ϵ-transition means a transition under the empty symbol ϵ. The result of following the ϵ-transitions are stored in the auxiliary FIFO.

c) If none of the previous conditions is true, then simply store the set of current states in the auxiliary FIFO.

Step 3) continues while the main FIFO is not empty. In other words, the current symbol will be kept in the input line while the main FIFO is not empty. This means that, for each current symbol, there might be a finite number of intermediate steps before the next symbol is fed.

Step 4) When the main FIFO is empty, the process restarts from Step 2). Hence, with this algorithm, the network can go on all distinctive paths that a string ω' can follow and keep track of the probability computed for the states used by the network.

There are different ways to calculate and store these probabilities [10]. For instance, since there can be only two different transitions at each time, a binary tree (probability tree) can be built to store the states

used by the network and their respective transition probabilities. At the end of the process, when the last symbol is fed, if both 1) the states at which the network ultimately arrives contains at least one of the network accepting (final) states and 2) the sum of the probabilities computed for these states is greater than the threshold (cut-point λ), the sentence is accepted by the network. In order to process a sentence, in the worst case, such an algorithm will make $l * n$ accesses to the memory locations in the network, where l is the length of the sentence and n the number of network valid states (i.e., those states which are not garbage states).

Theorem 2.1: [10] From the set of all languages, any language L which is recognized by a GSSWNN with the probabilistic recognition algorithm can be recognized by a PA.

The previous theorem implies that any GSSWNN, when used with the probabilistic recognition algorithm, has its computability restricted to the class of PAs. Thus, in this context, whatever problem a GSSWNN is given, such a problem can described in terms of a PA (WRL acceptors).

III. Synthesis of PAs in GSSWNNs

GSSWNNs are designed from PAs in a series of steps. The first is to assign unique binary tuples to each input symbol and state (the state-assignment problem). Second, tables for the transition function of the PA and its version with the current states codified, respectively, are produced. If a PA has ϵ-transitions, without loss of generality, a symbol ε is added to the input alphabet of this PA. Then, such a symbol is employed as the input symbol in the entries of the transition table that represent ϵ-transitions. Third, the contents of the nodes in the network are configured such that they can perform the transition function of the PA. Fourth and finally, the network is used with the recognition algorithm proposed in the previous section. This can be formally defined as follows.

Theorem 3.1: Let $\mathbf{A}_P = (\Sigma, Q, H, q_I, F)$ be a PA with cut-point λ. Then there exists a GSSWNN that implements $\mathbf{A}_P$.

Proof: $\mathbf{A}_P$ is reduced to a GSSWNN $\mathbf{N} = (X, U, y, \mathbf{f}, \mathbf{p}, \mathbf{x}_0)$ as follows.

1) A mapping $\Sigma \rightarrow \{0,1\}^{n_U}$, where $n_U = \lceil \log_2 |\Sigma| \rceil$, is defined. Such a mapping transforms each possible input symbol $\sigma_k \in \Sigma$ into a different vector $\mathbf{u}_k \in U$.
2) A mapping $Q \rightarrow \{0,1\}^{n_X}$, where $n_X = \lceil \log_2 |Q| \rceil$, is defined. Thus, each state $q_i \in Q$ is a assigned a vector $\mathbf{x}_i \in X$ such that if $q_i \neq q_j$, then $\mathbf{x}_i \neq \mathbf{x}_j$.
3) The network initial state $\mathbf{x}_0$ is defined as the vector standing for $q_I \in Q$.
4) The set R of network accepting states will consist of vectors $\mathbf{x}_i \in X$ representing the states $q_i \in F$.

From Step 1) and Step 2), it follows that $\mathbf{N}$ has to have $(n_X + 1)$ nodes with $(n_U + n_X)$ input lines. Next, in order to implement the transition function H of the automaton, functions $f_i (i = 1, \ldots, n_X)$ have to be assigned to each node in the hidden layer. This can be done by setting the contents of these nodes so that they output the current states for all pairs of input symbol and state. Likewise, function $\mathbf{p}$, which computes the probabilities assigned to transitions which lead the network to its current states, can be implemented by setting the contents of the output node (i.e., the feedforward node).

This can be done as follows. Represent the transition function H by a lookup table, [(current input previous state), current state(s) probabilities]. Since a pRAM node can also be seen as a lookup table, it is straightforward to regard the pairs (current input,previous state) as forming addresses to both the nodes in the hidden layer and the node in the output layer. In contrast, each entry current state(s) and probabilities, respectively, defines what each hidden node and the output node have to output for each pair. Without loss of generality, in each entry current state(s), the set of current states is assumed to be in lexicographic order. In addition, by using the output encoding scheme defined in Section II-B, each of these entries can be transformed into a single vector of continuous values. Moreover, it is assumed that there is a one-to-one relationship between the coordinates of the current state vector and the hidden nodes.

After this step, the vectors coding the current states could be directly stored in the contents of the hidden nodes. Now, recall that for a given state and input symbol, the set of possible current states in the lookup table representing the automaton transitions were encoded in lexicographic order. Also, there will exist at most two transitions leaving a given state under a given symbol. Thus, with regard to the probabilities assigned to the these transitions, it is only necessary to store the one concerning the first current state. The omitted one is the complement.

Finally, the locations of the state neurons that represent ϵ-transition will have a specific label assigned to them. Likewise, those locations which were not employed to represent the set of "valid states" (non-garbage states) in Q could be labeled as garbage (rejected) states. These labels will be important for the following reason. During the recognition process, if the current state is labeled as containing ϵ-transitions, the recognition algorithm will have to take a finite number of intermediate steps until the next symbol can be read. In case the current state is labeled as rejected state, the algorithm will stop the recognition process and reject the current string. Thus, the network will not be trapped in undefined configurations. ■

In summary, the algorithm in the previous proof generates a GSSWNN which has its number of nodes logarithmic in the number of automaton states, and each node has a number of memory locations linear in the number of such states. Once the network is generated, it can be used with the new recognition algorithm presented in the previous section, which makes such a network behave like a PA.

Remark 3.1: Theorems 2.1 and 3.1 imply that GSSWNNs and PAs are equivalent in terms of computational power (e.g., they recognize the class of WRLs). Thus, these two theorems together show the computability power of GSSWNNs. □

IV. Discussion

Probabilistic automata with cut-point set to 0 are very similar to hidden Markov models (HMMs). The main difference is that in the case of PAs, the sum of the probabilities of every sentence in the language is required to be equal to 1, whereas in HMMs, the requirement is that the sum of the probabilities for all sentences of length L be equal to 1 [12]. In terms of learning, one can also compare the task of learning HMMs to that of learning PAs with GSSWNNs. For instance, given a fixed topology of a HMM, there are well known algorithm for estimating the transition probabilities (e.g., forward-backward algorithm) and finding the most probable sequence of hidden states given a sequence (e.g., Viterbi algorithm) [12]. With respect to GSSWNNs, an algorithm to estimate the transition probabilities can be very simple. Given a pattern ω, if ω belongs to the language being learned, then reward the network (increase the probabilities of the transitions that the network has undergone when fed with ω), otherwise punish the network (decrease the probabilities of the transitions that the network has undergone when fed with ω). The recognition algorithm used with the GSSWNN is similar to Viterbi algorithm for HMM.

However, one fundamental problem in the application of HMMs is finding the HMM underlying topology, especially when there is no strong evidence toward a specific choice from the application domain (e.g., when doing black box modeling) [13]. In the context of GSSWNNs, learning the underlying topology of a PA (grammatical inference problem) from sentences is also a complex task: the existing

learning algorithms are based on reinforcement learning procedures, in which a reinforcement signal is required at each time step [4], [11], [14]. However, for formal language inference problems the classification of each sentence in the training set is available only after the whole sentence has been fed, not for each individual symbol fed. Thus, the current reinforcement procedures for RAM-based networks cannot be used. An alternative to overcome this problem is to train GSSWNNs with metaheuristics such as genetic algorithm and tabu search [15]–[17]. For example, [15] presented a method for inferring stochastic regular language using evolutionary computing. Whereas, [16] showed a method to train RAM-based networks by using genetic algorithms, tabu search and simulated annealing. Thus, the combination of both methods could give some useful insight in the developing of novel learning algorithms able to train GSSSWNs to learn PAs (or general time series).

Another question worth discussing is how to represent and operate on the probabilities associated with the GSSWNNs. In most computers, real numbers are represented by floating-point numbers. Thus, the "infinite" set of real numbers are represented by the "finite" set of floating-point numbers. This is a great drawback for scientific computation [18], as well as for those WRLs that need high precision along the path of computation. Interval methods and high-accuracy arithmetic have been successfully used to solve the problems facing scientific computation—there are already extensions of usual computer languages that perform interval computations [19], [20]. In fact, based on a probability interval method, in [21] a Java program to transform any PA into a GSSWNNs (including the probabilistic recognition algorithm) was presented. Another approach that could be used to minimize the roundoff errors is, like in HMMs [12], to apply the log to all transition probabilities and cut-point, and then use sums instead of multiplication to calculate the sentence probability along the path of computation.

However, even using interval probabilities methods or log transformation, the implemented system might have its computational power decreased because of its inability to deal with infinite precision. As shown in [9], only the class of PAs with *isolated* cut-point are robust to be implemented. Isolated cut-point means that that some "small" change in the cut-point will not lead the automaton to recognize another language. The class of languages accepted by PAs with isolated cut-point is the same as that of DFAs, that is, the class of regular languages [9]. Thus, although from a theoretical point of view the class of PAs are equivalent to that of GSSWNNs, from a practical perspective GSSWNNs implemented in software or hardware will be able to correctly represent only the class of PAs with isolated cut-point. This might seem a negative result, but if such a result is put into a broader context, one can see that such networks are very powerful when compared to the class of recurrent weighted neural networks with analog activation functions (e.g., the sigmoid function)—analog weighted neural networks [22]. For example, as their state–space is discrete, a GSSWNNs can be constructed—using the algorithm presented in this letter—from a PA with isolated-cut point (or a DFA) in such a way that resulting network will be able to recognizing correctly strings of arbitrary length. On the other hand, the analog weighted neural networks tend to classify correctly only short strings. When they are fed with longer strings, the clusters observed for the values of the hidden state vector (network state) start to blur and finally merge, leading to incorrect state representations [23]. Furthermore, Maass and Sontag [22] have proved that any analog neural network whose computational units are subject to unbounded Gaussian noise or other common noise distributions cannot recognize arbitrary regular languages. In fact, these networks will be able to recognize only definite regular languages, which is a small subclass of the regular languages. That is, in the case of hardware or software implementation, the GSSWNNs are strictly more powerful than the analog weighted neural networks.

REFERENCES

[1] I. Aleksander and H. Morton, *An Introduction to Neural Computing*, 2nd ed. London, U.K.: Chapman & Hall, 1995.

[2] T. B. Ludermir, A. C. P. L. F. de Carvalho, A. P. Braga, and M. C. P. de Souto, "Weightless neural models: A review of current and past works," *Neural Comput. Surv.*, vol. 2, pp. 41–61, 1999.

[3] I. Aleksander and H. Morton, "General neural unit: Retrieval performance," *Electron. Lett.*, vol. 27, pp. 1776–1778, 1991.

[4] D. Gorse and J. G. Taylor, "Enconding temporal structure in probabilistic RAM networks," in *Proc. Inst. Elect. Eng. Int. Conf. Neural Networks*, 1991, pp. 369–372.

[5] P. J. L. Adeodato and J. G. Taylor, "Recurrent neural networks with pRAM's," in *Proc. Int. Conf. Artificial Neural Networks (ICANN'95)*, Paris, France, pp. 607–611.

[6] C. Browne and I. Aleksander, "Digital general units with controlled transition probabilities," *Electron. Lett.*, vol. 32, no. 9, pp. 824–825, 1996.

[7] M. C. P. de Souto, P. J. L. Adeodato, and T. B. Ludermir, "Sequential RAM-based neural networks: Learnability, generalization, rule extraction, and grammatical inference," *Int. J. Neural Syst.*, vol. 2, pp. 203–210, 1999.

[8] G. G. Lockwood and I. Aleksander, "Predicting the behavior of G-RAM networks," *Neural Netw.*, vol. 16, no. 1, pp. 91–100, 2003.

[9] K. S. Fu, *Syntactic Pattern Recognition and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[10] M. C. P. de Souto, T. B. Ludermir, and M. A. Campos, "Encoding of probabilistic automata in RAM-based networks," in *Proc. Int. Joint Conf. Neural Networks*, 2000, pp. 439–444.

[11] D. Gorse and J. G. Taylor, "A continuous input RAM-based stochastic neural model," *Neural Netw.*, vol. 4, pp. 657–665, 1991.

[12] C. D. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press, 1999.

[13] A. Stolcke and S. M. Omohundro. (1994) Best-first model merging for hidden markov model induction, Berkeley, CA. [Online]. Available: citeseer.ist.psu.edu/stolcke94bestfirst.html

[14] D. Gorse, D. A. Romano-Critchley, and J. G. Taylor, "A pulse-based reinforcement algorithm for learning continuous functions," *Neurocomput.*, vol. 14, no. 4, pp. 319–344, 1997.

[15] B. J. Ross, "Probabilistic pattern matching and the evolution of stochastic regular expressions," *Appl. Intell.*, vol. 13, no. 2, pp. 285–300, 2000.

[16] L. A. C. Garcia and M. C. P. de Souto, "Global optimization methods for choosing the connectivity pattern of n-tuple classifiers," in *Proc. IEEE Int. Joint Conf. Neural Networks*, vol. III, Budapest, Hungary, 2004, pp. 2263–2266.

[17] A. Yamazaki, T. B. Ludermir, and M. C. P. de Souto, "Global optimization methods for designing neural networks," in *Proc. VII Brazilian Symp. Neural Network*, vol. 1, Brazil, 2002, pp. 136–141.

[18] D. Goldberg, "What every computer scientist should know about floating-point," in *ACM Comput. Surv.*, vol. 23, 1991, pp. 5–48.

[19] R. E. Moore, *Methods and Applications of Interval Analysis*. Philadelphia, PA: SIAM, 1979.

[20] R. Hammer, M. Neaga, and D. Ratz, "PASCAL-XSC," in *New Concepts for Scientific Computation and Numerical Data Processing*. New York: Academic, 1992, pp. 15–44.

[21] J. C. M. Oliveira, M. C. P. de Souto, and T. B. Ludermir, "Implementation of probabilistic automata in weightless neural networks," in *Proc. VII Brazilian Symp. Neural Networks*, 2002, p. 234.

[22] W. Maass and E. D. Sontag, "Analog neural nets with Gaussian or other common noise distributions cannot recognize arbitrary regular languages," *Neural Computat.*, vol. 11, pp. 771–782, 1997.

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