ELSEVIER Neurocomputing 15 (1997) 327-345

NEUROCOMPUTING

The complexity of language recognition by neural networks

Hava T. Siegelmann[a], C. Lee Giles[b,*]

[a] *Department of Information Systems Engineering, Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel*

[b] *NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA*

Received 19 April 1995; accepted 28 January 1997

Abstract

Neural networks are frequently used as adaptive classifiers. This research represents an attempt to measure the "neural complexity" of any regular set of binary strings, that is, to quantify the size of a recurrent continuous-valued neural network that is needed for correctly classifying the given regular set. Our estimate provides a predictor that is superior to the size of the minimal automaton that was used as an upper bound so far. Moreover, it is easily computable, using techniques from the theory of rational power series in non-commuting variables.

Keywords: Recurrent neural network; Grammatical inference; Grammars; Languages; Counters; Complexity; Automata; Learning

1. Introduction

One of the current applications of recurrent neural networks is as classifiers, or equivalently language acceptors. That is, given a string, the network's output is used to decide whether the string is in the language or not. This research represents an attempt to measure the "neural complexity" of regular languages, that is, to quantify the size of a recurrent neural network needed for acceptance of a given language. Regular languages are those that can be represented by finite automata, and indeed, our work can be thought of as parallel to the field of automata theory. There, a finite automaton that is associated with a regular language is being looked for, and the languages are ordered by the size of the minimal automata that accept them. Here we look at a similar hierarchy but the measure is the minimal network.

* Corresponding author. Tel.: + 1609 951 2642, fax: + 1609 951 2482; e-mail: giles@research.nj.nec.com.

We first provide a technique for estimating the number of neurons necessary to recognize a regular language by a linear activation net. This approach identifies languages with power series, and associates with each of these series its *Hankel matrix*. The rank of each of these Hankel matrices estimates the space complexity of regular languages when using linear-activation networks. We provide a polynomial time algorithm to bound this value and see, that roughly, this estimate is also an upper bound for the saturated and sigmoid models, and is a far better predictor than the size of minimal automaton that has been used as an upper bound so far. We then deviate from the rigoruous mathematical approach; the bound is heuristically improved and it is justified empirically.

1.1. The model

We focus on recurrent neural networks of finite size. Each neuron x_k is updated by a rule of the type

$$x_k(t+1) = \phi\left(\sum_{i=1}^{N}\sum_{j=1}^{M} a_{ij}^{k} I_j(t) x_i(t)\right), \quad k = 1, \dots, N, \tag{1}$$

where N is the number of neurons and M is the number of external input signals. The input vector at each time t

$$I(t) = I_1(t) I_2(t) \cdots I_M(t) \in \{0, 1\}^M$$

consists of $(M-1)$ 0's and a single 1, where the location of the single 1 implies the letter (i.e. unary representation). Note the multiplication $I \cdot x$; these neurons compose the so-called, "second-order networks."

The transfer function ϕ can be one of the following:

- Linear $\mathscr{L}(x) := x$
- Saturated function

$$\pi(x) := \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } 0 \le x \le 1, \\ 1 & \text{if } x > 1. \end{cases}$$

- The classical Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

- Heaviside (also called threshold)

$$\mathscr{H}(x) := \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \ge 0. \end{cases}$$

Our model for accepting languages is a "real time" model in the sense that the response to an input sequence appears s steps after reading it, where s is a fixed constant. The *response* of the network to the string $\omega = \omega_1, \omega_2, \ldots, \omega_l \in \Sigma_M^*$, represented as the input sequence $I(1), I(2), \ldots, I(l)$, is the value $x_1(l+s)$. The network *accepts* or *recognizes* a language $L \subseteq \Sigma_M^*$ if for every string $\omega \in \Sigma_M^*$

$$x_1(l+s) \geq \tau \Leftrightarrow \omega \in L, \qquad x_1(l+s) < \tau \Leftrightarrow \omega \notin L,$$

where τ is a fixed real number, and a fixed initial state is assumed. That is, the language accepted consists of all the input strings to which the network responds with a value larger than τ.

Our ultimate interest is in sigmoidal networks. Networks using the classical sigmoid were reported as relatively easy to train (see e.g. [3]) using gradient descent techniques, and hence have been popular in implementations. (Another reason to focus on the sigmoid activation function stems from its biological plausibility of describing the grade response of a neuron, and also from the fact that this function describes mixture of Gaussians.) As part of this work, we study linear and saturated networks as well.

1.2. Previous work

Previous work which dealt with finding the size of recurrent networks of threshold neurons was described by [1, 4], and recently there has been work on sigmoid encodings [17, 18]. These works calculated the size $f(N)$ of the largest (worst case) network required to simulate any finite automaton of size N. We, on the other hand, are not interested in the *worse case* but rather in the *exact answer*. We gain by dealing directly with recognizing the regular language without going through automata theory: our estimated bound is more natural for this domain and is much smaller than the ones found through automata studies. The correlation between the values we predict and the experimental values we obtain is good.

1.3. Paper organization

The remainder of this paper is organized as follows: Section 2 develops an upper bound on the size of linear networks, this bound is tighter than any non-deterministic finite-state automaton (NDFA) recognizing the language. This bound is based on the rank of Hankel matrices associated with regular languages. Section 3 describes an efficient algorithm to estimate the mathematical formula of Section 2. Section 4 compares different models of activation functions in terms of space complexity, thus suggesting that the space bound from Section 3 can be used to roughly bound the saturated and sigmoidal activation networks as well. From now, we deviate from the rigorous mathematics. Section 5 describes a method to tighten this bound up for saturated activation networks. In Section 6 we use the method from Section 5 as a heuristic bound for sigmoidal networks and justify this bound by a series of computer simulations.

2. Space complexity in linear networks

Let Σ be an alphabet; a *language* L over Σ is a function $f_L: \Sigma^* \to \{0, 1\}$. A *fuzzy language* over Σ^* is a function $f_F: \Sigma^* \to \mathbb{R}$, where $\mathbb{R}$ denotes the real numbers. The value $f_F(\omega)$ denotes the degree of ambiguity, or degree of membership of the string ω in the language. Another representation of fuzzy languages is as a formal *power series* r over Σ^*: $r = \sum_{w \in \Sigma^*} C_w w$, C_w is equivalent to $f_F(w)$. In these same terms, an *unambiguous* or *characteristic* series is one for which $C_w \in \{0, 1\}$ for all $w \in \Sigma^*$; this is correlated with a language. We say that a pair (r, τ), where r is a power series over Σ^* and $\tau \in \mathbb{R}$, *defines* the language $L = L(r, \tau)$ by $w \in L \Leftrightarrow C_w \geq \tau$. We also say that a power series r which belongs to some pair (r, τ) defining L, is *related to the language* L. Similarly, the response of the network $\mathcal{N}$ is a function $f_{\mathcal{N}}: \Sigma^* \to \mathbb{R}$, which is a fuzzy language. The language L *accepted by a neural network* $\mathcal{N}$ is defined *with respect to* a threshold $\tau \in \mathbb{R}$.

In this section we define the *l-complexity* of a language L as the size of the smallest linear-activation neural network accepting L with respect to some threshold $\tau \in \mathbb{R}$. We leave for future work the interesting case where L is defined according to two bounds τ_1, τ_2 such that the response of the network is never in the range $[\tau_1, \tau_2]$.

In what follows, we assume $\Sigma = \{0, 1\}^*$. This is assumed for simplicity, but the results hold for all number of input channels $M > 2$ in Eq. (1) as well.

Definition 2.1 (*Salomaa and Soittola* [8]). The Hankel matrix, H_r, of the power series r is the infinite matrix whose rows and columns are indexed by the strings over $\{0, 1\}^*$ – listed in the lexicographic order – and is defined by $H_r(u, v) = C_{uv}$ where uv denotes the concatenation of the sequences u and v.

Example 2.2. The Hankel matrix $H_r(L)$ of the characteristic power series of $L = 1^*$ is given in Fig. 1.

v/u	λ	0	1	00	01	10	11	⋯
λ	1	0	1	0	0	0	1	
0	0	0	0	0	0	0	0	
1	1	0	1	0	0	0	1	
00	0	0	0	0	0	0	0	
01	0	0	0	0	0	0	0	
⋮								⋱

Fig. 1. A Hankel matrix of the language 1^*.

If r is a power series related to a language L, than the matrix H_r is *associated* with L. Note that there are infinitely many Hankel matrices associated with L according to the choice of $\tau \in \mathbb{R}$.

Definition 2.3. The *H-complexity* of a language L is

$$H\text{-Complexity}(L) = \min_\tau\{\operatorname{rank}(H_r) \mid L = L(r, \tau) \text{ for some } \tau \in \mathbb{R}\}.$$

Theorem 1. *For every language L, l-complexity(L) = H-complexity(L). If L is regular, these numbers are finite.*

Proof. To prove the theorem, we first explain the equivalence between regular languages and a particular subset of power series: the rational series: $\mathbb{R}^{\text{rat}}\langle\langle\Sigma^*\rangle\rangle$. This subset, the set of rational series is defined as the smallest rationally closed subset of the power series over Σ^* which contains all polynomials. We consider the Schutzenberger representation of a rational series and its relationship with the rank of the Hankel matrix of the same series. We end the proof with identifying linear networks with Schutzenberger representations, thus acquiring our result.

1. *The regular languages over an alphabet Σ are the languages associated with the unambiguous series of $\mathbb{R}^{\text{rat}}\langle\langle\Sigma^*\rangle\rangle$, the set of rational series.*
   To argue it, we first cite [8, Theorem (5.2)]; if $r \in \mathbb{R}^{\text{rat}}\langle\langle\Sigma^*\rangle\rangle$ is unambiguous, then $L(r, \tau)$ is regular for every $\tau \in \mathbb{R}$. (Of course, the only interesting case is $\tau = 1$, or equivalently and $\tau \in (0, 1]$. See also [11, Theorem 2.3] for another proof.) Conversely, Theorem (5.1) in [8] states: Let L be a regular language. Then, $\tau_c(L)$ is in $\mathbb{R}^{\text{rat}}\langle\langle\Sigma^*\rangle\rangle$.

2. Given a word $\omega = u_1 \cdots u_n$, we denote by $\tilde{\omega}$ its transpose, $u_n \cdots u_1$.

Lemma 2.4 (Salomaa and Soittola [8]). *Let r be a power series over Σ^*. Then, $r \in \mathbb{R}^{\text{rat}}\langle\langle\Sigma^*\rangle\rangle$ iff the rank of H_r is finite. If H_r has finite rank N, then there exists a monoid representation (called Schutzenberger representation) $R : \Sigma^* \to \mathbb{R}^{N\times N}$ and vectors $a \in \mathbb{R}^{N\times 1}$ such that*

$$r = \sum_{w \in \Sigma^*} aR(\tilde{w})bw. \tag{2}$$

Moreover, if also $r = \sum_{w\in\Sigma^*} a'R'(\tilde{w})b'w$ with $a' \in \mathbb{R}^{1\times m}$, $b' \in \mathbb{R}^{m\times 1}$, $R'(\tilde{w}) \in \mathbb{R}^{m\times m}$, then $m \geq N$. Conversely, if there is any such representation then the rank of H_r is finite.

That is, let L be a language and r be a power series related to L. The power series r admits a presentation (2) of size equal to the rank of H_r, or none if the rank is infinite. The smallest dimension of a representation of a power series related to a language L, is equal to the H-complexity of the language. For regular languages, this is finite by item 1.

3. **Lemma 2.5.** *The response of a linear-activation network to a strong $\omega \in \Sigma^*$ can be written in the form $aR(\tilde{w})b$, for some representation of the same size as the network. Conversely, any power series admitting this representation is a response of some linear-activation network of the same size.*

To prove this lemma we describe a linear-activation network of size N as follows: Each of the $M = 2$ possible input letters, $I(t)_j$, is associated with a weight matrix A_j of size $N \times N$. At time t, the network receives an input letter $\omega(t)$, represented in unary by $I(t)$ and changes its state according to the corresponding associated weight matrices:

$$x(t+1) = \sum_{i=1}^{2} I_i(t) A_i x(t).$$

That is, if $I(t) = (1,0)$ or $I(t) = (0,1)$, then $x(t+1) = A_1 x(t)$ or $x(t+1) = A_2 x(t)$, respectively.

Given such a network, its output is designated by (without loss of generality) the first neuron. Let a be the row vector N, $a = (1,0,0,\ldots,0)$; let the column vector b of size N represent the initial state of the neural network. Assume that the input to the system is $w = w_1, w_2, \ldots, w_k$, where each letter $w_l \in \Sigma$ is associated with the either the matrix A_0 or A_1. Then, the response of the system to the input string w is $aA_{i_k} \ldots A_{i_2} A_{i_1} b$. We denote $R(w) = A_{i_1} \ldots A_{i_{(k-1)}} A_{i_k}$ and conclude that the responses of this network can be represented as in the statement of the lemma.

Conversely, given the Schutzenberger representation (2) for a power series, we can choose a basis in the state-space so that $a = (1,0,0,\ldots,0)$, and a linear-activation network results. Hence, response of linear-activation networks and power series are equivalent.

This completes the proof of Theorem 1. □

Comment. The representation suggested above was introduced first by Schutzenberger [9] and was rediscovered a few times later mainly in the context of stochastic automata. Representatons are called *sequential systems* [16], *generalized linear automata* [6], and *automata with multiplicities* [2]. For a nice summary and discussion, see [12].

3. Bounding the H-complexity

The H-complexity is the exact minimal size of linear-activation nets but it is not easy to calculate. We are interested in a bound that although less accurate, it is efficiently computable.

We define the $\mathscr{C}$-complexity as the size of the minimum linear-activation network that is constrained to output binary values only:

$$\mathscr{C}\text{-complexity}(L) = \operatorname{rank}(H_{r_c(L)}),$$

where $r_c(t)$ is the characteristic series of L. It is clear that H-complexity $\leq \mathscr{C}$-complexity, as the new bound is the size of particular network out of the class of nets included in the former bound. The gap between H-complexity and $\mathscr{C}$-complexity, however, can be arbitrarily large. This is easy to see from counting arguments. Indeed, there are an uncountable number of languages with l-complexity equal to 2 – see for instance [7, p. 311] – but it is easy to see that there are only a countable number of languages with finite $\mathscr{C}$-complexity.

This new bound is still tighter than the existing theory and even smaller than the size of the minimal non-deterministic finite automata realizing the same language. The reasoning is as follows: a non-deterministic finite automata can be realized as a bi-linear system of the type

$$x(t+1) = \tilde{A}_0 x(t) + w(t)\tilde{A}_1 x(t) = (1 - I_1(t))A_0 x(t) + I_1(t) A_1 x(t), \tag{3}$$

where $x(t)$ is a vector of length N of the activations of all neurons, $w = (I, 0, I_1)$ is an input considered as a binary scalar, A_0 and A_1 are transition matrices for the $w(t) = 0$ input and the $w(t) = 1$ input, respectively. Thus A_0 and A_1 are matrices of binary entries. In linear-activation nets, the entries of A_0 and A_1 can be general as long as the output is binary. (Note that it is useful to assume a special "end of string" symbol to handle the case that the NDFA resides in several finite states.) Hence, the NDFA is a special case of the linear-activation net. We conclude that

$$H\text{-complexity} \leq \mathscr{C}\text{-complexity} \leq |\text{NDFA}|.$$

The $\mathscr{C}$-complexity is of interest although not a very tight bound because it can be computed efficiently. The algorithm we suggest is strongly based on system theory. We next provide some of the required preliminaries as described in [12], but full details are outside of the scope of this paper.

A system $S = (X, P, Q, x_0)$ is defined by a vector space X, maps $P: X \times \Sigma \mapsto X$ and $Q: X \times \Sigma \mapsto Y$ and $x_0 \in X$. Σ denote an arbitrary set of input values and Y is the set of output values. A system can equivalently be written as a set of equations

$$x_{t+1} = P(x_t, u_t),$$
$$y_t = Q(x_t, u_t).$$

Our focus is on particular systems when

$$\begin{aligned} x_{t+1} &= \sum_{i=0}^{1} \delta_i(w) F_i x + \delta_i(w) G_i, \\ y_t &= \sum_{i=0}^{1} \delta_i(w) H_i x, \end{aligned} \tag{4}$$

where δ_i are piecewise-linear functions from Σ into $\mathbb{R}$. This family of systems includes our simple network from Eq. (3).

The system S is *span-reachable* iff X is the smallest affine manifold containing the states that are reachable from x_0 for all possible $w \in \Sigma^*$. S is *observable* iff $Q(x)$ are all distinct. A system is *span-canonical* iff it is both span reachable and observable.

Theorem 1.13 in [12] states that a system, for which P and Q are affine, is minimal (in the dimension of X) iff it is span-canonical. The span-canonical representations of a system are the same up to a morphism.

Sontag continues and provides a non-trivial algorithm that for a given system S of the type in Eq. (4), it transfers it to a span-reachable one and then to an observable one, until getting a minimal such a system that has the same input–output response as S. Note that this algorithm is an involved generalization of the simple minimization algorithm of linear systems as suggested by Kalman and is nicely summarized e.g. in [14, p. 91]. There, the "Kalman decomposition" transforms the linear system to reachable (not span-reachable) and then to observable matrices – leading to a canonical representation (rather than span-canonical) which is minimum dimension. The efficient and elegant algorithm of Sontag, on the other hand, is valid to much larger set of systems. In short, Sontag's algorithm is based on the *generalized Hankel matrix*: Rows are indexed (lexicographically) by $\{0, 1\}^*$ and columns by the subset of $\{0, 1\}^*$ that includes no word that starts with the letter 0. The details require wide understanding in system theory and thus we will use the algorithm of Sontag as a "black-box" for our purpose and send the interested reader to the original paper for more [12].

Lemma 3.1. *Given a regular language L represented as a regular expression, the estimate $\mathscr{C}$-complexity (L) is computable in polynomial time in the length of the expression.*

Proof. The following algorithm computes the $\mathscr{C}$-complexity:

1. Given regular expression representing L, one can construct a nondeterministic finite automaton (NDFA) (with "ε-moves") accepting it. The NDFA outputs 1 for the strings in L, and 0 for those that are not in L. The constructed NDFA *does not* have to be a minimal one.
2. The non-deterministic automaton is considered as the Schutzenberger representation of a rational series. (It is equivalent to a linear-activation network, for which the activations of the neurons are non-negative integers, and the transition matrices consist of binary entries only.) This gives rise to a linear-activation network accepting L with $\tau = \frac{1}{2}$. This representation is not necessarily the smallest in dimension.
3. Now we find the span-reachability matrix of the bi-linear realization. This was shown in [12] to have a polynomial-time algorithm. Continue with the last system and find its observability matrix. Again, this can be done in polynomial time [12]. See the related discussion in [13]. The resulting system is both span-reachable and observable and thus is a minimum size bi-linear system, equivalent in its dynamics to the original one. (The minimal network is not unique, but any two such networks can be shown to coincide up to a change of basis in the space of neuron states.)

The total algorithm takes time polynomial in the size of the regular expression. □

In Fig. 2, the first to third columns (ignore the column labeled "experiment" for now), we use nine languages to compare the size of the associated minimal DFA with the

	The Language	DFA's Size	$\mathcal{C}$-Complexity	Experiment
Tomita1:	1*	2	1	1
Tomita2:	(10*)	3	2	2
Tomita3:	no (1^{odd}) followed by (0^{odd})	5	3	3
Tomita4:	does not contain 000 substring	4	3	2
Tomita5:	([01+10] [01+10])*	7	4	4
Tomita6:	#1 - #0 is a multiple of 3	3	3	3
Tomita7:	0*1*0*1*	5	2	2
Parity:	even number of 1's	2	2	2
Dualparity:	even number of 0's, 1's	4	4	2

Fig. 2. Comparison of DFA, $\mathcal{C}$-complexity, and experimental results.

$\mathcal{C}$-complexity. The first seven are known by the name *Tomita languages* [15]; they were used in past grammatical inference studies base on neural networks, e.g. [3]. The last two languages are parity and dual-parity. The comparison with DFA sizes is given here as much of previous literature used this quantity as an estimate of "neural" complexity. (Clearly, the size of a minimal NDFA is bounded by that of the minimal DFA.)

Remark 3.2. In the more realistic case in which the language L is not available, and the only data consists of training strings together with information regarding their membership in the language, one can show that even the question of finiteness of rank for the Hankel matrix is undecidable [11]. However, different heuristics might be useful in this case, such as repetitively enlarging the Hankel matrix until no change in its rank results after a few iterations. We experimented with such heuristics only for too simple languages to be able to conclude favorite statements.

4. Bounds on networks with different activation functions

The space complexity of networks employing the *sigmoidal* activation function: $\sigma(x) = 1/(1 + e^{-x})$, is in a sense bounded by the H-complexity: For any finite number of strings, we may use a low-gain approximation so as to cause the sigmoid to be close enough to the identity function in any given bounded domain. Thus, for any finite sample of a strings, the number of neurons required to accept it in the sigmoidal model is bounded by the number in the linear-activation one.

To estimate better the space required in the sigmoidal model, we experimented with the sigmoid network as NECI [3] and compared the size of the sigmoid network found there with the bound developed above. We trained networks of different size with each of the regular-languages described in the first column and kept track of the minimum network that was successfully trained for each language. Note that the size found in the implementation is not necessarily the minimum possible for recognition, but the best one found in training. That is, it is an upper bound of the minimal network. The correlation between the values predicted by the $\mathcal{C}$-complexity and the experimental values for sigmoids is reasonable, especially in comparison with the

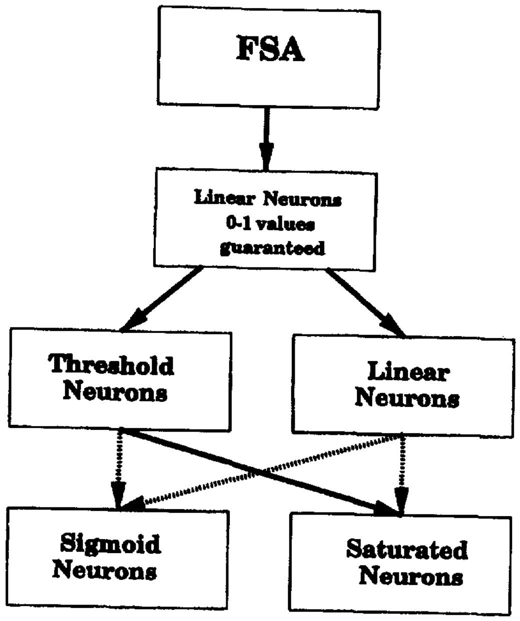


Fig. 3. The space complexity relations in a second-order network.

values that would be predicted from minimal automata size. Using high-gain approximations, one can show that the space complexity of σ nets is also bounded by the complexity of the threshold neurons, on any finite sample.

Generally, the picture is given in Fig. 3. Here, solid arrows denote true relations and dashed arrows stand for relations which hold for finite number of strings.

5. The size of saturated-linear networks

The $\mathscr{C}$-complexity discussed above provides a bound on the size of both the sigmoidal and the saturated nets, as for their "linearity" part around the origin. However, this bound does not use the properties of these functions which stem at their bounded range. Denote by π-complexity of a language L the size of a minimum saturated recurrent network that accepts L. In [10] Siegelmann and Sontag showed how to simulate a counter (also called a unary stack) along with its operation – Increase, Decrease, Iszero – via a fixed number of neurons using a fixed time quantum per operation. It is clear that a bounded-size counter (i.e., one that can count up to a constant c only) can test Isfull analogously to the test Iszero.

Many regular languages can be decomposed into counters and other, smaller regular languages. Consider, for example, the language L which recognizes input strings that include substrings of either 7 or 51 consecutive 1's. Such L can be decomposed into one 7-counter, one 51-counter, and a simple control L' that manipulates the counters. We call any language which is decomposed into counters and control in a non-trivial manner (i.e., the new control has less states than the original one) by the name *counting based language.*

Let L be a counting-based regular language; L can be decomposed into counters and a new regular language L'. (A similar idea of decomposing a language was suggested before in relation to context-free languages. The decomposition was into some super-counter language, named Dyck language, and a regular language, see [8].' The "input letters" of L' are not the external input of L anymore, but rather the Cartesian product of the external input along with the counters' readings. We estimate

$$\pi\text{-complexity } (L) \leq z \cdot v + H\text{-complexity } (L'), \tag{5}$$

where v is the number of counters required, and z is the fixed number of neurons needed to simulate each counter. (The value of z is determined by the operations defined on the particular counter and is typically between 2 and 5.) In practice, we compute a bound on the π-complexity using a decomposition algorithm (to be described below) that uses $\mathscr{C}$-complexity (L') as a subroutine. We start with examples:

Example 1. Consider a language $L \subseteq \{0, 1\}^*$ that accepts a string if it contains a substring of 23 or more 1's. We call it the "23-substring" language. The linear complexity of this language is 24. We decompose it into L' and one counter and show that 4 neurons suffice. Generally, the family of *n-counter* languages

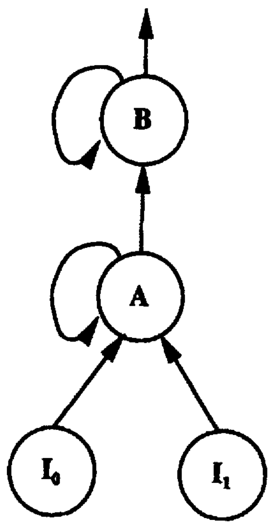
$$\mathscr{L} = \{L_n = \{0, 1\}^* 1^n \{0, 1\}^* | n = 1, 2, 3 \ldots\}$$

has a linear complexity of $n + 1$ while its π-complexity is 4, independently of n. The new language L'_n operates as shown in Fig. 4; its input consists of the pairs [counter-status, input-bits]. Fig. 5 demonstrates the neural network that accepts L_{23}. (I_0 and I_1 are counted as "input neurons".) □

Example 2. We now demonstrate languages that utilize multiple counter *simultaneously*. Dual-parity is the language that accepts strings consisting of even appearance of both "0"s and "1"s. Triple-parity tests for parity of "0"s, "1"s, and "2"s, and so forth. Fig. 6 demonstrates the difference in the complexity measure for the *n-parity* languages. The structure of the network suggested by the decomposition is described in Fig. 7.

The Input		Operation
[*full.counter*	0]	do nothing
[*full.counter*	1]	do nothing
[*full.counter*	*end*]	*accept*
[*nonfull.counter*	0]	reset
[*nonfull.counter*	1]	increment
[*nonfull.counter*	*end*]	*reject*

Fig. 4. The control L' of the n-counter language.



$$\begin{aligned} A(t+1) &= \pi(A(t) + \frac{1}{23}I_1(t) - I_0(t)) \\ B(t+1) &= \pi(23A(t) - 22 + 23B(t)) \end{aligned}$$

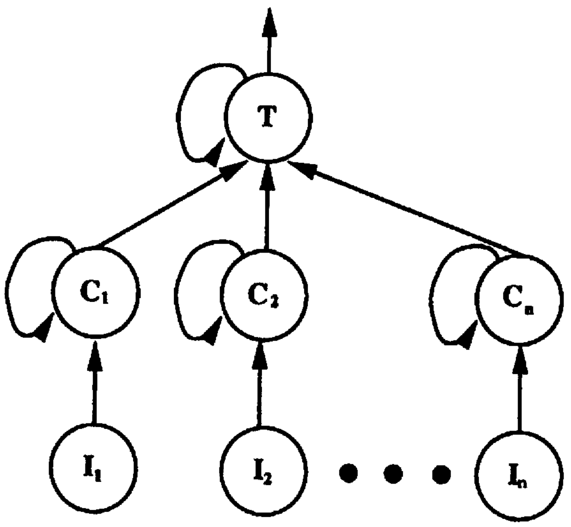
Fig. 5. Network recognizing n-counter languages (L_{23}).

The Language	$\mathcal{C}$-Complexity	π-Complexity
3-parity	8	7
4-parity	16	9
5-parity	32	11
⋮	⋮	⋮
n-parity	2^n	$2n+1$

Fig. 6. Complexity of parity languages.

Notice that for multiple counters (i.e., Cartesian product of counters), the π-complexity is logarithmic in the size of the linear and threshold [4] bounds. Note that the π-complexity is sensitive to the availability of precision in the neurons. It is, however, very useful for a language class of practically used languages.

There is no one general algorithm to decompose regular languages into all possible counters and a new, smaller controls. In Fig. 8 we demonstrate four specific patterns which we transformed into counter-like forms. Many more patterns should be split for minimization. We note that the symbol "1" could also be transformed to any regular



$$\begin{aligned} C_i(t+1) &= \pi(I_i(t) + C_i(t) - 2I_i(t)C_i(t)) \quad i = 1 \ldots M \\ T(t+1) &= \pi(1 - \sum_{i=1}^{M} C_i(t)) \end{aligned}$$

Fig. 7. Network recognizing n-parity languages.

expression. In this case the variable z of Eq. (5) should be increased accordingly. We would like to emphasize that the decomposition algorithm tracks only simple patterns and, by no means, provides an optimal decomposition.

6. Simulation results with a sigmoid network

Let L be a regular language. We denote by σ-complexity the size of a minimum sigmoidal network that recognizes L. Sigmoidal networks are very popular in applications of learning in neural nets. However, they are hard to analyze. In [5], Kilian and Siegelmann proved that sigmoidal networks (and a large class of sigmoidal-like nets) are Turing universal as for their ability to simulate counters (via a fixed number of neurons). In their construction, the time required to decide if a counter is empty grows polynomially with the "history" of the counter; that is, in all operations executed to it. This is mathematically true, however, in practice, one can simulate a counter in real-time for any finite number of strings; see Fig. 3.

We take this counting-ability of sigmoidal nets, along with the speed-up of operations on counters to motivate our heuristic use of the π-complexity as an estimate of the σ-complexity. To verify our heuristic bound, we experimented with the sigmoidal

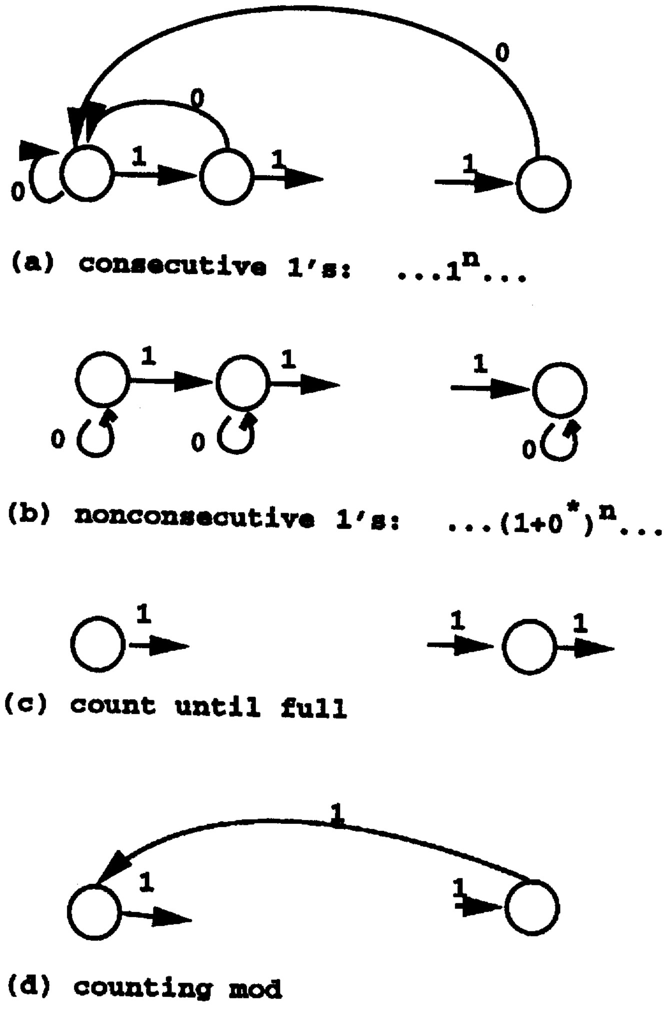

0
0
1
1
1
0
(a) consecutive 1's: ...1^n...
1
1
1
0
0
0
(b) nonconsecutive 1's: ...(1+0*)^n...
1
1
1
(c) count until full
1
1
1
(d) counting mod

Fig. 8. Counter heuristic: types of counters.

network developed by NECI [3]. As in Section 4, we trained networks of different size with each regular language described below, and kept track of the minimum network that was successfully trained for each language.

Training of the network was operated as follows: We started with a network of random coefficients (weights) $\in [-1, 1]$. The network is trained to recognize a set of 1000 random strings of length $\{0 \ldots 18\}$. An extra bit implies whether the string is in the language. The training utilizes the gradient descent technique. The network is trained gradually by subsets of increasing sizes of the given training data until generalizes well (up to strings of length 86). Only if it both converges well on the

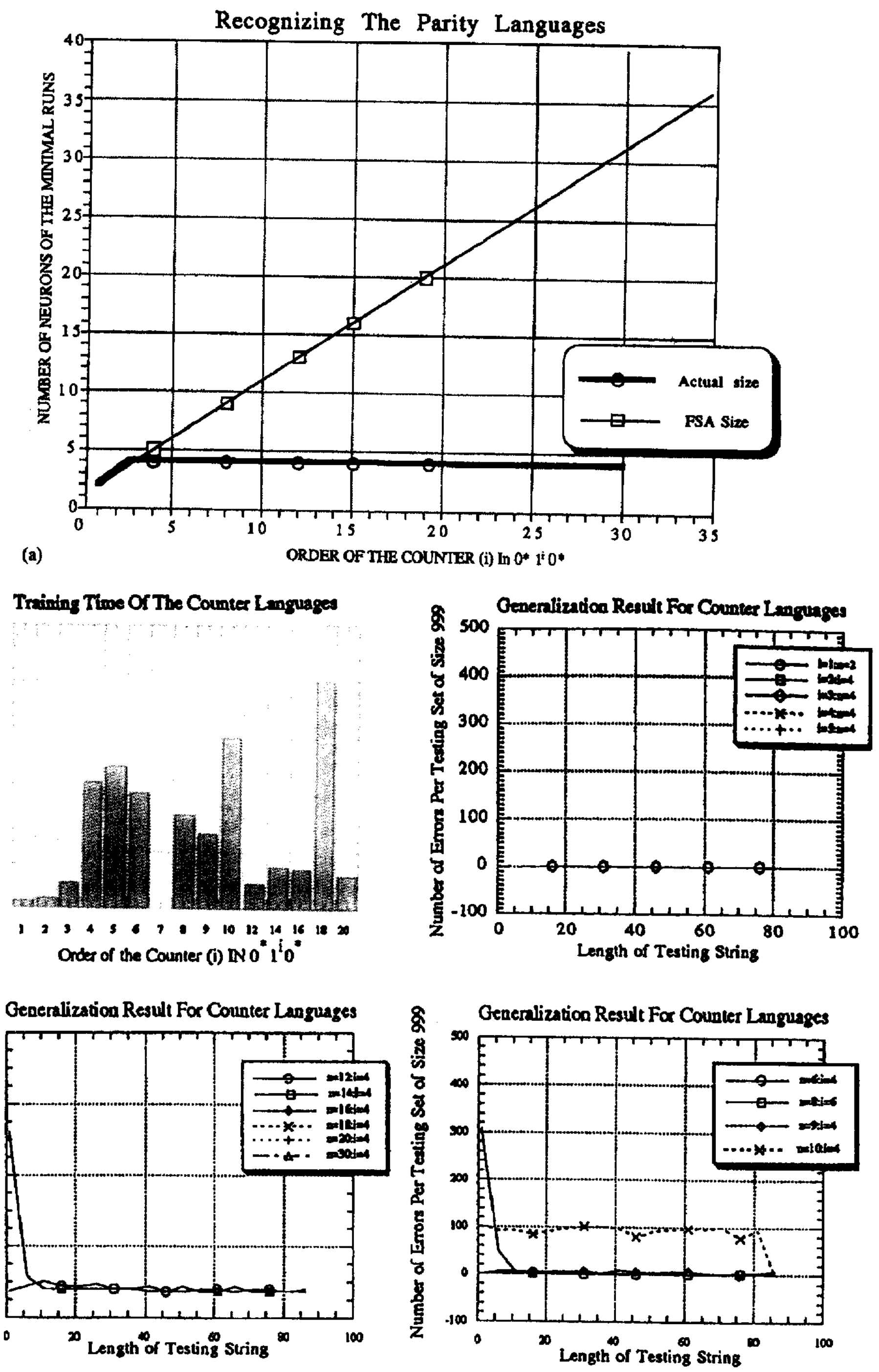


Fig. 9. The Counter Languages. (a) Size of the minimal net found in practice in comparison to FSA size and the counter bound; (b) Training time; (c) Generalization results.

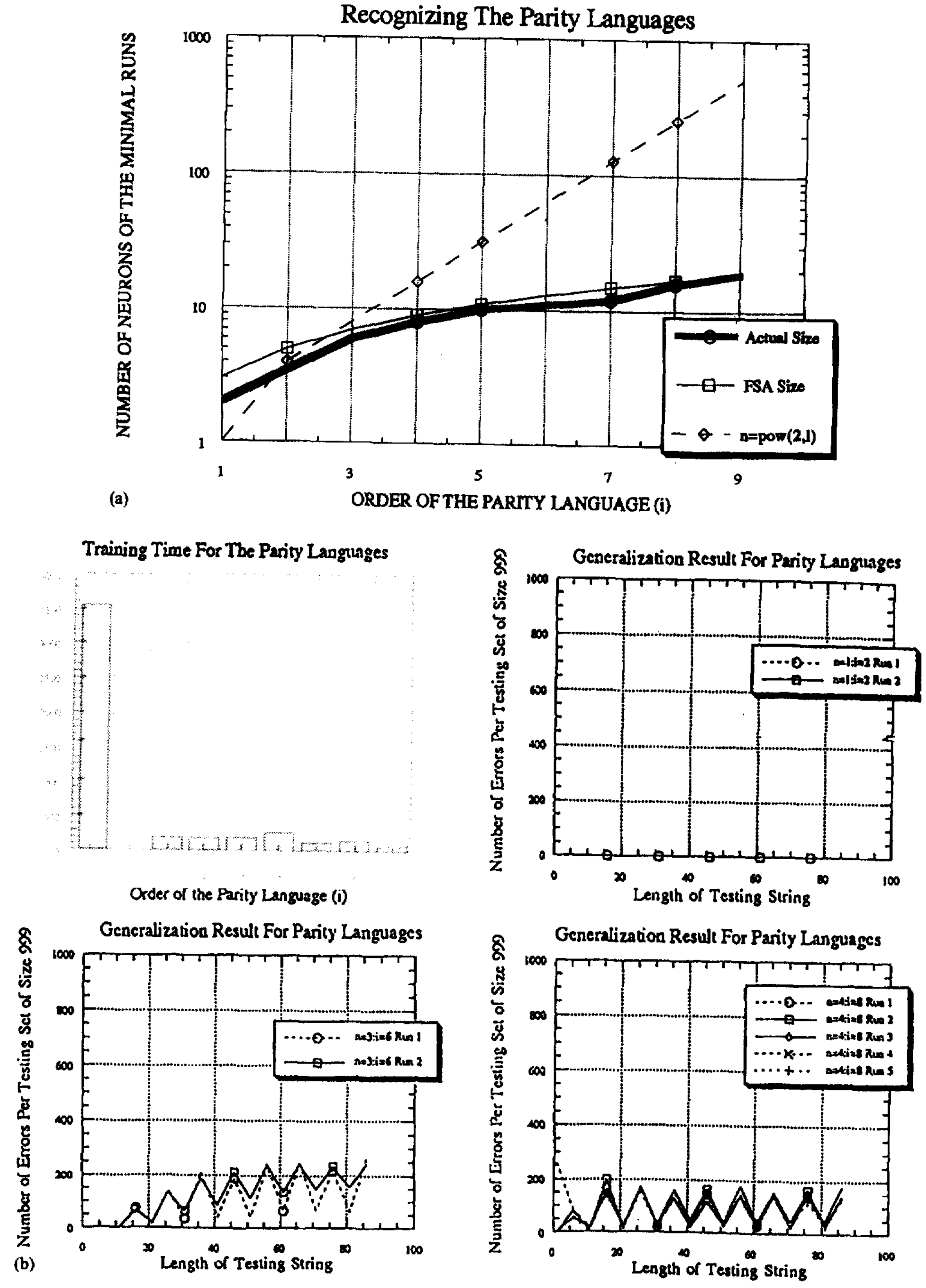


Fig. 10. The Parity Languages. (a) Size of the minimal net found in practice in comparison to the counter bond; (b) Training time; (c) Generalization results.

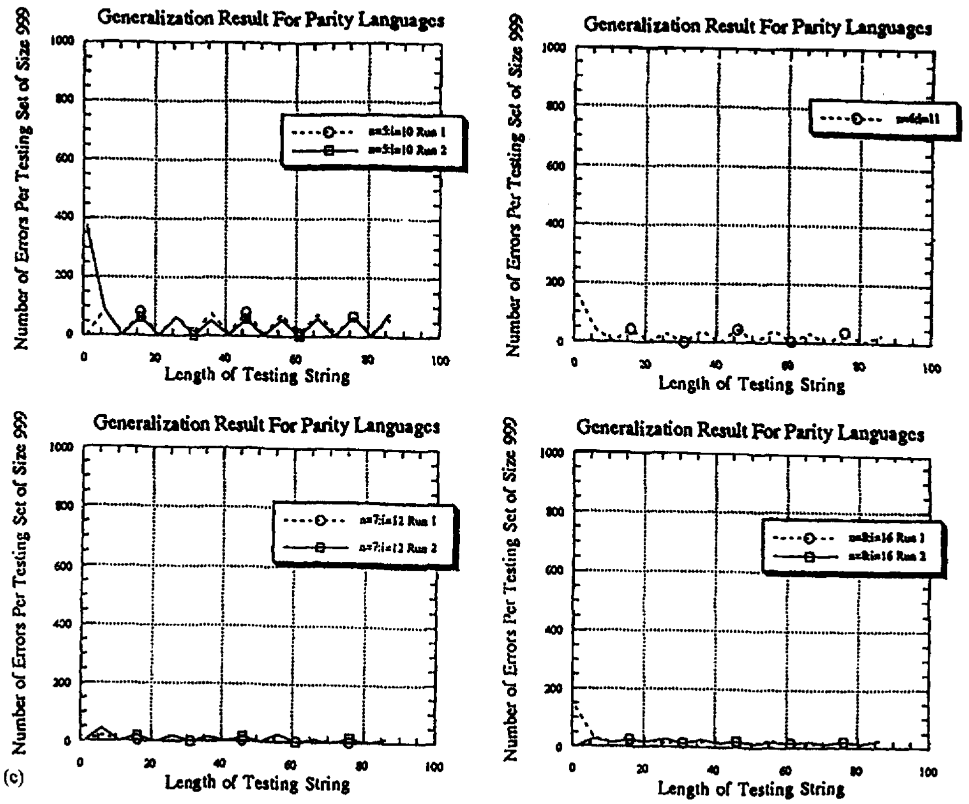


Fig. 10. Continued.

training data and generalizes well to a set of previously unseen strings, we conclude success in the training.

Fig. 9(a) demonstrates the size of the network that we managed to train for the counter languages (thick line) in comparison with the linear bound (Section 3). In this case, the size of the minimal FSA and threshold complexity have similar values to the $\mathscr{C}$-complexity. We experimented with counters of size up to 35. Although the π-complexity provides, in general, an upper bound, for this expected small size 4, the sigmoid network matches the size predicted by the π-complexity.

The training time required for each language is described in Fig. 9(b). We see that, for reasonably short sequences, the training time is not growing with the order of the counter. In Fig. 9(c), we demonstrate the generalization capability of the networks on strings which were unseen during the training phase. All these figures are averaged over several runs.

Fig. 10(a) demonstrates the size of the network that we managed to train for the parity languages, and compares it against the C-complexity and the π-complexity. We experimented with i-parity for $i = 1,2, \dots ,9$. Notice that the graph has a logarithmic scale. This result demonstrates the accuracy of the π-complexity as a heuristic bound for the sigmoidal nets.

7. Conclusions

In this work, we measure the complexity of regular languages by their minimal networks, rather than the minimal finite automata, as is used in automata theory and in previous works in neural networks. Prior to this work, conclusions were drawn from the analysis of automata field and were too weak and artificial in recurrent neural net content. Our results are preliminary and by no means are proved to be the best possible, however, the direction is worth noting. An important applied problem is to come up with a heuristic to estimate the required size of a network, given strings rather than a full description of the language to accept.

The space complexity problem discussed here deals with existence and not with the learning problem. We conjecture that the learning complexity is generally related to the required number of neurons. However, some languages which, although having a small space complexity, are likely to be difficult to learn. This might happen, for example, when a very small set of weights is acceptable. The problem of learning complexity is still open.

Acknowledgements

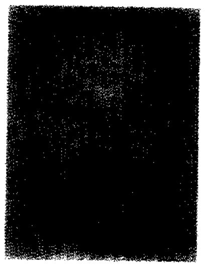
The authors are very grateful for useful discussions with Eduardo Sontag. We are also thankful to Lee Chun Ann, Christian Omlin, and Orna Schechter for various assistance during the work; and to the reviewers for their comments that helped in significantly improving the final representation.

References

[1] N. Alon, A.K. Dewdney, T.J. Ott, J. Assoc. Comput. Mach. 38 (1991) 495.
[2] S. Eilenberg. Automata, Languages, and Machines, vol. A, Academic, New York, 1974.
[3] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, Y.C. Lee, Neural Comput. 4(3) (1992).
[4] B.G. Horne, D.R. Hush, Bounds on the complexity of recurrent neural network implementations of finite state machines. Neural Networks 9(2) (1996) 243.
[5] J. Kilian, H.T. Siegelmann, On the power of sigmoid neural networks, in: Proc. ACM Workship on Computational Learning Theory, Santa Cruz, July 1993.
[6] A.A. Muchnik, Trans Systems Theory Res. 23 (1973).
[7] M. Rabin, Lectures on classical and probabilistic automata, in: E.R. Caianiello (Ed.), Automata Theory. Academic Press, London, 1966.
[8] A. Salomaa, M. Soittola, Automata-Theoretic Aspects of Formal Power Series, Springer, New York 1978.
[9] M.P. Schutzenberger, Inform. Control 4 (1961) 245.
[10] H.T. Siegelmann, E.D. Sontag, Appl. Math. Lett. 4(6) (1991) 77.
[11] E.D. Sontag, J. Comput. System Sci. 11 (1975) 375.
[12] E.D. Sontag, IEEE Trans. Circuits and Systems 26 (1979) 342.
[13] E.D. Sontag, SIAM J. Control Optim. 26(6) (1988) 1106.
[14] E.D. Sontag, Mathematical Control Theory: Deterministic Finite Dimensional Systems, Springer, New York, 1990.

[15] M. Tomita, Dynamic construction of finite-state automata from examples using hill-climbing, in: Proc 4th Annual Cognitive Science Conf., Ann Arbor MI, 1982, pp. 105–108.
[16] P. Turakainen, Ann. Acad. Sci. Fenn. A506 (1972) 1.
[17] P. Frasconi, M. Gori, M. Maggini, G. Soda, Machine Learning 23 (1996) 5.
[18] C.W. Omlin, C.L. Giles, J. Assoc. Comput. Mach. 43(6) (1996) 937.

Hava Siegelmann specializes in biologically motivated information processing systems – including neural networks and evolutionary algorithms – as well as in alternative models of computation (such as analog, distributed, and stochastic dynamics) that are relevant to natural systems. Dr. Siegelmann received her BA from the Technion (Summa Cum Laude) in 1988, MSc from the Hebrew University in 1992, and PhD from Rutgers University (where she was on a Fellowship of Excellence) in 1993 – all in Computer Science. Currently, she is an assistant professor on the faculty of Industrial Engineering and Management at the Technion. Siegelmann is a 1995–1997 ALON Fellow (Israeli Presidential Young Investigator). She has published in a variety of prestigious journals, including Science, Theoretical Computer Science, and the Journal of Computer Systems Science, and has given numerous invited talks throughout her career.



C. Lee Giles is a Senior Research Scientist in Computer Science at NEC Research Institute, Princeton, NJ. His research interests are in the following areas: neural networks, machine learning and AI; hybrid systems and integrating neural networks with intelligent systems and intelligent agents; AI and machine learning applications in finance, communications, world-wide web and computer systems and software; and optics in computing and processing. He has published numerous journal and conference papers and book chapters in these areas. Dr. Giles plays an active professional role in the neural networks and machine learning communities. He serves on many related conference program committees and has helped organize many related meetings and workshops. He has been an advisor and reviewer to government and university programs in both neural networks and in optical computing and processing. He has served or is currently serving on the editorial boards of *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Neural Networks*, *Journal of Parallel and Distributed Computing*, *Neural Networks*, *Neural Computation*, *Optical Computing and Processing*, *Applied Optics*, and *Academic Press*. In 1994, he coedited a special issue for *IEEE Transactions on Neural Networks* on "Dynamic Recurrent Neural Networks." He is a Fellow of the IEEE and a member of AAAI, ACM, INNS and the OSA.