JOURNAL OF COMPLEXITY **15**, 451–475 (1999)
Article ID jcom.1999.0505, available online at http://www.idealibrary.com on IDEAL®

Stochastic Analog Networks and Computational Complexity

Hava T. Siegelmann

Information Systems Engineering, Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel
E-mail:iehava@ie.technion.ac.il

Received July 7, 1997

The model of analog recurrent neural networks (ARNN) is typically perceived as based on either the practical powerful tool of automatic learning or on biological metaphors, yet it constitutes an appealing model of computation. This paper provides rigorous foundations for ARNN, when they are allowed to exhibit stochastic and random behavior of discrete nature. Our model is an extension of the von Neumann model of unreliable interconnection of components and incorporates a generalization of Shannon's random-noise philosophy. In the general case the computational class (P/poly) is associated with both deterministic and stochastic networks. However, when the weights are restricted to rational numbers, stochasticity adds power to the computation. As part of the proof, we show that probabilistic Turing machines that use a coin with a real probability rather than an exactly random ($\frac{1}{2}$) coin, compute the nonuniform version BPP/log* instead of the recursive class BPP. We also show that in the case of real probabilities only their first logarithmic number of bits are relevant for the computation. © 1999 Academic Press

1. INTRODUCTION

This paper examines the effect of random coins on the complexity of particular continuous computational systems, the analog recurrent neural networks (ARNN). The networks consist of multiple assemblies of basic processors interconnected in an intricate structure. Each basic processor, or "neuron," computes a scalar non-linear function of its input. The scalar value produced by a neuron affects other neurons, which then calculate new scalar values of their own. This describes the dynamical behavior of parallel updates. The network is analog in that it is defined on a continuous domain (as opposed to the discrete models of digital computers). The adjective “recurrent” emphasizes that the interconnection is general, rather than layered or symmetrical. Such an architecture, with possible loops, allows the system to evolve for a flexible amount of time by incorporating memory into the computation. This architectural property makes the network a candidate for a strong uniform computational model.

Randomness is a basic characteristic of large distributed systems. It may result from the activity of the individual agents, from unpredictable changes in the communication pattern among the agents, or even just from the different update paces. Most previous work that examined stochasticity in networks, e.g., [vN56, Pip90, Ade78, Pip88, Pip89, DO77a, DO77b], studied only acyclic architectures of binary gates, while we study general architectures of analog components. Due to these two qualitative differences, our results are totally different from the previous ones and require alternative proof techniques.

Our particular stochastic model can be seen as a generalization of the von Neumann model of unreliable interconnections of components: “the basic component has a fixed probability ε for malfunction at any step [vN56].” In contrast to the von Neumann model, here because the neurons have continuous values, it is natural to allow for real probabilities in ε, rather than the value 1/2 only. Furthermore, because we consider recurrent computation it is natural to let ε not only be a constant, as in the von Neumann model, but also a function of the history and the neighboring neurons. The latter, referred to as the “Markovian model,” provides a useful model for biologically motivated stochastic computation.

The element of stochasticity, when joined with exact known parameters, has the potential to increase the computational power of the underlying deterministic process. We find that it indeed adds some power, but only in some cases. To state the results precisely we need to describe the model.

1.1. *Deterministic Analog Recurrent Networks*

Analog recurrent neural networks (ARNNs) are finite collections of neurons. The *activation value*, or *state*, of each neuron is updated at times $t = 1, 2, 3, \dots$, according to a function of the activations (x_j) and the inputs (u_j) at time $t-1$, and a set of real *weights* (a_{ij}, b_{ij}, c_i). The network consists of N neurons; the input arrives as a stream of letters, and each letter appears on M input channels; each neuron's state is updated by the equation

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}x_j(t) + \sum_{j=1}^{M} b_{ij}u_j(t) + c_i\right), \qquad i = 1, \dots, N, \tag{1}$$

where σ is a "sigmoid-like" function called the saturated-linear function:

$$\sigma(x) := \begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{if} \quad 0 \leqslant x \leqslant 1 \\ 1 & \text{if} \quad x > 1. \end{cases} \tag{1}$$

A subset of p out of the N neurons (the *output neurons*) is used to communicate the outputs of the network to the environment. Similarly to the input, the output is also a stream of letters, transferred on p channels. With this input–output convention, the evolution of the ARNN can be interpreted as a process of computation. This model of ARNN is "uniform" in the sense that the same structure is fixed for computing on inputs of all different lengths.

Previous studies [Sie99] show that the deterministic network is a *parametric model of computation*: altering its constitutive parameters allows the model to coincide with other models which are computationally and conceptually different. In particular, the computational power depends on the type of numbers utilized as weights. When the weights are integers, the network is a finite state machine only. When the weights are rational numbers, the network is equivalent (in power and computation time) to the Turing model ([SS91, SS95]). Finally, when the weights require infinite precision, the finite networks are proved to be stronger than the Turing model and to compute, under polynomial time, the class P/poly. (The class P/poly, defined in Section 2, includes all P and, moreover, some of EXP and even a few non-recursive functions, e.g., a unary encoding of the halting problem. However, this class of functions consists of a very small fraction of all binary functions.) An infinite hierarchy of computational classes was associated with networks having weights with increasing Kolmogorov complexity; P and P/poly were recognized as the two extremes of that hierarchy [BGS97].

1.2. *Main Results*

In the cases of real weights and integer weights, the incorporation of random coins does not change the computational power of the underlying deterministic network. It does increase the power, however, when the weights are rationals.

In order to characterize stochastic networks with rational weights, we formulate a new-result in complexity theory, namely that a probabilistic Turing machine which uses real probability coins computes the nonrecursive class BPP/log$*$ rather than the class BPP, which is computed by Turing machines that use rational probability coins. We then associate stochastic networks with probabilistic Turing machines, thus characterizing their computational power.

It is perhaps surprising that real probabilities strengthen the Turing machine, because the machine still reads only the binary values of the coin flips. However, a long sequence of coin flips allows indirect access to the real-valued probability, or more accurately, it facilitates its approximation with high probability. This is in contrast to the case of real weight networks, where access to the real values is direct and immediate. Thus, the resulting computation class (BPP/log$*$) is of intermediate computational power. It contains some nonrecursive functions but is strictly weaker than P/poly. Because real probabilities do not provide the same power as real weights, this work can be seen as suggesting a model of computation which is stronger than a Turing machine, but still is not as strong as real weight neural networks.

It was proven in [SS94] that, although real weight neural networks are defined with unbounded precision, they demonstrate the feature referred to as "linear precision suffices." That is, up to the qth step of the computation, only the first $O(q)$ bits, in both weights and activation values of the neurons, influence the result. This means that for time-bounded computation, only bounded precision is required. This property can be viewed as time-dependent resistance ("weak resistance") of the networks to noise and implementation error. Complementary to the feature of "linear precision suffices" for weights, we prove that for stochastic networks "logarithmic precision suffices" for the coins' probabilities; that is, for up to the qth step of the computation, only the first $O(\log q)$ bits in the probabilities of the neurons influence the result. We note that the same precision characterizes the quantum computer [BV93].

1.3. *Noisy Models*

The notion of stochastic networks is closely related to the concept of deterministic networks influenced by external noise. A recent series of work considered recurrent networks in which each neuron is affected by an additive noise [Cas96, OM98, MS99, SR98a]. This noise is characterized by a transition probability function $Q(x, A)$ describing the transit from the state x to the set of states A. If the update equation of the underlying deterministic network is of the form $x^+ = \psi(x, u)$, where $u \in \Sigma$ is the current input, then this system first moves from x to $\psi(x, u)$ and then is dispersed by the noise according to the transition probability function Q. The probability that it reaches a state in A is defined by $Q(\psi(x, u), A)$. Although it was assumed in [OM98, MS99] that the noise is additive and $Q(x, A)$ has a density $q(x, y)$ with respect to some fixed (i.e., Lebesgue) measure μ, in later work [SR98a, SR98b] these assumptions were relaxed to include much larger classes of noise. Typically, the resulting networks are not stronger than finite automata [Cas96, OM98, RGS99], and for many types of noise they compute the strict subset of the regular languages called the *definite languages* [MS99, SR98b, SR98a]: Let Σ be an arbitrary alphabet $L \subseteq \Sigma^*$ is called a definite language if for some integer r any two words coinciding on the last r symbols are either both in L or neither in L. The ability of a computational system to recognize only definite languages can be interpreted as saying that the system forgets all its input signals, except for the most recent ones.

The underlying mechanism which leads to the generation of definite languages was revealed in [SR98b]. This theory, which builds on [RR99, Paz71], introduces a general model of "Markov computational systems." These systems can be defined on *any arbitrary* state space, and their evolution is described by the *flow of their state distributions*. That is, if the distribution of initial states is μ_0, then the state distribution on the $(n+1)th$ computational step (after receiving the input string $w = w_0, ..., w_n$) is defined by $P_w \mu_0 = P_{w_n} \cdots P_{w_0} \mu_0$, where P_u is a Markov operator corresponding to input $u \in \Sigma$. Particular cases of Markov computational systems include Rabin's probabilistic automata with cut-point [Rab66], the probabilistic automata by Paz [Paz71], and the noisy analog neural network by Maass and Sontag [MS99] and Maass and Orponen [OM98]. Interestingly, Markov systems also include many diverse computational systems, such as analog dynamical systems and neural networks with an unbounded number of components, networks with non-fixed dimensions (e.g., "recruiting networks"), hybrid systems that combine discrete and continuous variables and time evolution, stochastic cellular automata, and stochastic coupled map lattices.

It is proved in [SR98b] that any Markov computational system which is weakly ergodic can recognize only the class of definite languages. This computational power is an *inherent property* of weakly ergodic systems and is independent of the specific details of the system, whether defined by finite or infinite dimension, discrete or continuous variables, finite or infinite alphabet, or stochasticity specified in terms of a Lebesgue-like measure. In addition, a stability theorem concerning language recognition under small perturbations is proved there for weakly ergodic computational systems. In [SR98a] the principle of weak ergodicity is applied for various systems which generate definite languages. This includes deterministic systems, many kinds of noisy systems where the noise can be a function of the input and the state of the system, aggregates of probabilistic discrete-time models, and probabilistic hybrid computational systems. In [RGS99] an underlying mechanism leading to the generation of regular languages is identified as an extension of [RR99] and [OM98].

The stochastic networks considered in this paper can be thought of as a special case of noisy networks where the noise is characterized by a discrete probability which is nonzero for two domain points only: it is $1-p$ in 0 and p for some nonzero value. When such stochasticity is applied to *some* of the neurons, it can only increase the computational power. Therefore, the term "noisy" is not a good characterization of this network, and we prefer the term "stochastic."

1.4. *Organization of the Paper*

This paper is organized as follows: Section 2 provides the required preliminaries of computational classes. Section 3 defines our stochastic networks, distinguishing them from a variety of stochastic models. Section 4 states the main results. Sections 5–7 include the proofs of the main theorems. We close with Section 8, restating our model as a network of stochastically unreliable (biologically motivated) neurons.

2. PRELIMINARIES: COMPUTATIONAL CLASSES

Let us briefly describe the computational classes relevant to this work.

2.1. *Probabilistic Turing Machines*

The basis of the operation of the probabilistic Turing machine, as well as of our stochastic neural networks, is the use of random coins. In contrast to the deterministic machine, which acts on every input in a specified manner and responds in one possible way, the probabilistic machine may produce different responses for the same input.

DEFINITION 2.1 ([BDG90, Vol. I]). A *probabilistic Turing machine* is a machine that computes as follows:

1. Every step of the computation can have two outcomes, one chosen with probability p and the other with probability $1-p$.
2. All computations on the same input require the same number of steps.
3. Every computation ends with *reject* or *accept*.

All possible computations of a probabilistic Turing machine can be described by a full (all leaves at the same depth) binary tree whose edges are directed from the root to the leaves. Each computation is a path from the root to a leaf, which represents the final decision, or equivalently, the classification of the input word by the associated computation. A coin, characterized by the parameter p, chooses one of the two children of a node. In the standard definition of probabilistic computation, p takes the value $\frac{1}{2}$.

A word ω is said to be accepted by a probabilistic Turing machine $\mathscr{M}$ if its acceptance probability is high (above $\frac{1}{2}$) or, equivalently, if its error probability $e_{\mathscr{M}}(\omega)$ is low. PP is the class of languages accepted by polynomial-time probabilistic Turing machines with $e_{\mathscr{M}} < \frac{1}{2}$. A weaker class defined by the same machine model is BPP, which stands for *bounded-error probabilistic polynomial time*. BPP is the class of languages recognized by polynomial-time probabilistic Turing machines whose error probability is bounded above by some positive constant $\varepsilon < \frac{1}{2}$. The latter class is recursive but it is unknown whether it is strictly stronger than P.

2.2. *Nonuniform Turing Machines*

Nonuniform complexity classes are based on the model of advice Turing machines [KL80]; these, in addition to their input, receive also another sequence that assists in the computation. For all possible inputs of the same length n, the machine receives the same advice sequence, but different advice is provided for input sequences of different lengths. When the different advice strings cannot be generated from a finite rule (e.g., Turing machine) the resulting computational classes are called *nonuniform*. The nonuniformity of the advice translates into noncomputability of the corresponding class. The length of the advice is bounded as a function of the input and can be used to quantify the amount of noncomputability.

Let Σ be an alphabet and \$ a distinguished symbol not in Σ; $\Sigma_\$$ denotes $\Sigma \cup \{\$\}$. We use homomorphisms h^* between monoids like $\Sigma_\$^*$ and Σ^* to encode words. Generally these homomorphisms are extensions of mappings h from $\Sigma_\$$ to Σ^*, inductively defined as follows: $h^*(\varepsilon) = \varepsilon$ and $h^*(a\omega) = h(a)\, h^*(\omega)$ for all $a \in \Sigma_\$$ and $\omega \in \Sigma_\$^*$. For example, when working with binary sequences, we usually encode "0" by "00," "1" by "11," and \$ by "01."

Let $\omega \in \Sigma^*$ and $v: \mathbb{N} \to \Sigma^*$. Define $\Sigma_v^* = \{\omega\$v(|\omega|) \mid \omega \in \Sigma^*\} \subset \Sigma_\$^*$; the suffix $v(|\omega|)$ is called the *advice* of the input ω. We next encode Σ_v^* back to Σ^* using a one-to-one homomorphism h^* as described above. We denote the resulting words by $\langle \omega, v(|\omega|) \rangle \in \Sigma^*$.

Definition 2.2 (Nonuniformity). Given a class of sets C and a class of bounding functions H, the class C/H is the class of all the sets A for which there exist a set $B \in C$ and a function $v \in H$ such that

$$\forall n \in \mathbb{N}, \quad \forall \omega \in \Sigma^n: \qquad \omega \in A \Leftrightarrow \langle \omega, v(n) \rangle \in B$$

Some frequent H's are the space classes poly and log.

We next concentrate on the special case of prefix nonuniform classes [BHM92]. In these classes, $v(n)$ must be useful for all strings of length up to n, not only those of length n. This is similar to the definitions of "strong" [Ko91] or "full" [BHM92] nonuniform classes. Furthermore, $v(n_1)$ is the prefix of $v(n_2)$ for all lengths $n_1 < n_2$.

DEFINITION 2.3 (Prefix Nonuniformity). Given a class of languages C and a class of bounding functions H, we say that $A \in \text{Pref–}C/H$ if and only if there is $B \in C$ and a prefix function $v \in H$ such that $\forall n \in \mathbb{N}$ and $\forall \omega \in \Sigma^n$:

1. $\omega \in A \Leftrightarrow \langle \omega, v(n) \rangle \in B$, and
2. $\forall k \geqslant n \ \langle \omega, v(k) \rangle \in B \Leftrightarrow \langle \omega, v(n) \rangle \in B$.

For the sake of brevity, we use the notation of C/H^* for the prefix advice class.

A special case is that of a Turing machine that receives polynomially long advice and computes in polynomial time. The class obtained in this fashion is called P/poly, and in this case P/poly = P/poly*. When exponential time and advice are allowed, *any* language on $\{0, 1\}$ is computable. Every such language can be recognized in the following form: just prepare a table of length 2^n whose entries are associated with all the binary sequences of length n, in the lexicographic order. In each entry (sequence) write the bit "1" if the sequence is in the language, and "0" if it is not. Concatenate all 2^n bits into a sequence and use it as the advice for inputs of length n. This sequence encodes all the required information for accepting or rejecting any input sequence of length n.

Recall the probabilistic class BPP from Subsection 2.1. We will later focus on the class BPP/log$*$. It is not hard to see that BPP/log$*$ is a strict subset of BPP/log . Any tally set is in P/1 (and thus is clearly also in P/log and BPP/log). Let S be a tally set, whose characteristic sequence is completely random (say, Kolmogorov random). It can not be in any class Recursive/log$*$ because there is not enough information in $O(\log n)$ bits to get the nth bit of S. As a special case S is not in BPP/log$*$.

3. STOCHASTIC NETWORKS

Four main questions arise to be addressed when considering stochastic networks: How do we model stochasticity? What type of random behavior (or errors) should be allowed? How much randomness can be handled by the model? Finally, stochastic networks are not guaranteed to generate the same response in different runs of a given input; thus, how do we define the output of the network for a given input?

Modeling Stochasticity. The first question, how to model stochasticity, was discussed by von Neumann [vN56] and quoted by Pippenger [Pip90]:

> *The simplest assumption concerning errors is this: With every basic organ is associated a positive number ε such that in any operation, the organ will fail to function correctly with the (precise) probability ε. This malfunctioning is assumed to occur statistically independently of the general state of the network and of the occurrence of other malfunctions.*

We first adopt von Neumann's statistical independence assumption. This assumption is also consistent with the works of Wiener [Wie49] and Shannon [Sha48], where the "noise"—which is the source of stochasticity—is modeled as a random process with known parameters. Note that in this model, the components are stochastic in precisely the amount ε and "are being relied upon to behave unreliably in this exact amount" [Pip90] (see also [DO77a, DO77b, Pip89]). Similarly, in our work we assume either full knowledge of ε or only knowledge of the first $O(\log T)$ bits of ε, where T is the computation time of the network. We show these two options to be equivalent. We then continue and expand to a Markovian model of stochasticity in which ε depends on the neighboring neurons and the recent history of the system.

Types of Randomness. As for the second question, regarding the type of randomness, we consider any type of random behavior that can be modeled by augmenting the underlying deterministic process, either with independent identically distributed binary sequences (i.i.d.) or with Markovian sequences. We then abandon the stochastic coin model and substitute it with asynchronicity of update and with various nondeterministic responses of the neurons themselves. This will be described in Section 8, where we discuss stochastically amnesic neurons (each neuron forgets its activation value with some probability; this forgetting is modeled as resetting the activation value to "0"), the persevering neurons, and also the effect of probabilistic changes in the interconnections between neurons.

Amount of Randomness. The next question we consider is the amount of stochasticity allowed in the model. Von Neumann assumed a *constant failure probability* ε in the various gates, independent of the size of the network. Furthermore, he allowed *all components* to behave randomly. Thus, larger networks suffer from more unreliability than do smaller ones. In contrast, many later models allowed ε to decrease with the size of the net (see discussion in [Pip90]), and others assumed the incorporation of fully deterministic/reliable components in critical parts of the network (see, e.g., [Kir70, MG62, Ort78, Uli74]). It is not hard to verify that when ε is constant and all neurons are unreliable, no function requiring non-constant deterministic time $T(n)$ is computable by the network. This obviates the use of long term memory, an otherwise attractive property of recurrent networks. We thus focus on networks which combine both reliable/deterministic neurons with unreliable/stochastic neurons characterized by fixed ε's (It is interesting to know about tradeoffs between error-rate, network size, and reliable computation time, but that is outside of our current scope.)

Defining the Output Response. As for the question of defining an output for the probabilistic process, we adopt the bounded error approach. Given $\varepsilon < \frac{1}{2}$, we only consider networks that yield a wrong output on at most a fraction ε of the possible computations.

3.1. *The Model*

The underlying deterministic network is as introduced in Subsection 1.1. The following definition endows the network with stochasticity.

DEFINITION 3.1. A stochastic network has additional input lines, called *stochastic lines*, that carry independent identically distributed (iid) binary sequences, one bit per line at each tick of the clock. The distributions may be different on the different lines. That is, for all time $t \geqslant 0$, the stochastic line l_i has the value 1 with probability p_i $(0 \leqslant p_i \leqslant 1)$, and 0 otherwise.

Equivalently, stochastic networks can be viewed as being composed of two types of components, analog deterministic neurons and binary probabilistic gates/coins. A probabilistic gate is a binary device which outputs "1" with probability $p \in [0, 1]$.

Yet another way to view the same model is to consider it as a network of neurons, part of which function deterministically while others have "well-described faults;" that is, their faulty behavior can be described by a neural circuit. More on this and on other equivalent models appears in Section 8.

We define the recognition of a language by a stochastic network using the bounded error model as in BPP. We assume that the number of steps in all computations on an input ω is exactly the same. The classification of an input ω in a computation run is the reject or accept decision at the end of that computation. The final decision on ω considers the fraction of reject and accept classifications of the various computations.

DEFINITION 3.2. Let $T\colon \mathbb{N} \to \mathbb{N}$ be a total function on natural numbers. We say that the language $L \subseteq \{0, 1\}^+$ is *ε-recognized in time T* by a stochastic net $\mathcal{N}$ if every $\omega \in \{0, 1\}^+$ is classified in time $T(|\omega|)$ by every computation path of $\mathcal{N}$ on ω, and the error probability in deciding ω relative to the language L is bounded: $e_{\mathcal{N}}(\omega) < \varepsilon < \frac{1}{2}$. An equivalent definition is that, for any given run on an input ω, if ω is not accepted in time $T(|\omega|)$ then it is rejected.

A relatively standard lemma shows that, for probabilistic models of computation, the error probability can be reduced to any desired value [Par93]. This indicates that the following complexity class is well-defined.

DEFINITION 3.3. S-net is the class of languages that are ε-recognized by stochastic networks for any $\varepsilon < \frac{1}{2}$.

4. THE MAIN RESULTS

Now that we have defined the model, we are ready to state the theorems about stochastic networks and compare them with deterministic networks. Proofs appear in the following sections.

4.1. *Integer Networks*

In the deterministic case, if the networks are restricted to integer weights, the neurons may assume only binary activations and the networks become computationally equivalent to finite automata. Similar behavior occurs for stochastic networks.

THEOREM 1. *The class* S-net$_Z$ *of languages that are* ε*-recognized by stochastic networks with integer weights is the set of regular languages.*

The case of integer weights is considered only for the sake of completeness. Rational and real stochastic networks are of greater interest.

4.2. *Rational Networks*

In deterministic computation, if the weights are rational numbers the network is equivalent in power to the Turing machine model. Two different I/O conventions are suggested in [SS95]; in the first, input lines and output neurons are used, and, in the second, the discrete input and output are encoded as the states of two pre-fixed neurons.

In Section 2.2 we introduced nonuniform computational classes; in particular, Definition 2.3 described the special case of prefix nonuniformity. We will next focus on the class BPP/log$*$. Because it includes nonrecursive functions, just like other nonuniform classes, the following theorem that states the equivalence between rational stochastic networks and the class BPP/log$*$ is of special interest.

THEOREM 2. *The class* S-net$_Q[p]$ *of languages* ε*-recognized by rational stochastic networks in polynomial time is equal to* BPP/log$*$.

TABLE I

The Computational Power of Recurrent Neutral Networks

Weights	Deterministic	Stochastic
$\mathbb{Z}$	regular	regular
$\mathbb{Q}$	P	BPP/log*
$\mathbb{R}$	P/poly	P/poly

COROLLARY 4.1 (of Both Theorem 2 and the Proof of Lemma 6.3). *Up to the Tth step of the computation, only the first $O(\log T(n))$ bits of the probabilities are significant in a probabilistic computation.*

Remark 4.2. As a special case of this theorem we note that if the probabilities are all rational then the resulting polynomial-time computational class is constrained to the recursive class BPP. Recall that BPP is recursive; it is included both in P/poly ([BDG90, p. 144, Cor. 6.3]) and in $\Sigma_2 \cap \Pi_2$ ([BDG90, p. 172, Theorem 8.6). It is still unknown whether the inclusion $\mathrm{P} \subseteq \mathrm{BPP}$ is strict or not.

4.3. *Real Networks*

Deterministic real networks compute the class P/poly in polynomial time [SS94]. The addition of stochasticity does not yield a further increase in the computational power.

THEOREM 3. *Denote by* $\mathrm{net}_R(T(n))$ *the class of languages recognized by real networks in time* $T(n)$, *and by* $\text{S-net}_R(T(n))$ *the class of languages* ε*-recognized by real stochastic networks in time* $T(n)$. *Then*

$$\mathrm{net}_R(T(n)) \subseteq \text{S-net}_R(T(n))$$

$$\text{S-net}_R(T(n)) \subseteq \mathrm{net}_R(n^2 + nT(n)).$$

The results for polynomial stochastic networks are summarized in Table I.

5. INTEGER STOCHASTIC NETWORKS

In this section we prove Theorem 1, which states the correspondence between probabilistic automata and integer stochastic networks. The classical definition of a probabilistic automaton (see [Paz71], for example) is as follows:

DEFINITION 5.1. A probabilistic automaton $\mathscr{A}$ is the 5-tuple $\mathscr{A} = (Q, \Sigma, p, q_0, F)$, where Q is a finite set of states, Σ is the finite input alphabet, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ are the accepting states. The probabilistic transition function $p(m, q, a)$, where $m, q \in Q$, and $a \in \Sigma$, specifies the probability of getting into state m from a state q with input symbol a. This transition function satisfies that, for all q and a,

$$\sum_{m \in Q} p(m, q, a) = 1.$$

We consider the double-transition probabilistic automaton which is a special case of probabilistic automata: the transition function may transfer each state-input pair into exactly two states. That is, for all $q \in Q$ and $a \in \Sigma$, there are exactly two states m_{1qa}, m_{2qa} such that $p(m_{1qa}, q, a), p(m_{2qa}, q, a) > 0$ and $p(m_{1qa}, q, a) + p(m_{2qa}, q, a) = 1$. It is easy to verify that this automaton is indeed equivalent to the general probabilistic automaton (up to a reasonable slowdown in the computation). The proof is left to the reader.

A double-transition probabilistic automaton can be viewed as a finite automaton with $|Q|\,|\Sigma|$ additional input lines of i.i.d. binary sequences; these lines imply the next choice of the transition. From the equivalence of deterministic integer networks and finite automata, we conclude the equivalence between probabilistic automata and integer stochastic networks.

Rabin showed that bounded error probabilistic automata are computationally equivalent to deterministic ones [Paz71, p. 160]. Thus, stochastic networks with integer weights are computationally equivalent to bounded error probabilistic automata, and S-net$_Z$ is the class of regular languages.

6. RATIONAL STOCHASTIC NETWORKS

This section is devoted to the proof of Theorem 2 which places rational stochastic networks in the hierarchy of super-Turing computation.

We use a generalization of the classical probabilistic Turing machine [BDG90] that substitutes the random coin of probability $\frac{1}{2}$ by a finite set of real probabilities. (A set of probabilities is required in order to describe neurons with different stochasticity.)

DEFINITION 6.1. Let $S = \{p_1, p_2, ..., p_s\}$ be a finite set of probabilities. A *Probabilistic Turing machine over the set S* is a nondeterministic machine that computes as follows:

1. Every step of the computation can have two outcomes, one chosen with probability p and the other with probability $1 - p$.

2. All computations on the same input require the same number of steps.
3. Every computation ends with *reject* or *accept*.

We denote by BP[S, T] the class of languages recognized in time T by bounded error probabilistic Turing machines with probabilities over S. We use the shorthand notation BPP[S] = BP[S, poly]. Similarly, we denote S-net$_Q$[S, T] as the class of languages recognized by rational stochastic nets in time T, with probabilities from S.

LEMMA 6.2. *Let S be a finite set of probabilities, then BP*[S, T] *and* S-net$_Q$[S, T] *are polynomially time related.*

Proof. (1) BP[S, T] $\subseteq$ S-net$_Q$[S, $O(T)$]. Let $\mathcal{M}$ be a probabilistic Turing machine over the set S that computes in time T. We simulate $\mathcal{M}$ by a rational stochastic network $\mathcal{N}$ having stochastic streams l_i of probabilities $p_i \in S$. Consider the program

```
**Repeat**
  **If** ($l_i = 0$) **then** NextStep(0, cur-state, cur-tapes)
    **else** NextStep(1, cur-state, cur-tapes)
**Until** (final state),
```

where NextStep is a procedure that, given the current state of the Turing machine, the current tapes, and which of the two random choices is made, changes deterministically to the next configuration of the machine. This program can be compiled into a network that computes the same function, having no more than a linear slowdown [Sie95].

(2) S-net$_Q$[S, T] $\subseteq$ BP[S, poly(T)]. It is easy to verify that if a rational stochastic network $\mathcal{N}$ has s i.i.d. input channels, then it can be simulated by a probabilistic Turing machine over the same s probabilities. ∎

Next, we differentiate the case in which all probabilities of the set S are rational numbers from the case where S contains at least one real element.

6.1. *Rational Set of Choices*

Consider probabilistic Turing machines with probabilities over the set S, where S consists of rational probabilities only. Zachos showed that, in the error bounded model, if the transition function decides its next state uniformly over k choices (k is finite but can be larger than 2) this model is polynomially equivalent to the classical probabilistic Turing machine with $k = 2$ [Zac82]. When the probabilities are rational we can substitute them all by a common divisor which is written as $1/k'$ for an integer k'.

This process increases the number of uniform choices and implies polynomial equivalence between probabilistic Turing machines with one fair coin and probabilistic machines over a set S. We conclude Remark 4.2 which states the computational equality between the class S-net$_Q$[poly] and the class BPP. Thus, rational stochasticity adds power to deterministic rational networks if and only if the class BPP is strictly stronger than P. Note that S-net$_Q$[poly] must be computationally strictly included in net$_R$[poly], because BPP is included in P/poly ([BDG90, p. 144, Cor 6.3]).

6.2. *Real Set of Choices*

Lemma 6.2 relates probabilistic Turing machines to stochastic neural networks. The lemma below completes the proof of Theorem 2 by showing the equivalence between real probabilities and log prefix advice in the probabilistic Turing model. We define $\mathrm{BP}_R(T)=\bigcup_{p\in[0,1]}\mathrm{BPP}[\{p\},T]$ as the class of languages recognized by probabilistic bounded error Turing machines that use coins of real probability and compute in time T. $\mathrm{BP}_Q(T)/\log*$ is similarly defined, with the addition of prefix advice. Note that $\mathrm{BP}_Q(\mathrm{poly})/\log* = \mathrm{BPP}/\log*$.

LEMMA 6.3. *The classes* $\mathrm{BP}_R(T)$ *and* $\mathrm{BP}_Q(T)/\log*$ *are polynomially related.*

Proof. (1) $\mathrm{BP}_R(T)\subseteq\mathrm{BP}_Q(O(T\log T))/\log*$. Let $\mathscr{M}$ be a probabilistic Turing machine over the probability $p\in[0,1]$ that ε-recognizes the language L in time $T(n)$. We show a probabilistic Turing machine $\mathscr{M}'$ having a fair coin, which, upon receiving prefix advice of length $\log(T(n))$, ε'-recognizes L in time $O(T(n)\log(T(n)))$. We now describe the algorithm for the simulation and then bound its error probability:

> *Let p' be the rational number which is obtained from the $\log(T(n))$ most significant bits of the binary expansion of p. The advice of $\mathscr{M}'$ consists of the bits of p' starting from the most significant bits. One coin flip by $\mathscr{M}$ can be simulated by a binary conjecture of $\mathscr{M}'$, which is based on $\log(T(n))$ coin flips of its fair coin. $\mathscr{M}'$ tosses $\log(T(n))$ times and compares the resulting guessed string with the advise to make a binary conjecture. If the guessed string precedes the advice in the lexicographic order, $\mathscr{M}'$ conjectures* "0"; *otherwise, $\mathscr{M}'$ conjectures* "1."

The error probability of $\mathscr{M}'$ is the probability that it generates a sequence of conjectures which would yield a wrong decision. Denote by $r=r_1r_2\cdots r_T$, $r_i\in\{0,1\}$ a sequence of T binary bits. Let

$$\Pr_p(r)=p^{\Sigma_{k=1}^{T} r_k}(1-p)^{T-\Sigma_{k=1}^{T} r_k}$$

be the probability that r is the sequence of random choices generated by the coin of $\mathscr{M}$, and let

$$\Pr_{p'}(r) = p'^{\sum_{k=1}^{T} r_k}(1-p')^{T-\sum_{k=1}^{T} r_k}$$

be the probability that $\mathscr{M}'$ generates this sequence of binary choices during its computation.

For all r, if $p' < p$, we denote $q = 1-p$ and $q' = 1-p'$, so that $q' > q$, and calculate a connection between the two probabilities:

$$\Pr_{p'}(r) = \frac{\Pr_{p'}(r)}{\Pr_{p}(r)} \Pr_{p}(r) \leqslant \left(\frac{q'}{q}\right)^{T} \Pr_{p}(r) \tag{3}$$

We next substitute in Eq. (3)

$$\left(\frac{q'}{q}\right)^{T} = \left(1 + \frac{q'-q}{q}\right)^{T},$$

and use the approximation of small x,

$$1 + x \approx e^{x}, \tag{4}$$

to obtain the formula

$$\Pr_{p'}(r) \leqslant e^{((q'-q)/q)\,T} \Pr_{p}(r). \tag{5}$$

Denote by B the set of $T(n)$-long binary sequences which are "bad," i.e., are misleading conjectures. The error probability of $\mathscr{M}'$ is $\Pr_{p'}(r \in B)$, which is estimated as

$$\Pr_{p'}\,(r \in B) \leqslant e^{((q'-q)/q)\,T} \Pr_{p}\,(r \in B).$$

If p' approximates p with first $\log(T)$ bits, then $q' - q \leqslant a/T$ and

$$\Pr_{p'}\,(r \in B) \leqslant e^{a/q} \Pr_{p}\,(r \in B).$$

The error probability of $\mathscr{M}'$ is thus bounded by the constant $\varepsilon' = e^{a/q}\varepsilon$.

(2) $\mathrm{BP}_{Q}(T)/\log* \subseteq \mathrm{BP}_{R}(O(T^2))$. Given a probabilistic Turing machine $\mathscr{M}$, having a fair coin and a logarithmically long prefix advice A that ε-recognizes a language L in time $T(n)$, we describe a probabilistic Turing machine $\mathscr{M}'$ with an associated real probability p that ε'-recognizes L in time $O(T^2(n))$.

The probability p is constructed as follows: The binary expansion of p starts with ".01," i.e., $\frac{1}{4} \leqslant p \leqslant \frac{1}{2}$, the following bits are the advise of $\mathscr{M}$.

$\mathscr{M}'$ computes in two phases:

Phase I—*Preprocessing. $\mathcal{M}'$ estimates the advise sequence A of $\mathcal{M}$ by tossing its unfair coin $z = cT^2(n)$, $c \geqslant 1$ times.*

Phase II—*Simulating the computation of $\mathcal{M}$. $\mathcal{M}'$ simulates each flip of the fair coin of $\mathcal{M}$ by up to $2T(n)$ tosses using the following algorithm*:

(a) *Toss the unfair coin twice.*

(b) *If the results are* "01" *conjecture* "0"; *if they are* "10" *conjecture* "1".

(c) *If the results are either* "00" *or* "11" *and* (a) *was called less than $T(n)$, go to* (a).

(d) *Here* (a) *was called $T(n)$ times and the decision was not yet made*: *conjecture* "0."

We first bound the error of the estimated advice. Let #1 be the number of 1's found in z flips, and define the estimation

$$\tilde{p} = \frac{\#1}{z},$$

which will be used as an estimation of the advice. The Chebyshev formula states that for any random variable x with expectation μ and variance v, and $\forall \Delta > 0$, $\Pr(|x - \mu|) > \Delta) \leqslant v/\Delta^2$. Here x is the sum of i.i.d. random variables. The expectation of such z independent Bernoulli trials is $\mu = zp$, and the variance is $v = zp(1-p)$. We conclude that, for all $\Delta > 0$,

$$\Pr(|z\tilde{p} - zp| > \Delta) \leqslant \frac{zp(1-p)}{\Delta^2}.$$

Because $p(1-p) \leqslant \frac{1}{4}$, by choosing $\Delta = \sqrt{cz}$, for a constant c ($c \geqslant 1$) we get

$$\Pr\left(|\tilde{p} - p| > \sqrt{\frac{c}{z}}\right) \leqslant \frac{1}{4c}.$$

Thus, if in the first phase $\mathcal{M}'$ tosses its coin $z = cT^2(n)$ times, then the advice is reconstructed with logarithmically many bits and with an error probability ε_1 bounded by $1/4c$ (the first two bits "01" are omitted from the guessed advice).

We next prove the correctness of phase II, which is based on von Neumann's technique [vN51]. We compute the probability of $\mathcal{M}'$ to guess a bad sequence of coin flips. As above, we denote by r the sequence of binary conjectures of length $T(n)$ generated by $\mathcal{M}'$ during the algorithm and by B the set of misleading guesses. As the error of $\mathcal{M}$ is bounded by ε and $\mathcal{M}$ uses a fair coin, the cardinality of B is bounded by $2^{T(n)}\varepsilon$. We conclude that

$$\Pr(r \in B) \leqslant |B| \max_{b \in B} \Pr(r = b) \leqslant 2^T \varepsilon \max_{b \in B} \Pr(r = b) \tag{6}$$

The string with maximum probability is 0^T. This probability can be estimated as follows:

- The probability of getting the values "00" or "11" in two successive coin flips is $p' = p^2 + (1-p)^2$. Thus, the probability of ending a coin flip simulation in step (d) of the algorithm is bounded by $p'' = p'^{T(n)}$. Since $\frac{1}{4} \leqslant p \leqslant \frac{1}{2}$, we conclude that $p' \leqslant \frac{5}{8}$ and $p'' \leqslant \frac{5}{8}^{T(n)}$.
- The probability of ending one coin flip simulation with the conjecture "0" is $\frac{1}{2}(1-p'') + p'' = \frac{1}{2} + (p''/2)$.

We thus conclude

$$\max_b \Pr(r = b) \leqslant \Pr(r = 0^{T(n)}) = \left(\frac{1}{2} + \frac{p''}{2}\right)^{T(n)} \tag{7}$$

and we can substitute Eq. (7) in Eq. (6) to get

$$\Pr(r \in B) \leqslant 2^{T(n)} \varepsilon \left(\frac{1}{2} + \frac{p''}{2}\right)^{T(n)} \leqslant \varepsilon (1 + p'')^{T(n)} \approx \varepsilon e^{T(n)(5/8)^{T(n)}}, \tag{8}$$

which is bounded by a small error ε_2 for small enough ε. The error probability ε' of $\mathscr{M}'$ is bounded by

$$\Pr(\text{"wrong advice sequence"}) + \Pr(\text{"bad guess sequence"}) \leqslant \varepsilon_1 + \varepsilon_2,$$

which is also bounded by $\frac{1}{2}$. ∎

Remark 6.4. So far we have discussed stochastic networks defined by adding choices to deterministic networks. We can similarly define the stochastic nondeterministic network by adding choices to nondeterministic networks. When weights are rationals, the latter class is similar to the framework of interactive proof systems [BDG90, Vol. I, Chapt. 11].

7. REAL STOCHASTIC NETWORKS

In this section we prove Theorem 3 and show that stochasticity does not add power to real deterministic networks. It is trivial to show that stochasticity does not decrease the power of the model; we thus focus on the other direction and prove that S-net$_R$[poly] $\subseteq$ Net$_R$[poly].

We prove this inclusion in two steps. Given a real stochastic network $\mathscr{N}$ that ε-recognizes a language L, the first step describes a nonuniform family $\mathscr{F}$ of feedforward networks that recognizes L; this creates only a constant slowdown in the computation. The second step describes a deterministic recurrent network $\mathscr{M}$ that simulates the family $\mathscr{F}$, with a polynomial slowdown of $n^2+nT(n)$. (We could skip the first step with a more elaborate counting argument in the second step, but we prefer this method for its simplicity of representation.)

LEMMA 7.1 (*Step* 1) *Let* $0<\varepsilon<\frac{1}{2}$, *and let* L *be a language that is* ε-*recognized by a real stochastic network* $\mathcal{N}$ *of size* N *in time* T. *Then,* L *can also be recognized by a family of nonuniform feedforward real networks* $\{\mathcal{F}_n\}_{i=1}^{\infty}$ *of depth* $T(n)+1$ *and size* $cNT(n)+1$, *where*

$$c=\left\lceil \frac{8\varepsilon \ln 2}{(1-2\varepsilon)^2} \right\rceil.$$

Proof. The technique used in this proof is similar to the one used in the proof that BPP $\subseteq$ P/poly [BDG90].

Let $\mathcal{N}$ be a real network that ε-recognizes a language L as above. We show the existence of a family of networks that recognizes L. Let r be the number of probabilistic gates g_k, $1\leqslant k\leqslant r$; each outputs "1" with probability p_k. For a given input of length n, by unfolding the network to $T(n)$ layers, each a copy of $\mathcal{N}$, we get a feedforward network with $\tilde{r}_n=T(n)\,r$ probabilistic gates. Denote this feedforward stochastic net by $\tilde{\mathcal{N}}_n$.

We pick a string

$$\rho^{i,n}=\rho_1^i\rho_2^i\cdots\rho_{\tilde{r}_n}^i\in\{0,1\}^{\tilde{r}_n}$$

at random, with probability $p_{(j \bmod r)}$ that ρ_j^i is "1." Let $\tilde{\mathcal{N}}_n\{\rho^{i,n}\}$ be a deterministic feedforward net similar to $\tilde{\mathcal{N}}_n$, but with the string $\rho^{i,n}$ substituting the probabilistic gates (i.e., ρ_j^i substitutes $g_{j \bmod r}$ in level $(j \operatorname{div} r)$. We now pick cn such strings

$$\rho[n]=(\rho^{1,n},\rho^{2,n},...,\rho^{cn,n})$$

at random. The feedforward net $\mathcal{F}_n$ consists of the cn subnetworks $\tilde{\mathcal{N}}_n\{\rho^{i,n}\}$ $(i=1\cdots cn)$ and one "majority gate" in the final level. The majority gate takes the output of the cn subnetworks as its input. That is, for each n, the network $\mathcal{F}_n$ computes the majority over cn random runs of the stochastic net $\mathcal{N}$.

We next compute the probability that $\mathcal{F}_n$ outputs incorrectly on an input ω of length n. By the definition of $\mathcal{N}$, each $\tilde{\mathcal{N}}_n\{\rho^{i,n}\}$ has the probability ε of being wrong. Thus, picking $\rho[n]$ at random, the probability of $\mathcal{F}_n$ to fail is bounded by $B(cn/2, cn, \varepsilon)$; this is the probability of being wrong in at least $cn/2$ out of cn independent Bernoulli trials, each having the failure probability ε.

We use the bound $B(cn/2, cn, \varepsilon) \leqslant (4\varepsilon(1-\varepsilon))^{cn/2}$ from [Par93], and choose $c = \lceil (8\varepsilon \ln 2)/(1-2\varepsilon)^2 \rceil$ to get

$$B\left(\frac{cn}{2}, cn, \varepsilon\right) < 2^{-n}.$$

This is a bound on the error probability for any input of length n. Thus, the sum of failures for all the inputs of length n is less than 1. Hence, there must be at least one choice of random strings $\rho[n]$ that makes $\mathscr{F}_n$ correctly recognize any input of length n. ▮

The above lemma of the two-step algorithm introduces a family $\mathscr{F}$ of deterministic feedforward networks that decides L. This specially structured family will be shown in the next step to be included in P/poly. More specifically, in the following lemma we complete the proof of Theorem 3 with a construction of a real deterministic network $\mathscr{M}$ that simulates $\mathscr{F}$.

Lemma 7.2 (*Step* 2). *Any language that is recognized by the real family* $\{\mathscr{F}_n\}_{n=1}^{\infty}$ *described above can also be recognized by a real deterministic recurrent network* $\mathscr{M}$. *Furthermore, an input of length n that was recognized by the network* $\mathscr{F}_n$ *having depth* $T(n)$ *can be recognized by* $\mathscr{M}$ *in time* $O(n^2 + nT(n))$.

Proof. We remind the reader that the whole family $\mathscr{F}$ was constructed from a single recurrent neural network; call it $\mathscr{N}$. Each family member $\mathscr{F}_n$ can thus be described by the tuple

$$(\mathscr{N}, n, \rho[n])$$

where $\mathscr{N}$ is the underlying deterministic recurrent network, n is the index of the network, and $\rho[n] \in \{0, 1\}^{r'_n cn}$.

Let $\tilde{\mathscr{N}}$ be any binary encoding of $\mathscr{N}$ and $\tilde{\rho}$ be the infinite string

$$\tilde{\rho} = \rho[1] \quad 2 \quad \rho[2] \quad 2 \quad \rho[3] \quad 2 \cdots.$$

Let $\alpha = \alpha_1 \alpha_2 \cdots \in \{0, 1, 2\}^{\#}$ and denote by $\alpha|_6$ the value $\sum_{i=1}^{|\alpha|} ((2\alpha_i + 1)/6i)$. This encoding is Cantor-like, and a network can read weights of this form letter-by-letter in $O(1)$ time each (see [SS95]). We next construct the recurrent network $\mathscr{M}$ that has the weights $\tilde{\mathscr{N}}|_6$ and $\tilde{\rho}|_6$ and recognizes the language L. $\mathscr{M}$ operates as follows:

1. $\mathscr{M}$ reads the input ω and measures its length.

2. $\mathscr{M}$ retrieves the encoding $\rho[n]$ from the constant $\tilde{\rho}|_6$. (This takes $\sum_{j \leqslant n} |\rho[j]| \leqslant O(n^2)$ as proven in [SS94]).

3. $\mathscr{M}$ executes the code:

```
Func Net ($\omega$, $\rho[n]$, $n$, $\tilde{\mathscr{N}}$);
Var  Yes, No, z: Counter,
     A: Boolean,
     $\rho$: Real;
Begin
  $z = 1$;
  Repeat
    $\rho \leftarrow$ retrieve($\tilde{\rho}$, $i$, $n$)  %retrieving $\rho^{i,n}$
    $A \leftarrow$ simulate($\tilde{\mathscr{N}}$, $\rho$, $\omega$)
    If $A = 1$ then Increment(Yes) else Increment(No)
    Increment($z$)
  Until ($z > cn$)
  Net $\leftarrow$ Return(Yes > No)
End
```

We know that the command "simulate" is feasible from the constructive Turing machine simulation in [SS95]. Furthermore, it was shown in [Sie95] how to construct a net from this type of high-level language. This program, as well as the associated network, takes $nT(n)$ steps. Thus $\mathscr{M}$ fulfills the requirements of Lemma 7.2 and Theorem 3 is proved. ∎

8. UNRELIABLE NETWORKS

In this section we provide a different formulation of stochastic networks. Our von-Neumann-like modeling captures many types of random behavior in networks. It can describe the probabilistic "amnesic" (i.e., forgetting) neuron,

$$x_i^+ = \begin{cases} \text{regular update} & \text{with probability} \quad 1 - p_{\text{i}} \\ 0 & \text{with probability} \quad p_{\text{i}}, \end{cases} \tag{9}$$

as well as the probabilistic "persevering" (i.e., frozen) neuron,

$$x_i^+ = \begin{cases} \text{regular update} & \text{with probability} \quad 1 - p_{\text{i}} \\ x_i & \text{with probability} \quad p_{\text{i}}, \end{cases} \tag{10}$$

and the probabilistic "disconnecting" neuron x_i, which is defined by the update equation

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} \tilde{a}_{ij} x_j(t) + \sum_{j=1}^{M} b_{ij} u_j(t) + c_i\right),$$

where

$$\tilde{a}_{ij}^k = \begin{cases} a_{ij}^k & \text{with probability} \quad 1 - p_{ij} \\ 0 & \text{with probability} \quad p_{ij}. \end{cases} \tag{11}$$

We generally allow not only for these three types of errors but for more general faults. Each stochastic neuron computes one of d functions which may be more complicated than the above examples.

DEFINITION 8.1. A neuron g is said to have a *well-described fault* if it computes d different functions f_i each with a probability p_{gi} $(\sum_{i=1}^{d} p_{gi} = 1)$ such that all the f_i's are deterministically computable by a net in constant time.

It is not difficult to verify the following:

LEMMA 8.2. *A stochastic network that consists of both well-described faulty and deterministic neurons can be described by our stochastic modeling.*

The proof is left to the reader.

Remark 8.3. Note that if all neurons are stochastic ("catastrophic nets") the stochasticity can no longer be controlled, and the networks cannot compute more than definite languages, as discussed in Subsection 1.3. Introducing a varying error rate or a nonuniform architecture allows one to overcome the catastrophe.

Similarly, we can define the "Markovian" model of unreliability: unlike the model of independent erroneous neurons, let us consider devices whose unreliable behavior depends on the last c choices of all devices and the last c global states in the network. This better models biological phenomena such as dying neurons, toxification, and the Korsakoff syndrome [SF99]. Note that this model is not strictly Markovian because transitions do not depend only on the global states but on the choices as well.

DEFINITION 8.4. Let g be a well-described faulty neuron with d choices. Let $x(i) \in [0, 1]^N$ be the activation vector in time i, and let the vector

$$F_{c,t} = (x(t-c), ..., x(t-1)) \in [0, 1]^{cN}$$

represent the activation values of the neurons in the previous c steps. Similarly, let $j(i) \in \{1, 2, ..., d\}^N$ be the vector of choices made by all neurons in time i, and let the vector

$$J_{c,t} = (j(t-c), j(t-c+1), ..., j(t-1)) \in \{1, 2, ..., d\}^{cN}$$

represent the choices made by all neurons in the previous c steps. Observe that only $x(t-c)$ in $F_{c,t}$ is necessary: the rest can be computed from $x(t-c)$ and the choices $J_{c,t}$. We use this redundancy for simplicity of presentation.

A *c-Markov network* is a network with some unreliable neurons, for which the probabilities p_{gi} are functions of $J_{c,t}$ and $F_{c,t}$; there exist dN stochastic subnetworks that, upon receiving $p_{gi}(t)$ as input, output "1" with this probability.

We state without proof:

THEOREM 4. *For any well-described default and any constant $c \geqslant 1$, c-Markovian networks are computational equivalent to networks of independent unreliability.*

One final equivalent model that we note is the *asynchronous model.* To characterize the behavior of the asynchronous neural networks, we adapt the classical assumption of asynchronous distributed systems: no two neurons ever update simultaneously. An *asynchronous network* is a network with an additional N-level probabilistic gate, g, where level l_i appears with probability p_i ($\sum_i p_i = 1$). At each time t, only processor $g(t)$ updates, and the output is interpreted probabilistically.

ACKNOWLEDGMENTS

I thank Joe Kilian, Ricard Gavaldà, Michael Saks, Bill Horne, Seppo Törmä, and Felix Costa for the many helpful comments. This work was funded by the Israeli Ministry of Art and Sciences, the Binational U.S./Israel Foundation, the fund for promotion of research at the Technion, and the VPR fund at the Technion.

REFERENCES

[Ade78] L. Adelman, Two theorems on random polynomial time, *in* "Proc. IEEE Foundations of Computer Science," Vol. 19, pp. 75–83, IEEE, New York, 1978.

[BDG90] J. L. Balcázar, J. Díaz, and J. Gabarró, "Structural Complexity," Vols. I and II, EATCS Monographs, Springer-Verlag, Berlin, 1988–1990.

[BGS97] J. L. Balcázar, R. Gavaldà, and H. T. Siegelmann, Computational power of neural networks: A characterization in terms of kolmogorov complexity, *IEEE Trans. Inform. Theory* **43**, No. 4 (1997), 1175–1183.

[BHM92] J. L. Balcázar, M. Hermo, and E. Mayordomo, Characterizations of logarithmic advice complexity classes, *IFIP Trans. on Information Processing* **1** (1992), 315–321.

[BV93] E. Bernstein and U. Vazirani, Quantum complexity theory, *in* "Proc. 25th ACM Symposium on the Theory of Computing," pp. 11–20, 1993.

[Cas96] M. Casey, The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction, *Neural Comput.* **8** (1996), 1135–1178.

[DO77a] R. L. Dobrushin and S. I. Ortyukov, Lower bound for the redundancy of self-correcting arrangement of unreliable functional elements, *Problems Info. Transmission* **13** (1977), 59–65.

[DO77b] R. L. Dobrushin and S. I. Ortyukov, Upper bound for the redundancy of self-correcting arrangement of unreliable functional elements, *Problems Info. Transmission* **13** (1977), 346–353.

[Kir70] G. I. Kirienko, Sintez samokottektiruyshchikhsya skhem iz funktsionalnykh elementov dlya aluchava tastushchego chisla oshibok v skheme, *Diskret. Anal.* **16** (1970), 38–43.

[KL80] R. M. Karp and R. J. Lipton, Some connections between uniform and nonuniform complexity classes, *in* "Proc. 12th ACM Symposium on the Theory of Computing," pp. 302–309, ACM, New York, 1980.

[Ko91] K. Ko, "Complexity Theory of Real Functions," Birkhäuser, Boston, 1991.

[MG62] A. A. Muchnik and S. G. Gindikin, The completeness of a system made up of non-reliable elements realizing a function of algebraic logic, *Soviet Phys. Dokl.* **7** (1962), 477–479.

[MS99] W. Maass and E. Sontag, Analog neural nets with gaussian or other common noise distribution cannot recognize arbitrary regular languages, *Neural Comput.* **11** (1999), 771–782.

[OM98] P. Orponen and W. Maass, On the effect of analog noise on discrete time analog computations, *Neural Comput.* **10** (1998), 1071–1095.

[Ort78] S. I. Ortyukov, Synthesis of asymptotically nonredundant self-correcting arrangements of unreliable functional elements, *Problems Inform. Transmission* **13** (1978), 247–251.

[Par93] I. Parberry, The computational and learning complexity of neural networks, manuscript, 1993.

[Paz71] A. Paz, "Introduction to Probabilistic Automata," Academic Press, New York, 1971.

[Pip88] N. Pippenger, Reliable computation by formulae in the presence of noise, *IEEE Trans. Inform. Theory* **34** (1988), 194–197.

[Pip89] N. Pippenger, Invariance of complexity measure of networks with unreliable gates, *J. Assoc. Comput. Mach.* **36** (1989), 531–539.

[Pip90] N. Pippenger, Developments in the synthesis of reliable organisms from unreliable components, *in* "Proc. Symposia in Pure Mathematics," Vol. 5, pp. 311–324, 1990.

[Rab66] M. Rabin, Lectures on classical and probabilistic automata, *in* "Automata Theory" (E. R. Caianiello, Ed.), Academic Press, London, 1966.

[RGS99] A. Roiterstein, L. Gurvits, and H. Siegelmann, "On Markov Computational Systems," Technical report, Technion, 1999.

[RR99] M. Rabin and A. Roitershtein, On Markov computational systems, *Ann. Math. Statist.* **41** (1999), 539–550.

[Sha48] C. E. Shannon, A mathematical theory of communication, *Bell System Tech J.* **1948** (1948), 379–423, 623–656.

[Sie95] H. T. Siegelmann, On nil: The software constructor of neural networks, *Parallel Process. Lett.* (1995), to appear; previous version appeared *in* "Proc. 12th AAAI," pp. 887–882, Seattle, Aug. 1994.

[Sie99] H. T. Siegelmann, "Neural Networks and Analog Computation: Beyond the Turing Limit," Birkhäuser, Boston, MA, 1999.

[SR98b] H. Siegelmann and A. Roitershtein, "On Weakly Ergodic Computational Systems," Technical Report, Technion, 1998.

[SR00] H. Siegelmann and A. Roitershtein, On definite languages and noisy computational systems, *Discrete Appl. Math.*, to appear, 2000.

[SS91] H. T. Siegelmann and E. D. Sontag, Turing computability with neural nets, *Appl. Math. Lett.* **4**, No. 6 (1991), 77–80.

[SS94] H. T. Siegelmann and E. D. Sontag, Analog computation via neural networks, *Theoret. Comput. Sci.* **131** (1994), 331–360.

[SS95] H. T. Siegelmann and E. D. Sontag, On computational power of neural networks, *J. Computer System Sci.* **50**, No. 1 (1995), 132–150.

[Uli74] D. Ulig, On the synthesis of self-correcting schemes from functional elements with a small number of reliable elements, *Math. Notes Acad. Sci. USSR* **15** (1974), 558–562.

[vN51] J. von Neumann, "Various Techniques Used in Connection with Random Digits" (Notes by G.E. Forsythe, National Bureau of Standards), in Applied Math Series, Vol. 12, pp. 36–38, Pergamon, New York, 1951.

[vN56] J. von Neumann, Probabilistic, logics and the synthesis of reliable organisms from unreliable components, *in* "Automata Studies" (C. E. Shannon and J. McCarth, Eds.), Princeton Univ. Press, Princeton, NJ, 1956.

[Wie49] N. Wiener, "Extrapolation, Interpolation, and Smoothing of Stationary Time Series," MIT Press, Cambridge, MA, 1949.

[Zac82] S. Zachos, Robustness of probabilistic computational complexity classes under definitional perturbations, *Inform. and Control* **54** (1982), 143–154.