ON THE COMPUTATIONAL POWER OF FAULTY AND ASYNCHRONOUS NEURAL NETWORKS

Hava T. Siegelmann
Department of Computer Science
Bar-Ilan University, Ramat-Gan 52900, Israel
E-mail: hava@bimacs.cs.biu.ac.il

Abstract

This paper deals with finite size recurrent neural networks which consist of general (possibly with cycles) interconnections of evolving processors. Each neuron may assume real activation value. We provide the first rigorous foundations for *recurrent* networks which are built of unreliable *analog* devices and present asynchronicity in their updates.

The first model considered incorporates unreliable devices (either neurons or the connections between them) which assume fixed error probabilities, independent of the history and the global state of the network. The model corresponds to the random-noise philosophy of Shannon. Another model allows the error probabilities to depend both on the global state and the history. Next, we change the various faulty nets to update in a total asynchronity. We prove all the above models to be computationally equivalent and we express their power. In particular, we see that for some constrained models of networks, the random behavior adds nonunifromity to the computation. However, the general simple model of networks is robust to probabilistic failures and asynchronous behavior.

Nondeterministic unreliable nets are defined, and we show that in the faulty model the equality P = NP holds.

1 Introduction

"Neural networks" have attracted much attention lately as models of analog computation. Such nets consist of a *finite* number of simple processors, each of which computes a scalar —*real-valued, not binary*— function of an integrated input. This scalar function, or "activation," is meant to reflect the graded response of biological neurons to the net sum of excitatory and inhibitory inputs affecting them. The existence of feedback loops in the interconnection graph gives rise to a dynamical system.

We introduce neural network models in which neurons exhibit random behavior. Nondeterminism in this sense is a basic characteristic of large distributed systems consisting of many agents, and may occur both in the activity of an agent or during the communication among different agents. Another source of nondeterminism is the different update pace of the various neurons; that is the asynchronous rule of update. Such a model arises naturally both in the application of artificial neural networks to multiprocessor systems and in the study of biological models.

Previous models of faulty devices in distributed systems were studied long ago, see [vN56, Pip90] and the many references there. Von Neumann cited papers by Kleene, McCulloch an Pitts, Shannon, Szilard, and Turing, and emphasized the connection among mathematical, physical, and biological approaches of unreliable networks. Bounds on networks of faulty components were discussed in [Ade78, Pip88, Pip89, RO77a, RO77b]. All the studied networks, however, were acyclic and their probabilistic devices assumed binary values only.

In contradiction to previous work, we consider networks in which each neuron has an activation value in the continuous [0,1] range, that is, the neurons apply some kind of continuous sigmoidal transfer function on their net values rather than a hard threshold function. This model of neural networks is more appropriate than the discrete one and is motivated by the simple biological fact that neurons output voltages which are clearly continous. The continous-type model has recently received another push from the point of view of computational power. It was proved in [SS91] that the computational power of such nets consisting of a finite number of neurons is universal.

Thus, we provide the first foundations dealing with *analog unreliable devices* (rather than digital ones) and with *recurrent interconnection* (rather than combinational feedforward). This makes our work applicable and meaningful to the neural network field. In addition to unreliable devices, we remove the underlying assumption (that was made in all previous theoretical research regarding recurrent nets) of a global updating clock in the network. We, thus, introduce the *truly asynchronous* model (of unreliable devices).

Three main philosophical issues are to be addressed when considering unreliable devices. First, how to model failures. Second, what type of errors should be allowed. (Here, our response considers analog-type devices). Third, networks with faulty devices are not guaranteed to generate the same response in different runs of a given input. Thus, there is a need of defining the output of the network to a given input.

The question of how to model failures was discussed by von-Neumann in [vN56] and quoted by Pippenger in [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. A more general assumption which is a good deal more realistic, is this: The malfunctions are statistically dependent on the general state of the network and on each other ..."

In our further development, we consider both the independent model of failures and the Markovian model. In the first one, the unreliability behavior is independent of any history, while in the latter, an error probability of a device (either a neuron or a connection) may depend on both k (any fixed number) last choices (i.e. success/failure) of behavior by the devices as well as the k last global states of the system. The first model is consistent with the first approach of von Neumann as well as that adopted by Wiener [Wie49] and Shannon [Sha48] in which the "noise" which is the source of unreliability is modeled as a random process with known parameters. The Markov model of failures is a possible implementation of the second philosophical approach of von Neumann.

As for the type of errors, we consider a wide possible such range: we allow for any "well described fault," while emphasizing the fault of asynchronous updating (in which each neuron has some update probability and may keep its former activation rather updating to a new one) and the phenomena of (catastrophic) forgetting in networks. Here, each neuron has the probability to forget its activation value, which is modeled as resetting the activation value to 0. Changes in the connections between neurons is modeled here by weak-connections that may break (and recover) with some probability.

Collecting the various responses of the network to its input into a unified output is solved by adapting the *bounded error model*. Given $\epsilon < \frac{1}{2}$, we assume that only $100\epsilon\%$ of the runs of a given input may yield wrong output. The correct response, then, appears in at least $100(1-\epsilon)\%$ of the runs.

1.1 Main Results

We focus on recurrent neural networks which consist of a finite number of neurons. In these networks, the activation of each processor is updated according to a certain type of piecewise affine function of the activations (x_j) and inputs (u_j) at the previous instant, with real coefficients —also called *weights*— (a_{ij}, b_{ij}, c_i). Each processor's state is updated by an equation of the type

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

where N is the number of processors and M is the number of external input signals. The function σ is the simplest possible "sigmoid," namely the saturated-linear function:

$$\sigma(x) := \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \le x \le 1 \\ 1 & \text{if } x > 1 \; . \end{cases} \tag{2}$$

As part of the description, we assume that we have singled out a subset of the N processors, say $x_{i_1}, \ldots, x_{i_l}$; these are the l *output processors*, and they are used to communicate the outputs of the network to the environment. Thus a net is specified by the data (a_{ij}, b_{ij}, c_i) together with a subset of its nodes.

In our networks, neurons do not act deterministically, but rather with some probability of failure. We consider various models of error probabilities: including fixed error probability—independent of history—and Markov probability which depends on previous history of the various devices. The faults themselves cover a wide range, including changes in the weights (i.e. receptor at the post-synaptic side), forgetting nets - in which neurons may reset their values probabilistically, and more. A total asynchronous version of the above is provided as well.

Our results include:

1. Faulty networks with any of the well described faults in neurons or connections have the computational power as in the following table:

Weights	Deterministic - Power General	Deterministic - Power Polytime	Probability	Probabilistic-Power General	Probabilistic-Power Polytime
$\mathbb{Z}$	regular	regular	any	regular	regular
$\mathbb{Q}$	recursive	P	$\mathbb{Q}$	recursive	BPP
			$\mathbb{R}$	arbitrary	Pref-BPP/ log
$\mathbb{R}$	arbitrary	P/poly	any	arbitrary	P/poly

Table 1: The Computational Power of Probabilistic and Faulty Nets

2. We define nondeterministic faulty nets and show that in this model the equality P = NP holds.

3. We introduce Markovian models of failures rather than independent error probabilities. We prove that the computational power of the Markovian faulty nets is exactly the same as in the fixed error probability model.

4. We connect our results of unreliable analog recurrent networks with more classical models of probabilistic digital models. We do so by augmenting a deterministic network with extra input lines carrying IID sequences of bits. We show the equivalence of these models to the different models of unreliablilty.

5. Catastrophic faulty models are introduced. Here, *each neuron* has some positive error probability. As for the recurrence, the errors are magnified and the computational power is limited (more precisely, after some time, the net will start behaving chaotically). To avoid the catastrophy, non-uniformity in either the errors of the neurons or in the structure of the network is suggested.

6. Truly-asynchronous networks with Markovian faulty behavior are considered and their computational power is proved to adhere to Table 1. We also demonstrate a special type of asynchronous network which is totally robust to the unreliability of its neurons in the sense that its computational power is indifferent of whether all its neurons are reliable or a few of them are not so. The existence of such a robustness is a surprising phenomena.

Notice that in the above table of computational power, the general analog network (with real weights) is robust to failure and asynchronicity. This phenomena may serve as a support to the "Siso analog computation thesis" from [SS93], stating that all reasonable analog systems can be modeled by that deterministic network model (see, Equation 1).

1.2 Previous Work: Deterministic Recurrent Networks

The deterministic recurrent network which is the underlying of the many "unreliability variances" introduced in our work, is described by an iteration equation such as (1). For notational simplicity, we often summarize this equation, writing "$x^+(t)$" instead of "$x(t+1)$" and then dropping arguments t.

Much effort has been directed towards practical implementation of the applications, including those in the areas of pattern and speech recognitions, robot control, time series prediction, and more—see [HKP91]. Only recently, rigorous foundations to the recurrent neural network model were developed, see [SS91, SS49, BGSS93, SS93]. (See [Orp92] and the many references there for analysis of nonuniform networks and [KS93] for a uniform related model but with different activation functions.) The foundations characterize the discrete functions (or "formal languages") that the nets are able to compute. Thus the focus is on input and output channels that carry only discrete data. As opposed to the I/O, the computation inside the network will in general involve continuous real values.

The computational power of the recurrent network (with finite number N of neurons and analog activations values) depends of the of numbers utilized as weights (i.e. the constants a, b, c.)

1. If the weights are integers, the neurons may assume binary activation values only. Thus, the network computes a regular language.

2. If the weights are rational numbers (here, it is suffice to have simple, non-periodic and bounded rationals) the network is *equivalent in power to a Turing Machine* ([SS91]). In particular, given any function ϕ computed by a Turing Machine M in time T, one can construct a network $\mathcal{N}$ that computes the function ϕ in exactly time T. That is, there is no slow down in the computation ([SS49]). Furthermore, the size of the network is independent of the computation time T.

A corollary is the existence of a universal network consisting of 886 neurons and simple rational weights that computes all recursive functions.

3. When weights are general real numbers (still are bounded in the range [-2,2] but may assume unbounded precision) the network turns out to *have super-Turing capabilities.* However, it is sensitive to resource constrains and thus is not a tautology. In [SS93] and [SS93] there is an exact characterization of the computational class associated with such nets: Networks with real weights compute in exponential time any Boolean language, but in polynomial time they compute exactly the class P/poly. The class P/poly includes all P and, moreover, 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.

   The main observation that allowed for the result is as follows: Let $\mathcal{N}$ be a network. Then q-truncation ($\mathcal{N}$) is a similar network to $\mathcal{N}$, but in which the weights have binary precision of q bits only and so are the activations values of the neurons. It is proven in [SS93] that up to a computation of time q, only the first $O(q)$ bits in both weights and activation values influence the result. Hence, there is a constant c so that up to computation time q, cq-truncation ($\mathcal{N}$) computes equivalently to $\mathcal{N}$.

4. All above results are true in two different models of input and output [SS49]: In the first one, the I/O are streams (one letter at a time) of letters and the network is initialized to the 0 inactive state. In the second one, there is no external I/O. The net starts computing from an "initial state" until reaching a "final state". These two states are the encoding of the input and output strings, respectively.

5. A large class of dynamical systems, including high-order and heterogeneous networks having different sigmoidal-like activation (transfer) functions adheres to similar computational constrains ([SS93]). Thus, allowing to generalize the above results in items 1-3 to a large class of neural networks.

6. One may wonder of the essence of the discrepancy between the computational power of rational (3) versus real networks (4). This issue was addressed in [BGSS93]. There, rational and real numbers (and a proper hierarchy between them) were characterized in an information theoretic manner using Kolmogorov type characterization. The main theorem there states that networks associated with different types of computable numbers recognize different classes of languages. Thus, there is a proper hierarchy of classes of languages between P and P/poly that are recognized by nets.

1.3 Paper Organization

The rest of this paper is organized as follows:

In Section 2 we introduce a simple model of probabilistic nets which has roots in classical probabilistic models. In this model, a deterministic network is augmented by extra input lines carrying binary strings. We define there the meaning of acceptance by probabilistic nets and prove that the error in such models can be reduced as much as desired.

Section 3 describes the exact computational power of the probabilistic models from Section 2. As was the case with deterministic nets, the probabilistic networks computes differently depending on the type of weights. This section includes four related theorems including one for the case of integer weights, two for the rational case, and one for real weights.

All the above served as the mathematical introduction for the faulty models which are in interest of this work. In Section 4, the various models of unreliable analog networks are introduced. We prove their computational equivalence to the probabilistic models of Section 2. Thus, Table 1 describes the power of these models as well. We also discuss the models of catastrophic faulty nets, which are nets for which all neurons have positive error probabilities. Here, chaotic behavior is demonstrated unless we encounter non-uniformity of the error in neurons or non-uniformity in the architecture of the net.

In Section 5 we introduce the Markovian faulty model. Here, each neuron (connection) has different error probabilities depending on whether it was faulty in its k previous steps, and possibly on the last k choices made by all probabilistic devices in the network. In addition, the global activation of the net may be incorporated into the decision. These models are maybe more biologically adequate than the previous one. We show that the power of the various Markovian networks is equivalent to the one of the simple faulty model.

We consider in Section 6 the nondeterminism faulty model (which is applied to both faulty and faulty Markovian models). We differentiate between weak and strong nondeterminism, where the first considers binary strings in the extra input channel, and the later allows for general real value strings. The question of comparing such nondeterminism is open for the model of computing over the real numbers ([BSS89]). We show the equivalence of the two types of nondeterminism in our faulty model. Furthermore, we show that our model adheres to P = NP.

In Section 7 we introduce an asynchronous model which incorporates faulty, Markovian neurons. We prove their computational power to be equal to Table 1.

In Section 8 we provide the two main and hardest proofs in the paper. These demonstrate the computational power of the probabilistic model from Section 2. We end in Section 9 with a few comments and open problems.

2 Probabilistic Recurrent Networks: Definitions

We consider a special type of probabilistic device: a binary gate which outputs 1 with probability $p \in [0,1]$. We call such a gate a *probabilistic gate*.

Definition 2.1 A *probabilistic network* ($\mathcal{PN}$) is a network that consists of two types of devices: saturated neurons and probabilistic gates.

Alternatively, we can view probabilistic networks as networks having a certain number of additional input lines. These input lines carry independent streams of binary data, each is an IID (independent identically distributed) sequence, with possibly different distribution in the different lines.

Definition 2.2 A probabilistic network— in signs $\mathcal{PN}$ —is a network that has a few additional input lines of independent identically distributed binary sequences.

We use definitions 2.1 and 2.2 interchangeably.

We define now what we mean by a probabilistic network recognizing a language $L \subseteq \{0,1\}^+$. We proceed as in [SS49] and define formal nets with two binary input lines. The first of these is a *data line*, and it is used to carry a binary input signal; when no signal is present, it defaults to zero. The second is the *validation line*, and it indicates when the data line is active; it takes the value "1" while the input is present there and "0" thereafter. We use "D" and "V" to denote the contents of these two lines, respectively, so

$$u(t) = (D(t), V(t)) \in \{0,1\}^2$$

for each t. We always take the initial state $x(1)$ to be zero and to be an equilibrium state. We assume that there are two output processors, which also take the role of data and validation lines and are denoted $O_d(t), O_v(t)$ respectively. The probabilistic network is not guaranteed to generate the same output in different runs of a given input string. Given a formal net $\mathcal{N}$, with two inputs as above, we say that a word α is *classified* for a given run *in time* τ, if for that run the following property holds: the output sequence

$$y(t) = (O_d(t), O_v(t)) \in \{0,1\}^2$$

produced by u_α when starting from $x(1) = 0$ has the form

$$O_d = \underbrace{0 \cdots 0}_{\tau-1} \eta_\alpha 000 \cdots , \quad O_v = \underbrace{0 \cdots 0}_{\tau-1} 1000 \cdots ,$$

where $\eta_\alpha = 0$ or 1.

We say that a probabilistic network $\mathcal{PN}$ recognizes the language L with an *error* if there exists an input ω so that either $\omega \in L$ and $\mathcal{PN}$ rejects it in some run, or $\omega \notin L$ and $\mathcal{PN}$ accepts it in some run.

Definition 2.3 Let ϵ be a real number such that $0 < \epsilon < .5$. We say that a probabilistic network $\mathcal{PN}$ *ϵ-recognizes* a language L if for each $\omega \in \{0,1\}^+$ output is provided and probability of error in recognizing each input string ω is bounded by ϵ.

For example, if $\epsilon = .2$, for any input string ω, 80% of the runs —probabilistically speaking— yield the same output, and this response to the input ω is defined to be the output of the network. Not more than 20% of the runs will probabilistically yield an error.

Let $T : \mathbb{N} \to \mathbb{N}$ be a function on natural numbers. We say that the language $L \subseteq \{0,1\}^+$ is *ϵ-recognized in time T* by the formal probabilistic net $\mathcal{PN}$ provided that all the computations of each word $\omega \in \{0,1\}^+$ are computed in time $T(|\omega|)$. 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.

We show that one can reduce the acceptance error in networks. That is, if a language L is ϵ-recognized by a probabilistic network in time T, it can also be λ-recognized by another probabilistic network, for each $0 < \lambda < \epsilon$ with little increase in the size of the network or in its computation time. (The theorem is analogous to the one concerning threshold circuits, given in [Par92] pg 199.)

Theorem 1 *Let $f(\lambda, \epsilon) = \lceil \frac{2\log\lambda}{\log(4\epsilon(1-\epsilon))} \rceil$. For every $0 < \lambda < \epsilon < \frac{1}{2}$, any language that can be ϵ-recognized by a probabilistic network $\mathcal{PN}$ of size N and computation time T, can also be λ-recognized by a probabilistic network $\mathcal{PN}'$ either of size $N f(\lambda, \epsilon)$ and in time $(T+1)$, or of size $(N+10)$ and in time $T(f(\lambda, \epsilon) + 10)$.*

Proof. Assume we compute $b \in \mathbb{Z}$ independent computations on the same input ω in the network $\mathcal{PN}$, and decide upon the majority of all these trials. The probability of an error of this experiment is the probability of error in at least half of b independent Bernoulli trials, where each trial has a failure probability ϵ. Given b Bernuli trials having failure probability ϵ, the probability of failing m out of them, for $m \geq b\epsilon$ is

$$B(m, b, \epsilon) = \sum_{i=1}^{b} \binom{b}{i} \epsilon^i (1-\epsilon)^{b-i} ,$$

which can be bound by

$$B(m,b,\epsilon) \le (\frac{b\epsilon}{m})^m (\frac{b-b\epsilon}{b-m})^{b-m} \quad . \tag{3}$$

Thus, the error of this decision, which is $B(\frac{b}{2}, b, \epsilon)$ is bounded by

$$(4\epsilon(1-\epsilon))^{\frac{b}{2}} \ .$$

Choosing $b \ge \frac{2\log\lambda}{\log(4\epsilon(1-\epsilon))}$ the error is bounded by λ. Assume a probabilistic network $\mathcal{PN}'$ which consists of b copies of the network $\mathcal{PN}$ (thus computing all b trials in parallel) and decide acceptance upon the majority of the b parallel trials. Such net computes with an error bounded by λ. Another net that yields such a bounded error is a net that computes the b trials serially in one copy of the network, while keeping track of the history decisions, and decides upon the majority. Ten neurons suffice for the necessary bookkeeping. ∎

As for the above theorem, we can use the following notation:

Definition 2.4 $\mathcal{LPN}$ is the class of languages that are ϵ-recognized by some probabilistic network for any $\epsilon < .5$.

That is, there is no need in defining ϵ-$\mathcal{LPN}$ for different values of ϵ.

3 Computational Power of The Simple Probabilistic Model

The probability gates in the network adds randomness into the computation. It is reasonable to conjecture, then, that the gates add power to the deterministic net. Just as we did when investigating the computational power of deterministic networks, we distinguish among probabilistic models utilizing different types of constants (weights). We prove that when weights are integers, adding probabilistic gates does not add power to the model (Theorem 2). Similar is the case of networks with general real weights (Theorem 5). However, when the weights are rational numbers, the probabilistic model is more powerful than the deterministic one. In the rational weight case, we can distinguish two subcases, depending if the probabilities are restricted to be rational or not. If probabilities are rationals, $\mathcal{LPN}$ is equivalent to the class of languages accepted by probabilistic Turing Machines with bounded errors (Corollary 3.3). When the computation time is polynomial, this class, denoted BPP, includes P and is included both in $\Sigma_2 \bigcap \Pi_2$ and in P/poly. (None of the inclusions is known to be strict.) If the probabilities may be arbitrary real numbers, $\mathcal{LPN}$ is equivalent to a nonuniform extension of BPP that allows for an additional *prefix logarithmically long advice* (Theorem 4). The results are summarized in Table 1.

In the rest of this section, when referring to networks with integer weights we use the notation $\mathcal{N}_i$. Networks with rational weights are denoted by $\mathcal{N}_q$, and network with real weights by $\mathcal{N}_r$. We still use the generic network notation $\mathcal{N}$.

3.1 Probabilistic Integer Networks

We show that the class $\mathcal{LPN}_i$ of languages which are ϵ-recognized by probabilistic nets with integer weights is equivalent to the class of regular languages. The classical definition of probabilistic automaton (see [Paz71] for example) is as follows:

Definition 3.1 A probabilistic automaton A is the 5-tuple $A = (Q, \Sigma, p, q_0, F)$ 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 $p(m, q, a)$, $m, q \in Q$, $a \in \Sigma$ is the probability of getting into state m from a state q and the input symbol a. This transition function satisfies that $\forall q, a$

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

We consider the doubly-transition probabilistic automaton which is a special case of the probabilistic automata: the transition function may transfer each state-input pair into exactly two states. That is, $\forall q \in Q, a \in \Sigma, \exists$ exactly two states m_{1qa}, m_{2qa} so 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 slow down in the computation).

Let $\mathcal{LPA}$ be the class of languages ϵ-recognized by probabilistic automata; $\mathcal{LPN}_i$ is the class of languages ϵ-recognized by probabilistic networks with integer weights.

Theorem 2 *The class $\mathcal{LPN}_i$ is the class of regular languages.*

Proof. We start with proving the correspondence between probabilistic automata and probabilistic integer networks. A doubly transition probabilistic automaton can be viewed as a finite automaton with $|Q||\Sigma|$ additional input lines of IID binary sequences. (These lines imply the next choice of the transition.) From the equivalence of integer networks and finite automata ([Sie93]), we conclude the equivalence of the probabilistic automata with the probabilistic integer networks. That is, $\mathcal{LPN}_i = \mathcal{LPA}$.

Consider now Rabin's result about probabilistic automata ([Paz71], pg. 160):

> Let A be a probabilistic automaton. $P^A(\omega)$ is the probability of answering "yes" on input ω. A *cut point* λ is *δ-isolated* with respect to A if for any input ω, $|P^A(\omega) - \lambda| \geq \delta$. If a probabilistic automaton $\mathcal{A}$ has an isolated cut-point, then there is a deterministic automaton A' that recognizes the same language which is ϵ-recognized by $\mathcal{A}$.

In the case of ϵ-recognition by $\mathcal{PN}_i$, $\frac{1}{2}$ is an isolated cut-point with $\delta = \frac{1}{2} - \epsilon > 0$. Thus, probabilistic networks with integer weights are computationally equivalent to probabilistic automata with isolated cut points. That is, $\mathcal{LPN}_i$ is the class of regular languages. ∎

3.2 Probabilistic Rational Networks

As opposed to the integer case, we prove that the probabilistic networks with rational weights are computationally *strictly stronger* than the deterministic ones. Nonetheless, these networks are still less powerful than the deterministic networks that allow for real weights.

Definition 3.2 Let $S = \{p_1, p_2, \ldots p_s\}$ be a finite set of probabilities. A *Probabilistic Turing Machine over the set S* (PTM$[S]$) is a non-deterministic machine so that

1. Every step of the computation can be made in exactly two possible ways, with some probability $p_i \in S$ to choose one of them and the probability $(1 - p_i)$ to choose the alternative one.

2. The number of steps in each computation is exactly the same.

3. Every computation ends with Reject or Accept.

In the special case in which $S = \{\frac{1}{2}\}$, we get the classical probabilistic Turing machine (denoted as PTM), defined for example in Balcazar at al [BDG90].

Theorem 3 *Let S be a finite set of probabilities. Let $\mathcal{LPTM}[S]$ the class of languages recognized by probabilistic Turing Machines that use probabilistic decisions from the probability set S, and $\mathcal{LPN}_q[S]$ is the class of languages recognized by probabilistic rational nets, where probabilities are all from S. Then these two classes are polynomially related.*

Proof.

1. $\mathcal{LPTM}[S](T) \subseteq \mathcal{LPN}_q[S](O(T))$:
   Let M be a probabilistic Turing Machine over the set S that computes in time T. We simulate it by a probabilistic network $\mathcal{N}_q$ with probabilistic gates having probabilities from the set S. The proof is based upon the equivalence between deterministic Turing Machine and neural network with rational weights, as proved in [SS49]. The network $\mathcal{N}$ is provided implicitly by the following program, that uses probabilistic gates named random$_i$ ($i = 1..s$) and simulates M as follows:

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

   Here, NextStep is a procedure that given the current state, tapes, and which of the two choices to take, changes deterministically to next configuration of M. It was constructively proved in [Sie93] that such programs (under strict grammar of the language NEL) are compiled into simple, finite recurrent nets that compute in no more than linear slow down.

2. $\mathcal{LPN}_q[S](T) \subseteq \mathcal{LPTM}[S](T_1)$, where T_1 =poly(T):
   It is easy to verify that if $\mathcal{PN}_q$ has s probabilistic gates, it can be simulated by a probabilistic Turing Machine over these s probabilities.

∎

Next, we differentiate the case were all probabilities of the set S are rational numbers against the case in which at least one of its elements is real.

- Consider probabilistic Turing Machines with probabilities over the set S, where S consists of rational probabilities only. It is shown by Zachos in [Zac82] 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 TM (that is, to the case where $k = 2$). When probabilities are rationals, we can substitute them all by a common divisor which is written as $\frac{1}{k'}$ for an integer k', implying a polynomial equivalence to the probabilistic TM model.That is, we can substitute PTM[S] by PTM. When S is a set of rational numbers, we conclude the equality between the class $\mathcal{LPN}_q$(poly-time) and the class BPP (Bounded Probability model with Polynomial time, see Balcazar at al [BDG90] pg 138 def 6.4). Thus, $\mathcal{PN}_q$(poly-time) is included in $\mathcal{N}_r$ (as BPP is included in P/poly, [BDG90] pg 144, cor 6.3) and is in $\Sigma_2 \bigwedge \Pi_2$ ([BDG90] pg 172, theorem 8.6). Pis included in BPP, but whether this inclusion is strict is an open question.

Corollary 3.3 Adding rational probabilities to the model of deterministic network with rational weights adds power to the machine if an only if the class BPP is strictly stronger than P.

- In the case where S consists of at least one real number, the probabilistic model turns out to be stronger than the deterministic one. Similarly to the definition in [BHM92], we define the class Pref-PTM$(T(n))/log$ as follows:

Definition 3.4 A set A is in Pref-PTM$(T(n))/log$ if

$$\forall n \; \exists \omega_n \; (|\omega_n| \leq c \log T(n)) \; \forall x \; (|x| \leq n) \quad x \in A \iff <x, \omega_n> \in B$$

where $B \in \mathcal{LPTM}(T(n))$ and c is a constant. Furthermore, $\forall n \leq m$, ω_n is a prefix of ω_m.

String w_n in the definition is called "advice for length n". Observe that w_n must be useful for all strings of length up to n, not only those of length n.

If we consider only polynomial time computation then $|\omega_n| \leq c \log n$, we denote the accepted class as pref-BPP/log. If exponential time is allowed, the class is pref-BPExp/poly.

Theorem 4 *Let p be a real number in $[0,1]$. Denote by $\mathcal{LPTM}[\{p\}]$ the class of languages recognized by probabilistic Turing Machines over the probability p. Then, $\mathcal{LPTM}[\{p\}]$ and Pref-PTM/log are polynomially related.*

The proof is long and not trivial. To keep the paper readable, we postpone it to Section 8.

Corollary 3.5 Let S be a set of probabilities which are not all rational. Then, In exponential time, $\mathcal{LPN}_q$ is the class BPExp/poly, the class of functions recognized by probabilistic Turing machines given polynomially long advice in exponential time.

(As the advice is polynomially long, BPExp/poly and pref-BPExp/poly are equivalent.)

Corollary 3.6 Let S be a set of probabilities which are not all rational. Then, in polynomial time, $\mathcal{LPN}_q$ is the class Pref-BPP/log.

We conclude that real probabilities add power to neural networks with rational weights.

3.3 Probabilistic Networks With Real Weights

We next prove that real networks and probabilistic real networks have the same computational power up to a polynomial time slowdown:

Theorem 5 *Denote by $\mathcal{LN}_r(T(n))$ the class of languages recognized by networks in time $O(T(n))$, and by $\mathcal{LPN}_r(T(n))$ the class of languages ϵ-recognized by probabilistic real networks in time $O(T(n))$. Then*

$$\begin{aligned} \mathcal{LN}_r(T(n)) &\subseteq \mathcal{LPN}_r(T(n)) \\ \mathcal{LPN}_r(T(n)) &\subseteq \mathcal{LN}_r(n^2 + nT(n)) . \end{aligned}$$

∎

A similar result by [BG81] asserts that non-uniform deterministic polynomial size circuits are as powerful as probabilistic ones. Our theorem can be interpreted as a simple proof for a similar statement, as the class $\mathcal{LN}_r$ is equivalent to non-uniform polynomial size circuits ([SS93]). The proof is postponed to Section 8.

4 Probabilistic Faulty Analog Networks

Here we define formally various models of faulty analog networks. That is, we do not allow for Boolean gates but rather allow some of the analog neurons to have a (fixed) positive probability of failure. A probabilistic resetting neuron with probability p_i is defined by the update equation:

$$x_i^+ = \begin{cases} \text{update} & 1 - p_i \\ 0 & p_i \ . \end{cases} \tag{4}$$

A *probabilistic forgetting (partially reliable) network* ($\mathcal{PRN}$) is a network that consists of two types of devices: saturated neurons and probabilistic resetting neurons. The network is *catastrophic forgetting network* if all neurons have a positive probability of resetting their activation values. Denote by $\mathcal{LPRN}$ the class of languages ϵ-recognized by some probabilistic forgetting network.

A probabilistic asynchronous neuron with probability p_i is defined by the update equation:

$$x_i^+ = \begin{cases} \text{regular update} & 1 - p_i \\ x_i & p_i \ . \end{cases} \tag{5}$$

Analogously to the above definition, we define the *probabilistic asynchronous network* ($\mathcal{PAN}$), the *catastrophic asynchronous network*, and the class of languages $\mathcal{LPAN}$. (Note that this model is not totally asynchronous as a global clock still controls possibly simultaneous updates.)

A probabilistic weakly connected neuron x_i with probabilities p_{ij} 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

$$\tilde{a}_i^k = \begin{cases} a_i^k & 1 - p_i \\ 0 & p_i \ . \end{cases} \tag{6}$$

A *probabilistic weakly-connected network* ($\mathcal{PWN}$) is a network that consists of both reliable and faulty connections, where a few neurons have all their connections reliable. The *catastrophic probabilistic weakly-connected network* and the class $\mathcal{LPWN}$ are defined analogously to the above definitions.

We interpret the output of all the faulty nets as in the probabilistic model, see Definition 2.3.

Theorem 6 *Probabilistic forgetting networks, probabilistic asynchronous networks, probabilistically weakly-connected networks and probabilistic networks are computationally equivalent. So are all their combinations.*

Proof.

- $\mathcal{LPWN} \subseteq \mathcal{LPRN}$:
  We show how to simulate faulty-connections neurons with resetting neurons. Given a weakly connected network with N neurons, we simulate it by a semi-forgetting network having $N+N^2$ neurons.

  Each neuron $\tilde{x}_k$ $(k = 1 \dots n)$ in the weakly connected network is associated with $(1 + N)$ neurons in a probabilistic forgetting net:

$$\begin{aligned} x_k^+ &= \sigma(\sum a_i^k x_{ik} + \sum b_j^k u_j + c_k) \\ x_{ik}^+ &= \sigma(x_i) \qquad i = 1 \dots N \end{aligned}$$

  where x_k is a reliable neurons, and x_{ik} has reset probability of p_{ik} if and only if the weight a_i^k has the reset probability of the same amount. It is easy to see that the probabilistic forgetting net computes the same task as the probabilistic weakly connected one does.

- $\mathcal{LPRN} \subseteq \mathcal{LPAN}$:
  Let g_i be a probabilistic resetting gate with reset probability p_i and total weight W_i. We simulate it by the following probabilistic asynchronous circuit with output neuron x_i:

$$\begin{aligned} x_i' &= \begin{cases} \sigma(1 - x_i') & p_i \\ x_i' & 1 - p_i \end{cases} \\ x_i'' &= \sigma(x_i') \\ x_i &= \sigma(\text{update } - W_i(\sigma(x_i' - x_i'') + \sigma(x_i'' - x_i'))) \end{aligned}$$

- $\mathcal{LPAN} \subseteq \mathcal{LPN}$:
  Let g_i be a probabilistic asynchronous neuron that updates with the probability p_i. We simulate it with the following probabilistic net:

$$x_i^+ = \sigma[\sigma(\text{ update } - g_i) + \sigma(x_i + g_i - 1)]$$

  where g_i is a probabilistic gate with probability p_i of outputting the value 1.

- $\mathcal{LPN} \subseteq \mathcal{LPWN}$:
  Let g be a probabilistic gate with a probability p of outputting the value 1. It is simulated with a faulty connection neuron $x = \sigma(a)$ so that a has the resetting probability of $(1 - p)$.

∎

We generalize the above statement of faulty networks as follows:

Definition 4.1 A neuron x_i is said to have a *well described fault P* if it may compute d different functions f_i within fixed probabilities p_i, so that $\sum_{i=1}^{d} p_i = 1$. Each such f_i is a deterministic function of the neurons $x_j(t)$ $(j = 1 \dots N)$ and is computable via a net in constant time.

Special cases are the resetting neurons, asynchronous neurons, and weakly connected neurons.

Theorem 7 *Let P be a well described fault and $\mathcal{N}$ is a P-faulty network. Then $\mathcal{N}$ has the computational power as described in Table 1.*

The proof is not hard given the above details.

4.1 Catastrophic Faulty Networks

Here, we consider the case where all neurons have positive probability of having faults.

Lemma 4.2 No deterministic function can be computed by a probabilistic forgetting network that requires non-constant time T.

Proof. Assume the probability of error is δ in each neuron. The probability that there is a time t in which all neurons in the network reset is $\epsilon = \delta^N$. The probability that the net resets all its neurons during T successive steps is

$$1 - (1 - \epsilon)^T$$

which gets close to 1 for any growing function T. ∎

Furthermore, one can show that the behavior of this network is chaotic. A similar proof with show that

Lemma 4.3 Any P-faulty network can not compute deterministic function in time T which is not constant.

To handle such catastrophic networks, we consider the following cases:

1. Feedforward networks: Here, the computation time is bounded and thus, the total error is bounded by some ϵ. To lower the error ϵ, we copy the same network few times and take the majority decision.

   Proposition 4.4 Given a reliable feedforward network $\mathcal{N}$, there is a catastrophically faulty network that computes like $\mathcal{N}$ with a recognition error bounded by .5.

   Proof. For each N there is δ_N so that $(1 - \delta_N)^N > \frac{1}{2}$. The probability that the recognition is wrong is bounded by the probability that at least one neuron did not act correctly

$$\epsilon \leq 1 - (1 - \delta_N)^N < \frac{1}{2} \, .$$

   ∎

2. Recurrent networks in which the error is nonuniform:

   Proposition 4.5 A reliable recurrent net can be simulated by a family of catastrophically unreliable recurrent networks, in which all nets have the same architecture but with an error probability which is a decreasing function of the computation time.

   Proof. Let δ_T be a function so that $(1 - \delta(T))^{NT} > c > \frac{1}{2}$ for some constant c, and the proposition results. ∎

3. Recurrent network with a fixed fault rate δ but with a growing size of network as a function of the computation time.

   Proposition 4.6 A reliable recurrent network can be simulated by a family of catastrophically unreliable recurrent networks, in which the network grows as a function of the computation time.

   The above is equivalent to enlarging the width of the network in the feedforward model. The proof is omitted as for its simplicity.

5 Markovian Faulty Networks

As opposed to the model of independent erroneous neurons, here we consider devices that their unreliable behavior depends on their last choice of update (reflexive Markovian); or more generally, on the last c choices of all devices and the last c global states in the network (c-general Markovian devices).

Definition 5.1 A reflexive Markov neuron (connection) x_i is a neuron (connection) that acts by the probabilities:

$$\begin{array}{lcl} P[x_i(t) = \text{ success } (\cdot) \mid x_i(t-1) = \text{ success } (\cdot)] & = & 1-p \\ P[x_i(t) = \text{ failure } (\cdot) \mid x_i(t-1) = \text{ success } (\cdot) & = & p \\ P[x_i(t) = \text{ success } (\cdot) \mid x_i(t-1) = \text{ failure } (\cdot) & = & q \\ P[x_i(t) = \text{ failure } (\cdot) \mid x_i(t-1) = \text{ failure } (\cdot) & = & 1-q \end{array}$$

More generally, for any well described fault P with d possibilities (rather than 2 only) of functions f_i, the reflexive Markovian neuron is defined by: $P[x_i(t) = f_j(\cdot) \mid x_i(t-1) = f_l(\cdot)] = p^i_{jl}$ where $\sum_j p^i_{jl} = 1$ for each $i = 1 \ldots N$, $j, l = 1 \ldots d$.

In the probabilistic reflexive Markovian model, the output is interpreted as in Definition 2.3.

Theorem 8 *The probabilistic reflexive Markov models of asynchronous nets, forgetting nets and weakly-connected nets are computationally equivalent. The same is correct for any of the combinations of the above.*

The proof of this theorem is omitted as is very similar to the proof of Theorem 6

Theorem 9 *For any well-described fault, the reflexive Markovian faulty model is computationally equivalent to the faulty model.*

Proof.

- Given a neuron with a fixed independent fault

  $$x_i = \begin{cases} \text{success} & 1-p \\ \text{failure} & p\,, \end{cases}$$

  it is a special case of the reflexive Markovian faulty neuron as in Definition 5.1 for which $p + q = 1$. Here $P[x_t = \text{ success } (\cdot) \mid x_i(t-1) = \text{ success } (\cdot)] = P[x_t = \text{ success } (\cdot) \mid x_i(t-1) = \text{ failure } (\cdot)] = p$ and $P[x_t = \text{ failure } (\cdot) \mid x_i(t-1) = \text{ success } (\cdot)] = P[x_t = \text{ failure } (\cdot) \mid x_i(t-1) = \text{ failure } (\cdot)] = 1-p$.

- Given a reflexive Markovian neuron M (with a well described fault) as in Equation 5.1, we show how to simulate its behavior by a small probabilistic network. Assume that M may compute d different functions f_i with the probability $P[f_i(\cdot)|f_j(\cdot)] = p_{ij}$, where $\sum_i p_{ij} = 1$. For simplicity we assume $d = 2$ (that is, success or failure), but generalizing to a larger d is not hard. Thus, for us, $p_{11} \equiv p$ and $p_{12} \equiv q$, and $f_1 =$ success.

  We define two deterministic neuron S_1 and S_2 assuming binary values where $S_1 = 1$ if the last step of the reflexive Markovian neuron M was success and 0 otherwise; the neuron S_2 carries the complement value of S_1. We assume two probabilistic gates n_j $j = 1, 2$ which output 1 with the probabilities p_{1j}, respectively.

Thus, the function

$$y = S_1[n_{11} f_1(M) + (1 - n_{11}) f_2(M)] + S_2[n_{12} f_1(M) + (1 - n_{12}) f_2(M)]$$

computes just as M does. More in details, we define the neurons:

$$\begin{aligned} z_1 &= \sigma(S_1 + n_{11} - 1) \\ z_2 &= \sigma(S_2 + n_{12} - 1) \\ z_3 &= \sigma(S_1 - n_{11}) \\ z_4 &= \sigma(S_2 - n_{12})\,. \end{aligned}$$

Note that these are binary neurons and at each time exactly one of them carry the value 1. We also define the neurons:

$$\begin{aligned} z_5 &= \sigma(f_1(M) + z_1 + z_2 - 1) \\ z_6 &= \sigma(f_2(M) + z_3 + z_4 - 1)\,, \end{aligned}$$

and

$$y_7 = \sigma(z_5 + z_6)\,.$$

The value of y_7 is equivalent to M. The history-saving neurons S_1 and S_2 are updated by the equations:

$$\begin{aligned} S^+ &= \sigma(z_1 + z_2) \\ F^+ &= \sigma(z_3 + z_4)\,. \end{aligned}$$

This ends the proof for the special case $k = 2$. A similar proof holds for the case of general d. Here, we have to assume that the probabilistic gates output d different discrete values. We substitute these multi-level probabilistic gates with binary ones and the proof results. (We simulate a gate of d levels with d probabilistic gates a_i and we interpret their result as the smallest i that has the value 1. If the probabilities of the different levels of the d-probabilistic gate were p_i, $i = 1 \ldots d$, then the probabilities of the a_i gates to output 1 are

$$q_i = \frac{p_i}{1 \sum_{j<i} (1 - p_j)} \quad .$$

For more details, see the proof of Theorem 13 in Section 7 where simulating a few level probabilistic gate by binary probabilistic gates.)

∎

These results are true for a more general Markovian behavior: A c *general Markovian model* is a faulty Markovian model, where the probability of choosing the next function of each of the well-described faulty devices depends upon both the last c choices of all other devices and on the last c global activation values of the neurons.

More formally: A *saturated-threshold recurrent network* is defined as in Equation 1. However, some of its neurons have the saturated activation function (see Equation 2) and some other have the hard-threshold activation function which is described by

$$\mathcal{H}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0\,. \end{cases} \tag{7}$$

Definition 5.2 Let $J_{c,t}$ be a vector $[j_{1,t-c}, j_{1,t-c+1}, \ldots, j_{N,t-1}] \in p^{cN}$ that presents the choices made in times $t-c$ to $t-1$ by the different neurons. $F_{c,t}$ is the binary value defined by the function

$$F_c \ : \ [0,1]^{CN} \mapsto \{0,1\}$$

that receives as its input the last c activation values of all neurons and computes a binary decision. The function F_c is computable by a saturated-threshold network in a constant amount of time.

A c-general Markov network is a P faulty network, where P has d choices to make, for which for all $i = 1 \ldots N$, $j = 1 \ldots d$, the probabilities

$$P[x_i(t) = f_j(\cdot) \mid J_{c,t}(\cdot), F_{c,t}(\cdot)]$$

are well defined.

Theorem 10 *For any well-described fault and any constant $c \geq 1$, the c general Markov model is computationally equivalent to the faulty model.*

Proof. If it were known that the saturated-threshold network is computationally equivalent to the saturated network, then the previous proof could have been easily generalized to prove this theorem. (To overcome c when larger than 1, we have to add a few more "memory neurons.") Thus, we prove the following lemma:

Lemma 5.3 The computational power of the saturated-threshold network is polynomially equivalent to the (saturated) network model

Proof of Lemma. It is clear that networks are included in saturated-threshold nets. We prove the other direction: If weights are integers, the activation values of all neurons in the network model are binary. Hence, additional threshold devices add no power to the net. If the weights are rationals, as only linear combinations are computed in each neuron, there is a base in which all activations of all neurons at each time t consist of not more than poly(t) bits. This model was shown [SS49] to be equivalent to Turing Machines. Addition of a threshold device to the Turing Machine model, is equivalent to read the left most bit in the tape in one step. This can be simulated by adding a second reading head and may require up to a polynomial slow down when transferring to the original Turing Machine model. In the case of networks with real weights, it was shown in [Koi93] that the saturated-threshold networks compute not more that P/poly and thus are included in the network model. ∎

6 Nondeterminism in Probabilistic Nets

Both in the discrete complexity theory and in the model of computation over the real numbers [BSS89], the P = NP question is one of the most significant open problems. While in the discrete case, nondeterminism has only one meaning, it has two possible interpretations in continous-analog models. The weak nondeterminism is a model that incorporates strings of bits of guesses into the computation. The strong nondeterministic model incorporates a series of guesses, where each guess is a real number [BSS89]. The question $\mathrm{NP}_{\text{weak}} = \mathrm{NP}_{\text{strong}}$ is an important open problem in the BSS model of computation over the real numbers. Here, we show that in both the faulty and Markovian faulty nets, not only strong and weak nondeterminism are equivalent in power, but they both are equal to the deterministic model.

Theorem 11 *In the probabilistic models:* $P = NP_{\text{weak}} = NP_{\text{strong}}$.

(In the deterministic network, one can show that strong an weak models are equivalent using similar technique as here, but whether the nondeterministic models are equal to the deterministic one - is still open.)

We start with a few definitions.

Definition 6.1 A *probabilistic/faulty/Markovian network architecture* $\mathcal{A}$ is a network in which the probabilities are variables v_i.

Definition 6.2 A *nondeterministic probabilistic/faulty/Markovian architecture* is an architecture $\mathcal{A}$ that when given an input string ω, it guesses the values of the probabilities $v = v_1, v_2, \ldots v_N$ and outputs $\mathcal{A}(v, \omega) \in \{0, 1\}$. The *language accepted* is

$$L_{\mathcal{A}} = \{\omega \mid \exists v, \mathcal{A}(v, \omega) = 1\} .$$

As in the model of computation over the real numbers, we consider weak and strong nondeterminism. We say that L is a language accepted by $\mathcal{A}$ using a *strong model of nondeterminism* if $v \in \mathbb{R}^N$, and by a *weak model of nondeterminism* when $v \in \{0, 1\}^z$, z is a constant depending on the number of choices in the Markovian neurons.

*Proof. * Of Theorem* 11 *
Assume the input to a nondeterministic probabilistic architecture $\mathcal{A}$ is ω of length n, and it computes in time $T(n)$. As was proved in Theorem 4 (and combining it with Theorems 2, Theorem 3 and Theorem 5), only the first $O(\log T(n))$ bits of each probability p_i are significant for the computation of the input string ω. Thus a strong nondeterminism model that guesses the z probabilities can be simulated by a weak nondeterminism that guesses only $z \log(T(n))$ bits. Even more so, a deterministic model can simulate the latter by generating deterministically all possible sequences of $z \log(T(n))$ bits, simulating for each such a sequence the output $\mathcal{A}(v, \omega)$ and accepting ω if there exists at least one sequence v that causes a probabilistic acceptance. ∎

7 The Truly Asynchronous Models

In nature, each neuron updates by its own internal pace. Understanding asynchronous behavior is a hard but very important issue in the understanding of dynamical systems. In this section we solve the interesting open problem of the computational power of truly asynchronous networks.

To characterize exactly the behavior of the totally asynchronous neural network, we follow the classical assumption of asynchronous distributed system: no two neurons ever update simultaneously. Thus, it is commonly modeled as a synchronous network in which at each time only one neuron updates.

Definition 7.1 The *totally asynchronous recurrent network* ($\mathcal{AN}$) is a deterministic recurrent networks having an additional input line. This input channel S assumes values in the range $[1..N]$, where N is the size of the network. For each time t and each neuron i, $S(t) = i$ with probability p_i, $(i = 1 \ldots N)$ and $\sum_i p_i = 1$. (The probabilities match the relation in the speeds of the different neurons.) At each time t, only processor $S(t)$ updates. The output of such a network is interpreted by the probabilistic output definition, see Definition 2.3.

Equivalently, we may model the extra external input as a special probabilistic gate that has N different discrete levels of output. An equivalent definition is, thus, implied:

Definition 7.2 A *totally asynchronous network* is a deterministic network with an additional N-level probabilistic gate, where level l_i appears in probability p_i and $\sum_i p_i = 1$. At each time t, only processor $S(t)$ updates, and output is interpreted probabilistically.

This model provides truly asynchronous behavior. However, all its neurons are reliable. We next consider *unreliable networks which update asynchronously.* Throughout the rest of this section, the various models of asynchronous unreliable networks assume to incorporate a few reliable neurons.

Definition 7.3 Let P be a well described fault with d different functions f_l. A *P faulty asynchronous network* ($\mathcal{FAN}$) is a deterministic faulty network with an additional dN-level probabilistic gate S. Each level $l_{i,j}$ ($i = 1 \ldots N$, $j = 1 \ldots d$) appears in the probability $p_{i,j}$ so that $\sum_{i,j} p_{i,j} = 1$.

Each neuron n_i is associated with a set $\mathrm{id}(n_i) \subseteq \{(i,1), \ldots (i,k)\}$. In time t, if $S(t) = l_{ij}$, the neuron n_i will update to a new activation state using its f_j update function.

This models fixed probability faulty asynchronous behavior. We next suggest two possible models of asynchronous faulty models with some kind of memory: The Markovian model, and the sequence model.

Definition 7.4 A *Markovian faulty asynchronous network* ($\mathcal{MFAN}$) with the natural parameter $c \geq 1$ is a faulty asynchronous network, for which the probabilities $\mathrm{Prob}[l_{ik} \mid l_{j1}, \ldots l_{jc}]$ are well defined. The parameter c specifies the length of history considered.

The reflexive Markov model can be defined analogously, where only the last level associated to the same neuron will influence the outcome. In the general Markovian faulty asynchronous net, we take into account also the value F_c — i.e. the binary decision based on the last c global activation values. It is easy to see that all these Markovian models are equivalent. We, thus, stick to the one in the definition. The sequence faulty asynchronous network is very similar to the faulty asynchronous network, but each level causes some subset of neurons to wake up (rather than only one neuron).

Definition 7.5 Let P be a well described fault with d different possible update functions. A *sequence faulty asynchronous network* ($\mathcal{SFAN}$) of fault P with the natural parameter $c \geq 1$ is a deterministic faulty network with an additional probabilistic gate of

$$\begin{pmatrix} N \\ c \end{pmatrix} d^c$$

levels. Each level is associated with a subset of the neurons along with their different possible update functions. There is a fixed total ordering, and the chosen subset is being executed sequentially by this order.

Theorem 12 *The Markovian faulty asynchronous network and the sequence faulty asynchronous network are computational equivalent and their computational power is described by Table 1*

Before proving Theorem 12, we need the following definition:

Definition 7.6 An *ultra asynchronous network* ($\mathcal{UAN}$) is a deterministic network with an additional z-level probabilistic gate S; each level l_i appears in the probability p_i so that $\sum_{i=1}^{z} p_i = 1$.

Each neuron n_j is associated with a set $\mathrm{id}(n_j) \subseteq \{1, 2, \ldots z\}$. In time t, all neurons n_j which for them $S(t) \in \mathrm{id}(n_j)$ will update to a new activation state.

The *ultra faulty asynchronous network* ($UFAN$) allows for well-described (P with d choices) faulty neurons in the network. Here, each neuron i and a function choice j of its possible update rule are associated with a set $\mathrm{id}(i, j)$ and for any $i = 1 \ldots N$ and any different values $j, l = 1, \ldots, d$

$$\mathrm{id}\ (i, j) \cap\ \mathrm{id}\ (i, l) = \phi\ .$$

Now, each level activates a subset of the neurons and dictates their update choice.

Theorem 13 *The class of functions associated to the ultra asynchronous model of recurrent neural networks ($\mathcal{LUAN}$) is described in Table 1.*

Proof: We prove this theorem by showing an equivalence between the simple probabilistic model $\mathcal{PN}$ described in Section 2 and the ultra asynchronous model $\mathcal{UAN}$.

- $\mathcal{LUAN} \subseteq \mathcal{LPN}$:
  Let $\mathcal{N}$ be an ultra asynchronous network which consists of N neurons x_i. Let S be the z-level probabilistic gate that outputs level l_i with the probability p_i ($\sum_i p_i = 1$).

  We simulate it by a probabilistic network $\mathcal{N}'$ with z independent probabilistic gates and $z + 2N$ deterministic neurons as follows:

  1. The independent probabilistic gates a_i ($i = 1 \ldots z$) output 1 with probabilities

     $$\begin{aligned} q_1 &= p_1 \\ q_i &= \frac{p_i}{(1 - \sum_{j<i} p_j)} \qquad i = 2 \ldots N, \end{aligned}$$

     respectively. Note that $q_i \in [0, 1]$ by definition.
  2. The deterministic gates b_i ($i = 1 \ldots z$) update by

     $$b_i = \sigma(a_i - \sum_{j=1}^{i-1} a_j)\ .$$

     For each neuron b_i,

     $$\begin{aligned} \text{Prob}\ [b_i = 1] \quad = \quad & \text{Prob}\ [a_1 = 0, a_2 = 0, \ldots a_{i-1} = 0, a_i = 1] = \\ & (\pi_{j<i}\ \text{Prob}\ [a_j = 0])\ \text{Prob}\ [a_i = 1] = (\pi_{j<i}(1 - q_j))q_i = \\ & p_i\ \ . \end{aligned}$$

     Thus, each b_i may simulate the ith level l_i of the gate S.
  3. The deterministic neurons c_i ($i = 1 \ldots N$) collect the different probabilistic guesses into sets by neurons' identities:

     $$c_i = \sigma(\sum_{j \in\ \mathrm{id}\ (i)} b_j)\ .$$

4. Each of the neurons x_i $(i = 1 \ldots N)$ is now controlled by c_i:

$$x_i^+ \quad = \quad \sigma(\text{ previous update } + Cc_i - C) \ ,$$

where C is a constant larger than any possible value of the previous update.

- $\mathcal{LPN} \subseteq \mathcal{LUAN}$:
  Given a probabilistic network $\mathcal{N}$ with k binary probabilistic gates g_i, which outputs 1 with probability p_i, respectively. (The probabilities p_i are totally independent.) We simulate it by an ultra asynchronous network $\mathcal{N}'$ as follows:

  There are 2^k subsets of the gates g_i that may output 1 simultaneously. Each subset s_i $(i = 1 \ldots 2^k)$ has the probability q_i to appear. The q_i's adhere to $\sum_i q_i = 1$. $\mathcal{N}'$ has a 2^k-level probabilistic gate, each level l_i is associated with subset s_i and appears in probability q_i. We add k deterministic neurons b_i that interpret the values of the levels into the individual k values.

  All neurons have the whole 2^k levels as their identities. In order to be able to use the values of the b_is as the probabilistic gates, there is a slow down in the computation of the constant 2. (That is, x_i's at time $(t + 2)$ use the output of the b_i's from time $(t + 1)$, which were influenced by the gate S at time t.) ∎

Corollary 7.7 The computational power of reliable networks type $\mathcal{UAN}$ and the unreliable ones of type $\mathcal{UFAN}$ is equivalent. That is, networks of the type $\mathcal{UAN}$ are totally robust to failures of their components.

Sketch of Proof: To prove the corollary, we show that the function class associated with $\mathcal{UFAN}$ is a subset of $\mathcal{LPN}$. For this aim, we repeat the above proof, but rather than using N c_i neurons, we now allow for Nd c_{ij} neurons, where d is the number of possible choices of P faulty neurons. Each such c_{ij} represent the id set of neuron i with choice j. The c_{ij} neurons are now used to control the next update of the x neurons.

Corollary 7.8 The computational powers of the totally asynchronous network and the faulty asynchronous network are bounded by that of the probabilistic network.

Now we are ready to prove Theorem 12:
Proof of Theorem 12.

- Given an ultra faulty asynchronous network $\mathcal{N}$ of size N, we simulate it by a $2N$-sequence asynchronous network $\mathcal{N}'$ of size $2N$ as follows:
  - The neurons z_i, $i = 1 \ldots N$ update by the rule $z_i^+ = \sigma(y_i)$. These neurons are deterministic.
  - The neurons y_i $i = 1 \ldots N$ update similarly to the original neurons x_i of $\mathcal{N}$: If $x_i \in \mathcal{N}$ updates by

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

    then

    $$y_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}\mathbf{z_j}(t) + \sum_{j=1}^{M} b_{ij}u_j(t) + c_i\right) \ , \quad i = 1, \ldots, N \ .$$

In addition, faulty x_i is associated with faulty y_i of the same type and probabilities of fault.

Each update of a subset in $\mathcal{N}$ is simulated by updating a sequence that includes the y_i so that x_i was updated (including the dictation of the update choice for each neuron), followed by the sequence $z_1 \ldots z_n$. The probabilities of such sequences match the probability of the associated subset in $\mathcal{N}$, and other sequences has the probability 0 to occur.

- An s sequence faulty asynchronous network $\mathcal{N}$ of size N is simulated by Markov faulty asynchronous network $\mathcal{N}'$ as follows: Let R be the number of possible sequences and their length is bounded by s. Then $\mathcal{N}'$ has an Rs probabilistic gate. Level $l_{i,j}$ implies that the choice of sequence i was made, and currently the jth neuron there updates by the function chosen by the sequence. If j is not the last element of sequence sequence i, the next element in i has 1 probability to occur. If the last neuron in a sequence is being updated, the probabilities to start new sequences are according to the probabilities of the sequences in $\mathcal{N}$.
- We end this proof by simulating a Markov faulty asynchronous network by a Markov faulty network. This is very similar to the proof in Corollary 7.7 and thus is excluded. As Markovian faulty networks are computationally bounded by the class $\mathcal{LPN}$, the theorem results. ∎

8 Proofs

This section includes the two longest proofs of this paper.

8.1 The Proof of Turing Machine with Real Probabilities

Theorem 4 Let p be a real number in $[0,1]$; ϵ, ϵ_1 are in $(0, .5)$. Denote by $\mathcal{LPTM}[\{p\}](T(n))$ the class of languages recognized by probabilistic Turing Machines over the probability p and a bounded error ϵ in time T. Then $\mathcal{LPTM}[\{p\}]$ and Pref-PTM/log are polynomially related.

Proof:

- $\mathcal{LPTM}[\{p\}]$ is polynomially included in Pref-PTM/log :
  Assume a probabilistic Turing Machine M over the probability p that ϵ-recognizes the language L in time $T(n)$. We show a probabilistic Turing machine M', having a fair coin, that upon receiving a $c\log(T(n))$ long advice which is a prefix advice, M' ϵ'-recognizes L in time $O(T(n)\log(T(n)))$ (c is a constant).

  Assuming the probability p in a binary expansion, and let p' is a rational number that its first $c\log(T(n))$ bits after the decimal point are like those of p and the rest are 0. The advice of M' consists of the bits of of p' in the order from left to right.

  One coin flip by M is simulated by a *binary conjecture* of M' which is based on $c\log(T(n))$ of M' flipping its fair coin: M' tosses its fair coin $c\log(T(n))$ times and compares the resulting guessed string with the advice to make a binary conjecture. If the guessed string precedes the advice in the lexicographic order, M' conjectures 0, and 1 otherwise.

  The probability of error in M' is the probability that it generates a sequence of conjectures which yields a wrong decision. Denote by R the sequence $T(n)$ binary conjectures that M'

generates; B is the set of bad sequences of this length. As the error of M is bounded by ϵ, the cardinality of B is bounded by $2^{T(n)}\epsilon$. We conclude:

$$\begin{aligned}\text{Prob } (R \in B) &= \sum_{s\in B} \text{Prob } (R = s) \leq |B| \max_{s\in B} \text{ Prob } (R = s) \\ &\leq 2^{T(n)}\epsilon \max_{s\in B} \text{ Prob } (R = s)\end{aligned}$$

Assume w.l.o.g. that $p < \frac{1}{2}$. Then, the most probable sequence for M' to guess is the $0^{T(n)}$ sequence. As

$$|p' - p| \leq \frac{1}{T(n)^c} \quad ,$$

the total error of M' is bounded by

$$2^{T(n)}\epsilon(p + \frac{1}{T(n)^c})^{T(n)} = (2p)^{T(n)}\epsilon(1 + \frac{1}{pT(n)^c})^{T(n)} \; .$$

Using the approximation

$$(1 + x_n)^n \approx e^{nx_n} \tag{8}$$

for small x_n, we approximate the above as

$$\approx (2p)^{T(n)}\epsilon e^{T(n)(pT(n))^{-c}}$$

which is bounded by an error ϵ' for $c > 1$.

- Pref-PTM$(T(n))/log \subseteq \mathcal{LPTM}^{\epsilon}[\{p\}](O(T(n)^2))$:
  Given a probabilistic Turing Machine M with a fair coin and a prefix logarithmic long advice A that ϵ-recognizes L in time $T(n)$. We show a probabilistic Turing Machine M' over some real number p that ϵ'-recognizes the same language L in $O(T^2(n))$ time.

  We define p as follows: The binary expansion of p starts with .01, i.e. $\frac{1}{4} \leq p \leq \frac{1}{2}$. The following k bits are the advice of size k of M.

  M' computes in two phases:

  1. In the first phase, M' guesses the advice sequence A of M by tossing its unfair coin $z = O(T^2(n))$ many times: Let #1 be the number of 1's found in z flipping, and define

     $$p' = \frac{\#1}{z}$$

     We next show that $|p - p'|$ is bounded by a small constant. By Chebychev formula, for all $\epsilon > 0$, Prob $(|x - \mu|) > \epsilon) \leq \frac{v}{\epsilon^2}$ where μ is the expectation an v is the variance of the random variable x. The expectation of such z independent Bernuli trials is $\mu = zp$ and the variance is $\nu = zp(1 - p)$. Particularly, for all $\epsilon > 0$,

     $$\text{Prob } \; (|zp' - zp| > \epsilon) \leq \frac{zp(1-p)}{\epsilon^2} \; .$$

     As $p(1 - p) \leq \frac{1}{4}$, by choosing $\epsilon = \sqrt{25z}$, we get

     $$\text{Prob } \left(|p' - p| > \sqrt{\frac{25}{z}}\right) \leq \frac{1}{100} \; .$$

     Thus, if in the first phase, M' tosses its coin $z = 25T^2(n)$ times then the advice is reconstructed with an error probability bounded by $\frac{1}{100}$.

2. In the second phase, M' simulates each step of M as follows. Each coin flip of M is simulated by the following algorithm:
   (a) Toss the coin twice and increment the counter c.
   (b) If the results are "01" decide 0, and decide 1 if they are "10".
   (c) If the results are either "00" or "11" and the counter c is less than $T(n)$, go to (a).
   (d) Decide 0. (when the counter is $T(n)$ and decision was not yet made.)

   Then M' acts by its decision. This algorithm takes up to $T(n)$ steps for each step of M. Next, we compute the probability of M' to guess a bad sequence of coin flips. As above, we denote the set of bad guesses by B; R is the guess sequence of length $T(n)$ generated by M'.

$$\text{Prob }(R \in B) = \sum_{s \in B} \text{ Prob }(R = s) \leq |B| \max_{s \in B} \text{ Prob }(R = s)$$

   - The probability of getting in two successive coin flips the values "00" or "11" is $p' = p^2 + (1-p)^2$. Thus, the probability of ending a "step" in the fourth step of the algorithm is bounded by $p'' = p'^{T(n)}$. As $\frac{1}{4} \leq p \leq \frac{1}{2}$, $p' \leq \frac{5}{8}$ and $p'' \leq \frac{5}{8}^{T(n)}$.
   - The probability of ending one "step" in decision 0 is: $\frac{1}{2}(1-p'') + p'' = \frac{1}{2} + \frac{p''}{2}$.
   - $\max_s$ Prob $(R = s) \leq$ Prob $(R = 0^{T(n)}) = (\frac{1}{2} + \frac{p''}{2})^{T(n)}$.

   Thus,

$$\text{Prob }(R \in B) \leq 2^{T(n)} \epsilon (\frac{1}{2} + \frac{p''}{2})^{T(n)} \leq \epsilon (1 + p'')^{T(n)} \approx \epsilon e^{T(n) \frac{5}{8}^{T(n)}} < \epsilon_2 \ .$$

The probability ϵ' that M' is incorrect is bounded by

$$\text{Prob(“wrong advice sequence”)} + \text{Prob(“bad guess sequence”)} \leq \frac{1}{100} + \epsilon_2$$

which is bounded. ∎

8.2 The Power of Probabilistic Networks With Real Weights

Theorem 5 Denote by $\mathcal{N}_r(T(n))$ the class of languages recognized by networks in time $T(n)$, and by $\mathcal{LPN}_r(T(n))$ the class of languages ϵ-recognized by probabilistic real networks in time $T(n)$. Then

$$\begin{aligned} \mathcal{LN}_r(T(n)) &\subseteq \mathcal{LPN}_r(T(n)) \\ \mathcal{LPN}_r(T(n)) &\subseteq \mathcal{LN}_r(n^2 + nT(n)) \ . \end{aligned}$$

Proof. (* of theorem (5) *)
The first inclusion is trivial. We prove the second one in two steps. Assume a probabilistic network that ϵ-recognizes a language L. First step describes a non-uniform family $\mathcal{F}$ of feedforward networks that recognizes the same language L; this requires almost no slow down in the computation. The second step, describes a deterministic recurrent network that simulates the family $\mathcal{F}$, with a polynomial slow down only.

- **Step 1:**

Lemma 8.1 Let $0 < \epsilon < .5$ and L is the language that is ϵ-recognized by a probabilistic network $\mathcal{PN}$ of size N in time T. Then, L can also be recognized by a family of non-uniform feedforward networks $\mathcal{F} = \{\mathcal{N}'_n\}_{i=1}^{\infty}$ of depth $T(n)+1$ and size $cNT(n)+1$, where c is the constant

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

Proof. (* Proof of Lemma (8.1) *)
Assume $\mathcal{PN}_r$ is a network that ϵ-recognizes a language L. We show the existence of a family of networks that recognizes the same language.

Suppose that $\mathcal{PN}_r$ has r probabilistic gates g_k, $1 \leq k \leq r$, each outputs 1 with probability p_k. For a given input of depth n, by unfolding the network to length $T(n)$, we get a feedforward network with $r'_n = T(n)r$ probabilistic gates. Denote this feedforward probabilistic nets by $\mathcal{N}_{1,n}$.

We pick a string

$$\rho^{i,n} = \rho^i_1 \rho^i_2 \cdots, \rho^i_{r'_n} \ \in \{0,1\}^{r'_n}$$

at random with probability $p_{(j \bmod r)}$ that ρ^i_j is 1. Let $\mathcal{N}_{1,n} < \rho^{i,n} >$ be a deterministic feedforward net similar to $\mathcal{N}_{1,n}$, but with the string ρ^i substitutes the probabilistic gates (i.e., ρ^i_j substitutes $g_{j \bmod r}$ in level $(j \text{ div } r)$. We now pick randomly cn such strings

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

at random. The feedforward net $\mathcal{N}^{\rho[n]}_n$ consists of the cn subnetworks $\mathcal{N}_{1,n} < \rho^{i,n} > (i = 1 \ldots cn)$ and one "majority gate" in the final level, which takes the output of those cn subnetworks as its input. That is, for each n, the network $\mathcal{N}^{\rho[n]}_n$ computes the majority over cn random runs of the probabilistic net $\mathcal{PN}_r$.

We compute the probability that $\mathcal{N}^{\rho[n]}_n$ outputs wrongly on an input ω of length n. By definition of $\mathcal{PN}$, each $\mathcal{N}^{\rho_{i,n}}_{1,n}$ has the probability ϵ of being wrong. Thus, picking $\rho = (\rho^1, \ldots, \rho^{cn})$ at random, the probability of having a failure in the network $\mathcal{N}^{\rho[n]}_n$ equals the probability of being wrong in at least $cn/2$ independent Bernuli trials, having the failure probability ϵ each. We using the bound $B(\frac{cn}{2}, cn, \epsilon) \leq 4(\epsilon(1-e)^{cn/2}$, and choose $c = \lceil \frac{8\epsilon \ln 2}{(1-2\epsilon)^2} \rceil$ to get

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

Thus, for all the inputs of length n the sum of failures is less then 1. Hence, there must be at least one choice of random strings (ρ‘$[n]$) that makes $\mathcal{N}'_n \equiv \mathcal{N}^{\rho'[n]}_n$ recognizes correctly any input of length n. ∎

- **Step 2:**

Lemma 8.2 Any language that is recognized by the family $\mathcal{F} = \{\mathcal{N}'_n\}$ described above can also be recognized by a deterministic recurrent network $\mathcal{N}$. Furthermore, the strings of length n that where recognized by network $\mathcal{N}'_n$ (with depth $T(n)$) can be recognized by $\mathcal{N}$ in time $O(n^2 + nT(n))$.

Proof. The family $\mathcal{F}$ has a very special structure. Each member $\mathcal{N}'_n$ of this family can be described as a three-tuple

$$(\mathcal{N}_2, n, \rho'[n])$$

where $\mathcal{N}_2$ is a deterministic recurrent network, n is the index of the network, and $\rho'[n] \in \{0,1\}^{r'_n cn}$. The network $\mathcal{N}_2$ is the same for all family members of $\mathcal{F}$.

We use the "Cantor set encoding" introduced in [SS49] and encode $\rho'[n] = (\rho^{1,n}, \ldots \rho^{cn,n})$ as a of cn binary strings $\in \{0,1\}^{cnr'_n}$. The circuit family is encoded via two sequences

$$\begin{aligned} c_1 &= \rho_1 2 \rho_2 2 \rho_3 2 \rho_4 \cdots \\ c_2 &= \text{any Cantor encoding of the network } \mathcal{N}_2 \in \{0,1,2\}. \end{aligned}$$

Let $c = c_1 c_2 \cdots$ a finite or infinite string over the alphabet $\{0,1,2\}$. Denote by $C|_6$ the value $\sum_{i=1}^{|c|} \frac{2c_i+1}{6^i}$. We construct the recurrent network $\mathcal{N}$ that has the constants (weights) $c_1|_6$, and $c_2|_6$, and recognizes the language L.

1. $\mathcal{N}$ reads the input ω and measures its length.
2. $\mathcal{N}$ retrieves the encoding $\rho'[n]$ from the constant $c_1|_6$. (It takes $\sum_{j\leq n} |\rho'[j]|$ time as was proven in [SS93].)
3. $\mathcal{N}$ executes the code:

Func Net $(\omega, \rho'[n], n, c_2)$;
Var z: counter,
Yes, No, A: Boolean,
ρ: Real;
Begin
z=1 ;
Repeat
$\rho \leftarrow \text{retrieve}(c_1, i, n)$ % retrieving $\rho^{i,n}$
$A \leftarrow$ simulate (c_2, ρ, ω)
If $A = 1$ **then** Inc(Yes) **else** Inc(No)
$z = z + 1$
Until $(z > cn)$
Net $\leftarrow$ Return(Yes > No)
End

The command "simulate" is possible due to previous work on Turing Machine equivalence [SS49]. Furthermore, it was shown in [Sie93] how to construct a net from this type of high-level language. Thus, $\mathcal{N}$ is as required by lemma 8.2, and theorem 5 is proven. ∎

9 Addendum and Open Problems

We introduced and analyzed different faulty and asynchronous models which are more biologically adequate than the deterministic ones. In addition to the above results, we can also show that a deterministic network which is augmented with probabilistic gates of the following special type:

$$g(\alpha,\beta,p,x) = \begin{cases} 0 & x < \alpha \\ 1 \text{ with probability } p & \alpha \leq x \leq \beta \\ 1 & x \geq \beta \end{cases}$$

(where $0 \leq \alpha \leq \beta \leq 1$) is equivalent in power to the probabilistic networks discussed in this paper. The proof uses Lemma 5.3. We leave as an open problem the computational power of a network augmented with probabilistic gates of the type

$$g(b,x) = 1 \text{ with probability } b(x)$$

where $b(x)$ is a monotone non-decreasing function so that $\lim_{x\mapsto\infty} b(x) = 1$ and $\lim_{x\mapsto-\infty} b(x) = 0$.

Another unsolved question deals with the exact power of asynchronous reliable networks and asynchronous nets with the independent error probabilities assumption.

10 Acknowledgment

I with to thank Joe Kilian for his help with the proof of Theorem 4. Much gratitude are given to Eduardo Sontag for suggesting to look at these models and providing great comments during the first steps of this research. Appreciations are given to Eric Allender and Ricard Gavalda for helpful discussions.

References

[Ade78] L. Adelman. Two theorems on random polynomial time. In *IEEE Sympos. on Founcations of Computer Science*, volume 19, pages 75–83, New-York, 1978.

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

[BG81] C. G. Bennet and J. Gill. Relative to a random oracle a, $p^a \neq np^a \neq$ co-np^a with probability 1. *SIAM J. on Computing*, 10:96–113, 1981.

[BGSS93] J. L. Balcázar, R. Gavaldà, H.T. Siegelmann, and E. D. Sontag. Some structural complexity aspects of neural computation. In *IEEE Structure in Complexity Theory Conference*, pages 253–265, San Diego, CA, May 1993.

[BHM92] J. L. Balcázar, M. Hermo, and E. Mayordomo. Characterizations of logarithmic advice complexity classes. *Information Processing 92, IFIP Transactions A-12*, 1:315–321, 1992.

[BSS89] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: Np completeness, recursive functions, and universal machines. *Bull. A.M.S.*, 21:1–46, 1989.

[HKP91] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, 1991.

[Koi93] P. Koiran. A weak version of the Blum, Shub & Smale model. In *Proc. 34^{th} FOCS Symp.*, pages 486–495, 1993.

[KS93] J. Kilian and H.T. Siegelmann. On the power of sigmoid neural networks. In *Proc. Sixth ACM Workshop on Computational Learning Theory*, Santa Cruz, July 1993.

[Orp92] P. Orponen. Neural networks and complexity theory. In *Proc. 17th Symposium on Mathematical Foundations of Computer Science*, pages 50–61, 1992.

[Par92] I. Parberry. Knowledge, understanding, and computational complexity. Technical Report CRPDC-92-2, Department of Computer Sciences, University of North Texas, Feb 1992.

[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*, 34:194–197, 1988.

[Pip89] N. Pippenger. Invariance of complexity measure ofr networks with unreliable gates. *J. ACM*, 36:531–539, 1989.

[Pip90] N. Pippenger. Developments in: The synthesis of reliable organisms from unreliable components. In *Proc. of symposia in pure mathematics*, volume 5, pages 311–324, 1990.

[RO77a] Dobrushin R.L. and S.I. Ortyukov. Lower bound for the redundancy of self-correcting arrangement of unreliable functional elements. *Problems info, Transmission*, 13:59–65, 1977.

[RO77b] Dobrushin R.L. and S.I. Ortyukov. Upper bound for the redundancy of self-correcting arrangement of unreliable functional elements. *Problems info, Transmission*, 13:346–353, 1977.

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

[Sie93] H.T. Siegelmann. *Foundations of Recurrent Neural Networks*. PhD thesis, Rutgers University, 1993.

[SS49] H. T. Siegelmann and E. D. Sontag. On computational power of neural networks. *J. Comp. Syst. Sci*, to appear. Previous version appeared in "Proc. Fifth ACM Workshop on Computational Learning Theory," Pittsburgh, July 1992: 440-449.

[SS91] H. T. Siegelmann and E. D. Sontag. Turing computability with neural nets. *Appl. Math. Lett.*, 4(6):77–80, 1991.

[SS93] H. T. Siegelmann and E. D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, to appear. A preliminary version in: The second Israel Symposium on Theory of Computing and Systems, Natanya, Israel, June, 1993.

[vN56] J. von Neumann. Probabilistic, logics and the synthesis of reliable organisms from unrliable components. In C.E. Shannon and J.McCarth, editors, *Automata Studies*. Princeston U. 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. *Information and Control*, 54:143–154, 1982.