Doctoral Thesis ETH No. 14987

A Network Implementation of a Markov Model

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZÜRICH
for the degree of
DOCTOR OF NATURAL SCIENCES

presented by
ALESSANDRO USSEGLIO VIRETTA
Dottore in Fisica, Università degli Studi di Torino - Italy
born September, 15th 1970 in Torino
citizen of Italy

accepted on the recommendation of
Prof. Dr. Rodney Douglas
Dr. Shih-Chii Liu
Prof. Dr. Walter Senn
Dr. Stefano Fusi

2003

Contents

List of Figures

List of Tables

List of Boxes

Abstract

The processing of spatiotemporal patterns plays an important role in the life of humans and animals. However, the mechanisms that allow biological systems to learn, generate, recognize and predict sequences are poorly understood. In this thesis we show that a Hopfield-like network can emulate a Markov model of a sequence of stimuli. Various researchers have proposed that sequences of stimuli create, in the cerebral cortex, attractors of network dynamics. These attractors have spatial correlations with the patterns of activity elicited by temporally adjacent stimuli, so that temporal correlations are transformed into spatial correlations. Because of their simplicity and flexibility, Hopfield-like networks have been extensively used to model several of these phenomena. Simulations showed that transitions between attractors that are highly correlated are faster than the transition between less correlated attractors. A similar mechanism might underlie the effect of priming, where a stimulus is recognized in a shorter time if it is preceded by a cognitively correlated stimulus rather than by a non-correlated one. In the presence of noise, transitions can occur spontaneously. In this case, the probability of transition between different attractors is supposed to increase with the correlation of the attractors. The dependence of the transition probabilities between attractors as a function of the attractors correlation and network parameters is an important and interesting issue that has not yet been given careful consideration. For this reason, we investigated the properties of a Hopfield-like neural network, consisting of binary neurons and binary stochastic synapses. The states of the neurons are updated using the *Glauber's* dynamics, and the synaptic weights are potentiated or depressed according to a Hebbian learning rule. The learning rule includes an asymmetric term that links patterns of activity elicited by successive stimuli. When the noise level is within a given range, the attractors are metastable and the overlaps of the network state with the attractors change in time. We show that the network can learn, in an unsupervised way, the transition probabilities from a sequence of stimuli. The unsupervised network extracts the transition statistics of input sequences generated by a Markov process and encodes them in its synaptic matrix. When an input triggers the recall process, the network generates a temporal pattern sequence whose transition statistics reflects those of the input sequences used in training. A sequence generated by a non-Markov process can also be learned, provided that the input sequence is pre-processed. The pre-processing involves the mixing of the external stimuli with the current activity state of the network. This mechanism codes the temporal history of the input sequence, thus allowing the network to process history-dependent sequences. Our detailed study of the transition probabilities of the network state show that a Hopfield-like network with a stochastic learning rule, in the presence of noise can capture the temporal correlation statistics of input sequences.

Zusammenfassung

Die Verarbeitung von raumzeitlichen Mustern spielt eine wichtige Rolle im Leben von Menschen und Tieren. Trotzdem sind die Mechanismen, die es biologischen Systemen erlauben, Sequenzen zu erlernen, zu generieren und vorherzusagen, wenig verstanden. In dieser Arbeit zeigen wir, dass einem Hopfieldähnlichen Netzwerk ein Markov-Modell von Stimulisequenzen nachahmen kann. Mehrere Forscher haben vorgeschlagen, dass Sequenzen von Stimuli im cerebralen Kortex Attraktoren der Netzwerkdynamik erzeugen. Solche Attraktoren zeigen räumliche Korrelationen, mit denen durch zeitlich anliegende Stimuli hervorgerufene Aktivitätsmuster in räumliche Korrelationen transformiert werden. Wegen ihrer Einfachheit und Flexibilität wurden Hopfieldähnliche Netzwerke schon mehrfach benutzt, um viele dieser Phänomene zu modellieren. Diese Simulationen haben ergeben, dass Übergänge zwischen stark korrelierten Attraktoren schneller geschehen als Übergänge zwischen schwach korrelierten Attraktoren. Ein ähnlicher Mechanismus könnte dem Effekt des 'Priming' unterliegen: Ein Stimulus wird, wenn ihm ein kognitiv korrelierter Stimulus vorausgeht, in einer kürzeren Zeit erkannt als bei einem nicht korrelierten Stimulus. Wenn Rauschen vorhanden ist, können Übergänge spontan geschehen. In diesem Fall nimmt man an, dass sich die Übergangswahrscheinlichkeit zwischen verschiedenen Attraktoren mit der Attraktorenkorrelation erhöht. Die Abhängigkeit der Übergangswahrscheinlichkeit zwischen Attraktoren von Attraktorenkorrelation und Netzwerkparametern ist eine wichtige und interessante Frage, die jedoch noch nicht sorgfältig untersucht wurde. Aus diesem Grunde haben wir die Eigenschaften eines Hopfieldähnlichen Netzwerks, das aus binären Neuronen und binären stochastischen Synapsen besteht, untersucht. Die Zustände der Neuronen unterliegen der Glauber-Dynamik, und die synaptischen Gewichte werden nach einer Hebb'schen Lernregel potenziert oder abgeschwächt. Die Lernregel beinhaltet einen zeitlich asymmetrischen Teil, der Aktivitätsmuster, die durch darauffolgende Stimuli hervorgerufen werden, verbindet. Wenn der Rauschpegel innerhalb eines bestimmten Bereichs liegt, dann sind die Attraktoren metastabil, und die Überlappung des Netzwerkzustandes mit den Attraktoren verändert sich mit der Zeit. Wir zeigen, dass das Netzwerk die Übergangswahrscheinlichkeiten von einer Stimulussequenz *unsupervised* lernen kann. Das unbetreute Netzwerk extrahiert die Übergangsstatistik von Stimulussequenzen, die von einem Markov-Prozess generiert werden, und kodiert sie in seiner synaptischen Matrix. Wenn ein Stimulus den Erinnerungsprozess startet, generiert das Netzwerk eine zeitliche Mustersequenz, dessen Übergangsstatistik jene aus dem Training benutzte Stimulussequenzen reflektiert. Auch eine von einem nicht Markov'schen Prozess generierte Sequenz kann gelernt werden, vorausgesetzt dass die Stimulussequenz vorbearbeitet wird. Die Vorbearbeitung beinhaltet das Mischen der externen Stimuli mit dem aktuellen Aktivitätszustand des Netzwerks. Dieser Mechanismus kodiert den zeitlichen Ablauf der Inputsequenz und ermöglicht so dem Netzwerk die Bearbeitung von Sequenzen in Abhängigkeit der Vorgeschichte. Unsere detaillierte Studie über die Netzwerkübergangswahrscheinlichkeiten zeigt, dass ein Hopfieldähnliches Netzwerk mit einer stochastischen Lernregel, bei Vorhandensein von Rauschen, die zeitliche Korrelationsstatistik von Stimulussequenzen gewinnen kann.

Chapter 1

Introduction

In his paper "Can People Predict Chaotic Sequences?", the Australian psychologist Richard Heath [6] claims that some people can identify patterns in chaotic systems and predict their behavior. Although Heath's finding still need confirmation, it is clear that the processing of spatiotemporal patterns plays an important role in the life of humans and animals. For this reason, we believe that the neural structures that perform such tasks deserve a special attention. The mechanisms which let biological systems learn, recognize, generate, and predict spatiotemporal patterns are still poorly understood. Spatiotemporal patterns are here considered to be sequences of activity patterns of a set of neurons providing input to a neural network. Typically, research on artificial neural networks has focused on learning fixed sequences. Our work, inspired by the seminal paper by Griniasty et al. [7], focuses instead on the extraction of transition *probabilities* and their encoding in the connectivity pattern of a network. Griniasty showed that a Hopfield-type network provided with a slightly modified connectivity matrix, has attractors that are correlated with several learned patterns. The classical Hopfield synaptic matrix stores a set of patterns, which become attractors of the dynamics [8]. The modification introduced by Griniasty store in the matrix a fixed sequence of uncorrelated patterns. In this case, the network dynamics relax to attractors that have non-zero overlap with more than one pattern. This result provides a possible explanation to the phenomena observed by Miyashita and collaborators.

In the first experiment by Miyashita [9], a monkey is trained to recognize and match a set of visual stimuli. As a result, a stimulus-selective delay activity of neurons from the anterior ventral part of the temporal cortex was measured. The delay activity started in some neurons up to a few seconds after the presentation of the stimulus and lasted, sometimes without decline, up to 16 seconds. Moreover, the firing frequencies of the measured neurons during the delay activity were not necessarily correlated with those measured during the presentation. For these reasons, the delay activity probably represents a sort of memory trace, and not only a continuation of the stimulus-induced activity. The second study by Miyashita [10] discovered spatial correlations among the delay activity firing patterns. Being the visual stimuli spatially uncorrelated, the correlations were explained as being a consequence of the fixed temporal order of the presentations. So, the monkey's brain is apparently able to convert temporal correlations (in this case, temporal proximity) into spatial correlations of the attractors of the network dynamics. The delay activities were found to be stimulus-dependent, and the stimuli eliciting activity in a neuron were usually nearest neighbors in the presentation sequence during the training. These two facts made the authors hypothesize that the "selectivity acquired by these cells represents a neuronal correlate of associative long-term memory of pictures" [10].

Griniasty showed that the overlap of the attractor with the patterns is a function of the separation, in the presentation sequence, from the pattern having maximum overlap. Griniasty additionally showed that this function has the same qualitative behavior as showed by Miyashita [10]. Yakovlev et al. [11] went one step further in the modeling of these phenomena, by proposing a possible mechanism for the formation of the delay activities and correlations observed. In the model presented, the memories are formed in three stages: During the first one, uncorrelated attractors are built up. The attractors of the dynamics allow for the sustained level of firing rate observed during the delay activity. During the second stage, the delay activity leads to correlations between the delay activity patterns elicited by the presentation of stimuli contiguous in time. Eventually, the delay activity corresponding to neighboring stimuli show a higher level of correlations than the activities elicited by patterns more apart in the training sequence. Yakovlev uses a network of excitatory and inhibitory neurons. The presentation of visual stimuli in Miyashita's experiments corresponds in his model to an extra current injected in a sub-population of neurons. The activity produced by the external input, in combination with a Hebbian learning rule, strengthens the synaptic links between neurons active in correspondence to the same stimulus. After enough repetitions of the same stimulus, an attractor is formed, and the network can sustain an enhanced level of activity even in absence of an external input current. The formation of attractors can be observed as a stimulus-specific delay activity. Because of the presence of a global inhibition, the delay activity ends with the presentation of a different stimulus. The short ($<$100ms) overlap of the delay activity induced by a stimulus and the activity induced by the following one is sufficient to allow Hebbian strengthening of the synapses between the two neuron sub-populations. The final result is that, after a sufficient number of repetitions of the stimuli sequence, the neurons show a sustained level of activity also for the stimuli nearest neighbor of their preferred one. It is important to notice that this chain of events occurs automatically, independently from the relevance of the behavioral task. In this context, the simulation by means of an artificial neural network is key for the understanding of the mechanism leading to the formation of attractors and associative memories, and sheds light on the ways by which information about the temporal order of stimuli can be encoded in the activity of populations of neurons.

Griniasty ends his paper [7] with some speculations about the computational and behavioral utility of such synaptic development. One interesting suggestion is that the correlations between attractors and patterns might underlie the effect of priming. Priming is an effect commonly encountered in experiments involving the recognition of stimuli. The reaction time can be shortened if the stimulus to be recognized is preceded by a cognitively correlated pattern. This effect can be translated into the language of network dynamics: Let us suppose the network is presented a stimulus, after which the network relaxes to an attractor. When a new stimulus is presented, the network will move to another basin of attraction and eventually relax to another attractor. The transitions between attractors that are highly correlated is faster than the transition between less correlated attractors [12, 13]. In experiments of the Miyashita type, one would expect the transitions time between attractors to increase with the distance in the stimulus presentation sequence. An important extension is that, in the presence of noise, transitions can occur spontaneously, as described by Buhmann and Schulten [2]. In this case, the probability of transition between different attractors increases with the correlation of the attractors. Finally, Griniasty theorizes about the possibility to provoke transitions by random activation of the neural network. We followed this track by investigating further the properties of a stochastic Hopfield-like neural network, with particular regard to the probabilities of transition between different attractors and their dependence on the network parameters. In the next chapters, we show that a neural network can learn in unsupervised manner the transition probabilities from a sequence of stimuli. If the network is provided with a suitable level of noise, it can generate sequences of patterns (each pattern corresponding to a stimulus) whose transition probabilities closely reflect those of the sequence used during the training.

Learning stimuli sequences is a behaviorally relevant task: For example, when navigating in an unfamiliar environment, we can find the correct path by remembering a sequence of landmarks. Each landmark is used as a cue to lead us to the next one. Another relevant example is described by Bartlett

and Sejnowski, in their paper about "learning viewpoint invariant face representations from visual experience in an attractor network" [14]: Here they show how learning temporal sequences of stimuli can lead to viewpoint invariant representations of faces. Normally different views of an object (or a face, in this case) are perceived in close temporal proximity, as the object is manipulated or the face moves or changes expression. The authors devised an attractor neural network learning rule by which temporally neighboring stimuli are associated into the same basin of attraction. In this way, multiple views lead the network into the same attractor. This is equivalent to say the system has, within a certain approximation, a viewpoint invariant representation of the object (or face).
It would be easy to include several other examples: Indeed, the processing of temporal sequences of stimuli has a high relevance for animal and human behavior. Language, musical, manual, and more abstract skills rely on the learning, recognition, prediction, and generation of temporal sequences of stimuli. For this reason, we designed and analysed the properties of an extension of the known models [7, 11]. The network is shown to be able to learn in unsupervised manner sequences of stimuli, and generate sequences with statistical properties similar to those used during the training. Prediction and recognition abilities are two other features of the system that will be described in the next chapters.

Sequences can be better understood using the formalism of Markov processes and chains: Given a random variable X and the arbitrary times $\ldots < n-2 < n-1 < n$, a stochastic process is a Markov process if [15]

$$\text{prob}(X_n = x | X_{n-1} = y, X_{n-2} = z, \ldots) = \text{prob}(X_n = x | X_{n-1} = y). \quad (1.1)$$

A process is a *non-Markov* process or a Markov process of order $k > 1$ if

$$\text{prob}(X_n = x_n) = \text{prob}(X_n = x | X_{n-1} = x_{n-1}, \ldots, X_{n-k} = x_{n-k}).$$

We have developed a recurrent network that is able to extract the transition statistics from pattern sequences. The unsupervised network extracts the transition statistics of an input sequence generated by either a Markov or non-Markov process. When an input triggers the recall process, the transition statistics of the pattern of activity of the network reflect those of the input sequences.

In the following we review the use of neural network for the processing of pattern sequences. Firstly, Hopfield-like networks are described. Both networks with dynamic synapses and noise driven networks with instantaneous synapses are reviewed in Sections 1.2.1 and 1.2.2. Attempts have been done to solve grammar inference and prediction tasks using neural networks.

Some approaches are reviewed in Section 1.2.4. Section 1.2.5 presents an overview of the mostly used connectionist architectures used in conjunction with sequences generated by Markov and non-Markov processes. Two recent and innovative connectionist models are described in Section 1.2.6, including Hopfield's most recent work. The algorithmic touchstone for the processing of temporal sequences of pattern is provided by hidden Markov models. Hidden Markov models are the most efficient algorithmic method to model temporal sequences. In Section 1.3 hidden Markov models are briefly reviewed.

1.1 The associative memory problem

The associative memory problem, as defined by Hertz et al. [16] is to :

> Store a set of patterns ξ_i^μ in such a way that when presented with a new pattern ξ_i, the network responds by producing whichever one of the stored patterns most closely resembles ξ_i.

There is, of course, more then one way to solve this problem. One simple solution requires the definition of a distance between two patterns and an algorithm finding the pattern ξ_i^μ minimally distant from ξ_i. Our interest does not focus on algorithms, but on *dynamical systems* capable of solving this sort of problems. In particular, we are interested in systems of relatively simple interconnected units, called neural networks. Each neuron is, by itself, able to solve only one simple task, but, when several of them are connected to each other, the system as a whole develops *emergent* properties which are not to be found in the single components.

In the basic Hopfield [8] model of associative memory, the neurons can be in either of two states: $\xi = +1$ (firing) and $\xi = -1$ (quiescent, or not firing)[1]. The dynamics of the network is defined in this way:

$$S_i(t + \delta t) := \text{sgn}\left[\sum_j w_{ij} S_i(t) - \theta\right] \tag{1.2}$$

where the sign function sgn is

$$\text{sgn}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{1.3}$$

The update rule 1.2 can be applied sequentially to all neurons (*asynchronous* update) or to all neurons at the same time (*synchronous* update).

[1]Alternatively, one can chose to assign the firing and not firing state to, respectively, $\eta = +1$ and $\eta = 0$.

The condition of stability for one pattern ξ is $\text{sgn}[\sum_j w_{ij}\xi_j] = \xi_i, \ \forall i$. This condition is satisfied for $w_{ij} \propto \xi_i\xi_j$. If fewer of the half of the bits of the starting pattern S_i are wrong, the network will relax to ξ_i, correcting in this way the initial errors. The pattern ξ_i is called an *attractor* of the dynamics. In the case of p patterns, the synaptic matrix w_{ij} can be written as the superposition of the term to be used in the case of single patterns [16]:

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu, \ i \neq j \tag{1.4}$$

1.2 Neural networks for pattern sequences

In the following, we will mainly deal with the processing of pattern sequences. Hertz et al. (see [16], page 177) distinguish three different tasks: Sequence *recognition*, *association* and *reproduction*. Patterns recognition and association consist in the production of, respectively, a certain output pattern or pattern sequence after a specific sequence is presented to the network. A network can reproduce a pattern sequence, if it is able to generate the whole sequence after having been shown a part of it, working in this way as a sort of *content-addressable* memory for pattern sequences.

1.2.1 Synapses with dynamic memory

The Hopfield network can only be used as content-addressable memory for *static* patterns. Sompolinsky and Kanter [1] were the first to propose a neural network, based on the Hopfield model, capable of temporal association[2]. Sompolinsky and Kanter designed a network with symmetric bonds between neurons and synapses with dynamic memory. The network state moves along a sequence of temporarily stable states. Each state is stable over an average time period τ, which is the characteristic time constant of the synapse memory. In this model, the neurons are coupled by synapses described by two synaptic matrices. The first is the synaptic matrix $w_{ij}^{(s)}$ described in equation 1.4. The second is the asymmetric matrix

$$w_{ij}^{(a)} = \frac{\lambda}{N}\sum_{\mu=1}^{q} \xi_i^{\mu+1} \xi_j^\mu, \ i \neq j. \tag{1.5}$$

The last pattern can be followed by first one by defining $\xi_i^{q+1} = \xi_i^1$. In this way, a pattern loop is created.

[2]A similar model was independently developed by Kleinfeld [17].

The use of the weight matrix defined in equation 1.5 had been proposed by Hopfield [8] too. The network dynamics can be described by two regimes that depend on the magnitude of λ. For small λ, all patterns are stable and no transitions occur. For large λ, the transitions do occur, but they are too fast and eventually the network ends up in a state which has overlaps with all patterns. Sompolinsky and Kanter solve this problem by splitting the fields on the neurons in an instantaneous component $h_i^{(s)}(t) = \sum_j w_{ij}^{(s)} S_j(t)$, and in a retarded component $h_i^{(a)}(t) = \sum_j w_{ij}^{(a)} \bar{S}_j(t)$, where $\bar{S}_j(t) = \int_{-\infty}^{t} m(t-t')S_j(t')dt'$. The function $m(t)$ represents the memory of the synapses and is characterized by a time-decay constant τ. If the networks state S corresponds at time t_0 to the pattern ξ_i, a transition from pattern ξ_i to pattern ξ_{i+1} will be induced only after a period of time $\Delta t = t - t_0 \approx \tau$ (See Figure 1.1). Peretto and Niez [18], Nebenzahl [19] and Dehaene et al [20] have proposed different solutions. Instead of delayed synapses, they make use of synapses that operate after a delay. The result is similar to that of Sompolinsky and Kanter, that is, the network moves along a sequence of metastable attractors.

1.2.2 Noise-driven temporal association

Synapses provided with memory or delayed synapses are not necessary for storing and retrieving temporal sequences. Buhmann and Schulten [2] describe a Hopfield-like neural network with nonsymmetric *static* synaptic interactions capable of recalling temporal sequences. The only two conditions are the synaptic interactions to be sufficiently nonsymmetric and the presence of noise. The nonsymmetric projections define the order of the pattern sequence and the noise triggers the transitions between consecutive patterns. The frequency of the transitions is affected by noise and cannot be precisely set as in the model by described Sompolinsky and Kanter. Its fluctuations can anyway be decreased by using a larger number of neurons.

The network proposed by Buhmann and Schulten is composed by N neurons that can be in either of two states $S_i = 1$ (firing) or $S_i = 0$ (not firing). S_i=1 with probability

$$g_\beta(h) = \frac{1}{1 + \exp(-2\beta h)} \tag{1.6}$$

where β is the inverse of the temperature T of the network, $h_i = \sum_j J_{ij} S_j - I$ the synaptic input, or *field*, to neuron i, where J is the synaptic matrix, and I the threshold potential. The patterns used in the network are *biased*, that is, only a small fraction of the neurons are active. The patterns are supposed to be orthogonal, that is $\sum_i \varepsilon^\nu S_i^\nu S_i^\mu = \delta_{\nu\mu}$, where ε^ν is a normalization factor.

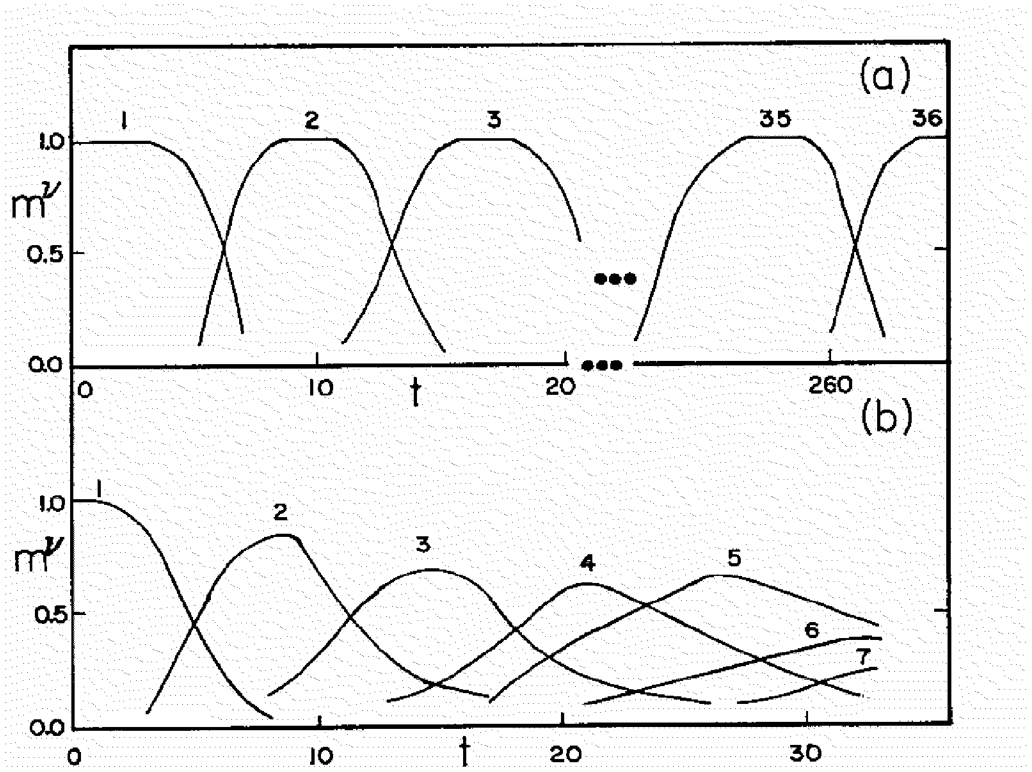


Figure 1.1: From [1], simulations of the network provided with symmetric instantaneous synapses and asymmetric synapses with memory, as described in Section 1.2. Number of neurons: 500, number of patterns: 10, $\tau = 8$, $\lambda = 2.5$. (a) $m(t) = \tau^{-1}$ for $t < \tau$ and $m(t) = 0$ otherwise. (b) $m(t) = \tau^{-1}e^{-t/\tau}$. The curves represent the overlaps $m^\nu(t) = N^{-1}\sum \xi_i^\nu S_i(t)$ of the network state with the patterns as a function of time. The numbers denote the index ν. The explanation for the parameters used can be found in the text.

Like the Sompolinsky and Kanter's model, the synaptic weight matrix is composed by a symmetric and a asymmetric part. The symmetric part

$$w_{ij}^{(s)} = \sum_{\nu=1}^{p} \left(\varepsilon^\nu S_j^\nu S_i^\nu - \sum_{\substack{\mu=1 \\ \mu \neq \nu}}^{p} \gamma \frac{p}{N} S_j^\mu S_i^\nu \right) \tag{1.7}$$

contains an excitatory term in common with equation 1.4 and an additional inhibitory term. The former stabilizes pattern ν, by providing excitation to all active neurons belonging to it. The latter inhibits the neurons not belonging to pattern ν. In this way, a sort of competition is created between all stored patterns. The network dynamics selects the pattern that has maximum overlap with the starting network state.

The asymmetric part

$$w_{ij}^{(a)} = \sum_{\nu=1}^{p} \left(\varepsilon^{\nu} S_j^{\nu} - \sum_{\substack{\mu=1 \\ |\mu-\nu|>1}}^{p} \gamma \frac{p}{N} S_j^{\mu} + \alpha^{\nu-1}\varepsilon^{\nu-1} S_j^{\nu-1} - \beta^{\nu+1}\varepsilon^{\nu+1} S_j^{\nu+1} \right) S_i^{\nu} \quad (1.8)$$

shows two additional terms that provide an excitatory projection from each pattern ($S^{\nu-1}$) to its successor (S^{ν}) pattern and add an inhibitory projection from each pattern to its predecessor. If the network is initialized with a state with maximum overlap with pattern S^{σ}, the network dynamics will move the network through a sequence of states with maximum overlap with patterns $S^{\sigma+1}$, $S^{\sigma+2}$, ..., $S^{\sigma+p}$ (See Figure 1.2). The time required to make

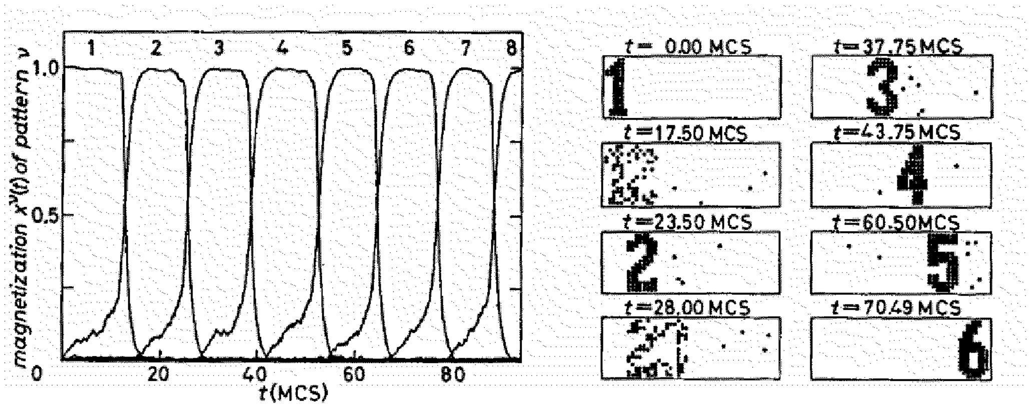


Figure 1.2: From [2], simulations of the network. In the left-hand part of the figure, the evolution of the overlaps $x^{\nu}(t) = \sum_i \xi_i^{\nu} S_i(t)$ of the network state with the stored patterns as a function of time are showed. The overlaps (see also Figure 1.1) have a small resting value, except for brief periods, when they reach a value close to one ($\nu = 8$, $\varepsilon^{\nu} = 0.001$, $\alpha^{\nu} = 0.1$, $\beta^{\nu} = 1.0 \, \forall \nu$, $T = \frac{1}{\beta} = 0.1$, $I = 0.35$). The right-hand part of the figure shows the network state, as a 2-dimensional matrix, as a function of time, during the retrieval of a sequence of six patterns representing the number 1, 2,...,6. The second and fourth network states show the transition between two patterns ($\alpha^{\nu} = 0.1$, $\beta^{\nu} = 1.0$, $T = \frac{1}{\beta} = 0.1$, $U = 0.35$).The explanation for the parameters used can be found in the text.

a transition from one patterns to the next one depends on the parameter α^{ν} and increases with decreasing α^{ν}. The temperature, which is directly linked to the noise of the systems, plays an important role. If the temperature is lower than a critical value T^{*}, any stored pattern becomes stable for an infinite time, and no transition can occur.

Buhmann's conditions concerning the orthogonality of the patterns and the presence of noise have been relaxed by Nishimori and Nakamura [21], who show that an asynchronous *deterministic* (i.e., noise-free) neural network can retrieve sequences of *random* patterns. The network used by Nishimori and Nakamura is formed by N neurons. The state S_i of each neurons can be either active ($S_i = +1$) or inactive ($S_i = -1$). The synaptic matrix has the form:

$$w_{ij} = \frac{1}{N} \sum_{\mu,\,\nu} \gamma_{\mu\nu} \xi_i^\mu \xi_i^\nu \tag{1.9}$$

where ξ_i^μ represents the state of neuron i of pattern μ, $\mu = 1, \ldots, p$. The overlap m^ν between the network state S and the pattern ξ^ν is defined, as usual, as $m^\nu(t) = N^{-1} \sum \xi_i^\nu S_i(t)$. The overlap m^ν evolves in time following the equation:

$$\frac{dm^\nu}{dt} = -m^\nu + \frac{1}{N} \sum_{\{\eta\}} \eta^\nu \tanh \left[\beta \sum_{\nu,\,\mu} \gamma_{\nu\mu} m^\nu \right] \tag{1.10}$$

where $\beta = 1/T$ is the pseudo temperature of the network. Equation 1.10 is valid in the limit $N \to \infty$ and for a finite number of patterns. The Hopfield's learning rule corresponds to the choice $\gamma_{\mu\nu} = \delta_{\mu\nu}$. In this case, equation 1.10 describes a dynamical system with a set of stable attractors, corresponding to the patterns embedded in the synaptic matrix. For a suitable choice of the coefficients $\gamma_{\nu\mu}$ (see [21]), Nishimori and Nakamura demonstrate that the networks evolves along a limit cycle, formed by the ordered sequence of stored pattern, even for $T = 0$.

1.2.3 Retrieval, counting, and recognition

An interesting application based on Sompolinsky's [1] and Kleinfeld's [17] models is described by Gutfreund and Mezard [22]. They also use two sets of interactions (see equations 1.4 and 1.5) and a stochastic updating rule at finite temperature (see equation 1.6). The authors distinguish three modalities of use of the network: *retrieval*, *counting*, and *recognition*. After having been in a state ν for a certain time and if λ (in equation 1.5) is sufficiently large, the network state will make a transition to state $\nu + 1$, as described in Section 1.2.1. In this way, the network can retrieve a temporal sequence in response to an external stimulus corresponding to the first pattern of the sequence. If the parameter λ is too small to induce a transition from state ν to state $\nu + 1$, the transition can be provoked by an external signal proportional to pattern $\nu + 1$, that is by using an additional external field on the i-th neuron $h_i = h_R\, \xi_i^{\nu+1}$. In this way, the pattern $\xi_i^{\nu+1}$ is recognized by the network. If the additional external field is proportional to a pattern η_i uncorrelated with ξ_i, $\forall\, i$, the network makes a transition to the next pattern each time the external field $h_i = h_C \eta_i$ is present. In this case, the network is said to be counting the number of external signals. The authors then extend the recognition mechanism to a case in which each state is connected to more then one possible successor. The field on the i-th neuron due to the asymmetric synaptic matrix $w_{ij}^{(a)}$ is

$$h_i^{(a)} = \lambda \sum (\xi_i^{1,\mu+1} + \xi_i^{2,\mu+1})[\bar{S}_{1,\mu} + \bar{S}_{2,\mu}] \tag{1.11}$$

(see Section 1.2.1), where $\xi_i^{1,\mu+1}$ and $\xi_i^{2,\mu+1}$ are two possible successors of $\xi_i^{1,\mu}$ and $\xi_i^{2,\mu}$, which belong to two intersecting sequences $\{\xi_i^{1,\mu}\}$ and $\{\xi_i^{2,\mu}\}$. Which transition occurs is determined by the external signal, which is therefore recognized. It is possible to choose the network parameters so that the network, which is in state ν, will not make a transition if the external signal is not correlated with any successive pattern $\nu + 1$. This means that it is possible to choose the network parameter so that the network recognizes the sequences without counting the number of external signals. Gutfreund and Mezard recognize the importance of noise. For a low noise level, the network remains in the initial state. When the noise is increased over a critical value, the network enters a region in which the external signals are recognized. A further increase in noise does not improve the performance. The overlaps with the successive patterns get more and more similar, as can be expected by inspection of the update equation 1.6. The external signal has to arrive before the combined action of the asymmetric synaptic links and noise induces a transition to one of the two successive patterns, or to a mixture of the two. In our study, we will analyze the link between the strength of the asymmetric synaptic weights, the level of noise (directly related to the pseudo temperature $1/\beta = T$), and the probability of a spontaneous transition from a state to a number of possible successive states.

Complex patterns

In all the mentioned cases, any pattern can occur only once in a given sequence. That means that the sequences cannot intersect by having a common pattern and no pattern can be repeated in any sequence. Guyon et al [23] addressed this problem, which was previously analyzed, in the context of bird-song learning, by Dehaene [20]. Guyon proposed to embed the patterns of the sequence at different times in a larger space, to be able to write the fields on the network neurons using the usual expression $h(t) = w\gamma(t)$, where

w is the synaptic matrix and γ the vector obtained by concatenating the input patterns at different times. The number of input pattern vectors concatenated corresponds to the minimal memory span required to disambiguate a sequence in which at least one pattern appears twice. Guyon's work includes a description of a number of non-local and local learning rules, the latter only valid if the set of input pattern are linearly independent vectors.

Kühn et al. [24] proposed a different solution to the same problem. The disambiguation of sequences with repeated patterns, or even repeated sub-sequences, is accomplished with the use of synapses with multiple time delays. The model is based on *three-neuron* interactions, the first and the second provided by symmetric (equation 1.4) and asymmetric synaptic links with memory (see Section 1.2.1). The third set of synapses are also provided of dynamic memory, but with a memory kernel characterized by a longer time constant and thus suitable for the disambiguation of sequences containing repetitions not only of single patterns but also of sub-sequences.

1.2.4 Grammars inference and prediction

A grammar is defined as a 4-tuple $(\mathcal{N}, \mathcal{V}, \mathcal{P}, \mathcal{S})$ where $\mathcal{N}$ and $\mathcal{V}$ are finite sets of variables and terminals (an alphabet), $\mathcal{P}$ is a finite set of production rules and $\mathcal{S}$ is the start symbol [25]. It is possible to associate a language to each grammar, corresponding to the set of strings the grammar can generate, and an automaton, a machine that that recognizes the grammar's strings. A simple example of a grammar is given in [25], page 81: $\mathcal{V} = \{S\}$, $\mathcal{V} = \{a, b\}$, $\mathcal{P} = \{S \to aSb, S \to ab\}$. By applying the first production rule $n-1$ times, and then the second production rule, starting from S, one obtains

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow a^3Sb^3 \Rightarrow \cdots \Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^nb^n \qquad (1.12)$$

So, the strings belonging to the language associated to this simple grammar are the set $\{a^nb^n\}$ for $n \geq 1$.

Originally, grammars were seen as possible models for natural languages. For a number of reasons, natural languages ended up being harder than expected to model using this formalism. Anyway, context-free grammars were adopted by computer scientists to model computer languages. Context-free languages are now of great practical importance in the definition of programming languages, in the formalization of the notion of parsing, in the translation of programming languages, and in other string-processing applications.

Grammar inference and prediction are fields in which the connectionist approach has been widely used. Grammar inference [26] is defined as the problem of finding or learning a grammar from a finite set of strings which results in the ability to classify strings as belonging to the grammar-generated language or not. The possibility of solving this problem using neural networks has been studied by a number of investigators: Elman [27, 3], Pollack [28], Giles et al. [29, 30], Lawrence et al. [31, 32], Hadley and Cardei [33], Das [34], Rodriguez et al. [35, 36], and Williams and Zipser [37], just to cite a few. Far from trying to review the subject, I just intend to point out some similarities grammar prediction bears with my research project. I have considered only those papers dealing with grammar prediction but not inference[3]. I have chosen the three papers whose goal and methods can be best compared with ours. In the first paper, by Williams and Zipser [37], a completely recurrent neural network is used in conjunction with a gradient-following learning rule for temporal supervised learning tasks. The network is shown to be able to learn complex tasks requiring the retention of information over time periods. The second paper, by Jeffrey L. Elman [3], illustrates the features of a recurrent neural network and its ability in solving the prediction task with, for example, a temporal version of the XOR problem. The third and last paper, by Paul Rodriguez, Janet Wiles and Jeffrey Elman [35] shows how a recurrent neural network can learn a language of the form $a^n b^n$.

A learning algorithm for continually running fully recurrent neural networks

The network used by Williams and Zipser has n units and m external inputs. Let $\vec{y}(t)$ be the vector of length n of the outputs of the network at time t and $\vec{x}(t)$ the vector of length m of external inputs to the network at time t. Let $\vec{z}(t)$ be the concatenation, of length $n+m$ of the vectors $\vec{x}$ and $\vec{y}$. If U is the set of indices k for which z_k is an output of the network and I the set of indices for which z_k is an external input, we have

$$z_k(t) = \begin{cases} x_k(t) & \text{if } k \in I \\ y_k(t) & \text{if } k \in U \end{cases} \tag{1.13}$$

The input s_k to the k-th unit at time t, for $k \in U$ is

$$s_k(t) = \sum_{j \in U \cup I} w_{kj} z_j(t) \tag{1.14}$$

where w_{kj} is the weight matrix, and its output at the next time step is

$$y_k(t+1) = f_k(s_k(t)) \tag{1.15}$$

[3]Pollack [28] describes how to turn the recognizer network into a generator, but the method is trivial and consists in enumerating strings and filter out those the recognizer rejects.

where f_k is a squashing function. The learning algorithm for this network adjustes the weight matrix w to match the units' output values with specified target values at each timestep. This is done by minimizing the network error by a gradient descent procedure. The learning can be done by accumulating the values of the weight matrix modifications for all timesteps (batch learning) or make the weights change while the network is running (real-time learning). The network has been tested on several tasks, two of which are summarized in the following.

Pipelined XOR For this task, two external inputs are used, each of which carry a randomly selected bit. The network is trained to match, at time t, a teacher signal being the XOR of the input bits at time $t - \tau$, with τ, with τ integer and $\tau \geq 2$. The network is showed to learn the task by configuring itself as a multilayer network.

Simple sequence recognition For this task, 2 or more units are used. Only two inputs, a and b are used, whereas the others are used as distractors. The task consist in having one output unit to be set to 1 each time activity on the b input is followed by activity on the a input, regardless of the time span between the two activity changes. The learning algorithm can accomplish the task by developing a flip-flop and a AND gate. The learning algorithm is not local, as each weight needs to have access to the weight matrix and the whole vector of errors.

Finding structure in time

Jeffrey Elman introduces this paper with a remark on the representation of time in parallel processing models. Usually, time is given an additional dimension in the input (see the *Window in Time* model in Kremer's review [38]). In this way, time can be treated like any other spatial input. Several drawbacks to this approach are described. For example, there must be some system buffering the inputs, to let the networks use them all at the same time, which is, physiologically speaking, not very realistic. Another problem lies in the size of the buffer, which limits the number of timesteps which can be used for computation. The author claims that time needs to be represented implicitly by the effects it has on processing, eliminating in this way the need to increase the dimensionality of the input. In a nutshell, the memory of the network should not be artificially introduced by using a buffer, but should be a result of the dynamics of the network itself.

An open question regarding this approach is the degree of dependence of the present state of the network on the past inputs. Systems whose computation at any point in time is modified by the present input can have a exceedingly high sensitivity to the inputs. That means that small differences in the presented sequences bring to network states with low correlation with each other. Low robustness with respect to noise in the input patterns is an important drawback of this way of encoding time. Unfortunately, the author does not mention these issues and limits its study to noiseless input sequences.

Architecture and learning rule Figure 1.3 shows the schematics of the network used by Elman. The recurrent connections allow the network's hid-

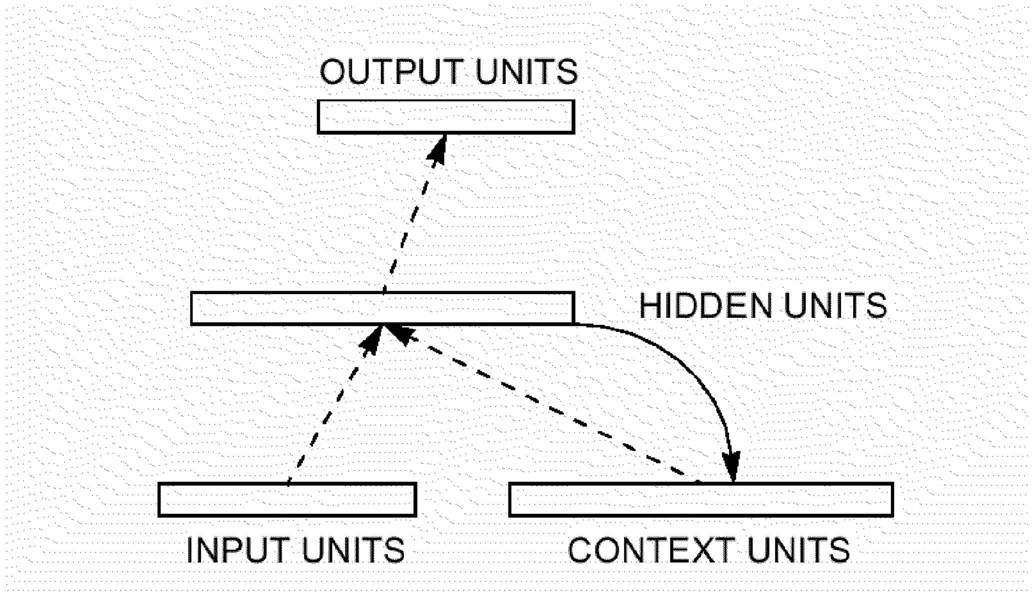


Figure 1.3: From [3], a simple recurrent network in which the activity pattern of the hidden units at any time t is used to form the activity of the hidden layer at time $t+1$. To do that, the hidden layer's activity pattern is copied at each timestep to the context layer.

den units to see their own previous output, so that the output at any time t is shaped by the input at time t and the output at time $t-1$. In this sense, the memory of the network is a feature of the dynamics of the network itself and does not need to be added as a separate system. In the learning phase the output of the network is compared with a teacher input and the synaptic weights are adjusted using backpropagation of error. The weights of the recurrent connections have a fixed value and are not subject to adjustment. The network has been tested on several tasks, two of which are described in the following.

Exclusive-OR This problem usually involves a two-bit input and a one-bit output. In this case, the problem has been translated into a temporal domain by constructing a sequence in which triplets of bits represent the first, the second and the answer bit in this order. A sample sequence is 1 0 1 0 0 0 0 1 1 1 1 0 1 0 1.... In this sequence, the first and the second bits (1 and 0) are the input to the XOR, the third bit the result (1) and so on. An input sequence, in which the input bit (the first and second bits) are randomly chosen, was presented to a network made by 1 input unit, 2 hidden units, 1 output units and 2 context units, one bit a time. The task the network had to learn was to predict the next bit in the sequence. It must be remarked that the network has no way of knowing whether a bit is an input input bit or an output (answer) bit and will try to make predict the next bit at all timesteps. After the learning phase, the network has acquired the ability to (partially) predict the bit sequence, as it is showed in Figure 1.4.

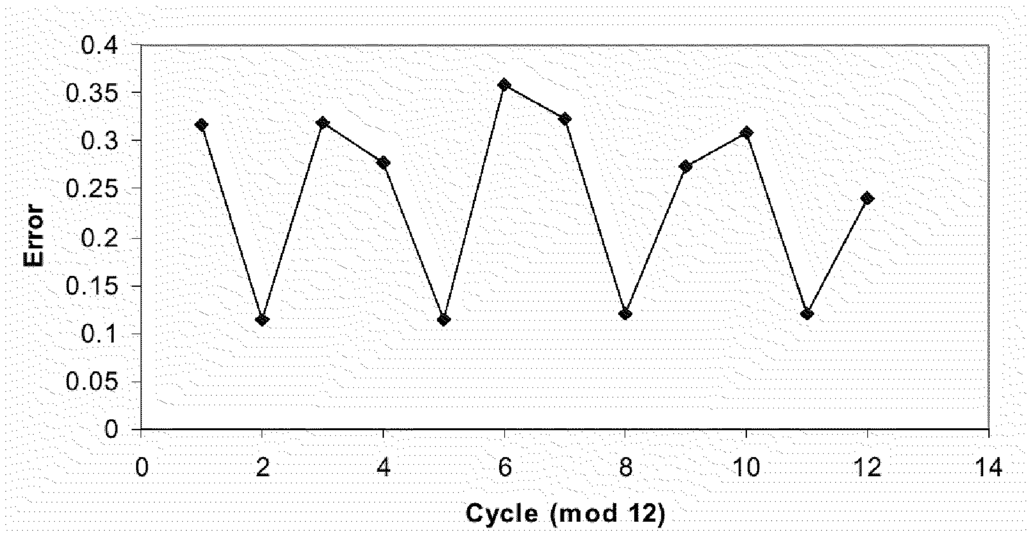


Figure 1.4: From [3], graph of the RMS error over 12 consecutive input bits. Data points are averaged over 1200 trials. The network tries to predict the sequence at each timestep. The error is low at the output (result) bits positions, where a correct prediction is possible.

Structure in letter sequences This task has a higher level of complexity: The sequence is composed by six different 6-bit binary vectors and requires a memory extending more than one timestep back time. The sequence used in the learning process is created in two steps. First, three consonants, b, d, and g were combined in random order, then each consonant is expanded using the rule b→ba, d→dii and g→guuu. So, for example, the sequence dbgbddg would become diibaguuubadiidiiguuu, where each letter is represented by a 6-bit vector. It is easy to see that the consonants appear randomly, but the vowels are correlated to the appearance of the consonants. The network used had 6 input units, 20 hidden units, 6 output units and 20 context units. The task of the network was to predict the next element in the sequence, by extracting the regularities used in forming the sequence from the sequence itself. Figure 1.5 shows the root mean squared error in this prediction task. The goal was to show that this class of problems *can* at all be solved by

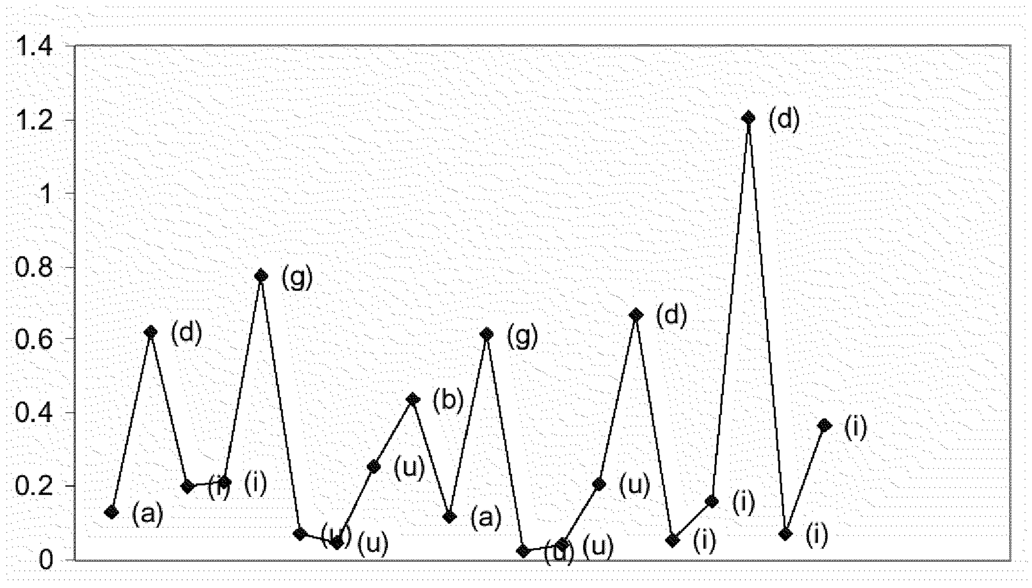


Figure 1.5: From [3], graph of the RMS error in the letter prediction task. The network tries to predict the sequence at each timestep. The error is high at the consonants, whose order is random, and low at the vowels, where a correct prediction is possible. Error is computed over the entire 6-bit vector.

using a neural network. In facts, the paper lacks a thorough analysis of the network's features and limits. Its capacity and the dependence of the capacity on the networks parameters are not examined. The learning rule is not local and is supervised. For this reason a direct comparison with our results is difficult.

A recurrent neural network that learns to count

In this paper, Rodriguez et al. try to answer two fundamental questions. Firstly, they explore the ability of recurrent neural networks in learning to predict a simple context-free language. Secondly they try to explain, by using the language of dynamical systems theory, the reasons for the results they got.

Architecture and learning rule The neural network used has two input units, two hidden units, two copy (context) units, two output units, and one bias unit. The architecture is thus the same as in Elman's paper [3] as well as the learning rule used, that is, backpropagation in time.

Task The task was to learn the context-free language $a^n b^n$ (see page 12). The sequences used in the learning phase had $1 \leq n \leq 11$. The sequences used in the evaluation of the networks included strings with $n > 11$ too, and a network was considered to have generalized if it had properly learned the training set and correctly predicted the strings with $n > 11$. The maximum length of the correctly predicted strings was used to quantify the performance of the network.

Results Out of 50 trials, the authors found that only 8 networks learned the task and were able to generalize. The only difference between the networks that could learn the task and those that could not is the initial weight set. The networks that successfully learned the task could properly predict strings up to $n = 16$.

1.2.5 Learning input-output relationships

The network reviewed in Section 1.2.4 can be considered as an instance of a large class of network developed to handle spatiotemporal patterns. Kremer [38] has carefully reviewed and classified a large number of recurrent and non-recurrent networks used to learn input-output relationships in the case of input data being a spatiotemporal pattern. His classification is based on the fundamental parts making up any spatio-temporal processing system. The state vector, the output function and the way they are computed, the parameters of the network (the so-called *long term memory*), the way they are updated (the *learning rule*), and the definition of the initial state of the system.

All the reviewed networks are characterized by a *supervised* learning rule. This means that the network parameters are updated with the aim of minimizing the difference between the actual output state of the network and a given target output.

An interesting part of Kremer's review is the one devoted to computing the present state vector from its past states and inputs. Thirteen alternatives are described. The importance of this part comes from the fact that we were confronted with same class of problems when designing a suitable way to merge past and present information to create a suitable input for the learning network. The problem, which will described in more detail in Chapter 5, originates from the necessity to incrementally compute the state of the network, without either storing the past states of the network or the external inputs to it.

1.2.6 Other approaches

A few models, because of the peculiar approach used, do not fit in any of the above mentioned classes. For example, the networks described by Natschläger, Maas and Zador [39] that make use of dynamic synapses, and a backpropagation-like algorithm to adjust the synaptic parameters. Natschläger and colleagues take inspiration from the activity-dependent forms of short-term plasticity such as facilitation and depression. In their framework, synaptic strengths change on a short time scale during performance and not only during the learning phase. In this way, a single synapse can produce quite different outputs for the same input, depending from the synapse parameters. By using a generalized form of the backpropagation learning algorithm, they succeeded in training the network to mimic the features of a quadratic filter of the form

$$\mathcal{L}x(t) = \sum_{l=1}^{m}\sum_{k=1}^{m} h_{kl}x(t-k\Delta)x(t-l\Delta) \tag{1.16}$$

where t is time, Δ is some time delay, $x(t)$ is the input, and the filter coefficient h_{kl} form an arbitrary symmetric $m \times m$ matrix.

In another work, Natschläger and Ruf [40] describe a network of spiking neurons capable of finding clusters in a high dimensional input space and recognizing temporal sequences, even if distorted in time. The network is based on a set of synapses with learnable weights connected to delay lines. The physiological realism of the network's architecture is questionable, as such use of delay lines has not been observed yet in living systems.

The following two approaches distinguish themselves for the novelty and originality. Both approaches are able to recognize spatiotemporal patterns, but are unable to generate or predict them.

Transient synchrony

Hopfield and Brody's model network is probably the most innovative model proposed in the last few years. The main question the authors try to answer is how can a biologically plausible neural network integrate information over times of the order of, at least, 500 msec. The model described in [4] makes use of a network of spiking neurons whose activities decay with time. A spatiotemporal pattern is recognized by the transient synchrony of a set of neurons having, for a short lapse of time, similar spike frequencies.

Architecture and learning rule Hopfield's network is arranged in three groups (see Figure 1.6).

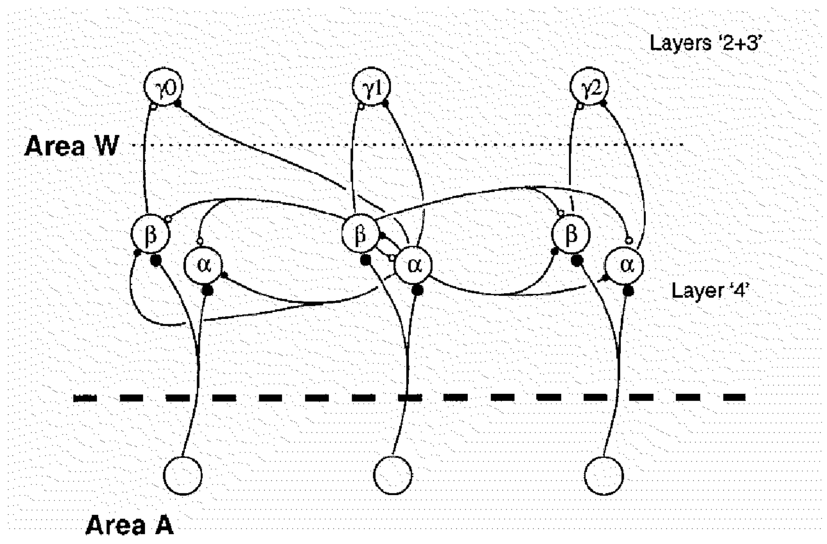


Figure 1.6: From [4], the architecture of the network. The thick dashed line separates area A from area W; the thin dotted line separates layers 2 and 3 from layer 4 in area W. Small filled circles indicate excitatory connections, whereas small open circles indicate inhibitory connections.

The input neurons belong to area A: Each neuron in this area is characterized by an initial firing rate and a decaying activity, whose time constant is the same for all the events that are able to trigger that neuron. Neurons of area A project to what has been called "layer 4" of area W. Layer 4 of area W contains two type of cells, α-type cells (excitatory) and β-type cells (inhibitory). Both α and β cells receive excitatory input from area A afferents and are found in similar quantities. Cells in layer 4 of area W connect to each other: approximately half of the connections from each cell are onto α cells, the other half are onto β cells. Both type of cells project onto layer 2 and 3, where they contact the output γ cells. The number of γ cells is about 3% of the number of cells in layer 4 and they do not feed back to them.

Input cells are sensitive to different features of the input patterns. These cells respond to stimuli with activities decaying with a variety of different timescales. Synaptic connections between neurons with similar firing rates often produce synchrony between the action potentials of these neurons. So, if α and β neurons are properly connected to each other, that is, after the learning procedure has finished, if the input pattern matches the stored pattern, the α and β neurons driven by the input neurons will synchronize and the γ cells will receive a high-amplitude oscillating input current that drives them to fire.

No learning rule has been developed yet for this network. A learning "procedure" is described instead. This procedure simply consist in selecting those α and β cells that have similar firing rates in response to the target spatiotemporal pattern and creating all-to-all connections between this set. The same set of neurons also need to be connected to an output γ neuron. The network has been successfully applied to the recognition of spoken words.

Liquid and 'echoed' states

Herbert Jaeger [5] and Wolfgang Mass et al. [41] arrived independently at a novel way of exploiting the features of the networks of neurons to encode spatiotemporal patterns. Jaeger [42] noticed how difficult it is to understand high-dimensional, nonlinear dynamical systems and, in particular, recurrent neural networks. Moreover, neural networks need a suitable learning rule, to solve tasks like associating an output data set given an input data set. For example, the *backpropagation through time* is one popular learning rule that has been widely used as a general-purpose supervised training algorithm. Some drawbacks of this algorithm are the computational cost and the lack of biological realism, as non-local information are needed. Another problem involved in learning in recurrent neural networks is that, in general, each unit can influence, directly or indirectly, any other unit. So, it is difficult to have old memories not being disrupted by the new ones. Jeager solves this problem avoiding learning from the outset. His idea is to use large recurrent networks as 'dynamical reservoir' from which an output set of units can tap information about the input spatiotemporal pattern. In this way, only network-to-output connection weights need to be learned. This can be done with known, highly efficient algorithms.

Architecture and learning rule Jaeger's model is a discrete-time neural network with K input units, N internal units and L output units. Activations of input units at time step n are $u(n) = (u_1(n), \ldots, u_K(n))$, of internal units are $x(n) = (x_1(n), \ldots, x_N(n))$, and of output units $y(n) = (y_1(n), \ldots, y_L(n))$. The network is updated according to

$$x(n+1) = \vec{f}(W^{\text{in}}\vec{u}(n+1) + W\vec{x}(n) + W^{\text{back}}\vec{y}(n)) \tag{1.17}$$

where $\vec{f} = (f_1, \ldots, f_N)$ are the internal unit's output functions (typically sigmoidal functions) and W^{in}, W, and W^{back}, respectively a $N \times K$, a $N \times N$ and a $L \times (K+N+L)$ weight matrix. The output of the network is computed according to

$$\vec{y}(n+1) = \vec{f}^{\text{out}}(\vec{u}(n+1), \vec{x}(n+1), \vec{y}(n)) \tag{1.18}$$

where $\vec{f}^{\text{out}} = (f_1^{\text{out}}, \ldots, f_L^{\text{out}})$ are the output unit's output functions and $(\vec{u}(n+1), \vec{x}(n+1), \vec{y}(n))$ the concatenation of the input, internal and previous output activation vectors (see Figure 1.2.6). In this way, the state $\vec{x}(n)$ can

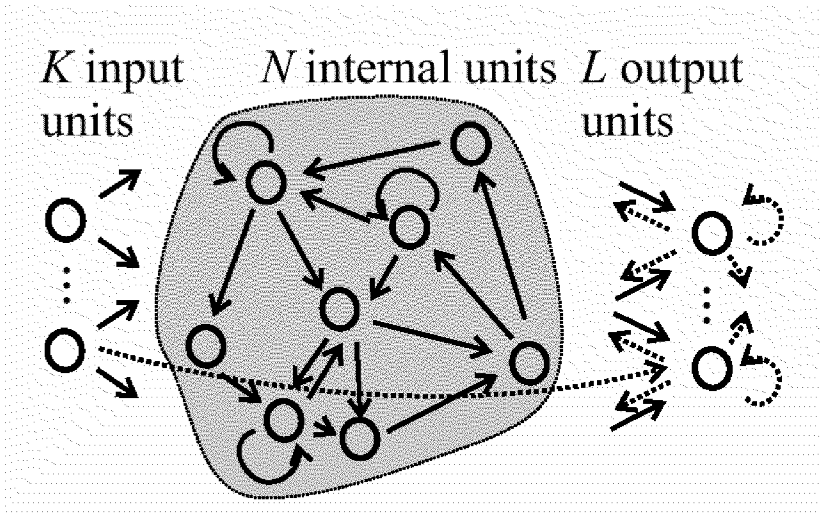


Figure 1.7: From [5], the network architecture. Dashed arrows indicate connections that are possible but not required.

be considered as an "echo" of the input history as well as the output of the network $\vec{y}(n)$, which can be written

$$\vec{y}(n+1) = \vec{G}(\ldots, \vec{u}(n), \vec{u}(n+1); \ldots, \vec{y}(n-1), \vec{y}(n)) \tag{1.19}$$

Jaeger's model avoids the necessity to convert a time series into a static input pattern by using windows in time or shift buffers.

The learning procedure minimizes the mean square error of the difference between the network's output and a training signal. Learning is applied only to the weights connecting to the output units; all other connections remain unchanged. Jaeger illustrates the features of his model in several examples. One particularly interesting example regards text learning and generation. In this example, the task is to process a text and recall it. The values of the output units are used to randomly draw the next character in the output sequence, which is then used as next input to the network.

1.3 Hidden Markov models

Hidden Markov models are statistical models of sequential data. A drawback of Markov models of order k (see equation 1.1) is that they become intractable for large k. Hidden Markov models do not assume that the *observed* data sequence has a Markov property of low order, which is, in general, not true. They are based instead on the assumption that a *hidden* (unobserved) state variable exists, which summarizes all the relevant past values of the observed and hidden variables. In this way, the values of the state variable can be used in the prediction of the value of the observed variable y_t or of the next state q_{t+1}.

A hidden Markov model is completely specified by the initial state probabilities $P(q_1)$, the transition probabilities $P(q_t|q_{t-1})$, and the emission probabilities $P(y_t|q_t)$. The probability of observing a given sequence y_1^T as the result of a sequence of hidden states q_1^T is given by

$$P(y_1^T|q_1^T) = P(q_1)\prod_{t=1}^{T-1} P(q_{t+1}|q_t)\prod_{t=1}^{T} P(y_t, q_t) \tag{1.20}$$

A review of the most commonly used learning methods can be found in the paper by Ghahramani [43] and in the one by Bengio [44], who also reviews some extensions of hidden Markov models like, for example, hybrids of hidden Markov models and neural networks.

1.4 Conclusions

The processing of temporal sequences of patterns is a task common to several fields. Brains are good at learning, recognizing, generating, and predicting patterns. Anyway, not much is known about the neuronal circuitry needed to implement such behaviors. Hopfield's model first gave an insight about the structure and dynamics of a network that is able to perform simple

tasks like the content-address recall of static patterns and pattern sequences. Hopfield's model was later augmented and enhanced by the use of dynamic 'non-instantaneous' synapses, which keep a short-term memory of the activity history of the network. Dynamic synapses can make the recall process dependent not only on the present state of the network, but also on its state back in time.
In neural networks, noise can play an important role. A stochastic update rule for single neurons can make a network show features that are not present in the deterministic version. Buhmann and Shulten showed that dynamic synapses are superfluous if a high enough level of noise is introduced into the system.

A Hopfield-like network is not the only architecture capable of processing pattern sequences. Kremer published a whole *taxonomy* of networks that, more or less successfully, can learn pattern sequences. The synaptic links of the networks reviewed are updated with a supervised learning rule, whereas an unsupervised rule would make these models more suitable for a comparison with the dynamics and mechanisms found in brains.

The problems tackled in the field of grammar inference and prediction also imply the processing of sequences of patterns. If, superficially, the tasks in neurosciences and grammar inference bear several resemblances, a closer analysis reveals that the goals are distinct. The use of neural network in grammar inference and prediction has been mainly aimed to the design of systems able to extract the production rules of the grammar rather than learn the sequences by creating an internal representation of them. The results are however of interest, as the use of neural networks to solve this class of problems might give an insight about the mechanism used by brains to solve similar tasks.

Hopfield's latest work is undoubtedly one of the most promising models for the processing of temporal sequences of patterns by physiologically realistic neurons. The development of a suitable, possibly unsupervised learning rule (only a learning 'procedure' is described) is probably going to be the next challenge, whose solution will determine the future development of this approach.

Another recent model is the so-called 'liquid states' [41]or 'echoed states' [5] network. In a nutshell, these networks use a set of neurons as a 'reservoir', containing memory traces of the past inputs to the network. The readout, which extracts information from the reservoir, is the core of the system. In some aspects, the 'echoed states' network is similar to the one we developed. In the following we wil maily refer to the Jeager's version of the network, being Maass' one substantially similar. Like our network, Jaeger's network is a dynamical system capable, of learning pattern sequences. The weights of the connections between at least some of its units can be modified by a learning rule. It encodes the history of the input stimuli (patterns) in the activity pattern of the network, by incremental modifications of the present activity, as it will be explained in detail in Chapter 4. The main differences between Jaeger's network and ours are described in Chapter 6.

We show how our network, based on a simpler architecture and on well-known dynamics and learning rule, displays a performance similar to the one of Jaeger's model. The most evident advantage of our model lie in the ability to generalize to time-warped stimuli sequences (or even to sequences with random inter-stimulus times), the unsupervised learning rule, and an architecture that, being based on binary synapses and neurons, is particularly simple to be implemented in hardware. The encoding mechanism (see Chapter 4) that mixes the present activity pattern of the network with the new stimulus is perhaps the part of our model that mostly needs improvements. In particular the buffer network, described in Section 4.1.1, should be considered as a placeholder for a not yet developed system: such a system should perform in a way very similar to the buffer network, but its properties should *emerge* from the network dynamics, instead of having been designed on purpose. Even if this might make many neuroscientists' eyebrow raise, we have anyway accomplished the task of implementing a Markov model as a dynamical system, suitable for aVLSI implementation. Our systems can also be employed for the modeling of brain functions, although to a limited extent, and represent a solid starting point for further research in this field.

Chapter 2

The model network

In Chapter 1, we showed how storing and recalling a temporal sequence of stimuli in a neuronal network can be achieved by creating associations between pairs of stimuli that are contiguous in time. This idea is illustrated by studying the behavior of a neural network model with binary neurons and binary stochastic synapses, whose architecture, dynamics, and learning rule we are going to describe in this chapter. The network is shown in Chapter 3 to be able to extract in unsupervised way the temporal statistics of the sequence of input stimuli. When a stimulus triggers the recalling process, the statistics of the output patterns reflects those of sequence used during the training. If the sequence of stimuli is generated through a Markov process, then the network faithfully reproduces all the transition probabilities. In Chapter 4 we show that, using a suitable pre-processing of the input stimuli, the network is also able to learn and generate stimuli sequences generated by a non-Markov process.

2.1 Introduction

Our current understanding of the neuronal mechanisms that allow biological systems to encode and recall temporal sequences of stimuli is still marginal. In the last decade many studies attempted to relate the problem of encoding temporal sequences to the generation of associations between visual stimuli [45, 2, 7]. Interestingly, in one simple case [10], in which the stimuli were presented in a fixed temporal order, it was possible to study the neural correlate of this kind of associative memory. Cortical recordings displayed significant correlations between the patterns of activity elicited by neighboring stimuli in the temporal sequence. Hence these internal representations of the visual stimuli encode the temporal context in which stimuli were repeatedly presented during training. These patterns of activity were stable throughout long time intervals and have been interpreted as global attractors of the network dynamics [7]. These attractors are correlated up to a distance of 5 in the temporal sequence, similar to that observed in the experiment of Miyashita [10], despite the fact that the learning rule makes use only of the information about the contiguity of two successive stimuli [7, 11]. Here we try to extend these mechanisms to a more general situation. First, we show that the network, in the presence of noise with a suitable amplitude, can spontaneously jump to a correlated pattern of activity representing a different stimulus. The degree of correlation between the internal representations of the stimuli encodes the transition probability, and the presentation of a single stimulus can trigger the recalling of a sequence of temporally correlated patterns of activity. Second, the spatial structure of the attractors, that encodes the transition probabilities, can be learned when the network is exposed repeatedly to the temporal statistics of the stimuli. The transition probabilities are automatically extracted during this training phase and encoded in the synaptic matrix, which, in turn, determines the structure of the attractors. The learning rule was inspired by the one introduced in [7] and it makes use of the information carried by the current stimulus and by the pattern of activity elicited by the previous stimulus. A possible mechanism for making this rule completely local in time has been suggested in [46, 11] and relies on the stable activity that is sustained by the network in the interval between two successive stimuli.

2.2 Methods

Concerning the choice of the architecture, we were guided by the wish to keep our model as simple as possible. For this reason, we employed a fully connected network of binary neurons linked by binary synapses, which is also a network model often employed by various researchers in this area. The dynamics of the network is stochastic, as described in [16]. Learning is also implemented as a stochastic process, in which the final probability for a synapse to be potentiated depends on the whole sequence of patterns to which the network was exposed during the learning phase. An adapting, dynamic global inhibition keeps the average activity of the network constant.

By using binary neurons with only two values ($+1$ and 0), excitation and inhibition have been kept separated. All neurons only provide an excitatory input to other neurons. The global activity is kept, on average, constant by the negative feedback provided by a global inhibition, which can be thought to be provided by a population of inhibitory neurons whose activity is proportional to the activity of the excitatory neurons population. Stochasticity was introduced in the dynamics of the network by means of the so called *Glauber dynamics* (see, for example, [47]). The Glauber dynamics is a popular way of updating the state of the neurons of a network: it depends on a parameter (the *pseudo-temperature* T, or its inverse β) and on the synaptic input to the neuron being updated. The stochasticity introduced by this dynamics reflects the noise which is normally present in brains. Far from being just a nuisance, noise is an essential component of the network. In the next chapter it will be shown how the performance of the network can be maximized by acting on the level of noise. Fedoroff and Fontana [48], on an article about gene networks, observed that

> The deterministic view of stable states [...] requires exogenous agents of change to initiate transitions, whether these transitions be developmental or adaptive. If stochasticity is a fact of life, states are by definition metastable, and fluctuations can cause transitions between them.

In this view, noise in neural networks, gene networks, and, in general, in biological systems, might play an important role in enabling the adjustments needed to cope with an ever-changing internal and external environment. Learning was also implemented as a stochastic process in the sense that the potentiation or depression of the synapses is a probabilistic function of the activity of the pre- and post-synaptic neurons.

For a wide class of networks, when the maximum storage capacity is reached, the network can no more retrieve information about any of the learned stimuli [47]. This problem can be avoided by allowing the network to forget. Our network, like a large class of networks [49], shows the 'palimpsest' property. This implies that information about old stimuli is forgotten to make room for the new ones. Of course, too fast forgetting is not a desirable feature. Remembered stimuli belong to a 'sliding window' in time: Stimuli laying outside the window are forgotten by the network. The width of the window depend on the learning rule, and namely, on the fraction of synapses that is changed in correspondence to any stimulus presentation. If only a small fraction of neurons is changed each time, learning is a slow process, and the window in time of the memory span is large. If the fraction is large, the network can quickly learn new stimuli, but the window width is relatively small. So, the speed by which information about old stimuli is forgotten can be tuned by changing the fraction of synapses taking part to the learning process each time a stimulus is presented. Additionally, Fusi [49] showed that this learning rule, in the case of slow learning and patterns with low activity, can reach the optimal storage capacity. This means that the information content of the pattern scales in the same way as the information content of the synapses. The maximum number of different patterns that can be stored and retrieved without errors is $N^2/(\log N)^2$, if the mean activity of the network scales as $\log N/N$.

In our network, we chose, for simplicity, to have binary neurons. This choice will ease the implementation of a simulation based on integrate-and-fire neurons, which will eventually lead to a hardware (aVLSI) implementation. In general, the performance of the network does not improve with the number of stable synaptic states if several neurons in the patterns of activity of the network have the same spiking rate, and so encode for the same information. As only two levels of neuronal activity are elicited by external stimuli, we chose binary synapses for our network, without worsening its performance. This choice in not physiologically implausible. Petersen [50] showed that individual synapses appear to have all-or-none potentiation. His results raise the possibility that some forms of synaptic memory may be stored in a digital manner in the brain.

At this point, it is natural to ask which is the mechanism that provokes the stochastic potentiation or depression of the synapses. Fusi [49] showed that the needed source of noise can be found in the inter-spike interval variability. His solutions finds justification in the irregularity of the spike trains recorded *in vivo* (see, for example, the work by Softy and Koch [51]). Fusi demonstrates that this source of noise can be used to achieve the stochastic potentiation or depression of any synapse, even if the synapses themselves are deterministic. The method described in [49] is shown to be suitable for implementation in hardware (aVLSI).

2.2.1 Architecture and dynamics

We implemented a recurrent neural network with N neurons, labeled by index i, $i = 1 \dots N$. The state of neuron i is described by the variable S_i: $S_i = 1$ ($S_i = 0$) corresponds to a firing (quiescent) neuron. The network is fully connected with binary synapses J_{ij} from neuron j to neuron i [46]. The fraction $F = \frac{1}{N}\sum_k S_k$ of neurons that are active, corresponding to the level of neural activity, is generally lower than 0.5. The neuron's state is updated using the Glauber dynamics, in which S_i=1 with probability

$$g_\beta(h) \equiv \frac{1}{1 + \exp(-2\beta h)} \tag{2.1}$$

where β is the *pseudo temperature* of the network and $h_i = \sum_j J_{ij}S_j - I$ the synaptic input, or *field*, to the neuron. The global inhibition I dynamically adjusts the activity of the network. The computed value I for the inhibition

Box 2.1 Glauber dynamics.

The network is composed by N neurons. Neurons are labeled by index *i*, $i = 1 \ldots N$. The state of neuron i is described by S_i: $S_i = 1$ (active neuron) or $S_i = 0$ (non active neuron). The synaptic (weight) matrix is J_{ij}. Each synapse can be potentiated ($J_{ij} = 1$) or depressed ($J_{ij} = 0$).

$$h_i = \sum_j J_{ij} S_j - I$$

is the synaptic input, or field to neuron i (see code in Section A.2) and I the global inhibition. Neurons are updated using the Glauber dynamics: The probability for a neuron with synaptic input h to be active when its state is updated is

$$g_\beta(h) \equiv \frac{1}{1 + \exp(-2\beta h)}$$

where β is the inverse of the *pseudo temperature* (see code in Section A.3). One timestep of simulation corresponds to the update of all neurons of the network in random order.

as a function of the global network activity F is expressed by the following equation:

$$I = \begin{cases} \frac{1}{\tau}(I(t) - s_0(F - s_1)) & \text{If } s_0(F - s_1) \geq I_m \\ \frac{1}{\tau}(I(t) - I_m) & \text{If } s_0(F - s_1) < I_m \end{cases} \tag{2.2}$$

where typically $1/\tau = 0.02$, I_m is a minimum activity, euristically chosen to be $I_0/5$, and s_0 and s_1 are chosen so that $s_0(f_0 - s_1) = I_0$ and $s_0(\kappa f_0 - s_1) = 0$, where tipically $\kappa = 0.7$. s_0 and s_1 are kept fixed during the whole simulation. I_0 is usually chosen between the maximum value of the field on quiescent neurons and the minimum value of the field on active neurons when the network state corresponds to a learned pattern of activity (Figure 2.1), and f_0 is the average activity of the learned patterns. The value of $L = s_0(F - s_1)$ represents the target inhibition for the network. The inhibition is not immediately set to L but slowly adjusted. The inhibition is typically adjusted once per timestep, where one timestep corresponds to the update of all neurons of the network. If $I(t)$ is the present value of the inhibition, $I(t+1)$ its value after the adjustment and L the target value,

$$I(t+1) = \frac{1}{\tau}(I(t) - L) \tag{2.3}$$

(typically, $1/\tau = 0.02$) shows how the inhibition is actually a low-pass filtered version of the function $L = L(F)$. This is made to let the system variables change continuously and to avoid the oscillations that might be caused by a too prompt response of the dynamic inhibition to changes in network activity. Different values of I_0 lead to different asymptotic value of the inhibition I,

Box 2.2 Dynamic inhibition

The network is composed by N neurons. Neurons are labeled by index i, $i = 1 \dots N$. The state of neuron i is described by S_i: $S_i = 1$ (active neuron) or $S_i = 0$ (non active neuron). F, the global actvity of the network, is defined as the fraction of active neurons:

$$F = \frac{1}{N} \sum_k S_k.$$

Each time the inhibition is updated, its value is modified according to the difference equation

$$I(t+1) = \begin{cases} \frac{1}{\tau}(I(t) - s_0(F - s_1)) & \text{If } s_0(F - s_1) \geq I_m \\ \frac{1}{\tau}(I(t) - I_m) & \text{If } s_0(F - s_1) < I_m \end{cases}$$

where typically $1/\tau = 0.02$, I_m is a minimum activity, euristically chosen to be $I_0/5$, and s_0 and s_1 are chosen so that

$$\begin{cases} s_0(f_0 - s_1) = I_0 \\ s_0(\kappa f_0 - s_1) = 0 \end{cases}$$

where tipically $\kappa = 0.7$. s_0 and s_1 are kept fixed during the whole simulation. I_0 is chosen between the maximum value of the field on quiescent neurons and the minimum value of the field on active neurons when the network state corresponds to a learned pattern of activity (see Figure 2.1), and f_0 is the average activity of the learned patterns (see code in Section A.4).

as shown in Figure 2.2. The asymptotic value of the activity depends on the chosen I_0, as it is shown in Figure 2.3. The temporal evolution of the inhibition, plotted in Figure 2.4 for several choices of I_0, shows that 5 updates of the network are sufficient to let the network's inhibition and activity relax.

Definition of overlaps and transitions

The overlaps $m^\mu(t)$ of the current state of the network with a pattern μ ($\eta_i^\mu = 1,\ 0$) are defined as $m^\mu = \frac{1}{N} \sum S_i \eta_i^\mu$ [47]. Given a set of n patterns and the set of integers $\mathcal{M} = \{1, \dots, n\}$, if, at time t, $m^\alpha > m^\mu$, $\forall \mu \in \mathcal{M} \setminus \alpha$ and, at time $t+1$, $m^\beta > m^\mu$, $\forall \mu \in \mathcal{M} \setminus \beta$, a network is said to have made a transition from pattern α to pattern β. This definition does not depend on the definition of timestep, which can correspond to the update of any number of neurons. The element $T_{\nu\mu}$ of a transition probability matrix for

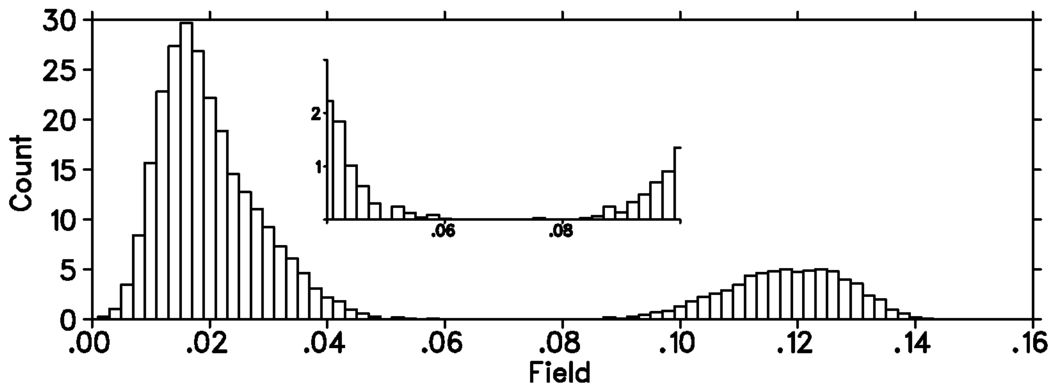


Figure 2.1: Histogram of the non-zero fields (synaptic inputs) on the neurons of a network of 490 neurons after the learning of 7 random patterns with activity 1/7. The network state has overlap 1 with one of the 7 learned patterns. The values of the bars of the histogram represent an average over 100 similar networks, which differ only in the synaptic matrix. The left-hand peak corresponds to the synaptic input to quiescent neurons, the right-hand peak to the synaptic input to active neurons. The inset shows a magnification of the central part of the histogram.

a network can be computed by measuring the probability for a network to make a transition from a pattern μ to pattern ν. The transition probabilities matrix can be calculated using the following protocol:

1. The network state is set to have overlap 1 with pattern μ: $S_i = \eta_i^\mu$.
2. The network is left free to evolve for a maximum time t_{max}.
3. If the network makes a transition to pattern ν, increase the element $T_{\nu\mu}$ of the transition probability matrix by a fixed quantity.
4. If the network makes a transition or if $t \geq t_{\text{max}}$, reset the network state to pattern $\mu + 1$, modulus the number of patterns.
5. Eventually normalize the transition probability matrix.

Error on the probability estimation

The probability of a set of mutually exclusive events can be estimated by direct measurement. Given the total number of observations m, the estimated probability P and the width of the confidence interval in standard deviations k, the upper and lower bound of the estimated probabilities are given by

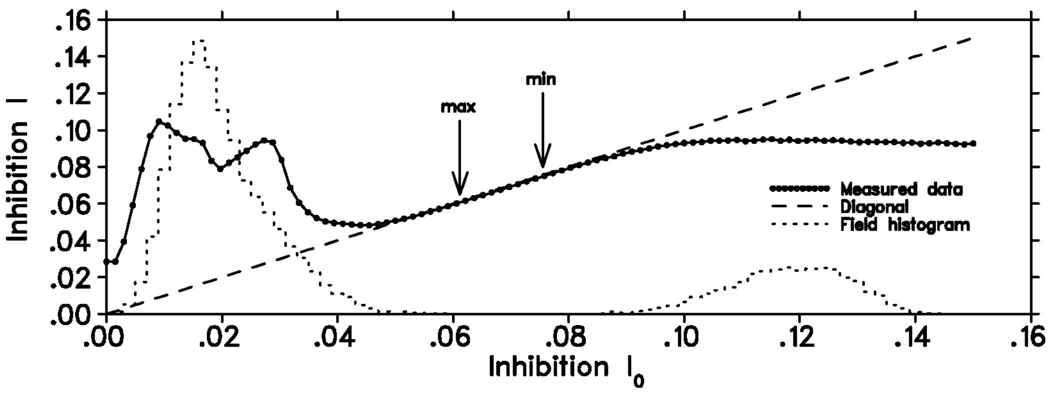


Figure 2.2: Asymptotic value of the inhibition I as a function of the initial inhibition I_0. All curves represent average values over 100 networks (the same networks used to produce Figure 2.1 have been used). The arrows point to the absolute maximum value of the fields on quiescent neurons and to the absolute minimum value of the fields on active neurons. When I_0 is chosen between these two values, the asymptotic value I of the inhibition (i.e., the value of the inhibition after a large number of updates of the whole network, 30 in our case. See also Figure 2.3) is a linear function of the initial inhibition I_0 for all networks. A good linearity is preserved for values of I_0 chosen outside the shown interval, but still in the range [0.05; 0.09]. The dashed line is the diagonal of the plot, which can be used as a reference. The dotted line corresponds to the histogram shown in Figure 2.1. For values of I_0 higher than 0.09, I eventually settles to a value close to 0.09. If I_0 is chosen lower than 0.05, I can be oscillating and its average value is always higher than I_0 ($\gamma = \frac{I_0}{0.3 * f_0}$, $1/\tau = 0.02$, $f_0 = 1/7 \approx 0.14$).

Meyer [52]

$$P_{\text{low/high}} = \frac{Pm + \frac{k^2}{2} \mp k\left[P(1-P)m + \frac{k^2}{4}\right]^{\frac{1}{2}}}{m + k^2} \tag{2.4}$$

2.2.2 The learning rule

Learning is also implemented as a stochastic process, in which the final probability for a synapse to be potentiated depends on the whole sequence of patterns to which the network was exposed during the learning phase. A pattern η_i^μ is presented to the network by setting the neuron states $S_i = \eta_i^\mu$ and it is learned, according to [53], by applying the following rule:

- If $J_{ij} = 0$ and the neurons η_i^μ and η_j^μ are both active, then a transition of the synaptic weight $J_{ij} : 0 \rightarrow 1$ occurs with a probability q_+.
- If $J_{ij} = 1$ and only one of the neurons η_i^μ and η_j^μ is active, then a

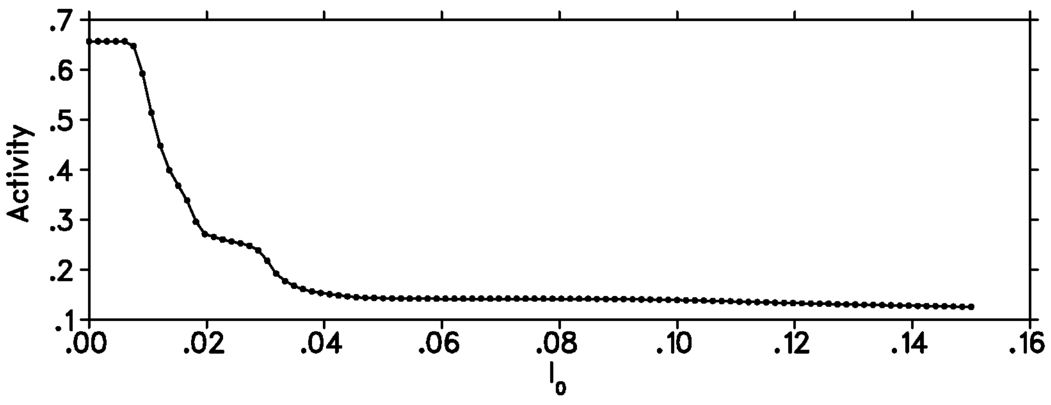


Figure 2.3: Asymptotic value of the activity f as a function of the initial inhibition I_0 ($f_0 = 1/7 \approx 0.14$). For I_0 greater than the maximum field on quiescent neurons (see Figure 2.2), the dynamic inhibition keeps the global activity (i.e., the value of the activity after a large number of updates of the whole network, 30 in our case, see also Figure 2.2) close to the desired value f_0. If I_0 is smaller than the maximum field on quiescent neurons, the asymptotic inhibition is greater than the desired activity f_0.

Box 2.3 Overlaps and transitions

The overlap of the network state S with a pattern η^μ is

$$m^\mu = \frac{1}{N}\sum S_i \eta_i^\mu$$

(see code in Section A.5). Given a set of n patterns and the set of integers $\mathcal{M} = \{1, \ldots, n\}$, if, at time t, $m^\alpha > m^\mu$, $\forall \mu \in \mathcal{M} \setminus \alpha$ and, at time $t+1$, $m^\beta > m^\mu$, $\forall \mu \in \mathcal{M} \setminus \beta$, a network is said to have made a transition from pattern α to pattern β (see code in Sections A.6 and A.7). The element $T_{\nu\mu}$ of a transition probability matrix is computed by measuring the probability for a network to make a transition from a pattern μ to pattern ν.

transition of the synaptic weight $J_{ij} : 1 \to 0$ occurs with a probability q_-.

- If both the neurons η_i^μ and η_j^μ are inactive, the corresponding synapses are left unchanged.

A transition from pattern η^μ to pattern η^ν is learned using the additional rule [46, 11]:

- If $J_{ij} = 0$ and the neurons η_i^ν and η_j^μ are both active, then a transition of the synaptic weight $J_{ij} : 0 \to 1$ occurs with a probability $q_\times = \lambda q_+$, where $\lambda < 1$: η_j^ν is usually the present activity pattern, and η_j^μ

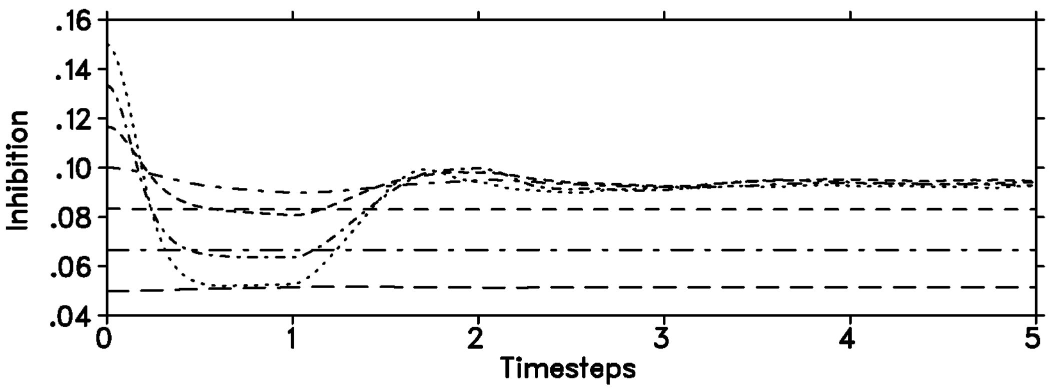


Figure 2.4: Evolution of the inhibition for different inhibitions I_0 (see Figures 2.2 and 2.3). One timestep corresponds to the update of a single neuron. The full network is evolved 5 times. For I_0 greater than 0.09, I oscillates and eventually settles to a value close to 0.09, as can be seen in Figure 2.2.

the activity pattern before the presentation of the latest stimulus. In Section 3.1.2 an extension of this rule has been used: if $J_{ij} = 0$ and the neurons η_i^ν and η_j^μ are both active, then a transition of the synaptic weight $J_{ij} : 0 \to 1$ occurs with a probability $q_\times = \lambda_\mathrm{f}\, q_+$ and a transition of the synaptic weight $J_{ji} : 0 \to 1$ occurs with a probability $q_\times = \lambda_\mathrm{b}\, q_+$, $\lambda_{b,f} < 1$.

The learning process needs to be repeated several times for each pattern and pattern pair. The number of times L the learning has to be repeated must be large enough to let the synaptic matrix relax to stable asymptotic configuration. To achieve that, $L \gg \frac{1}{q_\times} > \frac{1}{q_+}$ is needed. In the case of random patterns, if the average fraction of active neurons (activity) is f, the probability for two randomly chosen neurons to be both active is f^2, whereas the probability for the two neurons to have different activity is $2f(1-f)$. In order that the probability for long term depression is approximately equal to the probability of long term potentiation, we choose $q_- = \frac{fq_+}{2(1-f)}$. The result of a single pattern learning is to have the active neurons belonging to a pattern providing a synaptic input to neurons belonging to the same pattern. In this way stable attractors are created. The learning of a transition results in making neurons belonging to a pattern have a finite probability of providing a nonzero synaptic input to neurons belonging to a different pattern. Let us suppose the network has learned the transition from pattern η_i^μ to pattern η_i^ν. If the network state is made to correspond to pattern η_i^μ ($S_i = \eta_i^\mu$) and then let free to evolve, there will be a nonzero probability for the network to make a transition from pattern η_i^μ to pattern η_i^ν. In case several patterns and transitions have been learned, the transition probability is shown, for a suitable choice of the network parameters, to be a monotonically increasing function of the relative frequency of presentation of the patterns during the learning phase.

2.3 Summary

Learning temporally neighboring stimuli form correlated attractors in the cortical network dynamics. These correlations have been suggested to influence the probability for the network to make transitions between different attractors. To study such phenomena, we employed a Hopfield-like network with binary synapses and binary neurons. Noise is included in the model as the pseudo-temperature of the Glauber dynamics. When the network is exposed repeatedly to a sequence of stimuli, the synaptic matrix is modified according to a Hebbian learning rule.

Chapter 3

Learning Markov processes

A Markov process is a random process in which the probability for a certain state to occur depends only on the present state of the system, and not on the events leading up to the present state. Given a Markov matrix M, the element $M_{\nu\mu}$ is the probability of transition from state s_μ to state s_ν. To teach the network a pattern sequence generated by a Markov process, a pattern η_i^μ is randomly chosen from a pool of p patterns and presented to the network, which learns it. The next pattern η_i^ν is chosen according to the transition probability of the Markov process from state μ to state ν (see code in Section A.10). The two patterns are subsequently presented to the network, which thus learns the transition from the first to the second. This process is repeated until the synaptic matrix has reached its asymptotic configuration. Alternatively, the probability for each synapse to be potentiated can be analytically calculated in the limit for a pattern sequence of infinite length [54]. Calculating the weight matrix in this way is computationally less expensive than on-line learning. Given the auxiliary variables P_{ij} and Q_{ij}, which are respectively proportional to the number of events leading to synapse potentiation and depression (see code in Section A.9),

$$\begin{cases} P_{ij} = \sum_{\mu=1}^{p}(\eta_i^\mu \eta_j^\mu q_+ + \sum_{\nu=1}^{p}(\eta_i^\nu \eta_j^\mu M_{\mu\nu} \lambda q_+)) \\ Q_{ij} = \sum_{\mu=1}^{p}((1-\eta_i^\mu)\eta_j^\mu q_- + q_- \eta_i^\mu (1-\eta_j^\mu)) \end{cases} \tag{3.1}$$

the probability for the synapse J_{ij} to be 1 is given by $p_{ij} = \frac{P_{ij}}{P_{ij}+Q_{ij}}$.

3.1 Results

We start demonstrating the network's ability in learning and recalling a single sequence of patterns. The setup used now and in the following Sections is showed in Figure 3.1. A input layer of neurons is directly connected to the

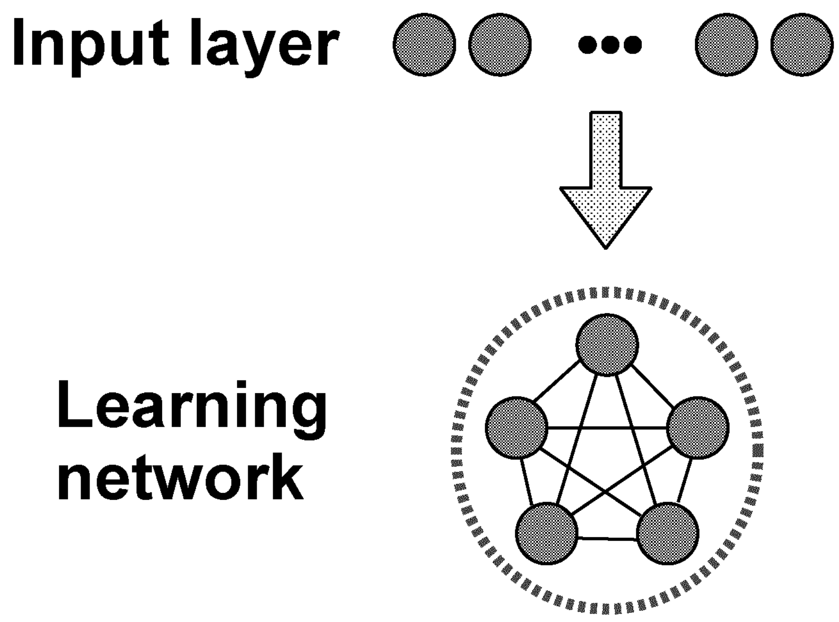


Figure 3.1: Schematic showing the direct connection of the input layer to the learning network for learning the transition statistic of a pattern sequence generated by a Markov process. The input layer does not play any active role, and is only used for clarity. The input layer neurons make a one-to-one synaptic contact with the learning network neurons. Input patterns provide a non-zero input to a fraction f (on average) of the learning network neurons. The input from the input layer is strong enough to drive the dynamics of the learning network neurons.

learning network. The input weights from the learning network to the input layer are all set to zero and the weights from the input layer to the learning input are set to dominate the network's dynamics, when an external input is present. A set of orthogonal patterns and a set of partially overlapping patterns have been chosen to illustrate some of the network features in two examples. The partially overlapping patterns were created to look like integer numbers. The network was able to reproduce the sequences. The effects of an external input on the network's dynamics are briefly examined: The external input affected the behavior of the network, by biasing its transition probabilities. The performance of the network was then evaluated in the case, where a Markov chain was used to generate the input sequence used in the learning phase. The network was able to reproduce the statistics of the input sequence, and its performance depended on β and I_0. The on-line learning procedure and the analytical derivation of the synaptic matrix led to equivalent results.

3.1.1 Learning pattern sequences

Our model neural network was able to learn a set of patterns η_i^μ, $\mu = 1 \dots p$, using the learning rule described in Section 2.2.2. The learning process created a set of stable attractors of the network's dynamics. When the transitions between patterns are learned, the attractors created are metastable. Let us consider a network that learned the transitions from pattern η_i^μ to pattern $\eta_i^{\mu+1}$ for $\mu < p$ and from pattern η_i^p to pattern η_i^1. The learning process has formed a closed chain of metastable attractors, where attractor μ corresponds to pattern η_i^μ. When it os left free to evolve, the network has, at each timestep, a finite probability of making a transitions from attractor μ to attractor $\mu + 1$ for $\mu < p$ and from attractor p to attractor 1.

The role of the pseudo-temperature

The pseudo-temperature, appearing as its inverse β (see Section 2.2.1), can alter the recall process by influencing the escape rate from the attractors of the dynamics. A low pseudo-temperature (high β) decreases the escape rate (increase the escape time) and so prevents the transition of the network to another attractor. The relationship between escape time and temperature is shown in Figure 3.2. Figure 3.3 shows the evolution in time of the overlaps $m^\mu(t)$ of the network state with the patterns η_i^μ, $\mu = 1 \dots p$ (see Section 2.2.1). Each time the plot lines cross each other, we say a transition has occurred (see definition in Section 2.2.1). As the network learned a closed chain of seven patterns, after seven transitions the network state has again

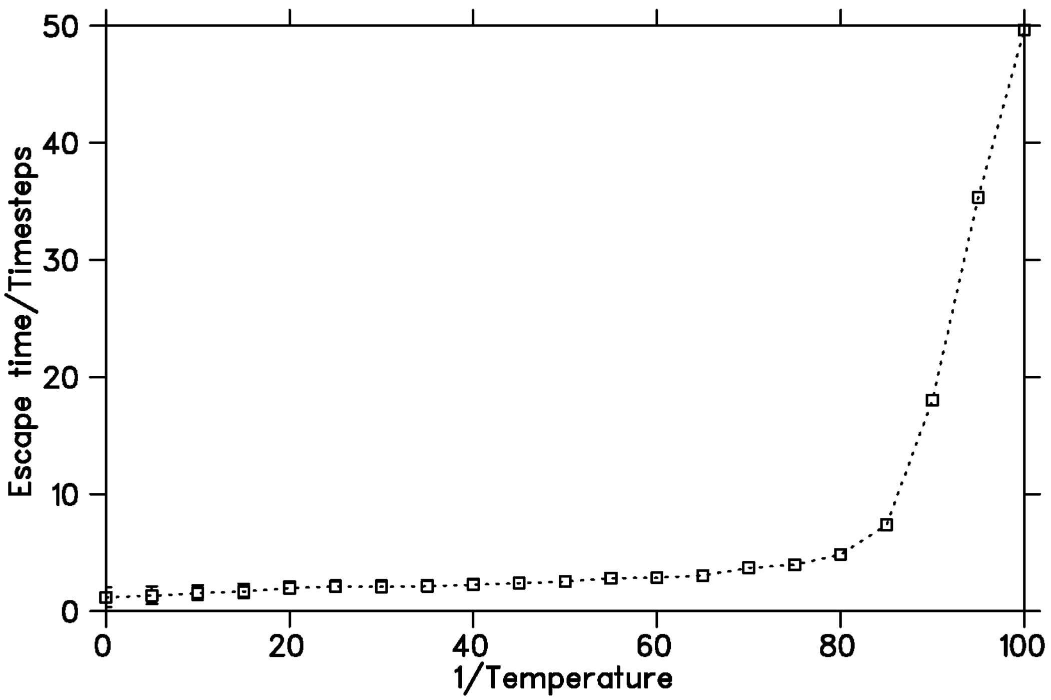


Figure 3.2: Escape time (one timestep corresponds to one update of the whole network) is plotted as a function of β (see Section 2.2.1). The escape time increases monotonically with the decreasing temperature. The values are averages over all learned transitions for 10 networks with different synaptic matrices but otherwise identical (490 neurons, 7 orthogonal patterns, activity = 1/7, $\lambda = 0.4$).

maximum overlap with the starting pattern.

About (non-) orthogonal patterns

In Section 3.1.1 we used both orthogonal and non-orthogonal patterns. In general, patterns can be randomly generated, while keeping the fraction of active neurons (activity) constant . In this case, some neurons belong to more than one pattern, making the attractors less stable. When the networks state corresponds to one attractor, the difference between the minimum field on active neurons and the maximum field on quiescent neurons is a function of the overlaps between patterns (figure 3.6). When this difference is negative, it is impossible to find a suitable fixed point for the inhibition (see Section 2.2.1) and no stable attractors can be formed. Figure 3.4 represents one familiar-looking set of non-orthogonal patterns. A network learned a sequence of these patterns, with the same protocol previously described. The evolution of the network state is displayed in figure 3.5.

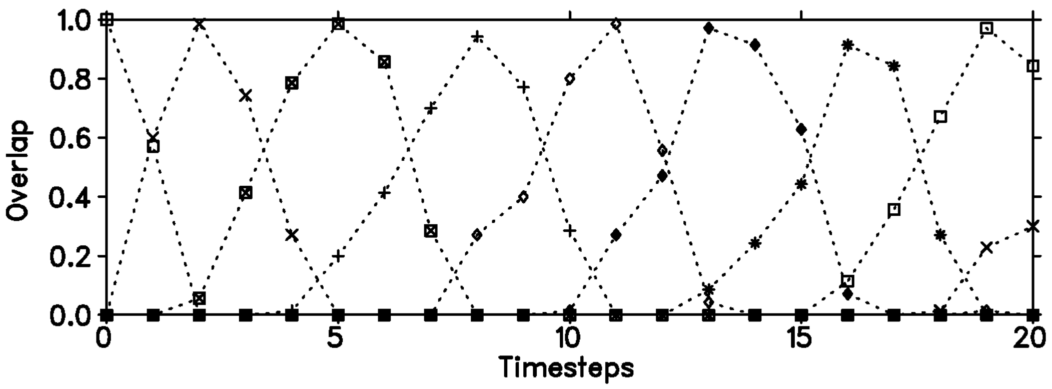


Figure 3.3: Evolution in time (timesteps correspond to one update of the whole network) of the overlaps $m^\mu(t)$ of the network state with the learned patterns η_i^μ. One timestep corresponds to the update of all network neurons. When the plot lines cross each other, we say a transition has occurred. When one overlap reaches its maximum, the network state has also a non-zero overlap with the previous and the next pattern. The escape time is consistent with those showed in figure 3.2 for the same temperature (490 neurons, 7 orthogonal patterns, activity = 1/7, $\beta = 50$).

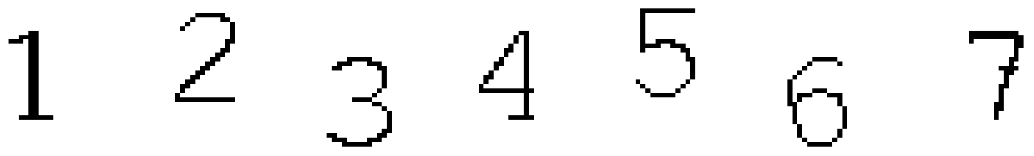


Figure 3.4: Seven patterns, embedded in a 35×23 matrix. In each matrix, only 50 elements are nonzero (activity = 0.062). The output of a network after learning this sequence is showed in figure 3.5.

Influence of external input

An external input consists of an additive term e_i to the fields h_i. External inputs can have a strong influence on the dynamics of the network. Figure 3.7 shows the influence of an external input on the transition probabilities for the generated pattern sequence of a network after having learned the transition probabilities from one pattern to two others. The Markov process used to generate the training sequence assigns the same transition probability (0.5) to the two possible transitions which can occur. An external input on the neurons belonging to one of the two patterns to which the transition can be made, biases the transition probability in favor of that pattern.

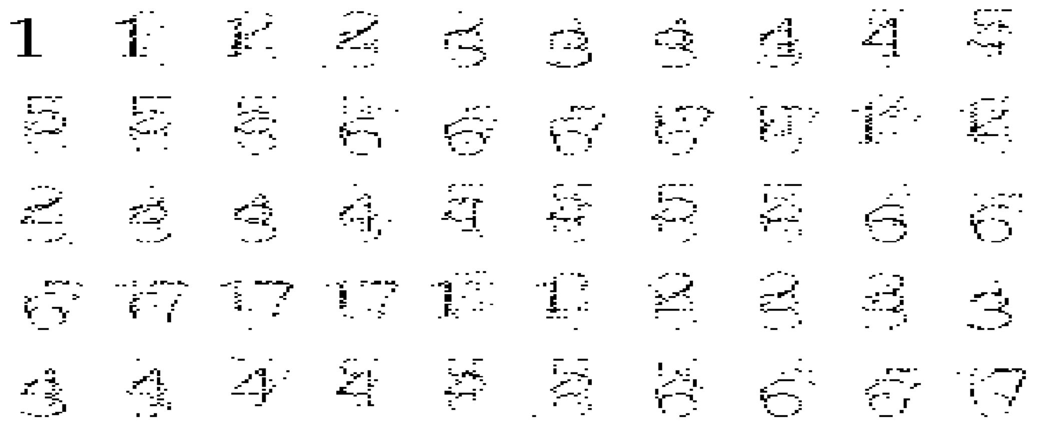


Figure 3.5: Evolution in time of the network state over 30 time steps, after having learned the sequence of patterns shown in figure 3.4. The network starts from pattern 1. The network state goes through the sequence three times (805 neurons, activity = 0.062).

3.1.2 Performance evaluation

The Markov matrix M used to generate the input patterns was compared with the transition probability matrix of the network T. The closer the elements of the transition probability matrix T are to the corresponding values of the Markov matrix M, the better the network is reproducing the statistics of the Markov process. We have analyzed the performance of the network with varying I_0 and β using a Markov matrix whose nonzero elements of the rows belong to the set $\{0.1, 0.2, 0.3, 0.4\}$ (see Table 3.1 and figures 3.9, and 3.8).

$$\Pi = \frac{1}{4}\sum_{\mu=1}^{4} \frac{|m_\mu - t_\mu|}{\frac{m_\mu + t_\mu}{2}} \tag{3.2}$$

was chosen as the performance index of a network, where $m_1, \ldots, m_q$ are all possible non-zero values of the elements of the transition probability matrix (in our case $m_1 = 0.1$, $m_2 = 0.2$, $m_3 = 0.3$, and $m_4 = 0.4$), and $t_1, \ldots, t_q$ are the average values of the elements of the transition probability matrix corresponding to elements of the Markov matrix equal to $m_1, \ldots, m_q$. Figures 3.8 and 3.9 show the performance of the network for different combinations of β and I_0. Figures 3.10, 3.11, 3.12, 3.13, and 3.14 show the the performance index Π and the mean value of the transition probabilities as a function of I_0 and β for different combinations of λ_f and λ_b (see Section 2.2.2).

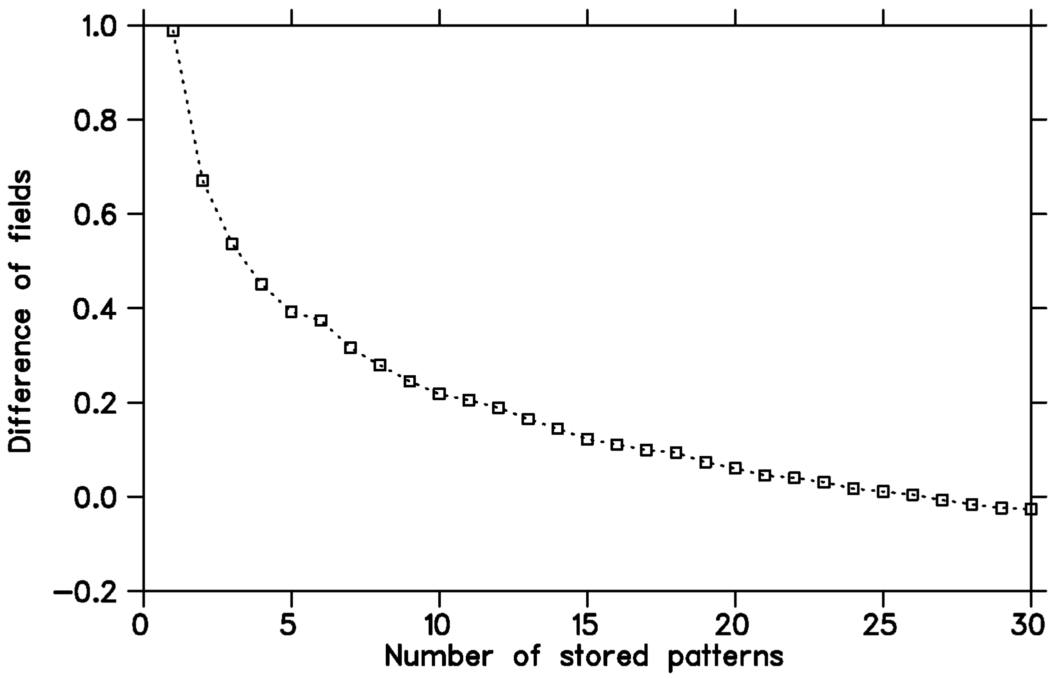


Figure 3.6: Difference between the minimum fields on active neurons and the maximum field on quiescent neurons, when the network state corresponds to one attractor of the dynamics. This difference corresponds to the width between the two peaks of the bimodal histogram plotted in Figure 2.1. The plotted values are averages over all attractors (stored patterns) and for 10 networks with different synaptic matrices, but otherwise identical (490 neurons, activity = 1/7).

Online learning

On page 37 we said that the analytical derivation of the synaptic matrix is equivalent to the learning rule described in Section 2.2.2, in the limit for the presentation of an infinite number of patterns and slow learning. Figures 3.15 and 3.16 qualitatively show this equivalence. The two figures show the transition probabilities of the network as a function of the Markov process transition probabilities, after learning using two different learning procedures. In each figure, one plot corresponds to a network trained using the learning rule described in Section 2.2.2, which we call *online* learning (see Section 3.1.3 for some remarks about the use of this learning procedure). In the same figure, the other plot corresponds to a network whose synaptic matrix has been analytically calculated using Equation 3.1, for the same λ_f, β and I_0.

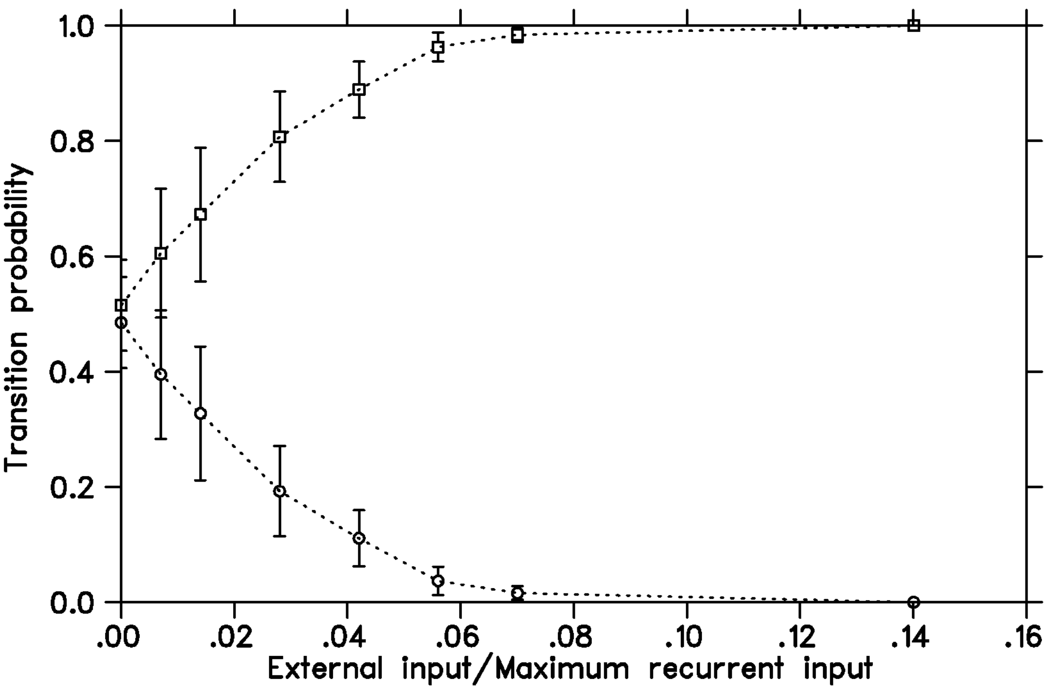


Figure 3.7: Transition probabilities from a starting pattern to two others. The neurons belonging to one pattern are receiving an external input, all others are not. With a zero external field intensity, the transition probabilities are the same (within the measurement error). When the external input is switched on, the transitions are biased toward the pattern whose neurons are receiving an input (490 neurons, 7 patterns, $\lambda = 0.1$, $\beta = 60$, $I_0 = 0.03$).

3.1.3 Classification of states in a Markov chain

Given a Markov matrix, the states of the corresponding Markov process (in this case, Markov chain), can be classified in distinct classes, depending on the behavior in the limit for an infinite number of transitions [15]. Given an initial state, if the process will return to this state with probability 1, the state is called *recurrent*. The time of first return is a stochastic variable called the *recurrence time*, and the state is called positive-recurrent or null-recurrent if the recurrence time is, respectively, finite or infinite. If the return to a state has probability less than 1, the state is called *transient*. If a state cannot be reached by any other state, it is called *ephemeral*. An ephemeral state can only happen to be a starting state. If the chain starts from a state a, which is then occupied at times iT, $i = 1, 2, 3, \ldots$, the state is said to be periodic. A state which is not periodic is called *aperiodic*, and a state which is positive recurrent and aperiodic is called *ergodic*.

To teach a network all transition probabilities of a given Markov matrix,

0.0	0.0	0.3	0.0	0.2	0.4	0.1
0.4	0.0	0.2	0.0	0.1	0.0	0.3
0.3	0.1	0.0	0.2	0.0	0.0	0.4
0.2	0.4	0.0	0.0	0.3	0.1	0.0
0.1	0.3	0.0	0.4	0.0	0.2	0.0
0.0	0.0	0.4	0.1	0.0	0.0	0.2
0.0	0.2	0.1	0.3	0.4	0.3	0.0

Table 3.1: Markov matrix used to study the performance of the network. Nonzero elements in a row belong to the set $\{0.1, 0.2, 0.3, 0.4\}$ with no repetitions.

it is necessary to generate a pattern sequence in which all possible transitions occur with a frequency sufficient to have the synaptic matrix reach its asymptotic configuration (see Section 3). The only kind of states suitable for the generation of the training pattern sequences are the ergodic and periodic states. Markov processes containing null-recurrent and ephemeral states cannot, per definition, produce a suitable training sequence. To go around this problem, it is possible to create patterns in pairs: The first pattern is chosen randomly with uniform probability among the available patterns, the second one is chosen according to the Markov matrix probabilities. To avoid the potentiation of the synapses between the neurons active before the presentation of the first pattern and the neurons activated by the first pattern (corresponding to a non-existing transition), all neurons are set to the quiescent state. The two patterns are then presented to the network, one after the other. The network learns the transition probability between the first pattern and the second one by potentiating the synapses connecting the neurons active after the presentation of the first pattern to the neurons active after the presentation of the second pattern. This process is repeated a number of time sufficient to have the synaptic matrix reach its asymptotic configuration. The minimum number of times depends on the average network activity f (that corresponds to the activity of the input patterns), the minimum probability m in the Markov matrix, and on the the learning rates q_+, q_-, and $q_\times$. As q_- and $q_\times$ are proportional to q_+, the minimum number of presentations is $1/(f^2 \lambda q_+ m)$, where λ is the minimum of λ_f and λ_b.

If all states of a Markov process are ergodic or periodic, all transition probabilities are present in a sequence generated by the Markov process. So, the sequence can be directly used to train a network.

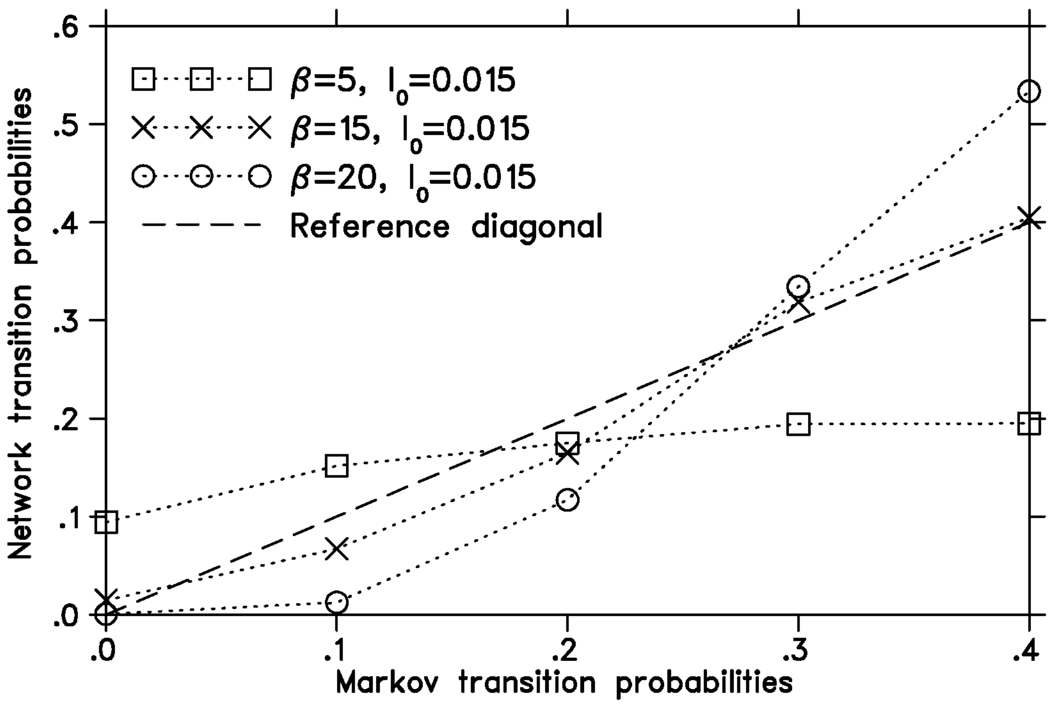


Figure 3.8: Network transition probabilities as a function of the Markov transition probabilities for fixed I_0 and variable temperature β. The error bars have been calculated using Equation 2.4 , with $k = 1$ and $n = 7000$, and are not visible because smaller than the symbols used in the plot. If compared with the case $\beta = 15$, an increase in β (decrease in temperature) decreases the network transition probabilities corresponding to low Markov transition probabilities and increases those corresponding to high Markov probabilities. A decrease in β (increase in temperature) makes the network transition probabilities depend less on the Markov transition probabilities. The corresponding plot is more shallow, indicating a reduced capability in reproducing the statistics of the input sequences (490 neurons, 7 patterns, activity $f = 1/7$, $\lambda_f = 0.1$, values are averages over 10 trials.)

Random Markov matrix

Until now we have used the Markov matrix showed in Table 3.1. The reason for that is our intention to separate the effects linked to the choice of the parameters from those linked to the Markov matrix. Keeping the Markov matrix constant is a way to enforce that. Figure 3.17 shows the transition probabilities of the network as a function of the Markov transition probabilities for different I_0 and constant λ and β, when using a Markov matrix with randomly generated elements. From Figure 3.17 one can notice that a decreasing I_0 tends to make the distribution more shallow and so the transition probabilities less dependent on the Markov probabilities. This is, qualita-

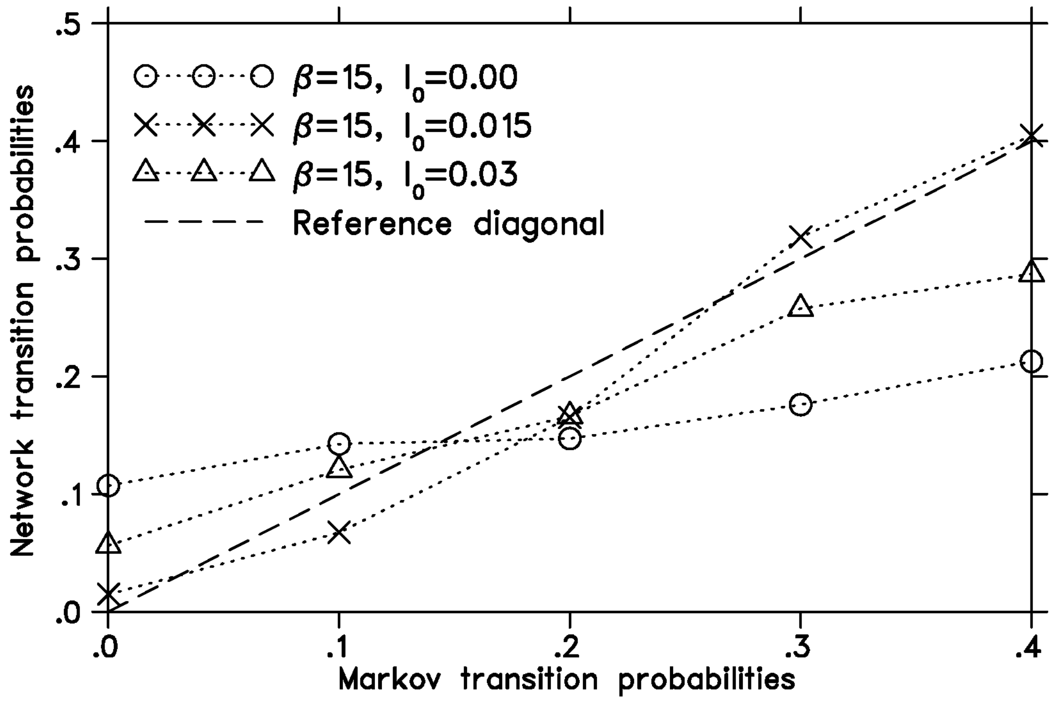


Figure 3.9: Network transition probabilities as a function of the Markov transition probabilities for fixed temperature β and variable I_0. The error bars have been calculated using Equation 2.4 , with $k = 1$ and $n = 7000$, and are not visible because smaller than the symbols used in the plot.. If compared with the case $I_0 = 0.015$, both an increase and a decrease in I_0 make the network transition probabilities depend less on the Markov transition probabilities. The corresponding plots are more shallow, indicating a reduced capability in reproducing the statistics of the input sequences (490 neurons, 7 patterns, activity $f = 1/7$, $\lambda_f = 0.1$, values are averages over 10 trials).

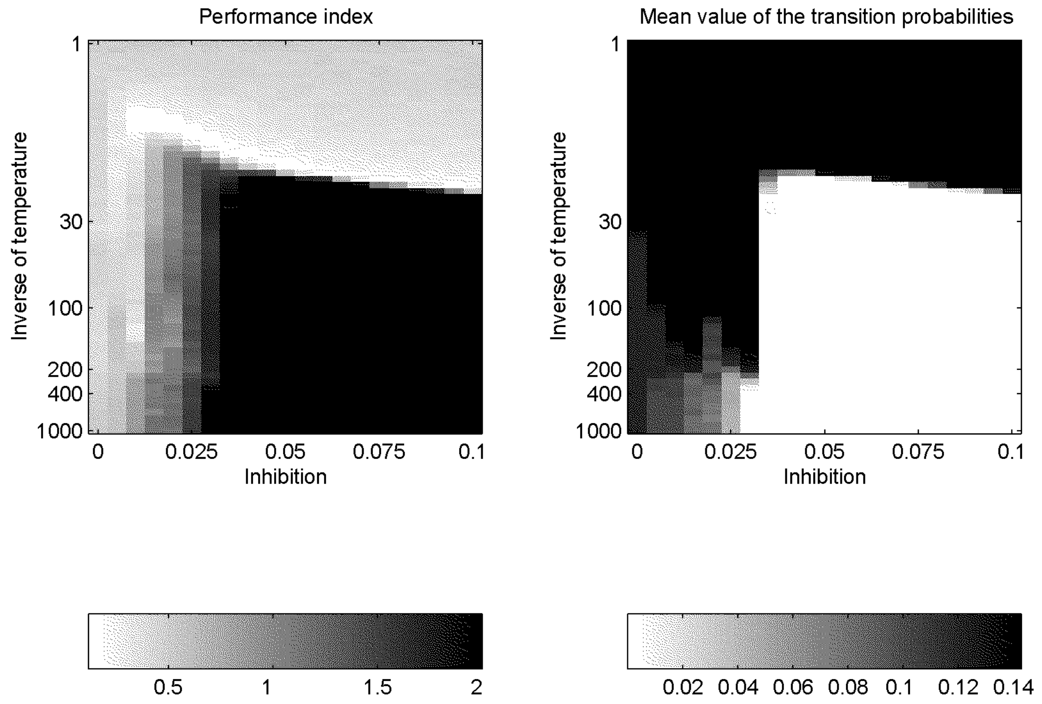


Figure 3.10: Diagram of the performance index and of the mean value of the transition probabilities. Left: Diagram of the performance index Π as a function of I_0 and β. Right: Diagram of the mean value of the transition probabilities as a function of I_0 and β ($\lambda_f = 0.1$ and $\lambda_b = 0$). Both diagrams show the average over 10 networks with different synaptic configurations, but otherwise identical. The transition probability matrix was measured by sampling the transitions 100 times per pattern. The performance of the network is maximal in a region around $I_0 = 0.01$, $\beta = 14$. The optimal β is roughly proportional to the square root of I_0, and for $I_0 \geq 0.04$, the best performance lies in the area just above the sudden change in performance. The change in performance goes together (see the right-hand plot) with a sudden decrease of the mean value of the transition probability for an increasing β. For any I_0, very low temperatures trap the network in the basin of attractions in which it is initially set. This can be seen in the diagrams for $I_0 \geq 0.03$. The same behavior can be observed for $\beta \geq 25$ and increasing I_0 (490 neurons, 7 patterns, activity 1/7).

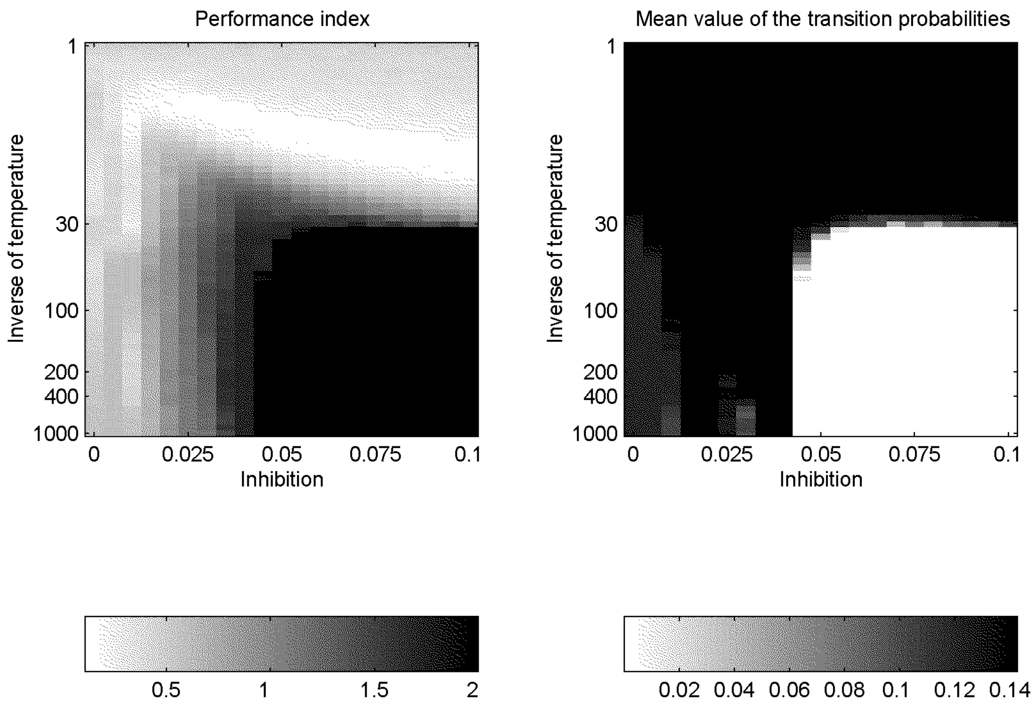


Figure 3.11: Same as Figure 3.10, but with $\lambda_f = 0.3$ and $\lambda_b = 0$. If compared to Figure 3.10, the area corresponding to the best performance is shifted toward lower β and higher I_0 and the sudden change in performance for a given I_0 can be found at a higher β. In this case too, the change in performance goes together (see the right-hand plot) with a sudden decrease of the mean value of the transition probability for an increasing β. (see also Figure 3.10). The optimal β is still roughly proportional to the square root of I_0.

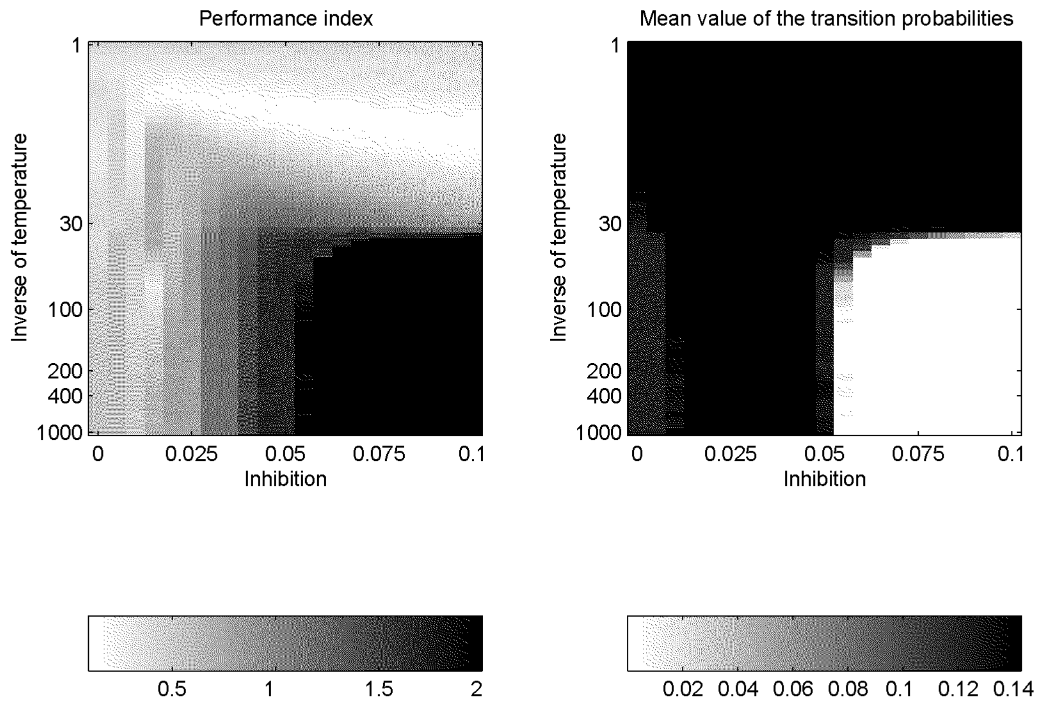


Figure 3.12: Same as Figure 3.10, but with $\lambda_f = 0.5$ and $\lambda_b = 0$. If compared to Figure 3.10 and Figure 3.11, the area corresponding to the best performance is even more shifted toward lower β and higher I_0, and the sudden change in performance for a given I_0 can be found at a even higher β. In this case too, the change in performance goes together (see the right-hand plot) with a sudden decrease of the mean value of the transition probability for an increasing β. The optimal β is in this case roughly proportional to I_0. This might be due to the shift of the best performance area toward higher I_0.

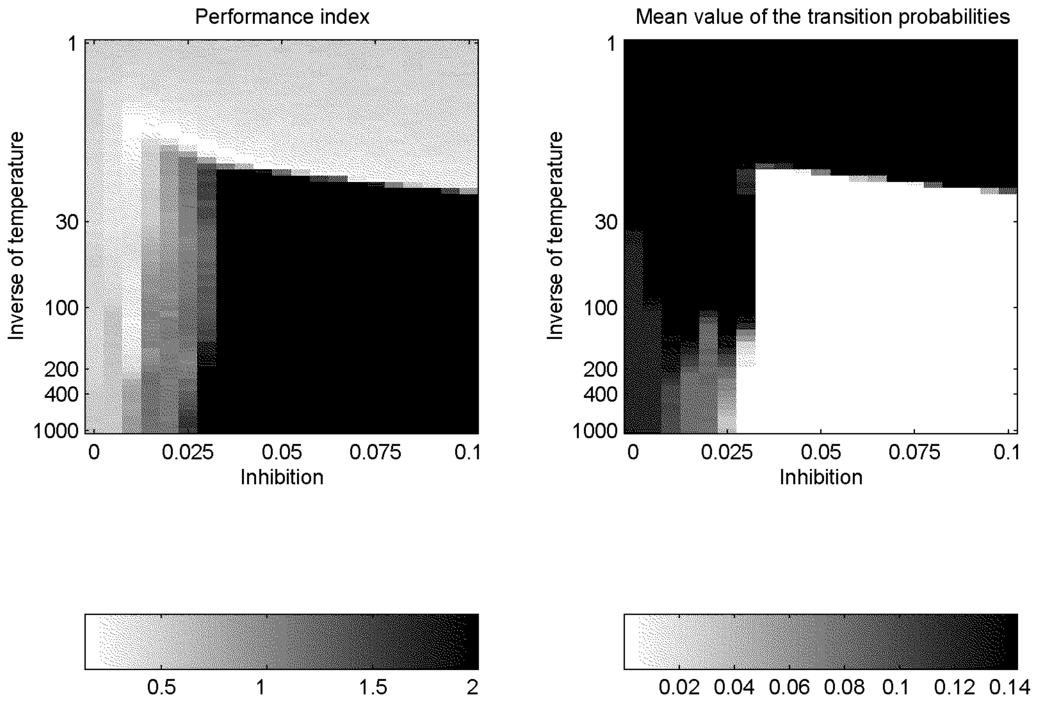


Figure 3.13: Same as Figure 3.10, but with $\lambda_f = 0.08$ and $\lambda_b = 0.02$. The behavior is indeed very similar to the one produced with $\lambda_f = 0.1$ and $\lambda_b = 0$, which is showed in figure 3.10.

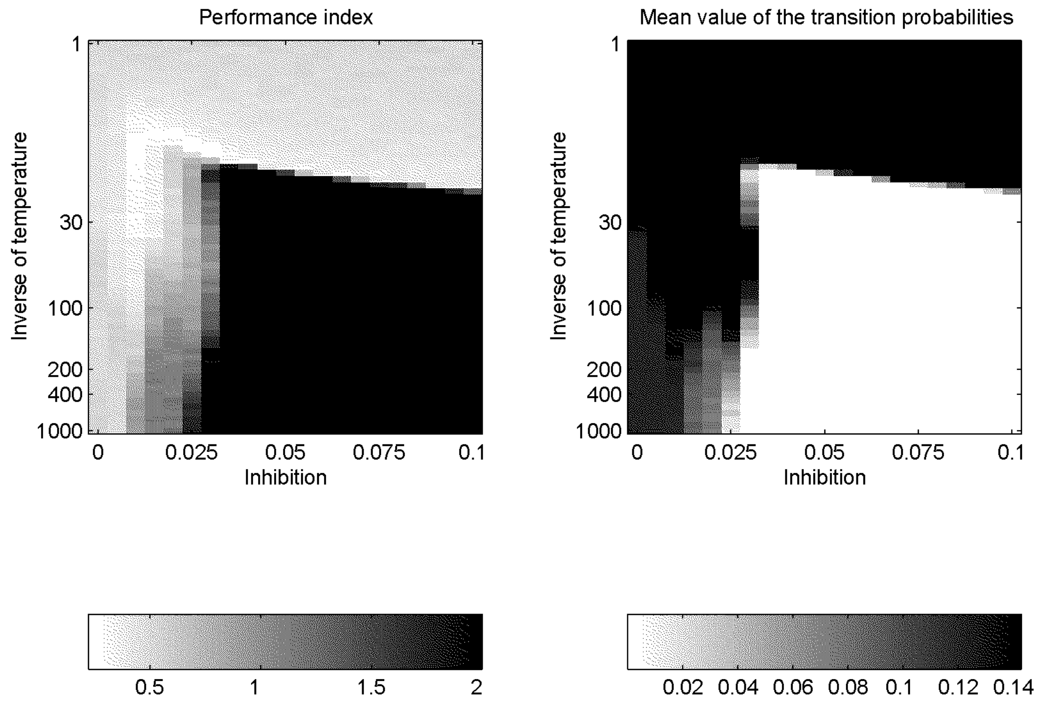


Figure 3.14: Same as Figure 3.13, but with $\lambda_f = 0.06$ and $\lambda_b = 0.04$.

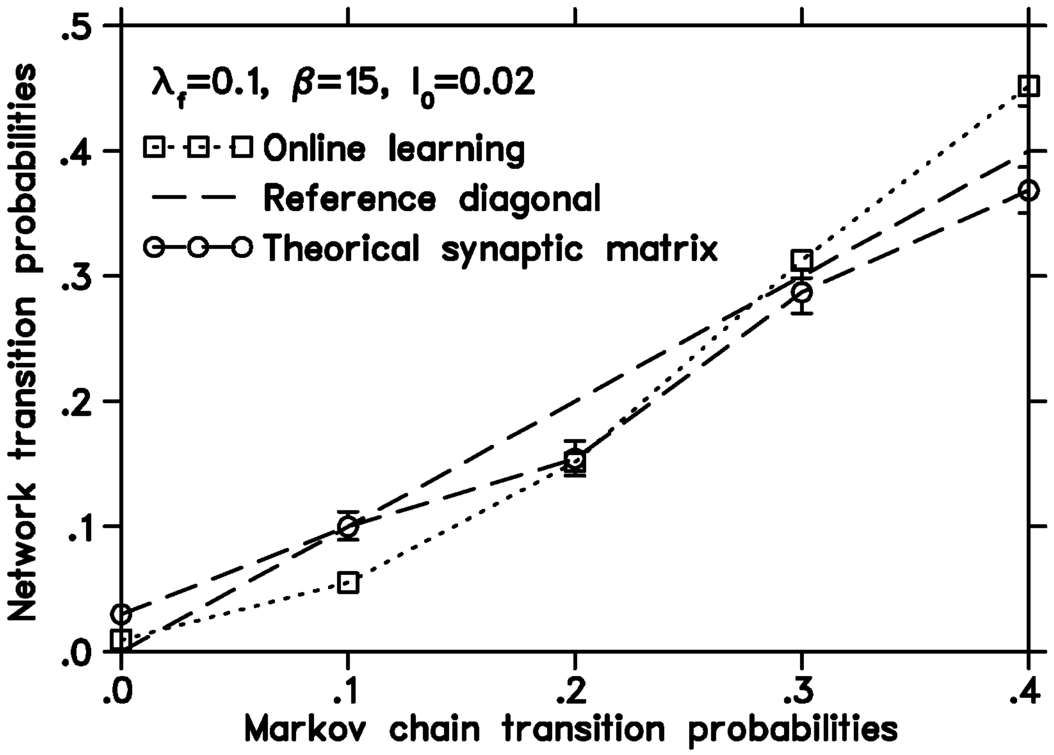


Figure 3.15: Transition probabilities as a function of the Markov transition probabilities for a network trained with the online learning rule described on page 37 and for a network whose synaptic matrix has been analytically calculated using Equation 3.1, for $\lambda_f = 0.1$, $\beta = 15$ and $I_0 = 0.02$. In both plots, upper and lower error bars have been calculated using Equation 2.4 , with $k = 1$ and $n = 700$ (490 neurons, 7 patterns).

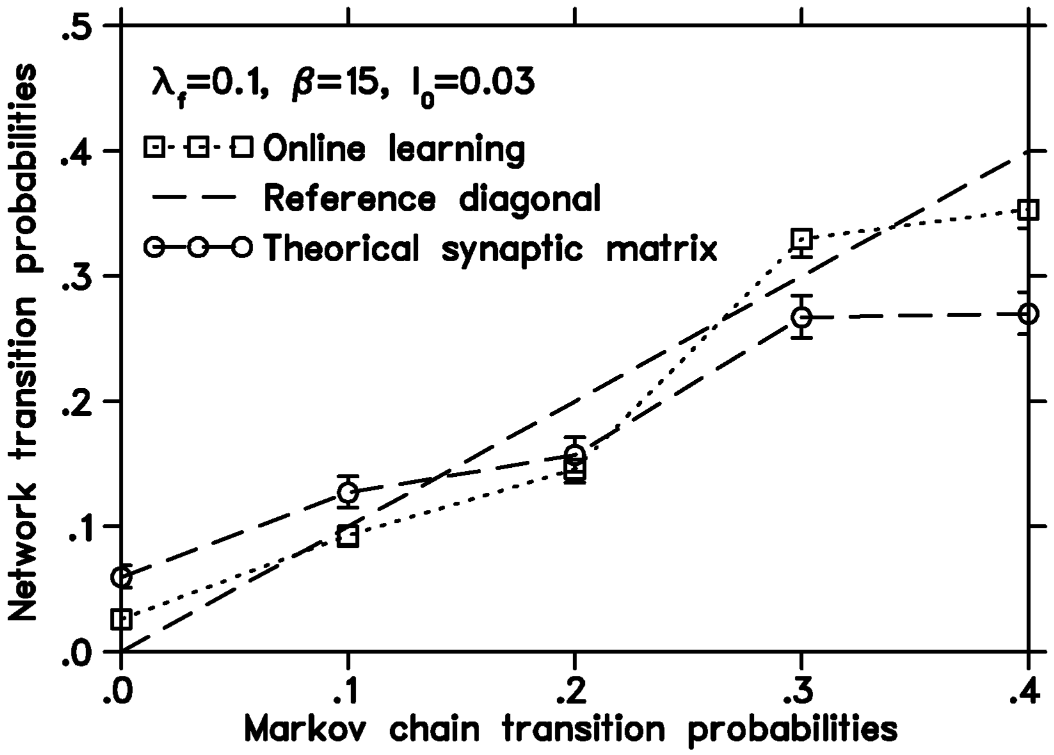
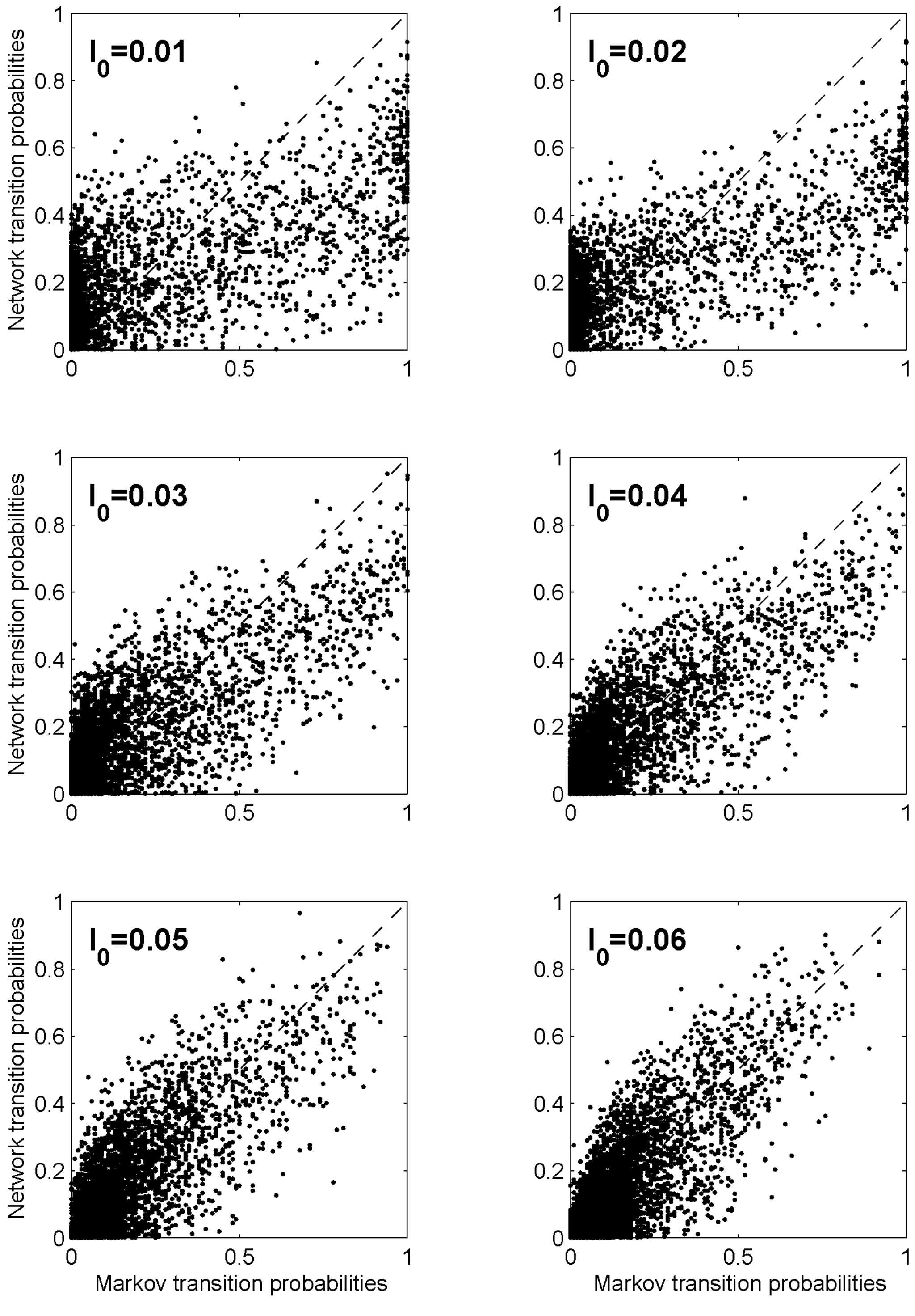


Figure 3.16: Transition probabilities of the networks as a function of the Markov transition probabilities for a network trained with the online learning rule described on page 37 and for a network whose synaptic matrix has been analytically calculated using Equation 3.1, for $\lambda_f = 0.1$, $\beta = 15$ and $I_0 = 0.03$. In both plots, upper and lower error bars have been calculated using Equation 2.4 , with $k = 1$ and $n = 700$ (490 neurons, 7 patterns).



I_0=0.01
I_0=0.02
I_0=0.03
I_0=0.04
I_0=0.05
I_0=0.06
Network transition probabilities
Markov transition probabilities
0
0.2
0.4
0.6
0.8
1
0.5

Figure 3.17: Values of the elements of the transition probability matrix T plotted against the corresponding values of the Markov matrix M for six different values of I_0 ($\beta = 15$, $\lambda_f = 0.1$, $\lambda_b = 0$). The distributions show the same qualitative behavior showed in Figure 3.9.

tively, the same behavior showed in Figure 3.9.

3.1.4 The palimpsest property

The neural network we employ shows the *palimpsest* property: During the learning process, information about stimuli that occurred earlier in time are forgotten as new information is accommodated in the synaptic matrix [49]. How long a stimulus is remembered by the network depends on how many synapses are changed after the presentation of a stimulus. The learning rule we use is implemented as a stochastic process: Synaptic long term potentiation and depression depend on the network state before and after the presentation of the last stimulus (pattern). As it is described in Section 2.2.2, the modification of the state of a synapse depends on the three parameters q_+, q_-, and $q_\times$ (both q_- and $q_\times$ are proportional to q_+). The effects of the palimpsest property can easily be shown by teaching one network several patterns, one after the other. After the learning of one pattern is complete, each neuron that is active for that pattern will receive a synaptic input from no more than $n = fN - 1$ neurons, where N is the total number of neurons and f the activity of the network. If the patterns are randomly chosen, that is, if the activities of each neuron for each pattern are not correlated, then n will decrease as the learning of other patterns progresses. The pattern remembered by the network belongs to a *window in time*, whose width is a function of the fraction of synapses updated after the presentation of a new stimulus, and thus is inversely proportional to q_+.

Another way to show the effects of the palimpsest property is to show how it affects the performance of the network. The test consists of two parts: In the first part, the network is trained, with a pattern sequence generated by the Markov process described by the Markov matrix 3.1. Then, the Markov matrix is slightly modified, by changing the position of some transition probabilities (see Table 3.2) A pattern sequence was then generated using the new Markov matrix. The pattern sequence was used to train the network. At regular intervals (every 100 presentations), the performance of the network was tested. The performance was computed against the original Markov matrix (Table 3.1). The results are showed in Figure 3.18 and 3.19 for different numbers of presentations. The performance of the network worsened as the learning of the pattern sequence generated by the new Markov matrix proceeded. After about 1500 presentations, the average performance converged to its asymptotic value. The asymptotic value of the performance is not much different from the original one due to the high similarity of the two Markov matrices used.

The role of the learning rate becomes evident when the network is trained

0.4	0.0	0.3	0.0	0.2	0.4	0.1
0.0	**0.4**	0.2	0.0	0.1	0.0	0.3
0.3	0.1	0.0	0.2	0.0	0.0	0.4
0.2	**0.0**	**0.4**	0.0	0.3	0.1	0.0
0.1	0.3	**0.0**	0.0	0.0	0.2	0.0
0.0	0.0	0.0	0.1	0.0	0.0	0.2
0.0	0.2	0.1	0.3	0.4	0.3	0.0

Table 3.2: Markov matrix used to study the effects of the palimpsest property on the performance of the network. The matrix is a modified version of the matrix showed in Table 3.1, where the elements in boldface in the same column have been swapped.

by presenting it patterns from a sequence produced by a stochastic process. In this case, q_+ is proportional to the number of synapses whose states are changed after each presentation of a new stimulus. The width of the memory temporal window of the network, as it has been explained above, depends on the magnitude of q_+. The larger the value of q_+, the narrower the time window. So, a large q_+ makes the network forget faster old stimuli and give more weight to the more recent ones. When learning the transition statistics of a pattern sequence, it is desirable to a have a learning rule that averages out information from a large number of presentations. This can be accomplished by choosing a small learning rate, which corresponds to a slow learning process. Euristically, a learning rate $q_+ = 0.01$ was found to be a good choice. A learning rate of this order of magnitude is also in accord with the learning rates measured by Miyashita [55]. It needs to be remarked that the choice of the learning rate is not critical. As $q_+ = 0.01$ represents a good choice, any smaller value will also do, and possibly also higher values, up to a limit which can be evaluated with numerical methods. In this sense, out learning rule is robust with respect to the learning rate.

3.1.5 Learning noisy patterns

Noise is ubiquitous in nature. Apparently, the nervous system is able to cope with noise and provides reliable responses to noisy inputs. A realistic model of brain functionality should show the same kind of robustness. In all our previous examples, we used only noise-free inputs. The input patterns correspond to patterns of activity of a layer of input neurons. Each pattern is imposed to the network, without errors or variability. It is reasonable to expect from a neural network some sort of robustness with respect to

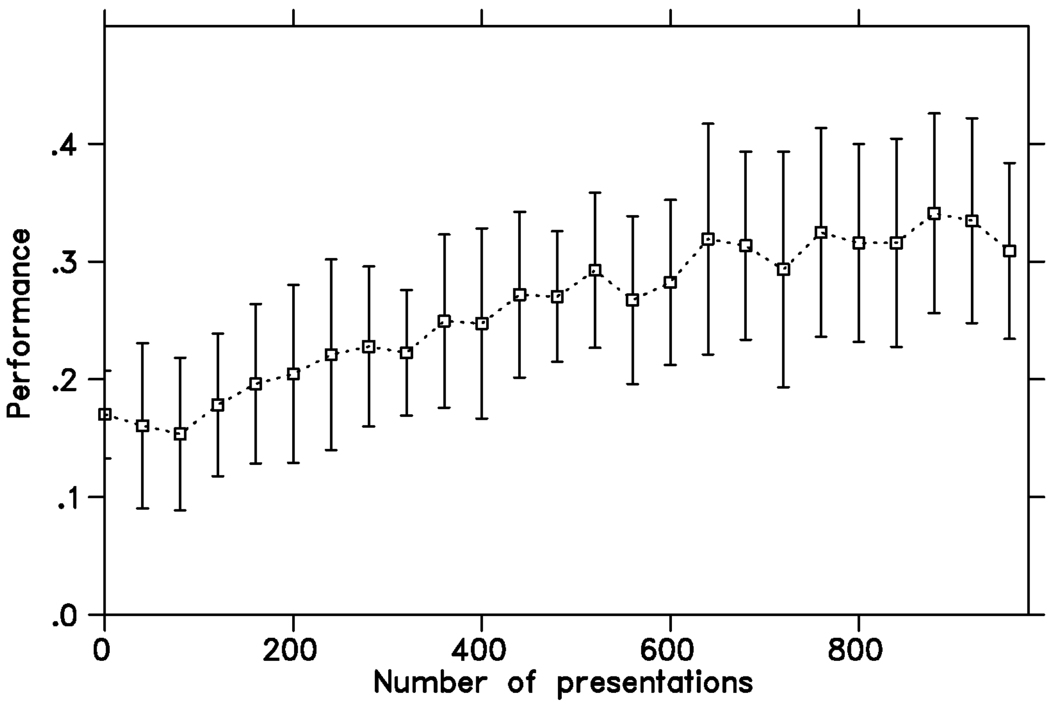


Figure 3.18: Performance of the network in generating a pattern sequence with statistics similar to those of the Markov process described by the matrix showed in Table 3.1 as a function of the number of presentations from a pattern sequence generated by a Markov process described by the matrix showed in Table 3.2 ($\beta = 15$, 7 orthogonal patterns, $\lambda_f = 0.1$, $I_0 = 0.015$, 490 neurons, data points are averages over 10 trials).

noisy inputs, although it would be excessive to expect the performance of the network to be unaffected by the level of noise in the activity patterns used as input. A performance that is 'gracefully decaying' with an increasing noise level would be a more sensible expectation.

We tested the performance of our network using input patterns corrupted by noise. The network was tested with increasing levels of noise. The noise level $\mathcal{L}$ is expressed as the probability for each neuron of the input layer to have its state flipped from 1 to 0 or viceversa. One of the effects of noise is to modify the activity of the input pattern. The activity f of the input patterns is defined as the fraction of active neurons, averaged over all instances of all patterns. Let us suppose the activity of a network after being presented with noiseless patterns is f_0 (the total number of neurons is N). The expected number of neurons going, becuase of the noise, from $S = 1$ to $S = 0$ is $f_0 N \mathcal{L}$, and the expected number of neurons going from state $S = 0$ to $S = 1$ is

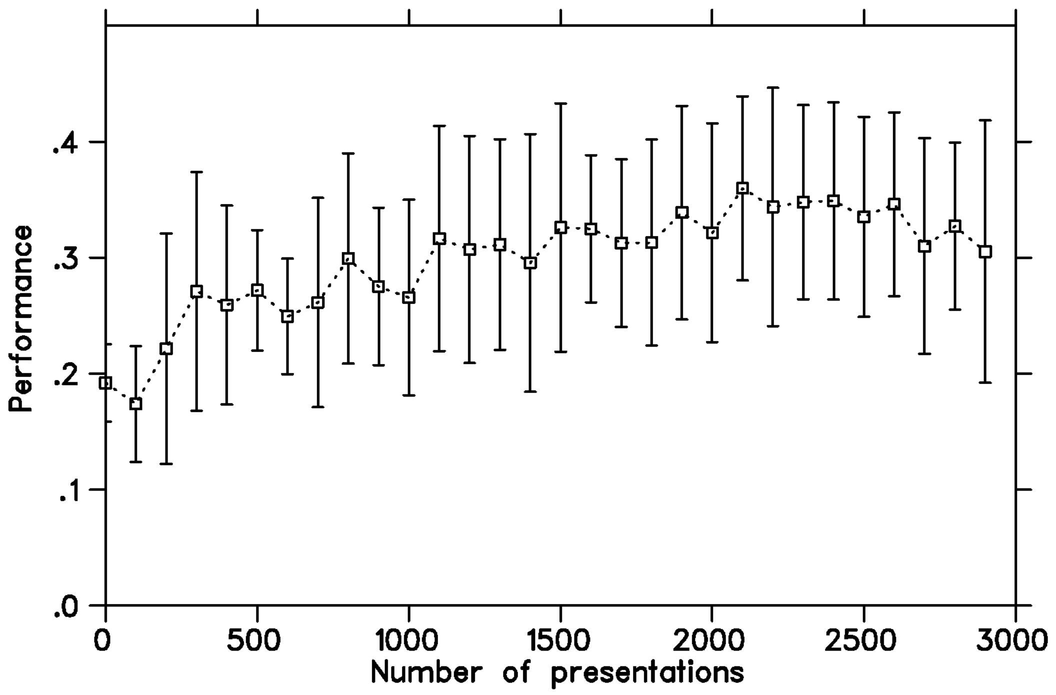


Figure 3.19: Performance of the network in generating a pattern sequence with statistics similar to those of the Markov process described by the matrix showed in Table 3.1 as a function of the number of presentations from a pattern sequence generated by a Markov process described by the matrix showed in Table 3.2 ($\beta =$ 15, 7 orthogonal patterns, $\lambda_f = 0.1$, $I_0 = 0.015$, 490 neurons, data points are averages over 10 trials).

$(1-f_0)N\mathcal{L}$. Finally, the number of active neurons is $f_0N-f_0N\mathcal{L}+(1-f_0)N\mathcal{L}$ and the expected activity is $\mathcal{L}(1-2f_0)+f_0$. For a starting activity f_0 lower than 0.5, noise will tend to make more inactive neurons active than viceversa, and the activity will increase toward the asymptotic value $f = 0.5$. For this activity level, the probability to have an active neuron going inactive equals the probability of having an inactive neuron going active. In this way, the average activity is constant.

To train the network, we used the learning rule described in Section 2.2.2. Instead of a single pattern sequence, we used the Markov matrix in Table 3.1 to generate pairs of patterns (a, b). Pattern a is chosen randomly and pattern b is chosen according to the transition probabilities of the Markov matrix. Pattern a is presented to the network, and then pattern b. In this way, the network learns the transition from pattern a and pattern b. After the synaptic matrix is updated, the memory of the previous pattern is erased, so that the network will not learn the transition from pattern b to the next pattern. The reasons for this protocol are the following: If the sequence is generated from a start pattern according to the transition probabilities of a Markov matrix, there is no guarantee that the generated sequence will contain all transitions described by the Markov matrix, as explained in Section 3.1.3. A possible solutions is to generate a sequence of patterns $a_0 b_0 a_1 b_1, \ldots, a_n b_n$, where patterns a_i are randomly chosen and patterns b_i are chosen according to the Markov transition probabilities. In this case the network learns the transition probabilities (data not shown), but some noise is introduced in the synaptic matrix because of the transitions from pattern b_i to pattern a_{i+1} which do not reflect the transition probabilities of the Markov matrix. The method we use let us isolate the effects of the noise in the input patterns from other sources of noise.

Figure 3.20 shows the performance of the network, as defined in Equation 3.2 for noise levels going from 0 (no noise) to 0.033. Figure 3.21 shows the performance on a wider range of noise levels, between 0 (no noise) and 0.33. The performance quickly degrades for a noise level up to 0.02, where each neuron has a 2% probability of having its state flipped. For higher noise levels, the performance slowly approaches the value 0.4687, which is the performance of the network in the case in which all transitions have the same probability of occurring (1/7 in our case). The transition probabilities of the network depend on the inhibition I_0, the pseudo-temperature $1/\beta$, and, of course, on the synaptic matrix. At the beginning of the learning process, the synaptic matrix is set to a random state, where each synapse has the same probability of being potentiated (1) or depressed (0). The learning rule (see Section 2.2.2) modifies the synaptic links to encode information about the transition probabilities of the pattern sequence presented to the network during the learning phase. Normally, several thousand presentations are needed (see page 45). We can monitor the learning process by building a suitable set of observables. Given a synaptic matrix J and a pair of patterns η^a and η^b, $\sum_i \eta_i^b \sum_j J_{ij} \eta_j^a$ is the number of active neurons in pattern η^a connected to active neurons of pattern η^b. The number of connections between any pair of patterns changes during the learning process and when it reaches a stable value, the synaptic matrix is said to have reached an asymptotic configuration and the learning process can be terminated. If the learning process goes on, the synaptic matrix will still be modified, but the number of connections between active neurons of all patterns will be, on average, constant.

If noisy patterns are presented to the network during the learning phase, two effects of noise on the number of connections between neurons can be seen. The first one is a general increase of the number of connections between

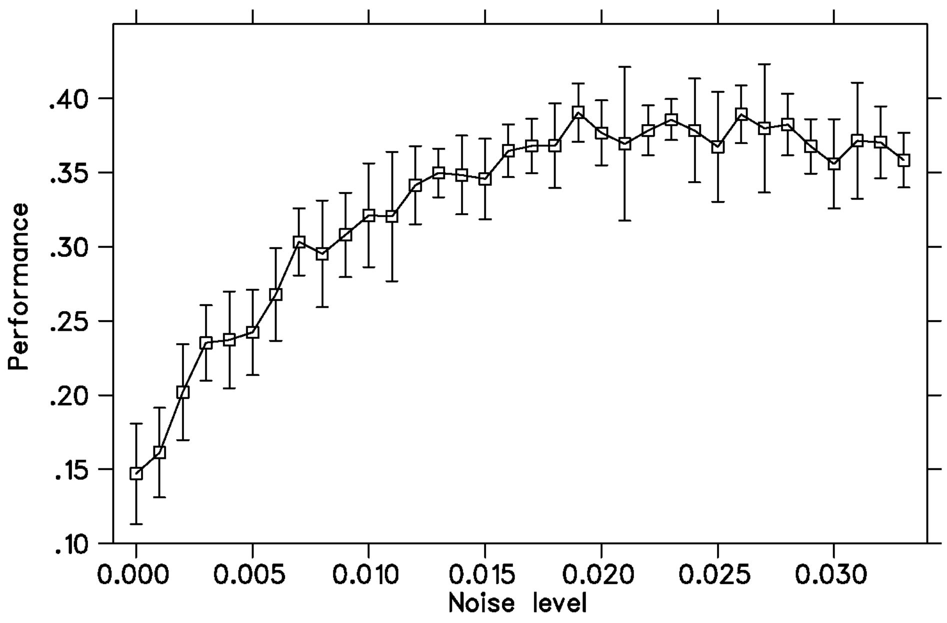


Figure 3.20: Performance of the network as a function of the noise level. The performance for a network for which all transition occur with the same probability (1/7) in our case) can be calculated from Equation 3.2 to be 0.4687. The values and error bars are calculated over 11 trials. (490 neurons, 7 patterns, $f = 1/7$, $\lambda_f = 0.1$, $I_0 = 0.01$, Markov matrix showed in Table 3.1, $\beta = 14$, 20000 transition presented during the learning phase).

all neurons. The second one is a convergence of the number of connections between any pattern toward a common value, independent of the transition probabilities of the pattern sequence presented during the learning phase. This value is reached for a high level of noise, for which the synaptic links are potentiated or depressed with the same probability. These two effects can be seen in Figure 3.22, which shows the number of connections or *projections* between patterns. One consequence of the fact that the number of connections between patterns converges toward a common value is that the transition probabilities of the network will be less and less dependent on the Markov transition probabilities used for generating the training pattern sequence. This decreases the performance of the network, as it is shown in Figure 3.20 and 3.21.

Given a fixed noise level, the evolution of the number of connections between patterns as a function of the number of presentations of single transitions is showed in Figure 3.23. After about 20000 presentations, the number of connections relaxes to an asymptotic value. This indicates that the learn-

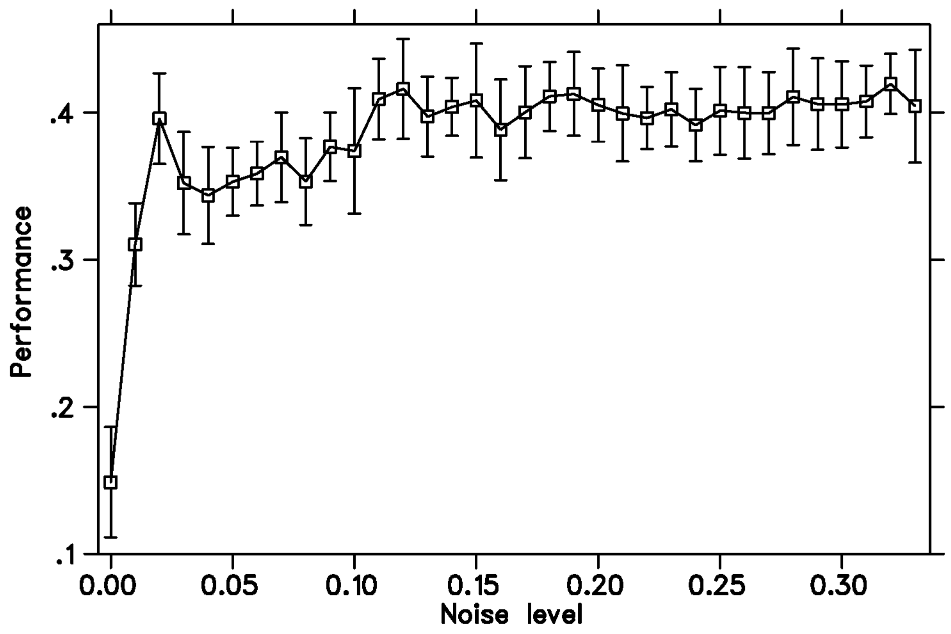


Figure 3.21: Same as Figure 3.20, but for a wider range of noise levels.

ing process can be terminated. The asymptotic values for the number of connections depend on the level of noise, as it can be seen in Figure 3.22.

3.1.6 Triggering recall with noisy patterns

In Section 3.1.5 we argued that natural systems, in particular brains, show a high degree of robustness with respect to noise. We showed that the artificial network we designed can be used with noisy patterns. Noisy input patterns can be used during the learning phase. The network's robustness consists in a 'graceful decay' of its performance with increasing noise. For all tests, the recall of the learned sequences was triggered using noiseless patterns, in order to isolate the effects of noise during the learning phase and during the recall phase. In this section we show how the performance of the network degrades when the recall is triggered by a noisy input. Again, to avoid mixing the effects of noise on the learning phase and on the recall phase, noiseless patterns have been used for training the network.

The level of noise is defined, as in Section 3.1.5 as the probability for each neuron of a given pattern to have its state flipped from 1 to 0 or vice versa. The network is trained using Equation 3.1, as described on page 37. Noiseless patterns were used in the learning phase. We tested the ability of the network in reproducing the transition statistics of the pattern sequence

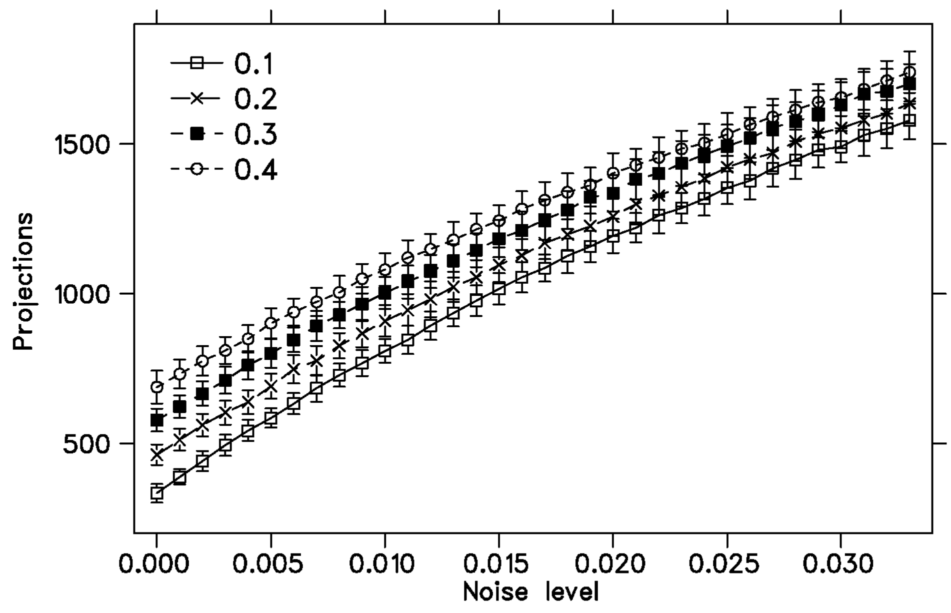


Figure 3.22: Number of projections (connections) between neurons active in (noiseless) patterns as a function of the noise level applied to the patterns used in the training. The number of projections are averaged over all pairs of patterns whose Markov transition probability is, respectively, 0.1, 0.2, 0.3, and 0.4. The number of projections increases with noise and, for all 4 values of the Markov transition probabilities, it converges toward a common value (490 neurons, 7 patterns, $f = 1/7$, $\lambda_f = 0.1$, $I_0 = 0.01$, Markov matrix showed in Table 3.1, $\beta = 14$, 20000 transition presented for each noise level during the learning phase).

used in the learning phase for two different cases. In the first case, the Markov matrix in Table 3.1 is used in generating the training pattern sequence. The performance Π of the network is evaluated by triggering the recall process with noisy patterns. Each transition probability is measured by setting the network state to a starting pattern and letting the network evolve until it makes a transition. The starting patterns used are noisy versions of the patterns used in the learning phase. The results are showed, for two different network pseudo-temperatures, in Figure 3.24 and 3.25. A second test was performed, by using a different Markov matrix (showed in Table 3.3) for generating the pattern sequence used in the learning phase. The generated sequence is a simple looping sequence going through seven different patterns. The network was trained with noiseless patterns. After the end of the learning phase, we tested the ability of the network in reproducing the whole sequence, from pattern 0 to pattern 6 and back to 0. The network state was initially set to correspond to an activity pattern that was obtained by corrupting

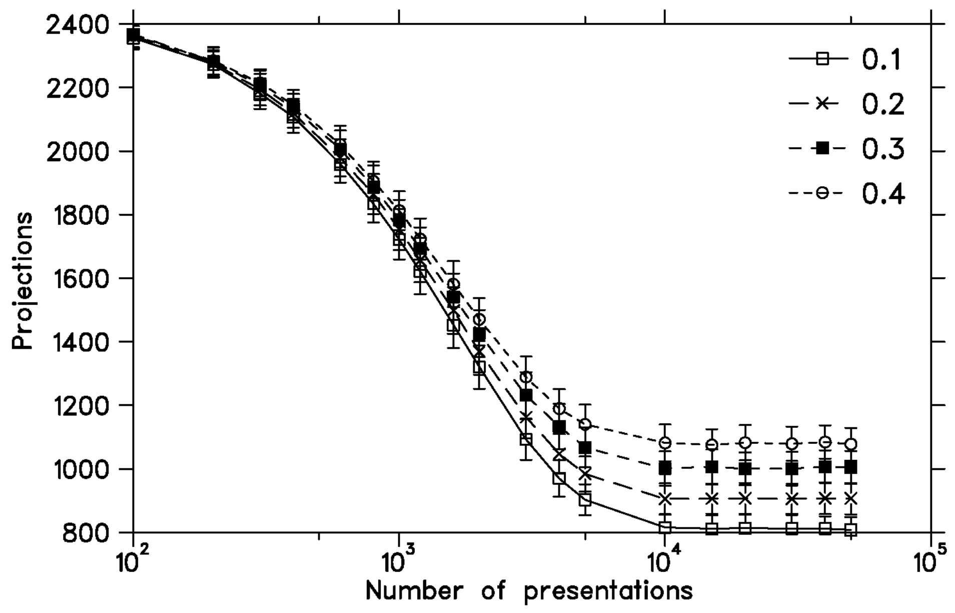


Figure 3.23: Number of projections (connections) between neurons active in (noiseless) patterns as a function of the number of presentations of single transitions during the learning phase. The starting value of the projections is obtained from a synaptic matrix in which synapses have the same probability of being potentiated or depressed. The number of projections are averaged over all pairs of patterns whose Markov transition probability is, respectively, 0.1, 0.2, 0.3, and 0.4. The number of projections decreases with the number of presentations and reaches an asymptotic value that is roughly proportional to the transition probability between patterns (490 neurons, 7 patterns, $f = 1/7$, $\lambda_f = 0.1$, $I_0 = 0.01$, Markov matrix showed in Table 3.1, $\beta = 14$, noise level=0.01, averages over 40 trials).

pattern 0 with increasing levels of noise (because of the symmetry between all patterns, pattern 0 can be chosen without losing generality). The result is showed in Figure 3.26.

3.1.7 Limits of the system

Up to now, the network was studied in a regime supposedly distant from its capacity limits. This means that the number of patterns used, as single patterns or in a sequence, was always much smaller than the number of patterns that could be stored by network. The theoretical maximum number of patterns that can be stored in a Hopfield network, called *capacity*, can be analytically calculated to be about $0.14N$ [47], where N is the number of neurons. We are not aware of any theoretical derivation of the network

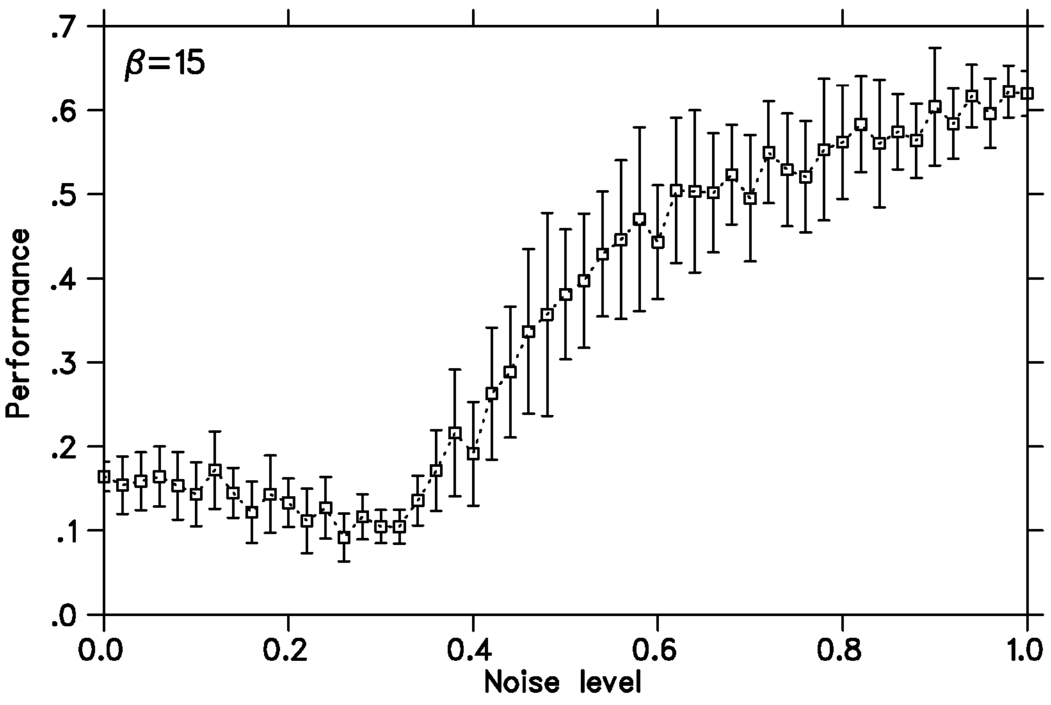


Figure 3.24: Performance of the network in reproducing the transition statistics of the training sequence generated by the Markov matrix showed in Table 3.1 as a function of the noise level. The performance improves (the performance index decreases) with the increasing noise up to a level of about 0.3. If the noise level is higher, the network is less able to generate a transition to one of the next patterns with the right transition probability. The minimum of the performance index around for a noise level close to 0.3 indicates that the pseudo- temperature of the network ($\beta = 15$) does not maximize the performance. Figure 3.25 shows the same plot for a higher pseudo-temperature ($\beta = 5$). In this case the performance index monotonically increases with the increasing noise level ($\beta = 15$, 7 orthogonal patterns, $\lambda_f = 0.1$, $I_0 = 0.015$, $f = 1/7$, 490 neurons).

capacity for simple pattern sequences nor for the more general case of several patterns linked by different transition probabilities. We show instead how the performance of the network in recalling a pattern sequence decreases with an increasing number of patterns. A sequence of patterns was generated using the Markov matrix showed in Table 3.3. The network was trained using Equation 3.1, using sequences of different lengths. For each step, the network was trained and its performance in recalling the sequence was tested. Partially recalled sequences were considered as failed recalls. Figure 3.27 shows the result of the test as the fraction of successful recalls as a function

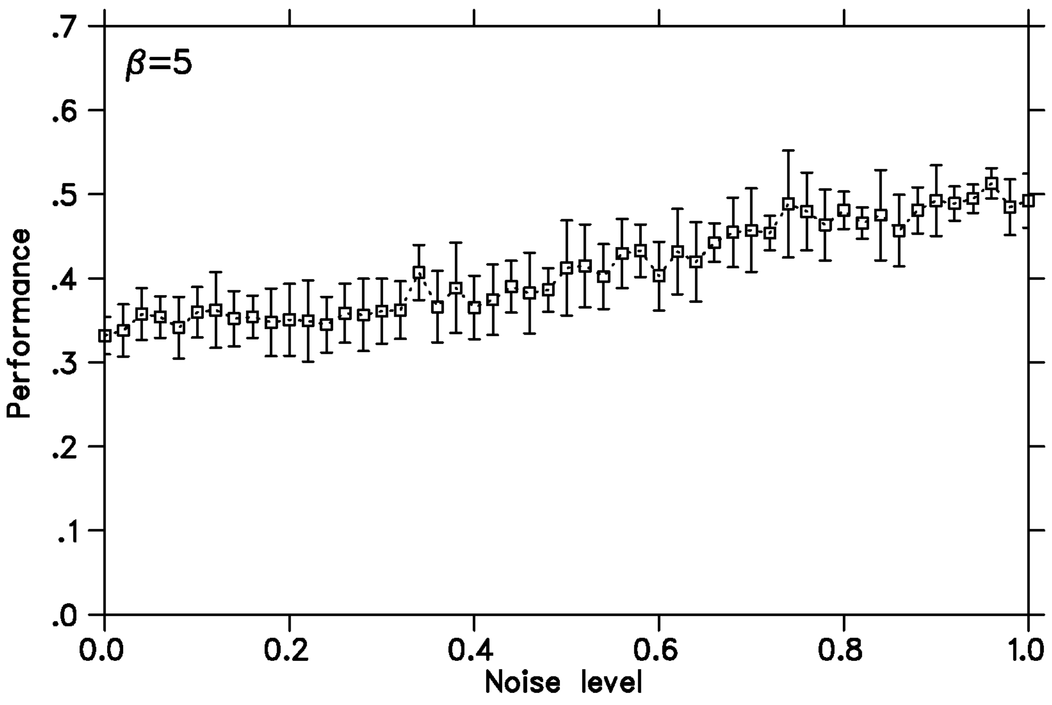


Figure 3.25: Performance of the network in reproducing the transition statistics of the training sequence generated by the Markov matrix showed in Table 3.1 as a function of the noise level. The performance index monotonically increases with the increasing noise level. To be compared with Figure 3.24 ($\beta = 5$, 7 orthogonal patterns, $\lambda_f = 0.1$, $I_0 = 0.015$, $f = 1/7$, 490 neurons).

of the number of patterns in the sequence. The fraction of correctly recalled sequences decreases with the number of patterns. The network is unable to correctly recall a sequence of 16 patterns, or a longer one. In general, the number of patterns and the Markov matrix used in generating the training sequence will affect the capacity of the network. The results showed in Figure 3.27 are valid only for the Markov matrix used. While it is clear that the performance of the network, for a fixed number of neurons, will certainly decay with the number of patterns, it is not clear how the Markov matrix influences the decay, and in which amount.

3.2 The synaptic matrix

For a given network temperature $1/\beta$ and inhibition I_0, the network's dynamics is solely determined by the synaptic matrix J. This matrix describes

0.0	0.0	0.0	0.0	0.0	0.0	1.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0

Table 3.3: Markov matrix used to study the performance of the network.

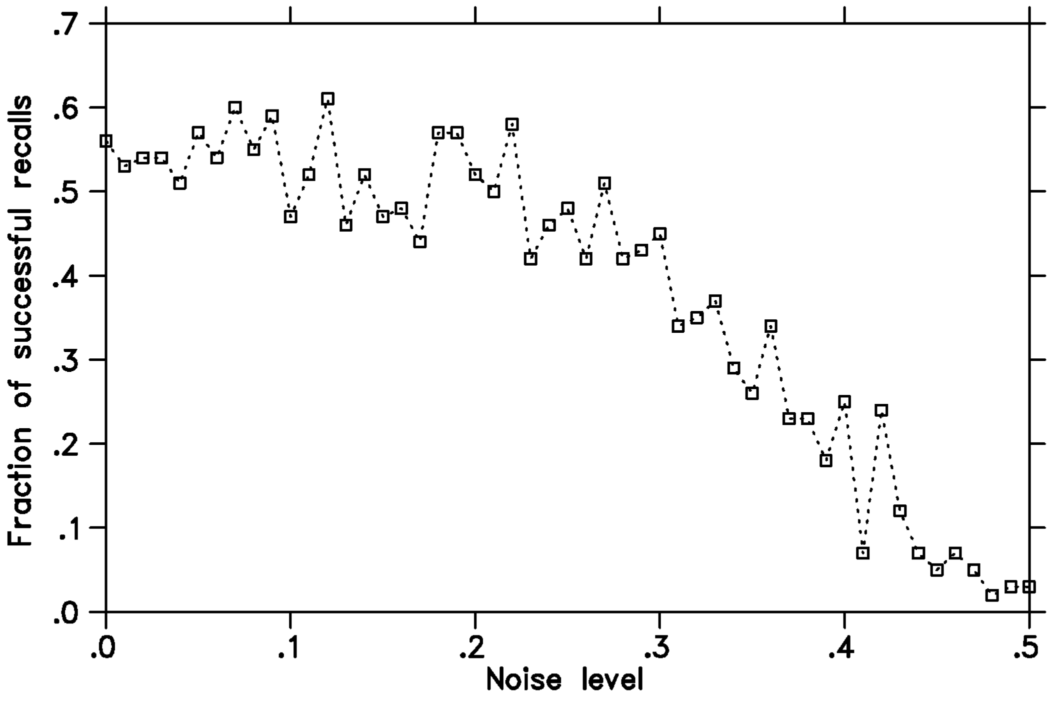


Figure 3.26: Fraction of successfully recalled sequences as a function of the noise level of the start pattern. The network was started from a pattern of activity obtained from pattern 0 by flipping the state of the neurons with a probability corresponding to the noise level. The sequence was considered to have been successfully recalled if the network made 6 transitions to patterns 1...6. Partially recalled sequences are not counted as successfully recalled ($\beta = 15$, 7 orthogonal patterns, $\lambda_f = 0.1$, $I_0 = 0.015$, $f = 1/7$, 490 neurons, data points obtained from 100 trials for each noise level).

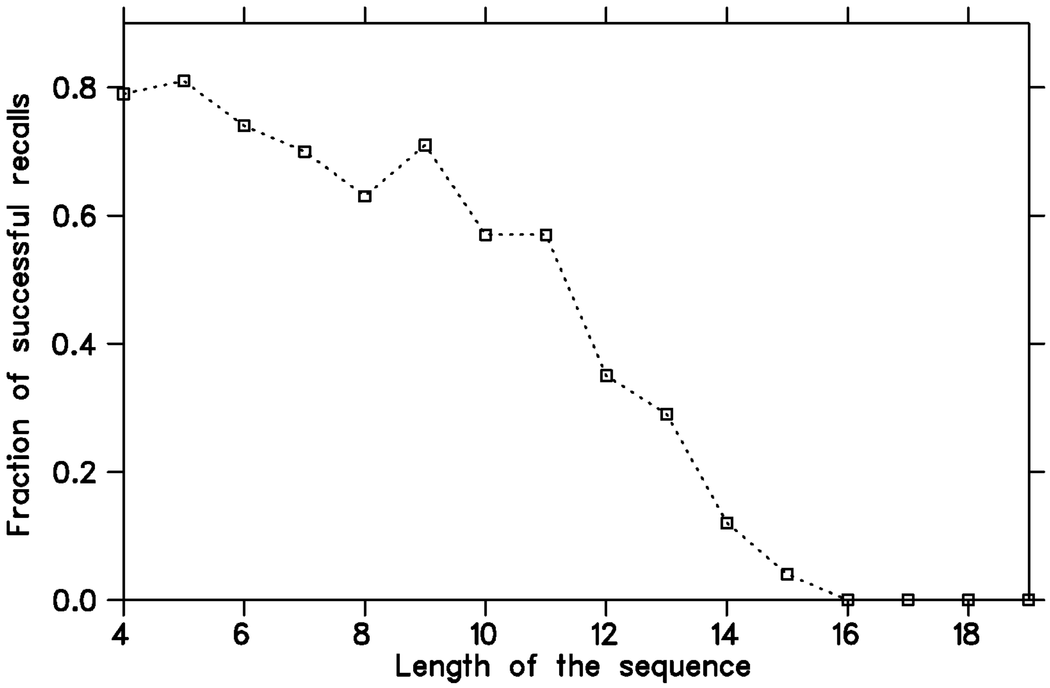


Figure 3.27: Fraction of successfully recalled sequences as a function of the length of the sequence, expressed in number of patterns. Data point are fractions measured over 100 trials. During each trial the network was trained and then its ability to correctly recall the sequence was tested. Partially recalled sequences were counted as failures. ($\beta = 15$, random patterns, $\lambda_f = 0.1$, $I_0 = 0.015$, 490 neurons, activity $f = 0.1$).

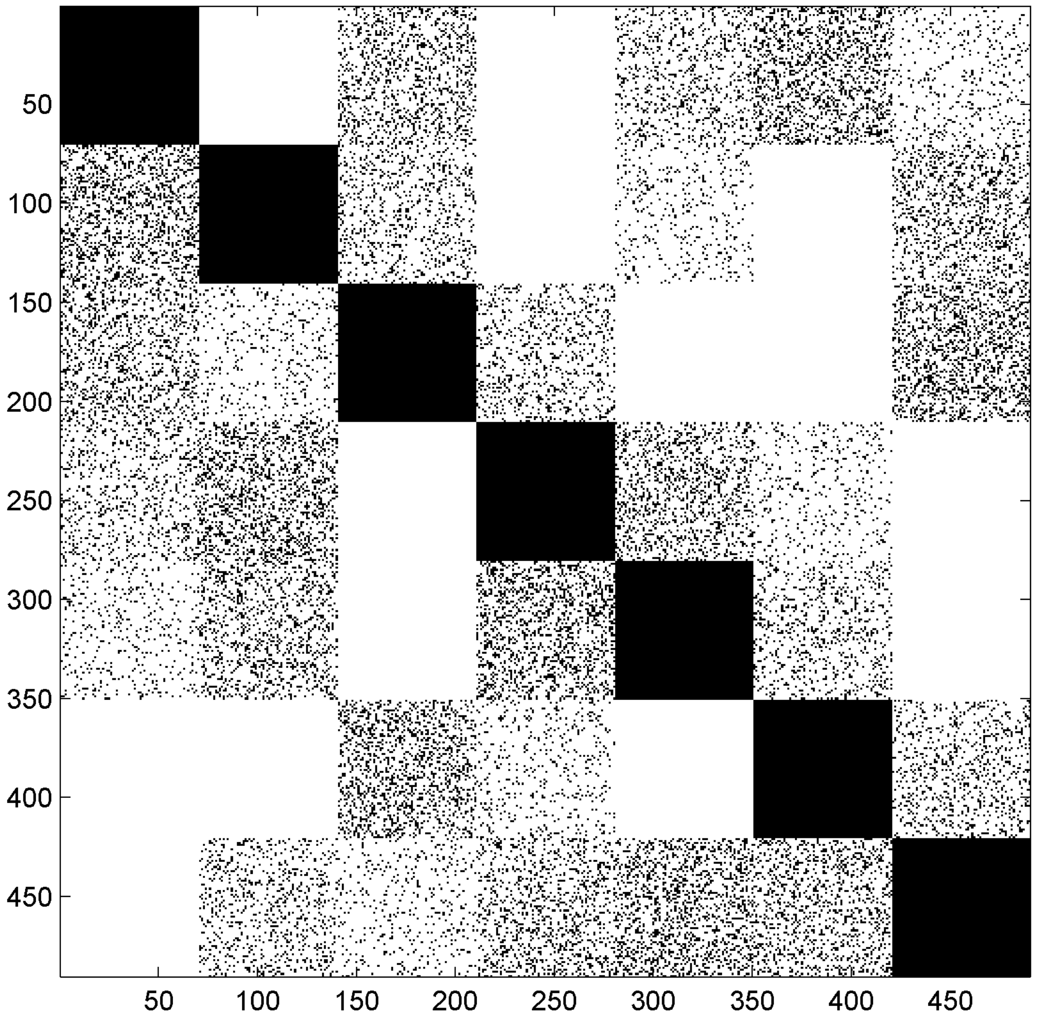

50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350
400
450

Figure 3.28: Synaptic matrix obtained applying the learning rule described in Section 2.2.2 using the Markov matrix showed in Table 3.1, as described on page 37. Potentiated synapses appear as black squares. This matrix describes the synaptic links between the 490 neurons of a network. The neurons are logically divided into 7 patterns, and each pattern corresponds to a group of neurons. Neuron numbers from 1 to 70 belong to pattern number 1, neuron numbers from 71 to 140 to pattern 2, and so on. The synaptic links appearing as black squares along the main diagonal correspond to the connections of each group of neurons with itself, as described on page 72. Column i corresponds to the axon of neuron i. The first 70 columns correspond to the axons of the group neurons belonging to pattern 1. By inspection of the first column of the Markov matrix used to generate the input sequence, it can be inferred that pattern 1 has been followed by pattern 2 with probability 0.4, by pattern 3 with probability 0.3, by pattern 4 with probability 0.2 and by pattern 5 with probability 0.1. These transition probabilities affect the ratio of potentiated over depressed synaptic links, which is mirrored in the figure by a correspondingly decreasing density of black spots. The same applies to the other 6 patterns.

how the neurons of the network are connected to each other. The element of the synaptic matrix J_{ij} corresponds to the strength of the connection from neuron j to neuron i and can have only the two values $+1$ and 0. As described in Section 2.2.1, we have used a network with N neurons. Each neuron can be either active or quiescent, and its state variable can be assigned the values $S_i = 1$ and $S_i = 0$ respectively. If the j-th neuron is active, the i-th neuron will receive a synaptic input h_{ij} proportional to J_{ij}, where $h_{ij} = \frac{1}{N} J_{ij} S_j$. The probability for a neuron i to be active at a time t depends, for a given β and I_0, from the synaptic input $h_i = \sum_j h_{ij} = \frac{1}{N} \sum_{j=1}^{N} J_{ij} S_j$ from all other neurons, and is calculated according to Equation 2.1. The synaptic matrix is built using the learning rule described in Section 2.2.2 applied to a sequence of external input patterns. After the synaptic matrix has reached its asymptotic configuration, by inspection of the synaptic matrix (see Figure 3.28) we notice that:

1. Neurons belonging to a pattern project onto all neurons belonging to the same pattern. In absence of other synaptic links, this would guarantee that the pattern is a stable attractor of the dynamics of the network.

2. Neurons belonging to each pattern project to neurons belonging to other patterns too.

3. The number of connections from the neurons associated to pattern i to the neurons associated to pattern j increases with the corresponding element of the Markov probability matrix M. The average field $\bar{h}_{i \to j}$ from the neurons of pattern i to the neurons of pattern j is defined as the average field on the neurons belonging to pattern j, when the network state has overlap 1 with pattern i. $\bar{h}_{i \to j}$ is roughly proportional to the corresponding element M_{ji} of the Markov probability matrix.

3.2.1 Variability of the synaptic projections

The learning rule described in Section 2.2.2 is stochastic. This choice makes the network learning rule more realistic, as the events that bring to synaptic potentiation and depression—like anything else happening in living cells—are stochastic in nature. One consequence of having implemented the learning rule as a stochastic process is that the average fields from pattern i to pattern j $\bar{h}_{i \to j}$ are not constant within a sample of synaptic matrices learned for the same network and the Markov matrix. Figures 3.29, 3.30, 3.31, 3.32, and 3.33, show the histograms of the mean synaptic input to all neurons of the network. The values used to fill the histogram have been generated by setting

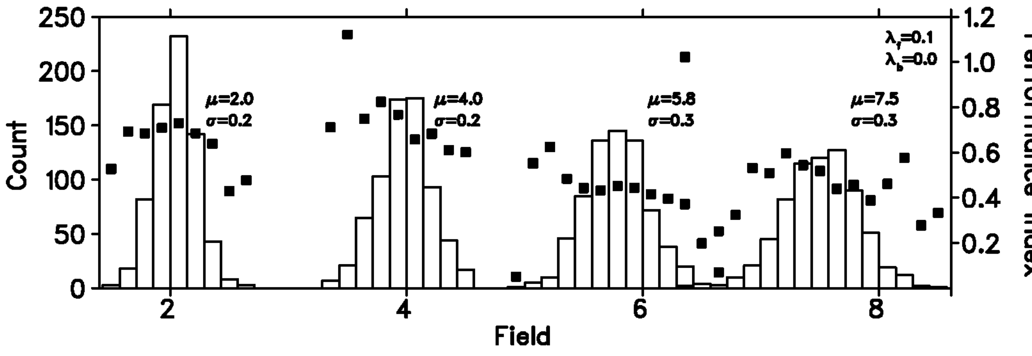


Figure 3.29: Histogram of the mean synaptic input $h_{\nu\to\mu}$ from the group of neurons corresponding to pattern ν to the group of neurons corresponding to pattern μ (Left-hand scale). The four peaks correspond to the four possible nonzero values of the corresponding element $M_{\nu\mu}$ of the Markov transition probability, 0.1, 0.2, 0.3 and 0.4. The mean value μ and the standard deviation σ of the distributions corresponding to each of the four values of $M_{\nu\mu}$ are printed close to the peaks of the distributions. The performance (right-hand scale) of the network in reproducing the transition probabilities in the Markov matrix (see Section 3.1.2) is plotted with filled squares ■. The performance was evaluated for each single element of the Markov matrix $M_{\mu\nu}$ and transition probability matrix $T_{\mu\nu}$. The mean value plotted above each bin of the histogram was computed on those elements for which $h_{\nu\to\mu}$ belongs to that bin ($\lambda_f = 0.1$ and $\lambda_b = 0.0$).

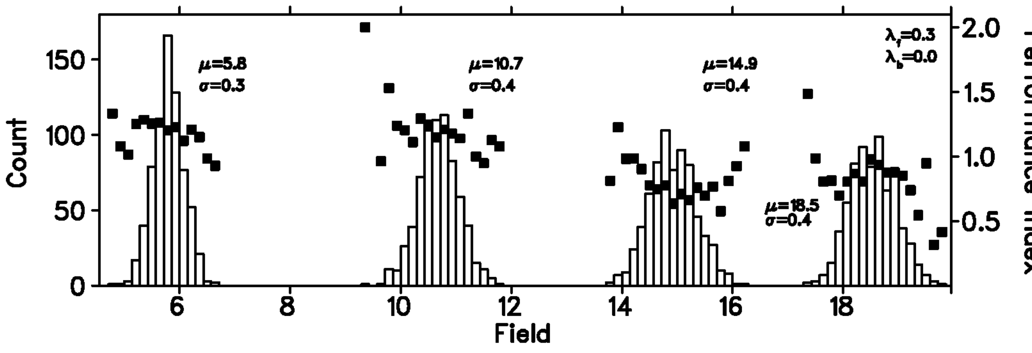


Figure 3.30: Same as Figure 3.29, but with $\lambda_f = 0.3$ and $\lambda_b = 0.0$.

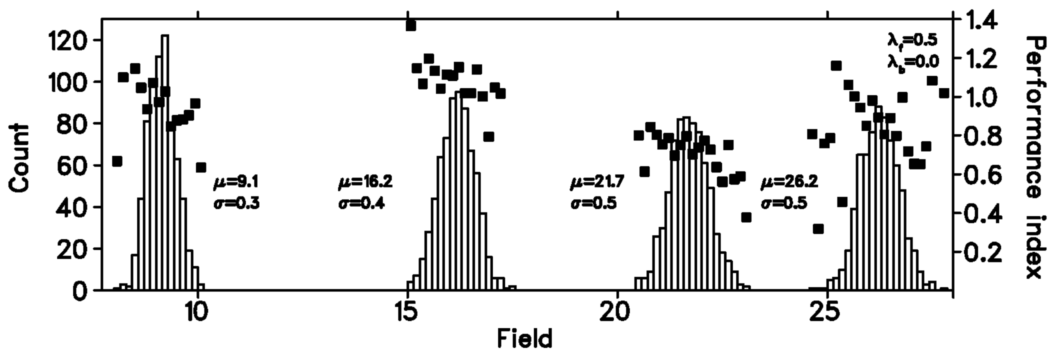
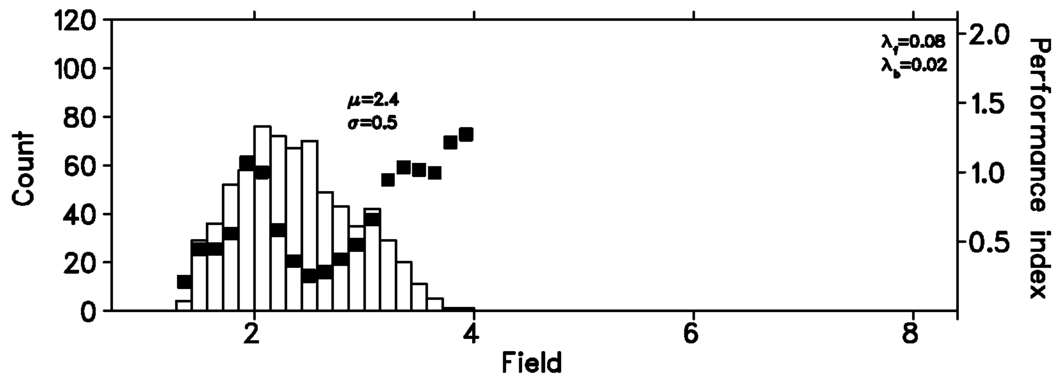
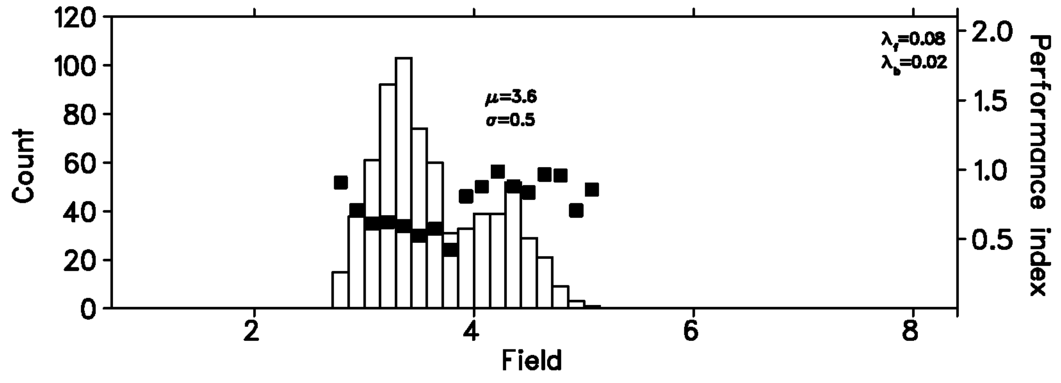


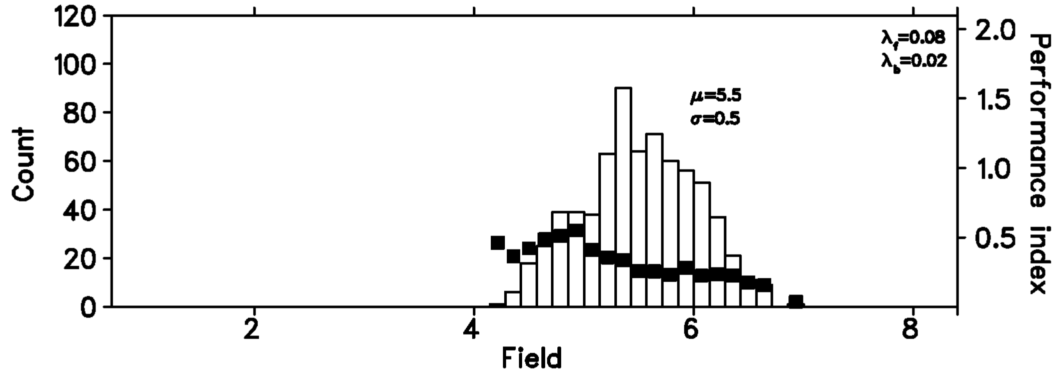
Figure 3.31: Same as Figure 3.29, but with $\lambda_f = 0.5$ and $\lambda_b = 0.0$.



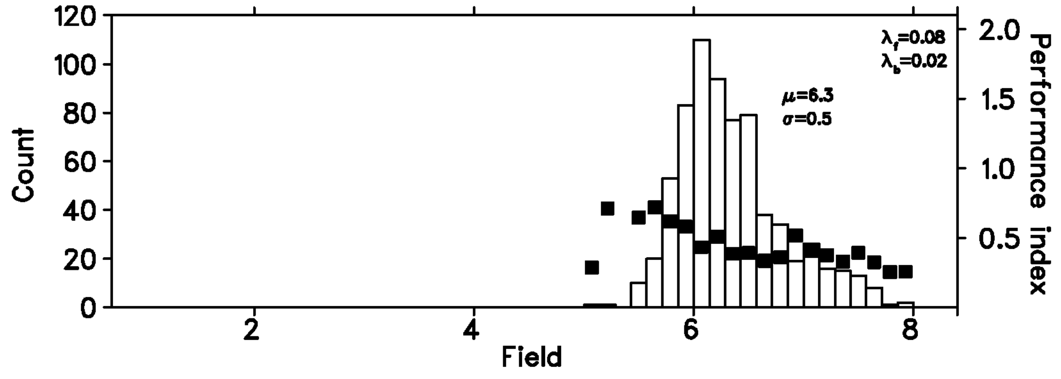
Count
Performance index
Field
λ_f=0.08
λ_b=0.02
μ=2.4
σ=0.5



Count
Performance index
Field
λ_f=0.08
λ_b=0.02
μ=3.6
σ=0.5



Count
Performance index
Field
λ_f=0.08
λ_b=0.02
μ=5.5
σ=0.5



Count
Performance index
Field
λ_f=0.08
λ_b=0.02
μ=6.3
σ=0.5

Figure 3.32: Same as Figure 3.29, but with $\lambda_f = 0.08$ and $\lambda_b = 0.02$.

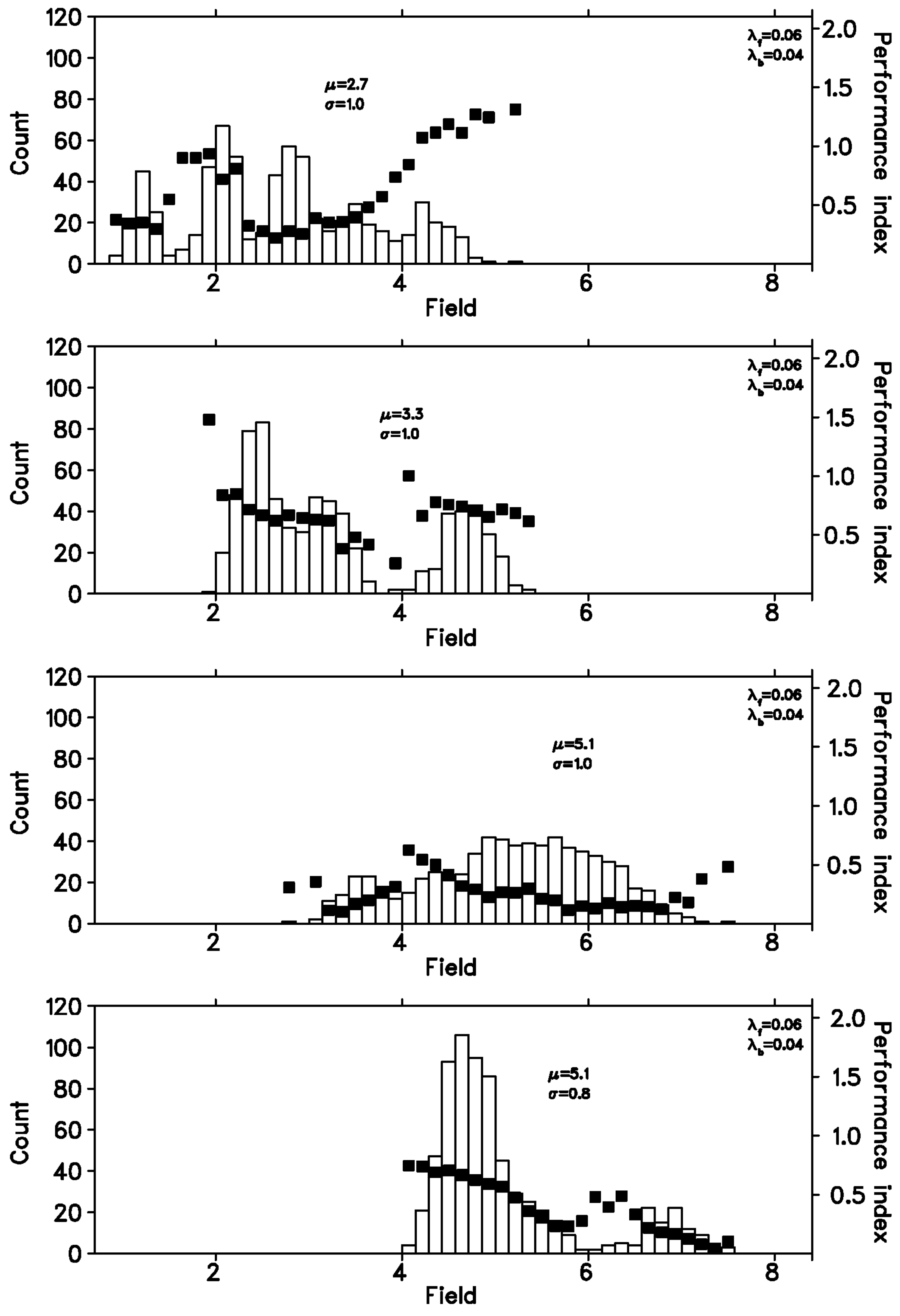

Count
Field
Performance index
λ_f=0.06
λ_b=0.04
μ=2.7
σ=1.0
μ=3.3
σ=1.0
μ=5.1
σ=1.0
μ=5.1
σ=0.8

Figure 3.33: Same as Figure 3.29, but with $\lambda_f = 0.06$ and $\lambda_b = 0.04$.

the network state S to pattern μ, that is $S_i = \eta_i^\mu$ with $\mu = 1, \ldots, p$, where p is the number of patterns used, and then calculating the average synaptic input to pattern μ from pattern ν expressed as the number of active neurons belonging to pattern ν providing an input to neurons belonging to pattern μ: $h_{\nu \to \mu} = \sum_{i=1}^{N} \eta_i^\mu \sum_{j=1}^{N} J_{ij} \eta_j^\nu$, where N is the number of neurons, and μ and ν have been chosen so that the element of the Markov matrix $M_{\nu\mu} \neq 0$. The data are obtained from 100 synaptic matrices trained with the same set of orthogonal patterns and the same Markov matrix. The number of potentiated synapses connecting a set of neurons corresponding to patterns i to the set of neurons corresponding to pattern j depends on the corresponding value of the element of the Markov matrix M_{ji}. The larger the value of the matrix element, the larger the number of potentiated synapses connecting the first group of neurons to the second.
The four peaks in Figures 3.29–3.31 correspond to the four possible non-zero transition probabilities (0.1, 0.2, 0.3 and 0.4) in the Markov matrix (Table 3.1) used to generate the training sequence. The histograms in Figures 3.32 and 3.33 have been separated in four parts for clarity. Each part shows the histogram of the average fields $h_{\nu \to \mu}$ for a different value of $M_{\mu\nu}$. The average value of the field for each possible value of the element of the Markov matrix is roughly proportional to it (Figure 3.34). As expected (it follows from basic statistics), the ratio of the standard deviation of the distributions over its mean value decreases with an increasing Markov transition probability and λ_f (Figure 3.35).

3.2.2 Transition probabilities as a function of the synaptic projections

In Section 3.1.2 we showed how the pseudo-temperature $1/\beta$ and the inhibition I_0 influence the performance of the network. The performance index Π, defined in Equation 3.2 is a measure of the network's ability in reproducing the transition probabilities of the patterns sequence used during the learning phase. To study how the variability in the number of potentiated synapses from the group of neurons corresponding to pattern μ to the group of neurons corresponding to pattern ν influences the performance of the network, we calculated the performance of the network in reproducing the transition probabilities in the Markov matrix for each single element of the Markov matrix M and transition probability matrix T. We have then grouped the data according to the average field of the synaptic input $h_{\mu \to \nu}$, using the same binning used in histogramming $h_{\mu \to \nu}$ in Figures 3.29–3.33. The average of the data points falling in the same bins are plotted as filled squares in Figure 3.29–3.33. The data does not show a strong dependence of the performance on the number of synaptic projections. This indicates a good amount of robustness of the network's dynamics with respect to fluctuations in the number of potentiated synapses, which are unavoidable, due to the stochastic nature of the learning rule.

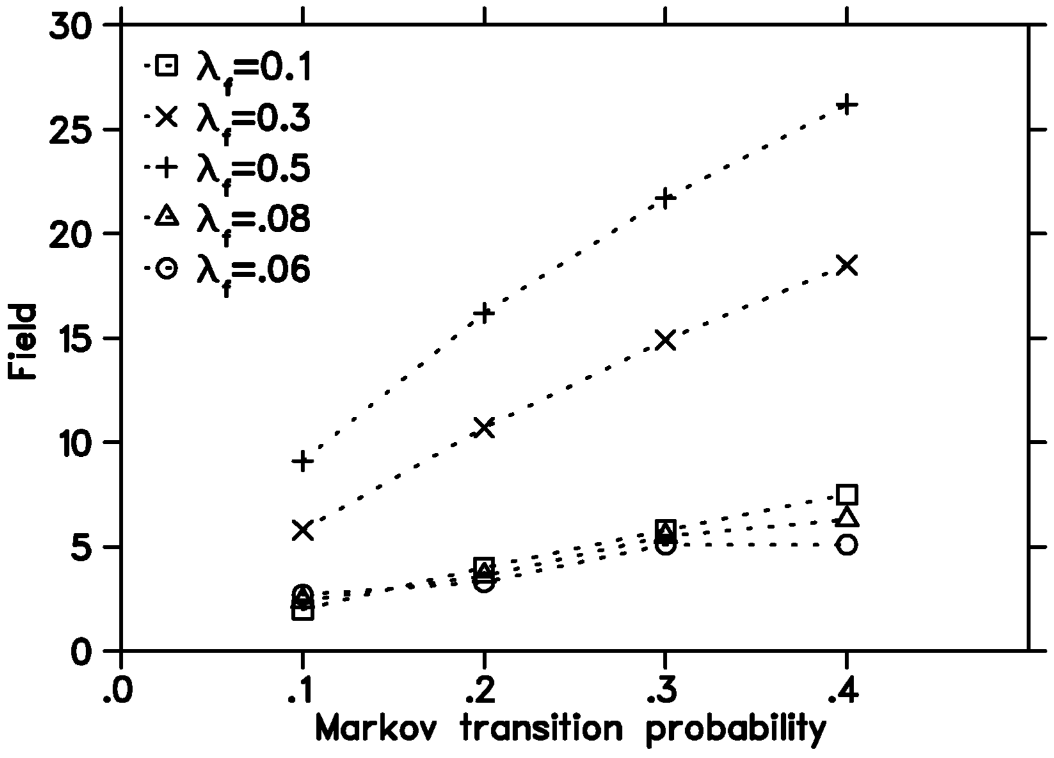


Figure 3.34: Mean synaptic input μ, as printed in figures 3.29–3.33, as a function of the Markov transition probability, for all four choices of λ_f. The mean field is roughly proportional to the Markov transition probability, and the constant of proportionality decreases with λ_f.

3.3 Summary

The network introduced in Chapter 2 can learn in an unsupervised way the transition probabilities of a pattern sequence generated by a Markov process. The performance of the network is evaluated by comparing the Markov matrix used in generating the training sequence with the transition probability matrix measured from the pattern sequence generated by the network after the learning phase. The network can learn also the transition probabilities of patterns *partially corrupted by noise. The performance of the network depends on the pseudo-temperature and the function defining the dynamic inhibition.*

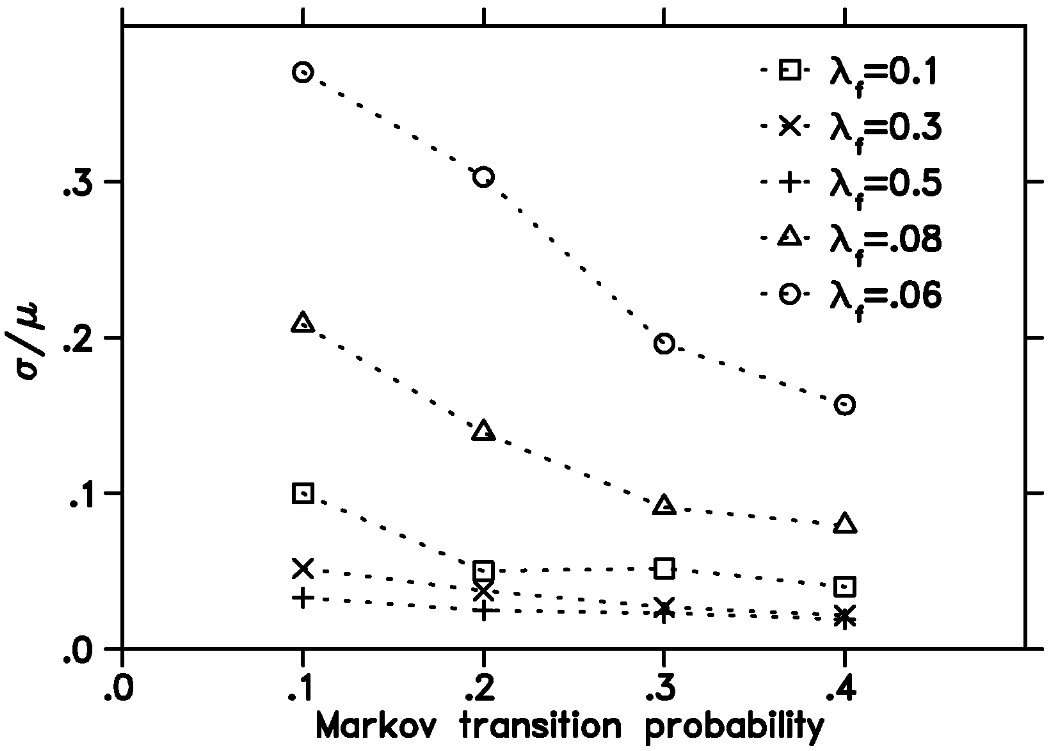


Figure 3.35: Ratio of the mean synaptic input μ, and its standard deviation σ, as printed in figures 3.29–3.33. For a fixed λ_f, the ratio decreases with the increasing Markov transition probability, and, for a given Markov transition probability, it decreases with the increasing λ_f. This behavior is readily explained by the stochastic nature of the learning rule, for which the number of potentiated synapses between two groups of neurons corresponding to two patterns is proportional to the product of the corresponding Markov transition probability and λ_f.

Chapter 4

Learning non-Markov processes

Several phenomena can be modeled with Markov models. In general, the state of a system can be modeled using a Markov model if its states at any arbitrary time in the future only depend on the present state. The system state is said to be a *Markov process.* Some phenomena cannot be properly modeled by Markov models. For example, if the future state of a system cannot be predicted by knowing its state at the present time, but information about its states at past times are needed. In this case, the state taken by the system can be modeled by a *non*-Markov process, or, equivalently, by a Markov process of order higher than 1 (see page 1).
A way to cope with non-Markov processes is to transform them in Markov processes by redefining the state space [15]. For example, let us suppose that a given system can be in either of two states 1 and 0, and let X_n be the state of the system at time n. The process described by the pair of equations,

$$\begin{aligned} \text{prob}(X_{n+1} = 1 | X_n = 1,\ X_{n-1} = 1) &= 0 \\ \text{prob}(X_{n+1} = 1 | X_n = 1,\ X_{n-1} = 0) &= 1 \end{aligned}$$

with all other possible transition probabilities at zero, is clearly non-Markov. The probability for the system to remain in state $X = 1$ does not only depend from the present state X_n, but also on the state one timestep earlier (X_{n-1}). This non-Markov process can be converted into a Markov process by extending the state space. For example, the original state 1 can be divided into two states $\{1,1\}$ and $\{1,0\}$, where $\{1,1\}$ is the state corresponding to $X_n = 1,\ X_{n-1} = 1$ and $\{1,0\}$ the state corresponding to $X_n = 1,\ X_{n-1} = 0$. By using these newly defined states, the process is Markov and it is possible to describe it by using a Markov matrix, as it is showed in Table 4.1.

A Markov process can be described by a Markov matrix whose columns correspond to the transition probabilities from any state to any other state. It is possible to define a non-Markov process by using a Markov matrix whose

	(1,1)	(1,0)	(0,1)	(0,0)
(1,1)	0	1	0	0
(1,0)	0	0	0	0
(0,1)	0	0	0	0
(0,0)	0	0	0	0

Table 4.1: Markov matrix: The columns describe the transition probabilities between all combinations of states. This matrix has been obtained by augmenting the state space of a non-Markov process to convert it into a Markov process. Conversely, such a matrix can be used to define a non-Markov process by using a Markov matrix.

elements are the transition probabilities between composed states, following the opposite reasoning used above. An example of how this can be done is given by the same Markov matrix showed in Table 4.1. The process it defines is non-Markov and the sequence of states generated by such a process cannot be modeled by a Markov model.

4.1 Learning pattern sequences

The synapses of the network we have described in Section 2 and used in Section 3 are instantaneous. One consequence is that the state of the network after each single neuron update just depends on its state before the update is done. It also follows that the transition probabilities only depend on the starting state of the network. In particular, the transition probabilities cannot depend on the previous history of the network. For this reason, the sequence of states taken by the network can be thought to be generated by a Markov process. We have used several times this assumption in Section 3 while computing the transition probabilities matrix of the network, for evaluating its performance in generating pattern sequences with a given transition statistics.

The fact that the transition probabilities for the network do not depend on the history of the network state, but only on its present state, makes it impossible for the network to deal with pattern sequences generated by non-Markov processes. Information about the past states must be explicitly encoded in the network state to make the transition probabilities depend not only on the present state but also on earlier states. The performance of the network in learning and generating pattern sequences produced by non-Markov processes depends on the width of the 'window in time' provided by the encoding mechanism, and by the weights assigned to the patterns presented at different times. For example, a shift register has a width equal to the number of memory slots, where all patterns in memory are given the same weight. Another, more natural way to encode the state history in a single pattern of activity, might be to assign more weight to recent patterns.

4.1.1 The encoding network

The problem of encoding the temporal history of the input sequence in a pattern of activity has more than one solution. A suitable solution will normally have to satisfy a set of requirements. Such requirements can be arbitrarily chosen by the experimenter or satisfy experimental constraints. We examined several approaches: All tests were aimed to combine the present state and the external input in some way to produce a pattern of activity to be used as the input to the learning network. Our first attempts focused on sequences with one single pattern in common. For example, let us consider the pattern sequences $\eta_i^1 = A, \eta_i^2 = B, \eta_i^3 = C$ and $\eta_i^4 = D, \eta_i^2 = B, \eta_i^5 = E$, $\eta_i = 0, +1$, leading to the formation of the attractors A', B', C' and D', B'', E' respectively. After having presented both sequences a sufficient number of times, the network synaptic matrix encodes the transition probabilities between the presented patterns. If B' and B'' are sufficiently uncorrelated, when the network is triggered by the external input patterns A or D, it should preferentially make a transition to, respectively, B' followed by C' or B'' followed by E'. This would demonstrate the ability of the network in learning and generating history-dependent pattern sequences. One of our very first approaches combined the network state at time t, $S_i(t)$, and the external input e_i using two matrices M and N. The two matrices had elements randomly chosen to be either $+1$ or -1 with the same probability. The resulting input vector $r_i = (M \cdot S_i + N \cdot e_i)$ is a combination of S_i and e_i. The state at time $t+1$, $S_i(t+1)$ is obtained by thresholding r_i. A fixed threshold was heuristically chosen to keep the average activity of the network constant. The same result might have been reached using a dynamic threshold, with a mechanism similar to the dynamic inhibition described in Section 2.2.1. For the two sequences described above, the patterns B' and B'' have an average overlap of about 0.7. This means that information about past stimuli is quickly forgotten, and the patterns depends much on the lastest pattern showed. The variability in the activity of the produced patterns is quite high, and, more importantly, it is difficult to set the system to forget more slowly. For these reasons, this solution was considered impractical and abandoned.

In the search for a method to produce patterns B' and B'' with the least overlap possible, we had to abandon physiological realism. The following method is illustrated only to explain its pathological behavior. A matrix $M(t=0)$ is used at time $t=0$, with elements +1 or -1, randomly chosen with the same probability. Given that the external input at time $t=0$, $e_i(t=0)$, the resulting input vector is $r_i(t=0) = M(t-0) \cdot e_i$. A new matrix $M(t=1)$ is then computed for the next external input $e_i(t=1)$. $M(t=1) = Me_i r_i^{\top}$, where all factors on the right-hand side are calculated at $t=0$. In this case, the patterns B' and B'' have a very small average overlap. The system produces patterns of activity r_i that are highly dependent on the stimuli history. Unfortunately, the dependence is so high that a very low level of noise can be catastrophic. At the limit, flipping the activity of one single neuron creates, for the same input stimuli sequence, states with small overlaps. Some robustness to noise is a requirement which cannot be left out, both in view of a possible implementation in hardware or use in the modeling of brain function. For this reason, also this method was abandoned.

A more physiologically realistic approach exploits the dynamics of the learning network itself. During the presentation of the stimuli, attractors of the network dynamics start to form. This corresponds to the 'first state' described on page 2. Once the attractors are formed, the network can sustain a pattern of activity without any external input. This corresponds to the delay activity described in Section 1. The delay activity represents a memory of the previous external input to the network. This memory can be combined with the external input pattern to create a pattern of activity coding for, at least, a few stimuli of the sequence being presented to the network. The induced, mixed, patterns of activity were studied for different external input pattern strengths. When setting the strength of the external input to be roughly equal to the *recurrent* synaptic input, the resulting mixed state was composed by about 50% of neurons already active (delay activity) and 50% of neurons selected among those receiving an external input. Due to the asynchronous, stochastic update of the neurons, the neurons selected to form the mixed state were picked in random order. Following different presentations, all neurons that are members of the active set during the delay activity and of the set receiving an external input participate in the mixed state. The resulting attractor involves twice as many neurons as it should be, that is all neurons active during the delay activity and those receiving an external input. When the network tries to relax to this attractor, the activity raises, and the dynamic inhibition randomly shuts down a number of neurons. Eventually, the patterns learned are just a function of two temporally adjacent stimuli. This does not fulfill the requirement of having patterns encoding the temporal history of the input sequence.

After several trials, on the base of the experience acquired, we decided to encode the state history using an implementation satisfying these two requirements:

- Information should be local in time. Past states of the network or past external inputs should not be explicitly stored. No buffer needs to keep track of past activity patterns, but the whole history needs to be coded in one single activity pattern. Only the present and the immediately preceding network state should be available to the synapses.

- The width of the 'window in time' should be adjustable. This means that it should be possible to adjust the forgetting rate and so the number of neurons coding for the present input pattern, the last one, the second last, and so on. Equivalently, it should be possible to adjust the weights assigned to the various patterns presented to the network in different points in time.

The encoding we decided to use is accomplished by combining the external input and the present activity pattern of the network. The encoding is such that the pattern of activity of the network carries information about the external input pattern at the present time and a temporally weighted influence of the network states from previous timesteps. Figure 4.1 shows the setup for the processing of pattern sequences generated by non-Markov processes. The input layer is connected to the *encoding layer* (compare with Figure 3.1), which provides synaptic input to the learning network. Alternatively, the learning network can receive input from the buffer network, the only difference being the time interval during which the activity pattern is preserved. The activity in the encoding layer is preserved only as long as the input layer provides an input. The activity in the buffer network sustains itself for the whole interval between the presentation of two different stimuli. The input layer serves no functional purpose and is only used for clarity. Each neuron in the input layer connects to only one other neuron in the encoding layer. The encoding layer is composed of a set of uncoupled binary neurons. The neurons receive input both from the input layer and the *buffer network*. The buffer network is a fully connected Hopfield network with fast learning non-stochastic synapses. It needs to be initialized with a random pattern of activity f before the first input pattern is presented. If both a pre-synaptic neuron j and a post-synaptic neuron i are active, then a transition to the potentiated state of the synaptic weight $J_{ij} : 0 \rightarrow 1$ occurs with a probability 1. If only one of the two neurons connected to the synapse is active, then a transition to the depressed state of the synaptic weight $J_{ij} : 1 \rightarrow 0$ occurs with a probability 1. If both neurons are inactive, the synapse is left unchanged. The buffer network receives input from the encoding layer. The synaptic weights between the encoding layer and the buffer network are

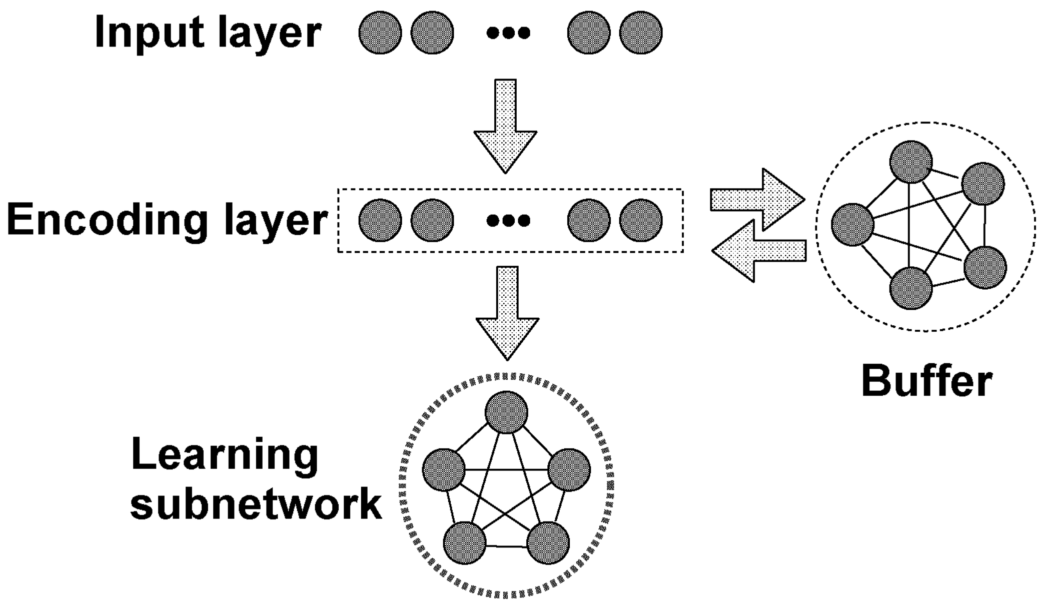


Figure 4.1: Schematic showing the connection of the input layer to the encoding layer, which is mutually connected to the buffer network. The encoding layer also drives the learning network.

all positive and equal and the weight value is set so that, when an input is present, the dynamics of the buffer network is completely dominated by the input from the encoding layer. The buffer network in return provides input to the encoding layer. The synaptic weights of the connections between the buffer network and the encoding layer are also all equal (see Box 4.1).

In our simulations, the connectivity was chosen so that, on average, 50% of the neurons code for the last pattern presented to the network, 25% for the second last, and so on. This choice is arbitrary and the temporal weights could have been chosen in a different way, thus making the window in time wider or narrower. Each input pattern sequence generates a unique pattern of activity in the encoding layer. By observing the state of the encoding layer it is possible, in principle, to associate the patterns of activity with the input pattern sequence presented to the network, as it is described in Section 4.1.2. On average, there will be at least one neuron encoding a pattern that occurred t timesteps earlier if $N/2^{t+1} \geq 1$. The number of neurons coding for a given pattern decreases as new patterns are presented to the network. Eventually, the number of neurons encoding a given pattern goes to zero and no traces of it are left in the network state. This considerations can be used to roughly evaluate the number of neurons needed to learn and generate successfully pattern sequences generated by non-Markov processes of any order.

Although it satisfies our requirements, the encoding mechanism is not optimal and should be seen as a placeholder for a functionally equivalent mechanism. A better mechanism should be more simple. For example, although the function of the buffer network can be justified by the presence in neural circuits of the delay activities described in Chapter 1, it would make sense to design a system in which this functionality emerges from the dynamics of the system, instead of having an ad-hoc external network implementing it. This lack indicates the way toward improvements that might be developed in the future. The implementation we used shows, at least, that the system is capable of dealing with non-Markov sequences. The implementation itself needs improvements as also discussed in Chapter 6.

The input layer provides the encoding layer with two kinds of input. A subset of $fN/2$ neurons receives a super-threshold synaptic input that always drives the neurons active. These subsets of neurons, uniquely assigned to each pattern, are normally chosen to be non-overlapping. In addition to the super-threshold input, the input layer also provides a sub-threshold to one half of the remaining neurons in the encoding layer. The neurons in the encoding layer can only become active if they receive either a super-threshold input or a sub-threshold input from both the buffer network and the input layer. In this way the number of active neurons of the encoding layer is, on average, fN.

The synaptic matrix from the encoding layer to the buffer network contains only one nonzero element on each column and row. The synaptic matrix from the buffer network to the encoding layer is generated by randomly permuting the columns of the first one. The neurons in the encoding layer and in the buffer network are updated synchronously, as well as the synapses in the buffer network. The only condition on the dynamics is that the duration of the input presented by the input layer should allow one only synchronous update of all neurons in the encoding layer and the buffer network.

The workings of the encoding networks are exemplified in Figure 4.2, which shows pictorially the external inputs, the inputs from the buffer network, and the states of the encoding network during the presentation of three different external input patterns.

4.1.2 The readout

On page 31 we have defined the overlap of the network state S ($S_i = 1,\ 0$) with a generic pattern μ ($\eta_i^\mu = 1,\ 0$). The overlaps were calculated using the same patterns used in the training phase. The knowledge of the patterns η_i^μ was taken for granted. In general, the number of possible external stimuli is unknown. In Section 1.4 we stated our intention to build a system that

Box 4.1 Dynamics of the encoding network

The encoding and buffer networks are composed of N neurons. The neurons are labeled by index i, $i = 1 \dots N$. The state of neuron i of the encoding (buffer) network is described by E_i (B_i). The buffer network is initialized to a random pattern of activity f (fN neurons are active). The inputs from the buffer to the encoding network are $e_i^{B\to E}$, the total synaptic inputs to the neurons of the encoding layer are h_i^E, the synaptic input to the neurons of the network buffer (only from the encoding network) are h_i^B, the inputs fro the input layer to the encoding layer are $e_i^{I\to E}$. The synapses connecting the buffer network to the encoding network are low-passed filtered with a time constant τ_b. The neurons of the encoding layer go active or inactive with a time constant τ_e. The neurons of the buffer network respond instantaneously to the external input from the encoding layer. This is justified by the fact that their output is low-pass filtered with a time constant 10 times larger than their response time constant (see code in Section A.12, time constants `ty` and `tz`). The set of difference equations implementing the encoding is:

$$e_i^{B\to E}(t+1) = e_i^{B\to E}(t) + \frac{1}{\tau_B}(B_i - e_i^{B\to E}(t))$$

This set of equations low-pass filters the input from the buffer layer to the encoding layer. B_i corresponds to the activity of the buffer during the presentation of the previous stimulus. The buffer works as a one-shot learning fully connected Hopfield network with binary synapses and binary neurons (see code in Section A.13)

$$h_i^E = e_i^{I\to E} + \frac{3}{2} e_{R_i}^{B\to E}$$

where R is a random permutation of the set of integers $1, \dots, N$. $e_i^{I\to E}$ can be either a (in the code a=3, over threshold) or b (in the code $b = \frac{3}{2}$, below threshold). Both a and b can be arbitrarily chosen. Finally, the state of the encoding buffer can be described by the difference equation

$$E_i(t+1) = E_i(t) + \frac{1}{\tau_E}(\Theta(h_i^E - \theta) - E_i(t))$$

where Θ is the Heaviside function, and θ can be arbitrarily chosen to discriminate neurons either getting super-threshold input from the input layer or a dual sub-threshold inputs from the input layer and the buffer network (in the code $\theta = 2$). The optimal dynamic inhibition parameter I_0 depends on the order of the Markov model the network should emulate. For example, let us consider a second order Markov model and the two sequences $\mathcal{S} = A, B, C$, and $\mathcal{U} = D, B, E$. Let $h_{x,y\to z}$ the mean synaptic input provided by the neurons active (as explained on Page 4.1.2) after the presentation of the sub-sequence $\mathcal{W} = x, y$ to the neurons corresponding to pattern z. The optimal I_0 is the average value of $h_{A,B\to C}$, $h_{A,B\to E}$, $h_{D,B\to C}$, and $h_{D,B\to E}$.

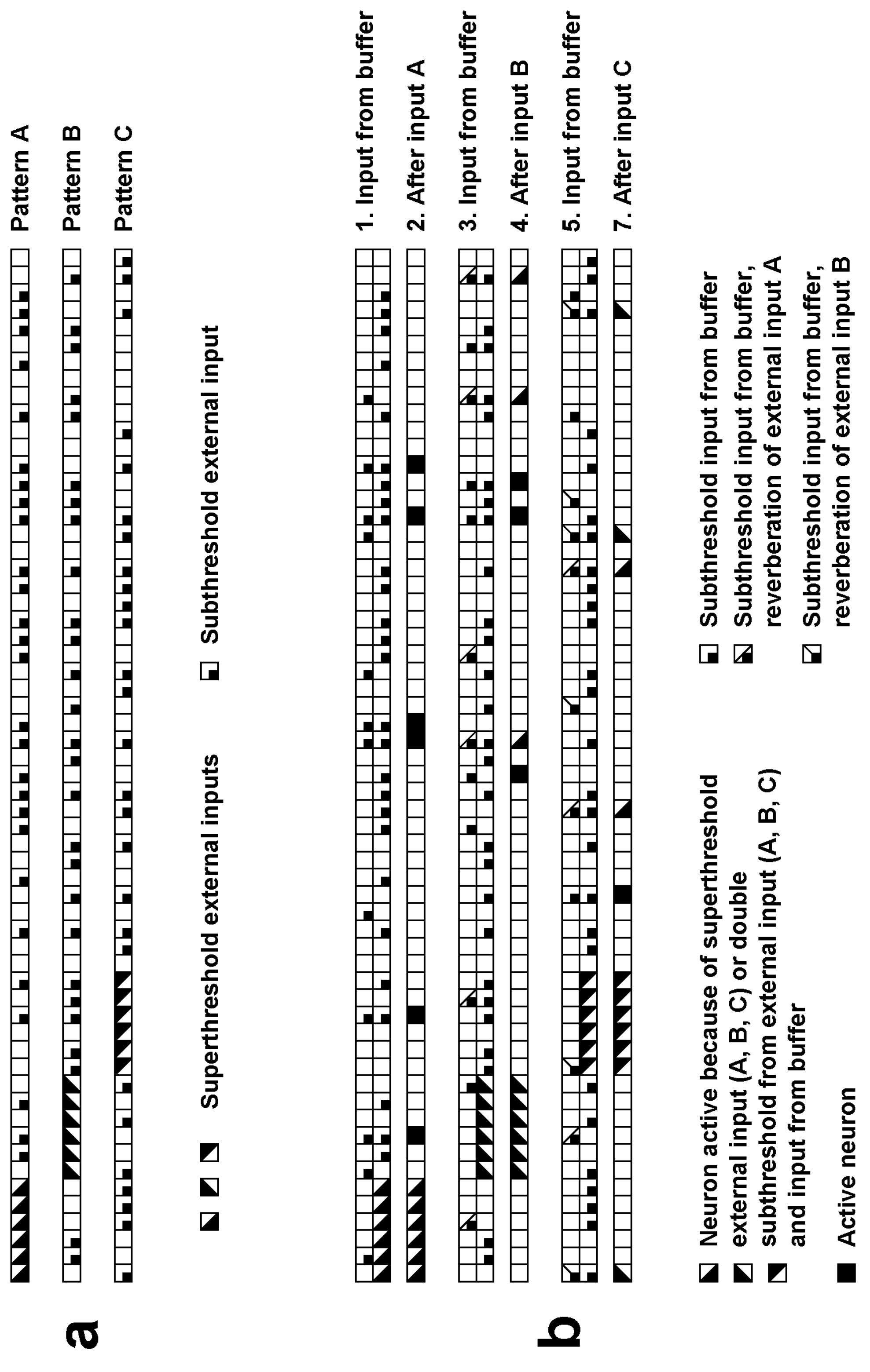

a
Pattern A
Pattern B
Pattern C
Superthreshold external inputs
Subthreshold external input
b
1. Input from buffer
2. After input A
3. Input from buffer
4. After input B
5. Input from buffer
7. After input C
Neuron active because of superthreshold external input (A, B, C) or double subthreshold from external input (A, B, C) and input from buffer
Active neuron
Subthreshold input from buffer
Subthreshold input from buffer, reverberation of external input A
Subthreshold input from buffer, reverberation of external input B

Figure 4.2: a: Schematic showing the structure of the external inputs corresponding to three different patterns (A, B, and C). Six neurons (see legend) receive a super-threshold input. On average, one half of the neurons also receive a sub-threshold input.
b: Schematic showing the input from the buffer, external input, and activity of the encoding layer during the presentation of three external input patterns. The first three rows are, from top to bottom, the input from the buffer, the external input pattern (same as in a), and the activity of the encoding layer. The buffer delivers only a sub-threshold input to the encoding layer. Neurons in the encoding layer can become active only if they either get a super-threshold external input or both a sub-threshold external input and a sub-threshold input from the buffer. In the absence of an input from the input layer, then no neurons of the encoding layer are active. The encoding layer provides an input to the learning network and to the buffer network. The strength of the inputs to these two networks is such that the external inputs dominate the network dynamics (in other words, the external input provided by the encoding network is higher than the recurrent input). The learning network learns the patterns and the transition probabilities between them, as previously described. The buffer network learns the pattern of activity provided as external input from the encoding layer. The input from the buffer network to the encoding layer is a shuffled version of the learned pattern of activity (see page 88). In the lines corresponding to the input from the buffer (first, fourth, and seventh rows), it is possible to distinguish the inputs from the buffer caused by the memory trace of the activity evoked by the initial random input from the buffer and by all external input patterns. The same structure is repeated three times, once for each of the input patterns A, B and C.

uses only information local in space and time. This property gives the system some sort of biological realism. In a real brain, no subsystem has a 'privileged' point of view, from which it can observe the state of areas other than those that it has synaptic contact with. For this reason, we cannot rely on the possibility of inspecting the inner workings of the system.

Computationally speaking, it is likely to be more convenient to use the knowledge of the internal connectivity and structure of a network to identify the output pattern(s) corresponding to a given input pattern(s). The goals we set forbid this kind of shortcuts and require all relevant information necessary for the functioning of the system to be available without a priori knowledge of the inner structure and dynamics of the system itself.

Reading out information from the network consists in the operation described on page 31. Project the network state onto a suitable set of patterns (vectors). The elements needed are the network state S and the external inputs. We want to show that, by simple observation of the output of the network, that is, of its state S, it is possible to infer the network state elicited by any pattern or pattern sequence presented as input to the network. The only knowledge needed is *which* pattern is presented to the network at any point in time. No information is needed about the synaptic input provided by a pattern, nor it is necessary to use the network in a regime different from its normal one.

Let us suppose that we want to measure the pattern of activity generated by a given input pattern sequence $\mathcal{S}$. For a sequence $\mathcal{S} = A, B, C$, the task consists in finding those neurons which are always active after the presentation of patterns A, B, and C, in this order. Not all active neurons code for the sequence $\mathcal{S}$. Some of the neurons active after the presentation of the sequence code for some of the patterns presented to the network before presenting the sequence $\mathcal{S}$. The neurons coding only for the pattern sequence $\mathcal{S}$ can be found by averaging over the initial state of the buffer. The initial state of the buffer codes for the input patterns presented to the network before the presentation of the sequence $\mathcal{S}$. Averaging over the initial state of the buffer network means presenting the pattern sequence $\mathcal{S}$ starting from several different uncorrelated initial states of the buffer, which is equivalent to presenting the sequence $\mathcal{S}$ after the presentation of a randomized sequence of patterns. After having presented the sequence, the neurons coding the sequence $\mathcal{S}$ will have probability 1 of being active, all other neurons will have a lower probability. By histogramming the number of times each neuron has been active after the presentation of the sequence, it is possible to classify those neurons always responding to the given pattern sequence. Figure 4.3 shows the histogram of the activity of the learning network or, equivalently, of the encoding layer, after the presentation of the pattern sequence $\mathcal{T} = A, B$, starting from a random initial states of the buffer. The neurons that are more often active code for the sequence $\mathcal{T}$. In the upper part of the plot we show, for comparison purposes, the 'theoretical' pattern of activity, elicited in the encoding network by the same pattern sequence. This pattern of activity was easily obtained using information about the connectivity of the network and the structure of the network. The matching between the theoretical pattern of activity and the (thresholded) histogram of the activity shows that all the information needed about the network state can be found without inspection of the system. Our network makes only use of information local in space and time, which is available to single neurons.

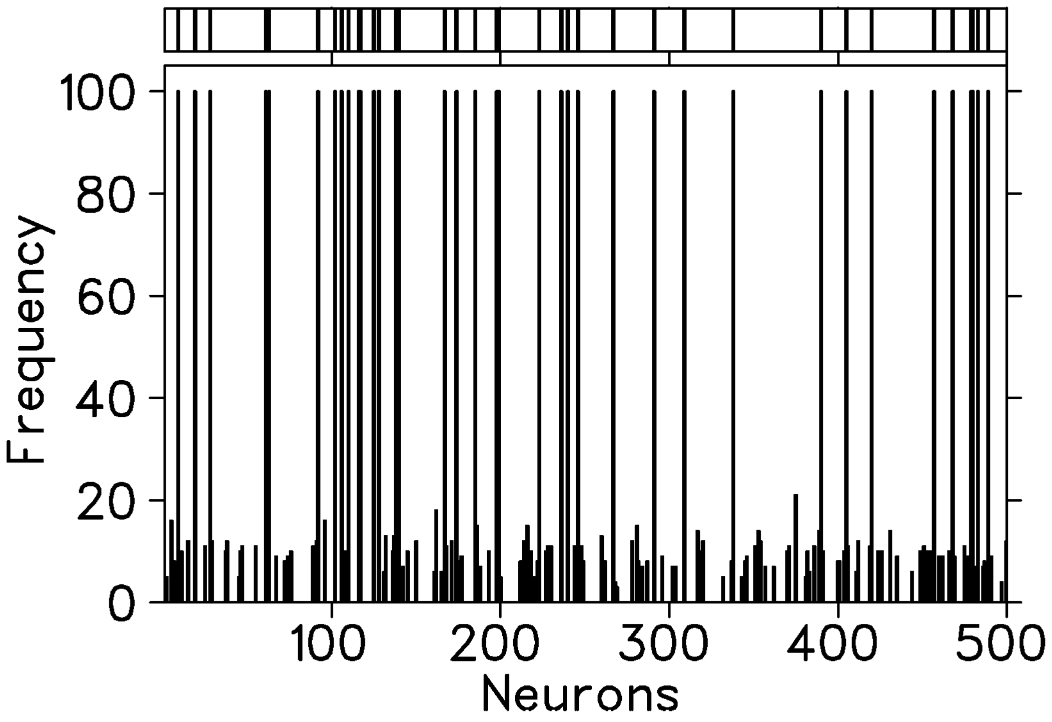


Figure 4.3: Histogram of the activity of the learning network, corresponding to the activity of the encoding layer, after the presentation of the pattern sequence $\mathcal{T} = A, B$ starting from 100 randomly selected initial states of the buffer. The learning and the buffer networks and the encoding and input layers are composed by 500 neurons. The average activity of the networks and layers is 0.1. For comparison purposes, the top plot shows the theoretical pattern of activity elicited in the encoding layer by the same pattern sequence. The encoding mechanism assigns about 50% of the active neurons to the last pattern, 25% to the second last, and so on. The expected number of neurons coding the sequence $\mathcal{T}$ therefore is $500 \cdot 0.1 \cdot (0.5 + 0.25) = 37.5$. The actual number of active neurons is 37.

Figure 4.4 shows the same plot for a sequence of the kind $\mathcal{U} = A, x$, where the x indicates a randomly chosen input pattern. In this case, the task consists in finding those neurons which are always active after the presentation of A as second last pattern, or, in other words, finding those neurons which are always active after the presentation of A followed by any other pattern. The neurons that code exclusively for the pattern sequence can be found by averaging the encoding network activity over the initial state of the buffer (it should be randomly chosen) and over the pattern following A (all possible patterns should be used). The theoretical activity pattern can be computed analogously to the previous case. The theoretical pattern is plotted in the top part of Figure 4.4. It must be remarked that the theoretical pattern shows all neurons that can take part to the pattern coding for $\mathcal{U} = A, x$. Each time the sequence is presented to the network, on average only one half of those neurons will go active, randomly selected by means of the last, random input pattern.

4.1.3 Repeated patterns

Because of the definition of transition given in Section 2.2.1, the network cannot generate sequences with repeated patterns. For this reason, the diagonal elements of the Markov and transition probabilities matrix are always zero. Anyway, the encoding mechanism allows the creation of patterns of activity corresponding to the repeated presentation of the same input pattern. The pattern generation occurs with the same modalities as for different patterns. Figure 4.5 shows those neurons responding to the presentation of the pattern sequence $\mathcal{V} = A, A$. The data plotted have been computed as explained above for the other two cases. Unfortunately, the use of these patterns of activity is impractical. The transition a network should make would be between a pattern of activity f and a pattern of activity $f + 1/4\, f$, the latter being the pattern of activity elicited by the pattern sequence $\mathcal{V} = A, A$. The detection of this transition is problematic, as the global activity is kept close to a fixed value f by the dynamic inhibition of the network.

4.2 Recognizing pattern sequences

From what has been explained in Section 4.1.2, it follows that, without need for modifications or additional elements, the encoding network provides all necessary information to recognize pattern sequences. The recognition of pattern sequences can be done by computing the overlaps of the encoding layer state with the patterns of activity described in Section 4.1.2, corresponding to the activity elicited by the input pattern sequences.

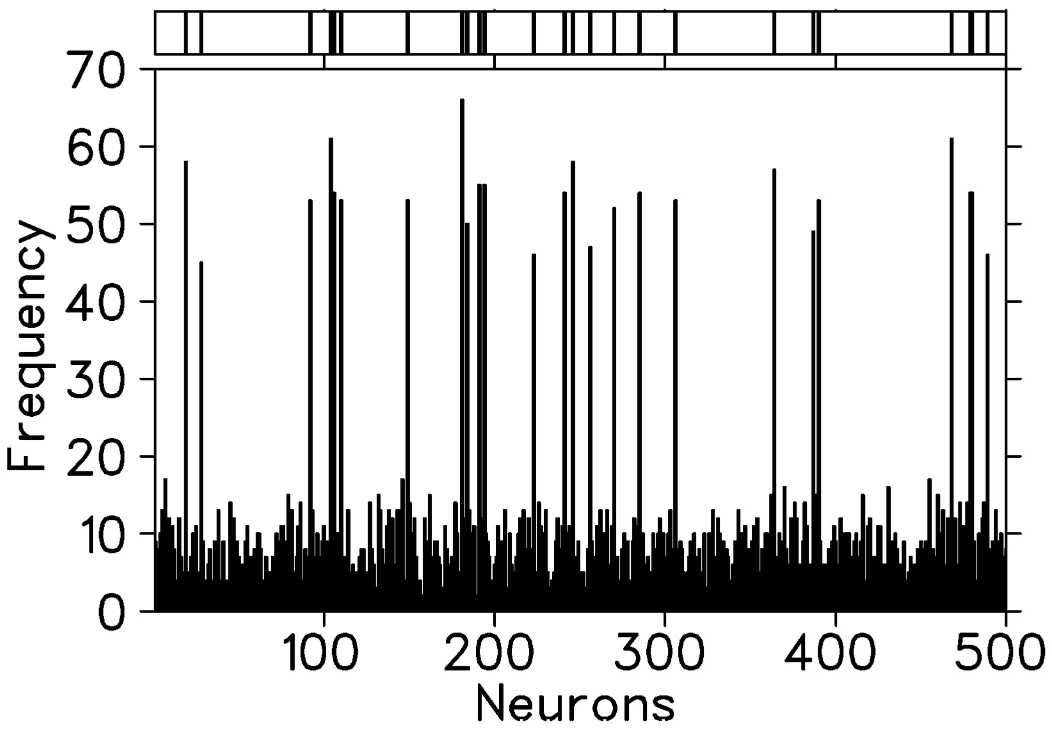


Figure 4.4: Histogram of the activity of the learning network, corresponding to the activity of the encoding layer, after the presentation of the pattern sequence $\mathcal{U} = A, x$ starting from 100 randomly selected initial states of the buffer, where the x indicates a randomly selected pattern. The learning and the buffer networks and the encoding and input layers are composed by 500 neurons. The average activity of the networks and layers is 0.1. For comparison purposes, the top plot shows all possible neurons (25) which can contribute to the pattern of activity elicited in the encoding layer by the same pattern sequence. The encoding mechanism assigns about 50% of the active neurons to the last pattern, 25% to the second last, and so on. The expected number of neurons coding the sequence $\mathcal{U}$ therefore is $500 \cdot 0.1 \cdot 0.25 = 12.5$. The active neurons after each presentations are a subset of the active neurons showed in the top plot.

Let us suppose that the pattern sequence $\mathcal{A}$ elicits in the encoding layer the pattern of activity α, and the sequence $\mathcal{B}$ elicits the pattern of activity β, both averaged over several initial states of the buffer network, as explained in Section 4.1.2. If the state of the encoding layer is S_i ($i = 1, \ldots, N$), where N is the number of neurons, the overlap of S with the pattern α (β) is $a = \frac{1}{N}\sum_{i=1}^{N} S_i\alpha_i$ ($b = \frac{1}{N}\sum_{i=1}^{N} S_i\beta_i$). When the pattern sequence $\mathcal{A}$ is presented to the network, we have that $a > b$. Conversely, when pattern sequence $\mathcal{B}$ is presented to the network, we have $b > a$. A readout system implemented with a trained perceptron and a *winner-take-all* (as explained

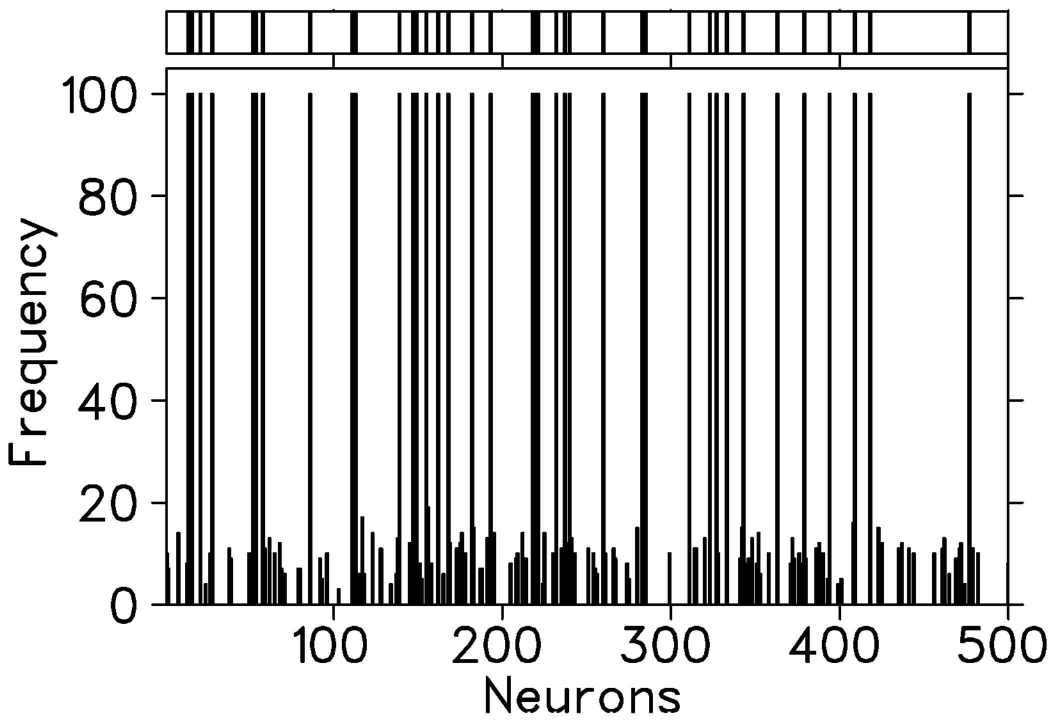


Figure 4.5: Histogram of the activity of the learning network, corresponding to the activity of the encoding layer, after the presentation of the pattern sequence $\mathcal{V} = A, A$ starting from 100 randomly selected initial states of the buffer. The learning and the buffer networks and the encoding and input layers are composed by 500 neurons. The average activity of the networks and layers is 0.1. For comparison purposes, the top plot shows the theoretical pattern of activity elicited by the same pattern sequence. The encoding mechanism assigns about 50% of the active neurons to the last pattern, 25% to the second last, and so on. The expected number of neurons coding the sequence $\mathcal{V}$ therefore is $500 \cdot 0.1(0.5 + 0.25) = 37.5$. The actual number of active neurons is 38.

by Jeager [56]), can in this way recognize pattern sequences.

As an example, we assign five different input patterns to the letters *f*, *o*, *w*, *l*, and *b*. Using the encoding network (no learning is necessary at this point), we analyze the encoding layer states s_1 and s_2 elicited after presenting the pattern sequences *fowl* and *bowl* several times starting with different, randomly chosen, buffer states. The two sequences have the last three patterns (out of four) in common. This means that we cannot expect a difference in the overlaps greater than $1/16 \cdot N = 1/16 \cdot 1000 = 6.25$. The average overlaps (over 100 trials) of the state of the encoding layer after one single presentation of the pattern sequences *fowl* and *bowl* with the patterns s_1 and s_2 are:

- (s_1,*fowl*)=87.0
- (s_1,*bowl*)=82.2
- (s_2,*bowl*)=88.0
- (s_2,*fowl*)=82.2

The differences, albeit small, are large enough for distinguishing the two pattern sequences.

Jaeger's 'echoed states network' [56] has functionalities that are very close to our encoding network. In this model, at any point in time, the state of the network reflects the sequence of input patterns presented to the network. A set of linear output units is trained to predict the next pattern. The output is then used to predict the next pattern in a sequence. For example, when presented with the first 3 letters *fow* of the sequence *fowl*, the overlap of the network state with the learned pattern for *fow* is maximum. This indicates that the next pattern is a *l*.
In one of his examples, Jaeger turns the output vector into a probability vector and the next pattern in the sequence is predicted by a weighted random draw. One of the advantages of our network over Jaeger's model is that the learning network itself predicts the next pattern in a sequence when presented with a partial sequence. The pattern selection mechanism is stochastic, and is based on the transition probabilities of the pattern sequence used in the learning phase. The selection mechanism is intrinsic to the network dynamics, in this way, there is no need for an external system that randomly selects the next pattern.

4.3 Performance of the network

4.3.1 Sequences generated by a Markov process

Before starting testing the network capability in processing pattern sequences generated by a non-Markov process, we want to verify that the network, in its new, extended configuration, is still able to learn pattern sequences generated by a Markov process. The answer is not trivial. Because of the way the coding mechanism works, the active neurons in the encoding layer, that provide input to the learning network, code for different patterns. More precisely, on average 50% of the neurons code for the last pattern, 25% for the second last, and so on. If the pattern sequence is generated by a Markov process, the probability for the next pattern in the sequence just depends on the last pattern. All information about previous patterns is not relevant. It follows that only about one half of the neurons code for useful information, the rest being noise that affects the dynamics of the system by increasing the rate of unsuccessful recalls. The decrease in the performance of the network can be numerically computed by training the network with a pattern sequence generated by a Markov process. As explained in Section 3.1.3, the patterns are created in pairs, using again the matrix showed in Table 3.1. The first pattern is chosen randomly with uniform probability, the second pattern is chosen according to the Markov matrix. The two patterns are then presented to the network, one after the other. The network learns the transition probability between the first pattern and the second one by potentiating the synapses connecting the neurons active after the presentation of the first pattern to the neurons active after the presentation of the second pattern, as described in Section 2.2.2. Some synapses that connect neurons active *before* the presentation of the first pattern to neurons active *after* the presentation of the first pattern will be potentiated as well. The state of the network before the presentation of the first pattern reflects the sequence of patterns previously presented to the network. In the case of pattern sequences generated by Markov processes, and for the training procedure described above, the patterns previously presented to the network are uncorrelated with the pattern presented next. This means that the pattern of activity of the network before the presentation of the first pattern is uncorrelated with the pattern that is going to be presented, and that the long term potentiation that will occur just represents a noise term in the synaptic matrix. Figure 4.6 shows the network transition probabilities as a function of the Markov transition probabilities for $I_0 = 0.015$ and $\beta = 15$ (compare with Figures 3.8 and 3.9 and see code in Section A.11). The plot is almost horizontal. The network is not able to generate pattern sequences with statistics similar to those of the input sequence. The fact that the four measured transition probabilities are quite similar to each other indicates that the transition happens with almost the same probability. The same effects can also be observed for high pseudo-temperatures (low β) for the network used in Chapter 3 in combination with sequences produced by Markov processes. A lower pseudo-temperature does not improve the performance. The intrinsic noise dominated the dynamics and does not allow the network to properly learn the transition statistics of the input sequence.

The Markov process described by the matrix showed in Table 4.2 produces a pattern sequence with 3 periodic states and 4 *positive-recurrent* states. The sequence is of the kind $\mathcal{S}$=$A,(B,C,D,E)F,G,A$, where the pattern following A can be any of the set A,B,C,D, with different probabilities. As it has been explained above, the network state at any point in time, depends on the patterns previously presented to the network. When the

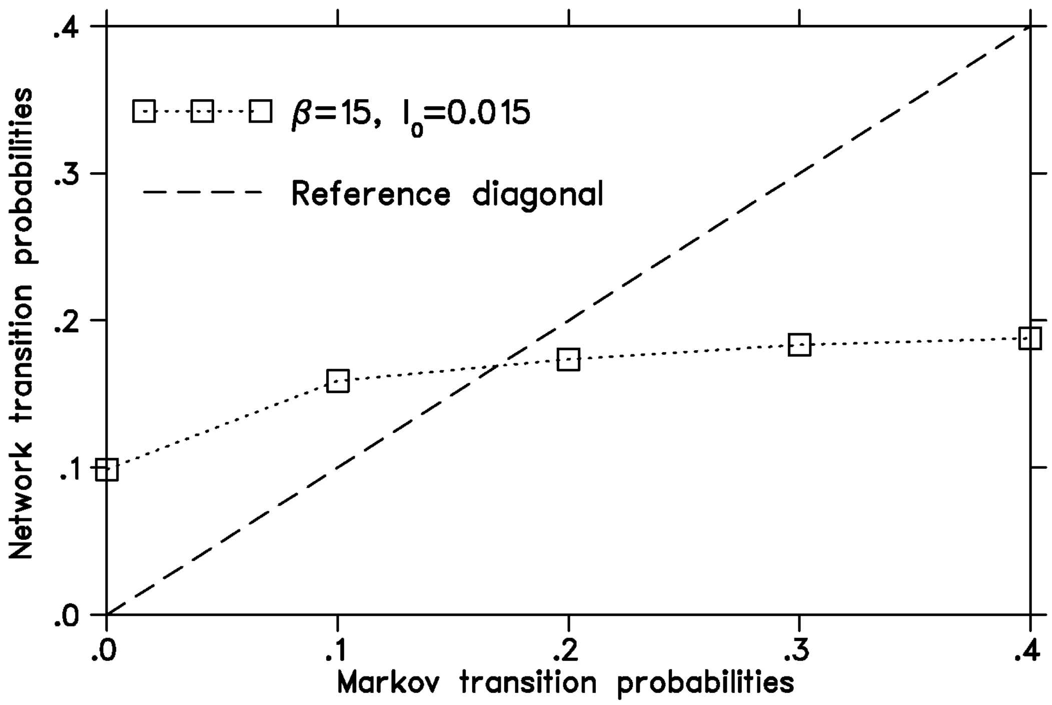


Figure 4.6: Network transition probabilities as a function of the Markov transition probabilities. The error bars are smaller than the plot symbols (data points averaged over 15 trials, $I_0 = 0.015$, $\beta = 15$, 20000 presentations during the training, $\lambda_f = 0.3$, Markov matrix showed in Table 3.1).

pattern sequence F, G, A is presented, 50%+25%+12.5%=87.5% of the active neurons of the encoding network are active and code for the past three inputs. The remaining 12.5% code the pattern presented previously to the network. The activity of the network state has a smaller random component than in the more general case described at the beginning of this Section. As a consequence, the network will perform better in generating pattern sequences with transition statistics similar to those of the training sequence. Figure 4.7 shows the measured transition probabilities of the generated pattern sequence as a function of the Markov transition probabilities. The performance of the network is similar to that of the network described in Chapter 3 in the case of sequences generated by Markov processes.

4.3.2 Higher order Markov processes

The evaluation of the network performance for pattern sequences generated by a non-Markov process is analogous to the evaluation for pattern sequences generated by a Markov process, as described in Section 3.1.2. On page 83

0.0	0.0	0.0	0.0	0.0	0.0	1.0
0.4	0.0	0.0	0.0	0.0	0.0	0.0
0.3	0.0	0.0	0.0	0.0	0.0	0.0
0.2	0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	1.0	1.0	1.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0

Table 4.2: Markov matrix used to study the performance of the network showed in Figure 4.1 in learning and reproducing the same transition probabilities used in the tests described in Chapter 3.

we showed how a Markov matrix can be used to describe a non-Markov process. This can be done if the matrix elements correspond to the transition probabilities between composed states, instead of single states. Given a suitable Markov matrix M and a set of states, the corresponding process can be used to generate a pattern sequence. The pattern sequence is then presented to the network that modifies its synaptic matrix according to the learning rule described in Section 2.2.2. After the learning process is ended, the network is left free to run, by updating the neurons using the rule described in Section 2.2.1.

The parameter I_0 of the dynamic inhibition should be assigned a suitable value. Let us suppose the network is taught the sequences $\mathcal{S} = A, B, C$, and $\mathcal{U} = D, B, E$. Let $h_{x,y \to z}$ the mean synaptic input provided by the neurons active (as explained on Page 4.1.2) after the presentation of the sub-sequence $\mathcal{W} = x, y$ to the neurons corresponding to pattern z. In our specific case, the maximum performance in generating the learned sequences correctly can be achieved by choosing I_0 as the mean value of the four mean synaptic inputs $h_{A,B \to C}$, $h_{A,B \to E}$, $h_{D,B \to C}$, and $h_{D,B \to E}$. This is clearly the best way to maximize the probability for the neurons belonging to the right pattern in the sequence to go active and for the others to be inactive. This is another shortcoming of our mechanism for the encoding. The optimal I_0 is a function of the order of the Markov model that should be emulated by the network. The higher the number of stimuli back in the past the network should be sensitive to, the lower the I_0 that should be chosen.

When left free to run, the network generates a sequence of patterns, whose transition statistics can be measured and represented in a transition probability matrix T. The matrices M and T are then compared to evaluate the ability of the network in capturing the transition probabilities of the input sequence and generating a sequence with similar transition statistics. The

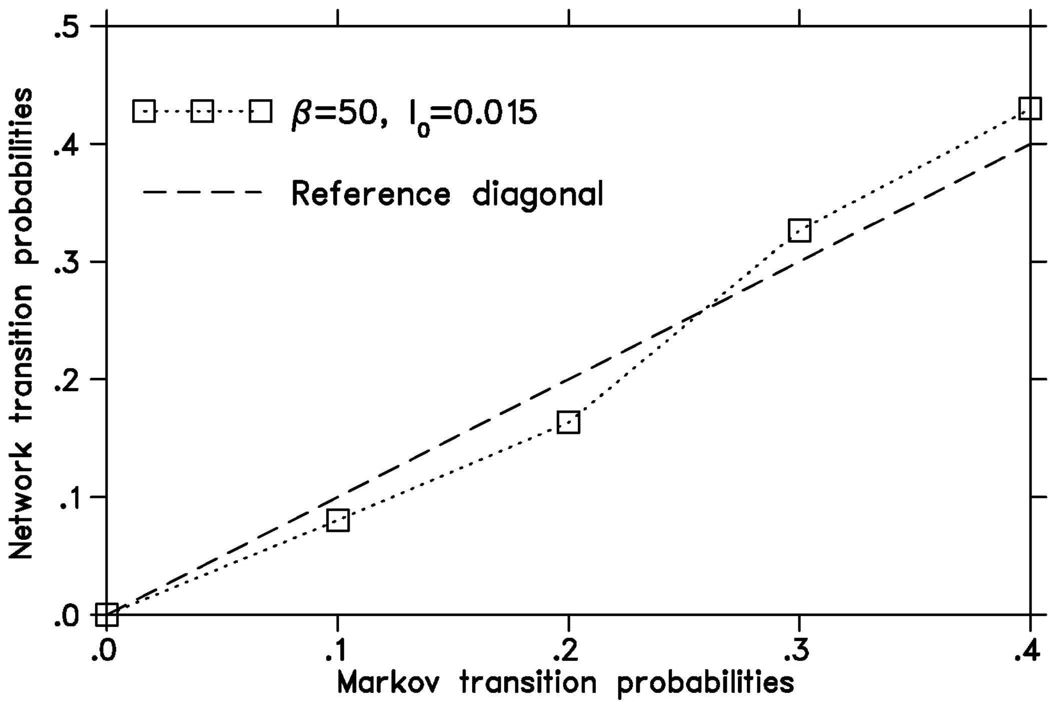


Figure 4.7: Transition probabilities of the networks as a function of the Markov transition probabilities for a network trained with a pattern sequence generated by a Markov process defined by the matrix showed in Table 4.2 ($\lambda_f = 0.1$, $\beta = 55$ and $I_0 = 0.015$, 1200 neurons, 7 patterns, 15000 pattern pair presented during the training). The temperature used is lower than the one used, for example, to produce the data showed in Figure 3.8 and 3.9, where $\beta = 15$. This partially compensate for the residual noise in the encoding mechanism.

more similar the matrices M and T are, the better the network performs.

Second order Markov process

A second order Markov process is a non-Markov process for which the probability for the next state (pattern, in our case) not only depends on the present pattern, but also on one pattern earlier. A second order Markov process can be described by a Markov matrix whose elements correspond to the transition probabilities between single patterns or couple of patterns. For example, given the five states A, B, C, D, and E, the Markov matrix showed in Table 4.3 describes a Markov process of order 2. The sequence generated by this process is ABCDBEABC.... With the sole knowledge of the last pattern generated, it is not possible to correctly predict the transition probabilities to the next patterns. In fact, information about the previous pattern

	AB	C	DB	E
AB	0	0	0	1
C	1	0	0	0
DB	0	1	0	0
E	0	0	1	0

Table 4.3: Markov matrix used to generate an input training sequence corresponding to a Markov process of order 2.

	AB	C	DB	E
AB	0.0001	0	0.0265	0.9994
C	0.9931	0	0.0248	0.0004
DB	0.0069	0.9999	0.0019	0.0002
E	0	0	0.9469	0

Table 4.4: Transition probability matrix obtained from the pattern sequence generated by the learning network after the end of the learning phase. The Markov matrix used to generate the training sequence is displayed in Table 4.3 (1000 neurons, $I_0 = 0.03$, $\beta = 10^3$, $q_+ = 0.01$, $\lambda = 0.3$, and $f = 0.05$, 750 neurons). The 1-sigma lower and upper bounds calculated with Equation 2.4 are all lower than 5%.

is necessary for correctly predicting the next one. From the matrix showed in Table 4.3, if the last *two* patterns are 'AB', then the probability for the next pattern to be 'C' will be one. If the last two patterns are 'DB', the probability for the next pattern to be 'E' will be one. A pattern sequence generated by the Markov process defined by the matrix showed in Table 4.3 was used to train the network. The transition probability matrix of the pattern sequence generated by the network after the learning phase is showed in Table 4.4. The transition probabilities match quite closely the corresponding probabilities of the Markov matrix used to generated the training sequence. This indicates a very good ability of the network in capturing the transition statistics of the training sequence, and in generating a pattern sequence with similar transition statistics.

Third order Markov process

Analogous to the second order Markov case, described in Section 4.3.2, a third order Markov process is a non-Markov process for which the probability for the next pattern not only depends on the latest pattern, but also on

	ABC	D	EBC	F
ABC	0	0	0	1
D	1	0	0	0
EBC	0	1	0	0
F	0	0	1	0

Table 4.5: Markov matrix used to generate an input training sequence corresponding to a Markov process of order 3.

	ABC	D	EBC	F
ABC	0.017	0.0003	0.0238	0.9997
D	0.9007	0	0.0588	0
EBC	0.0454	0.9991	0.0040	0.0003
F	0.0522	0.0006	0.9134	0

Table 4.6: Transition probability matrix obtained from the pattern sequence generated by the learning network after the end of the learning phase. The Markov matrix used to generate the training sequence is displayed in Table 4.5. (900 neurons, $I_0 = 0.03$, $\beta = 10^3$, $q_+ = 0.01$, $\lambda = 0.3$, and $f = 0.05$). The 1-sigma lower and upper bounds calculated with Equation 2.4 are all lower than 5%.

the two preceding patterns. A third order Markov process can be described by a Markov matrix whose elements correspond to the transition probabilities between single patterns or couple of patterns, or triplets of patterns. For example, given the five patterns A, B, C, D, E, and F, the Markov matrix showed in Table 4.5 describes a Markov process of order 3. The sequence generated by this process is ABCDEBCFABC.... The knowledge of the latest or of the last two patterns is not enough to correctly predict the transition probabilities to the next patterns. To correctly associate a pattern with a probability of being followed by any other pattern, it is necessary to know the last *three* patterns generated. The performance of the network has been evaluated as in the case of the second order Markov process. The transition probabilities of the generated pattern sequence are showed in Table 4.6.

The transition probability matrix clearly shows that the network can learn in an unsupervised way a Markov process of order greater than 1, up to at least order 3. The network dynamics can store the information provided by the encoding network in the synaptic matrix to correctly generate a sequence whose transition probabilities reflect those of the input sequence.

4.4 Summary

The network introduced in Chapter 2, together with a pre-processing layer named 'encoding network', can learn in an unsupervised manner the transition probabilities from a pattern sequence generated by a non-Markov process (Markov process of order higher than 1). The performance of the network is showed for sequences generated by Markov processes of second and third order. One way of recognizing pattern sequences, which does not make use of the learning network, is also described. The problems posed by the readout system and by the occurrence of repeated patterns are also discussed.

Chapter 5

Games networks play

Playing is normally regarded as a sign of intelligence. Animals which are usually regarded as most intelligent, mammals, spend quite a lot of time playing. There is little or no evidence that fishes, reptiles, or amphibians, which are normally considered less intelligent than mammals, are ever engaged in playful activities. The problem might lie in the difficulty of telling what it means for a lizard to play. We would not even expect insects or bacteria to be able to play and probably they do not, even if, due to the lack of experimental evidence, this is only an hypothesis. The length of the juvenile period might also be linked to propensity to play. Animals that take longer to reach maturity have longer to learn and practice new behaviors, without needing to care about adult life. Human children have the longest juvenile period in the animal kingdom, and they happen to have the highest intelligence. Playing is for young animals a way to train skills that will be then needed in adult life, for example motor, social, and manual skills. Human children also show to like games that develop skills linked to more abstract thinking, for example symbol processing and verbal abilities.

There is not really an unique definition of intelligence. One possible definition could be the ability to learn or understand or deal with new situations. Such a definition just shifts the problem to the definition of what it means, for example, to *understand* a situation. I will not even try to review the subject or to propose myself a new definition. The concept of intelligence is anyway so loose that even bacteria, viruses or computer programs might be seen as being provided with some degree of intelligence. Even atoms might be considered to have some intelligence, in the way they plug together to, for example, form a crystal. In this sense, intelligence is a vaguely defined feature of systems governed by laws that make them act in way we like to define as 'intelligent'. Often, this feature is an 'emerging' feature, in the sense given to this term by complex systems science. An emerging feature

does not belong to the single parts of a system, but *emerges* from the interactions of these parts (For a pleasant introduction to this sort of issues, see Hofstadter and Dennet's book 'The mind's I' [57]). Single neurons are not able of abstract thinking, but human brains do. Single ants cannot do much, but an ant nest has abilities that ants, by themselves, do not have. The slime molds *Dictyostelium discoideum* usually live as free-living amoebae. Upon starvation they undergo a complex developmental cycle in which about 10.000-50.000 individual amoebae aggregate to form a multicellular body around a few amoebae acting as the aggregation center. Somehow, this slime mold is dealing 'intelligently' with a new situation. The fact that this is the only behavior with which D. Discoideum reacts to food shortage does not really matter. We are here concerned about quality, not quantity. On another scale, the whole gene network in living cells displays some degree of intelligence, adaptation, sociality, and fault tolerance that are proper to more complex systems.
In this chapter I will show how the network I have described in the previous chapters is able to play a simple game. Several conclusions can be drawn from this. Firstly, my network is still in a very juvenile phase of development. Secondly, following the very loose definition of intelligence we have given above, we claim that our network displays some sort of intelligence, as an emerging feature of the interacting units (neurons) composing the network.

The game we have chosen, is the widely known 'scissors, paper, stone' game[1]. Two players at the same time display one of three hand symbols, signifying a stone (player keeps the hand in a fist), a piece of paper (player holds the hand open and flat) , or a pair of scissors (player holds the hand in partial fist with two fingers sticking out). A stone beats the scissors (it blunts). A piece of paper beats the stone (it wraps). Lastly, the scissors beat the paper (they cut). In Japan, the game is called "Jan-ken-pon", and presents a small variation: When one player wins, he says "achi muite hoi!" ("Look that way now!") and points up, down, left or right. When the winner says "hoi!"the other player must look in one of the four directions, trying to avoid looking in the same direction the winner points at. If the loser looks in the same direction as the winner points, he is even more disgraced, and the game is over. If he doesn't look in the same direction, the rock paper scissors game continues [59].

[1]In north America, it is also known as 'rock, paper, scissors game [58]

5.1 Network fun

Playing the 'scissors, paper, stone' game with the network consists in presenting to the network a sequence of patterns taken to from the pattern set $\{A = scissors, B = paper, C = stone\}$. During the game, the network continuously learns. In this way, the network extracts the transition statistics of the presented sequences and, given the last 3 or more patterns, tries to predict the next pattern in the sequence. A correct prediction allows the network to win the game, by presenting stone (pattern C) in response to a predicted scissors (pattern A), scissors (pattern A) in response to a predicted paper (pattern B), and paper (pattern B) to a predicted stone (pattern C).

The game is played as follows:

1. The synaptic weights are initialized by setting them to 0 and 1 with the same probability. Another possibility is to set all weights to 0.

2. The network is updated up to a maximum number of times (20 times in the following tests) until a transition occurs. The pattern to which the transition is made is used as prediction for the next pattern in the sequence.

3. A suitable pattern is played, as explained above, and the prediction is compared to the played pattern to evaluate the performance of the network.

4. The network learns the transition from the starting pattern to the pattern that had to be predicted, independently on the fact that the pattern has been correctly predicted or not. The played pattern becomes the new starting pattern.

5. The procedure is repeated from point 2.

Three different sequence orderings were used in the game, A, B, and C corresponding to, respectively, scissors, paper, and stone:

$\mathcal{S}$ = **A, B, C, A, B, C, ...** The network is expected to be able to predict all patterns.

$\mathcal{T}$ = **C, B, A, C, B, A, ...** The network is expected to be able to predict all patterns.

$\mathcal{U}$ = **A, B, C, B, A, C, ...** Sequence in which each triplet of patterns is a permutation of the set $\{A, B, C\}$. As in the case of the sequence $\mathcal{U}$, the network is expected to correctly predict the patterns with probability $1/3 \cdot (1/3 + 1/2 + 1) = 0.61$.

The expected prediction rates correspond to the case in which the network makes a transition to a pattern. This happens when the maximum number of times the network is updated is substantially higher than the escape time of the network state from any attractor of the dynamics. If the maximum number of updates is not large enough, the network does not always make a transition to another pattern. In this case, the starting pattern, that is the pattern the network state has maximum overlap with, is used as a prediction of the played pattern. Because of the way the pattern sequences are generated, any pattern can happen to be used in a sequence twice in a row. For example, it can happen when using the sequence ordering $\mathcal{U}$, if the last pattern of a triplet is the same as the first pattern of the next triplet, e.g., ..., A, B, C, C, A, B, ... Anyhow, because of the definition of transition, the network cannot make a transition to the same pattern, and so it cannot correctly predict the occurrence of the same pattern in a sequence twice in a row. A correct prediction can happen if the network parameters and the maximum number of updates are set in such a way that the network does not always make a transition to a different pattern, as explained above. In this case, the performance can be increased, in the best case, by up to $1/9$, in correspondence to the correct prediction of all first patterns of the triplets composing the pattern sequence. In this case, the maximum possible performance is 0.72.

The sequences $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ were generated and presented to the network. The performance of the network in predicting the next pattern in the sequence, and so in winning the game, is plotted in Figures 5.1, 5.2, 5.4, and 5.5. The sequences used to produce the data showed in all figures start with a short sub-sequence in which the patterns are randomly chosen. In all cases, the network is expected to be able to correctly predict the random sub-sequences with a probability $1/3$. Figure 5.3 shows the effects of an increased (or decreased) temperature on the performance of the network.

Figure 5.5 shows the performance of the network when the presented sequences are generated by alternating two orderings, and namely, ordering $\mathcal{S}$ and ordering $\mathcal{T}$. When the network is presented for the second time the sub-sequences ordered according to $\mathcal{S}$, the performance does not drop to a value close to zero, as when the network is presented for the first time a non-random sequence. This indicates that some memory about the past learned transition probabilities is still present in the synaptic matrix. This effect is due to the stochastic learning rule, which slowly replaces old memories with the new one, as explained in Section 2.2.2 (see Figures 5.6, and 5.7).

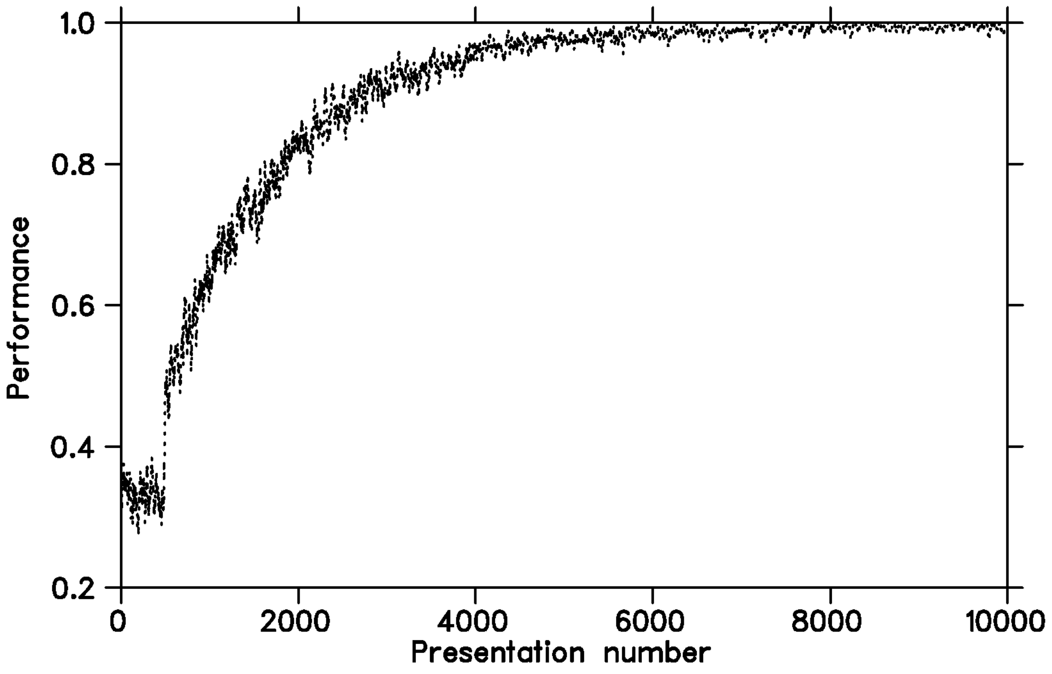


Figure 5.1: Performance of the network in playing the 'scissors, paper, stone' game as a function of the number of presented patterns. The maximum performance, 1, corresponds to 100% of correctly predicted patterns. The presented sequence is composed by 10000 patterns. Patterns from 0 to 499 are randomly chosen. Patterns 500 to 10000 are ordered according to $\mathcal{S}$. The synaptic matrix weights were initialized to 1 and 0 with the same (0.5) probability. As the learning modifies the synaptic matrix, the network performance increases toward the asymptotic, expected value 1. The figure shows the cumulative average (window width: 20 patterns) of the average of the performance calculated over 16 trials (800 neurons, $f = 0.05$, $\lambda_f = 0.3$, $\lambda_b = 0$, $I_0 = 0.015$, $\beta = 100$).

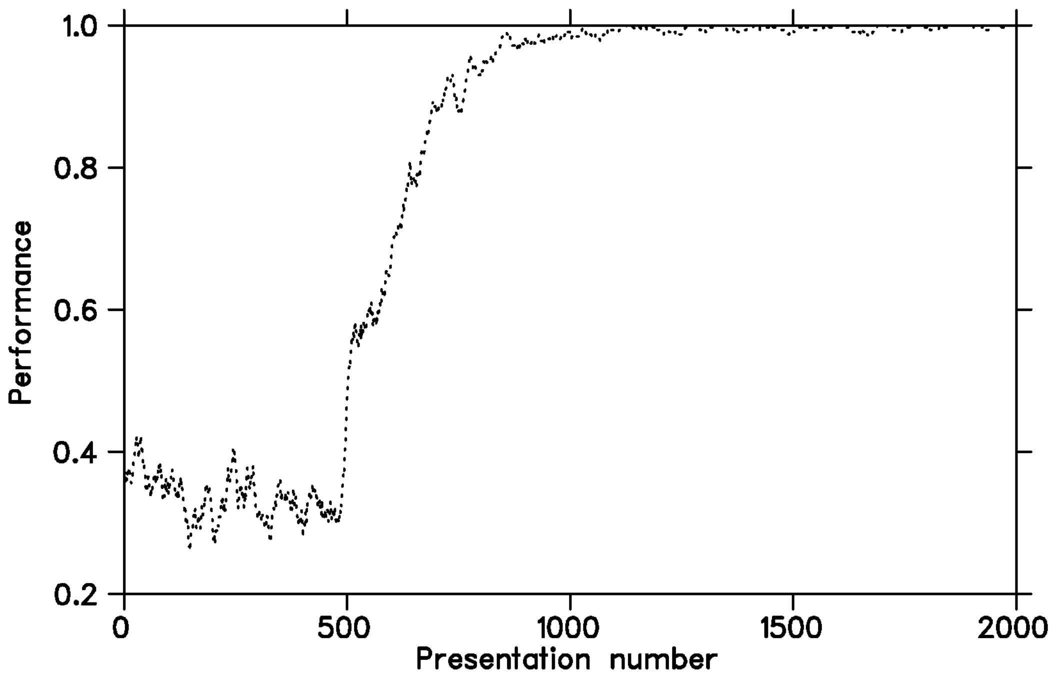


Figure 5.2: Performance of the network in playing the ‘scissors, paper, stone’ game as a function of the number of presented patterns. The maximum performance, 1, corresponds to 100% of correctly predicted patterns. The presented sequence is composed by 10000 patterns. Patterns from 0 to 499 are randomly chosen. Patterns 500 to 10000 are ordered according to $\mathcal{S}$. The synaptic matrix weights were all initialized to 0. As the learning modifies the synaptic matrix, the network performance increases toward the asymptotic, expected value 1. The performance increases more rapidly with the number of presentations than in the case showed in Figure 5.1. This is due to the different initialization of the synaptic matrices. In the case showed in Figure 5.1, the network dynamics is affected by the noise related to the random initialization of the synaptic matrix. The noise slows down the response of the network dynamics to the on-going learning process. The figure shows the cumulative average (window width: 20 patterns) of the average of the performance calculated over 16 trials (800 neurons, $f = 0.05$, $\lambda_f = 0.3$, $\lambda_g = 0$, $I_0 = 0.015$, $\beta = 100$).

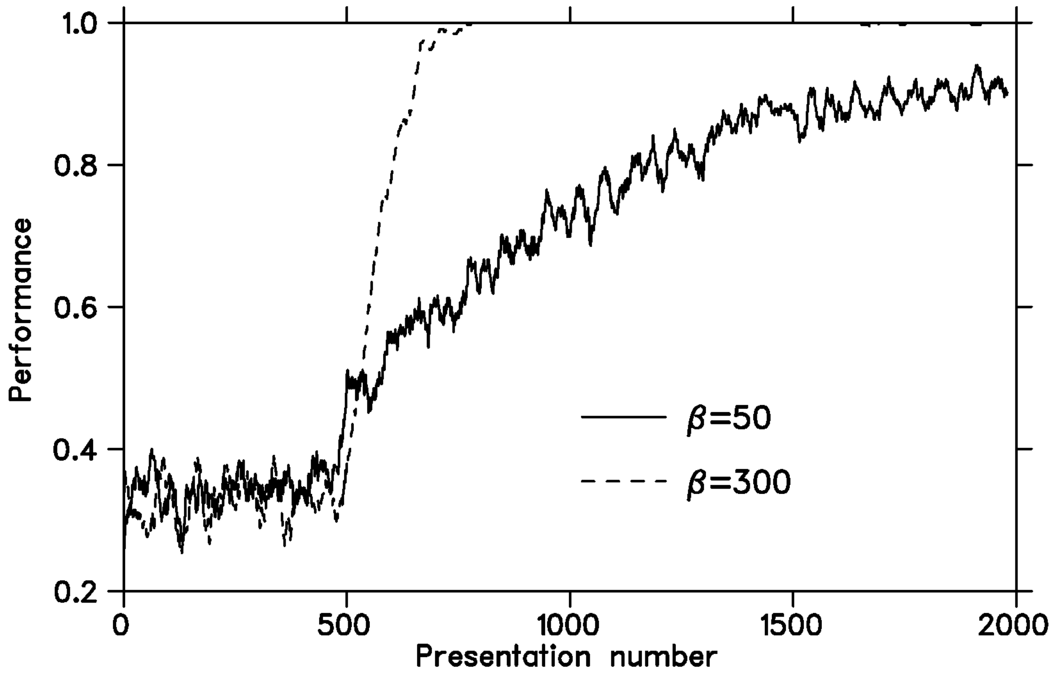


Figure 5.3: Performance of the network in playing the 'scissors, paper, stone' game as a function of the number of presented patterns for two network temperatures, $\beta = 50$ and $\beta = 300$, to be compared with the plot showed in Figure 5.3. A decrease in temperature (increase in β) does not significantly affect the learning, whereas an increase in the temperature (decrease in β) slows down the learning. This result is consistent with the fact that an increased level of noise increases the time necessary to reach the asymptotic configuration of the synaptic matrix.

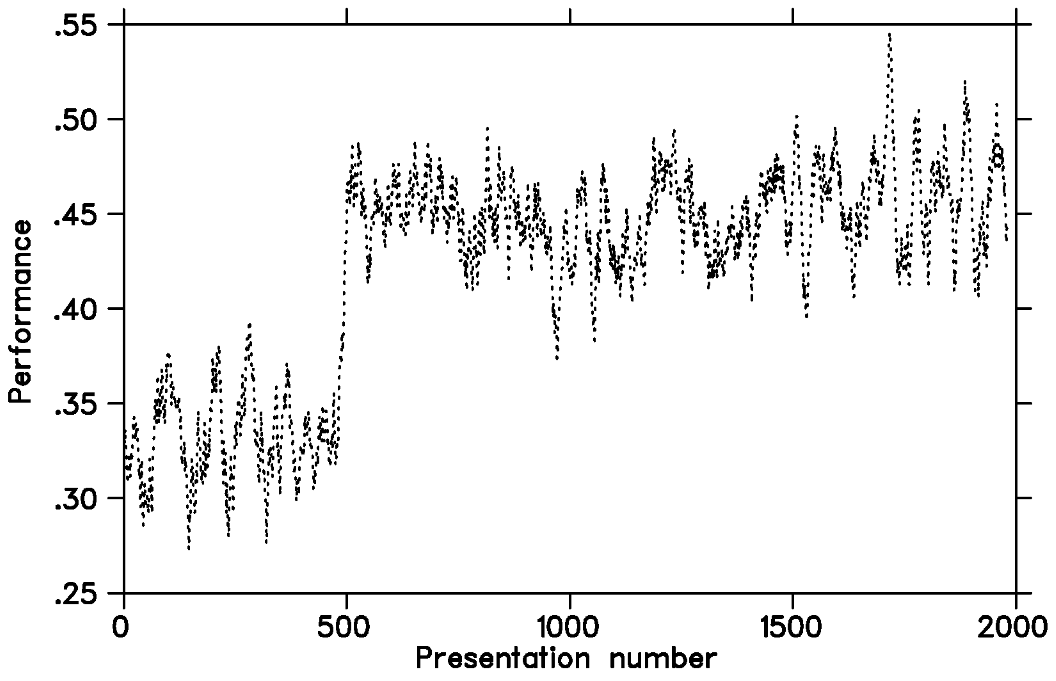


Figure 5.4: Performance of the network in playing the ‘scissors, paper, stone’ game as a function of the number of presented patterns. The maximum performance, 1, corresponds to 100% of correctly predicted patterns. The presented sequence is composed by 2000 patterns. Patterns from 0 to 499 are randomly chosen. Patterns 500 to 2000 are ordered according to $\mathcal{U}$. The synaptic matrix weights were initialized to 0. As the learning modifies the synaptic matrix, the network performance increases toward is average asymptotic value. The figure shows the cumulative average (window width: 20 patterns) of the average of the performance calculated over 16 trials (800 neurons, $f = 0.05$, $\lambda_f = 0.3$, $\lambda_g = 0$, $I_0 = 0.015$, $\beta = 100$).

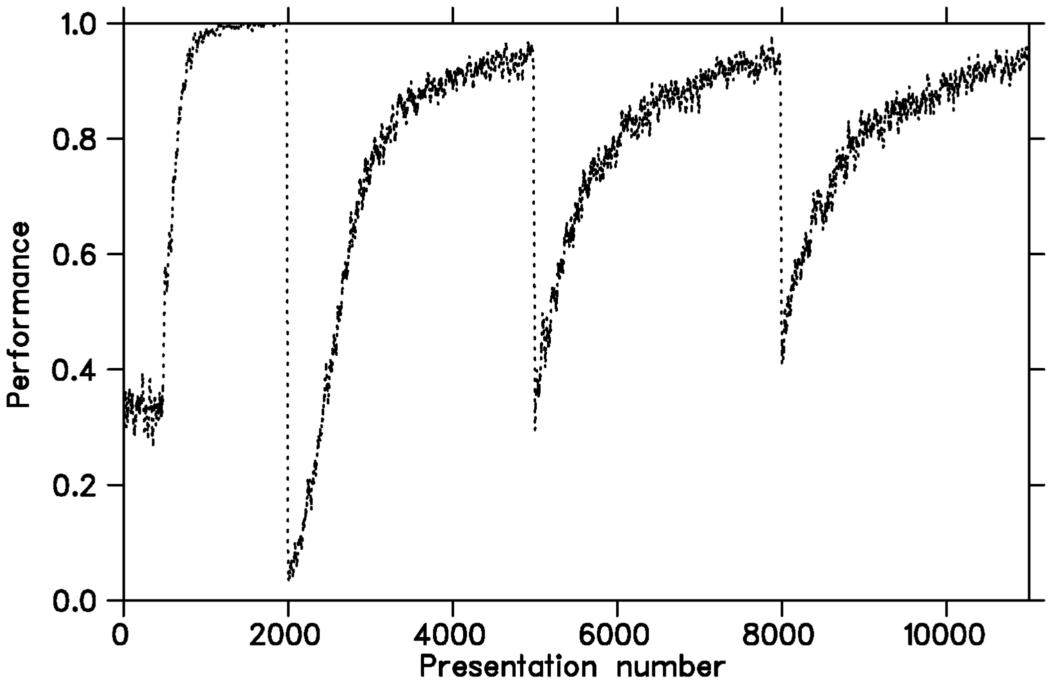


Figure 5.5: Performance of the network in playing the 'scissors, paper, stone' game as a function of the number of presented patterns. The maximum performance, 1, corresponds to 100% of correctly predicted patterns. The presented sequence is composed by 2000 patterns. Patterns from 0 to 499 are randomly chosen. Patterns 500 to 1999 are ordered according to $\mathcal{S}$, from 2000 to 4999 according to $\mathcal{T}$, from 5000 to 7999 again according to $\mathcal{S}$), and finally, patterns from 8000 to 11000 are ordered again according to $\mathcal{T}$. Each time the pattern ordering of the pattern sequence is changed, the performance drops to a value which is directly related to the memory of the presented sequence the network still has. The fact that the performance does not drop to a value close to zero indicates that still some memory of the previously presented sequence ordering is encoded in the synaptic matrix. This is a feature of the stochastic learning rule, which slowly removes old memories, replacing them with newer ones. This effect is more clearly showed in Figures 5.6, and 5.7.

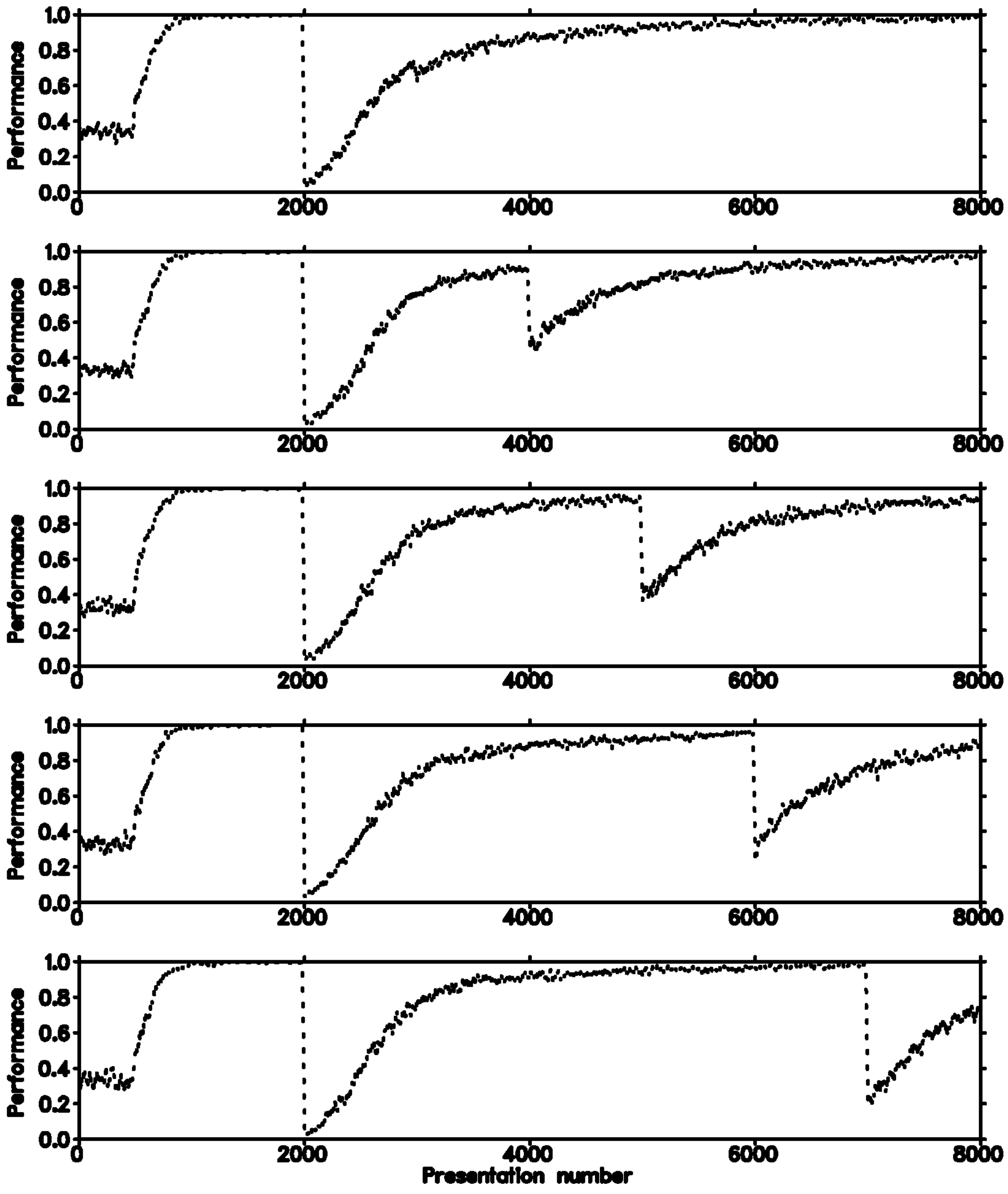

1.0
0.8
0.6
0.4
0.2
0.0
Performance
0
2000
4000
6000
8000
Presentation number

Figure 5.6: Performance of the network in playing the ‘scissors, paper, stone’ game as a function of the number of presented patterns. The maximum performance, 1, corresponds to 100% of correctly predicted patterns. The presented sequences are composed by 2000 patterns. The five plots show the performance of the network for five different pattern sequences. Each sequence is composed by three sub-sequences. The first sub-sequence, from pattern 0 to pattern 499, is generated by choosing the patterns in random order. The second sub-sequence, ranging from pattern 500 to pattern 1999 was generated choosing the sequence ordering $\mathcal{S}$. After this sub-sequence is presented to the network, the network is able to correctly predict the patterns in the sequence with a probability very close to 1. The five plots corresponds to five different lengths of the third and fourth sub-sequences. In the top plot, the third sub-sequence, generated using the ordering $\mathcal{T}$, is 1000 patterns long. In the other plots, the third sub-sequence is increasingly long: 2000 patterns for the second plot (from the top), 3000 patterns for the third plot, 4000 for the fourth plot, and 5000 patterns for the bottom plot. The fourth sub-sequence, generated using the ordering $\mathcal{S}$ again, lasts the number of patterns necessary to have a sequence length equal to 8000 patterns. In all plot, the performance drops to a value close to zero at the beginning of the third sub-sequence. This is due to the fact that, in all five cases, the network has never been exposed to a sequence ordered according to the sequence ordering $\mathcal{T}$. When presented with the third sub-sequence, the learning process gradually modifies the synaptic matrix, accommodating information regarding its transition probabilities. In this way, the performance of the network gradually increases. After a variable number of presentations, at the beginning of the fourth sub-sequence, the sequence ordering is switched back to $\mathcal{S}$. In this case, the performance does not drop down a value close to zero. This fact shows that some memory traces of the transition probabilities previously learned is still present in the synaptic matrix. This is a characteristic of the stochastic learning rule (see Section 2.2.2), which slowly removes old memories to make room for information about the transition probabilities of the pattern sequence being showed.

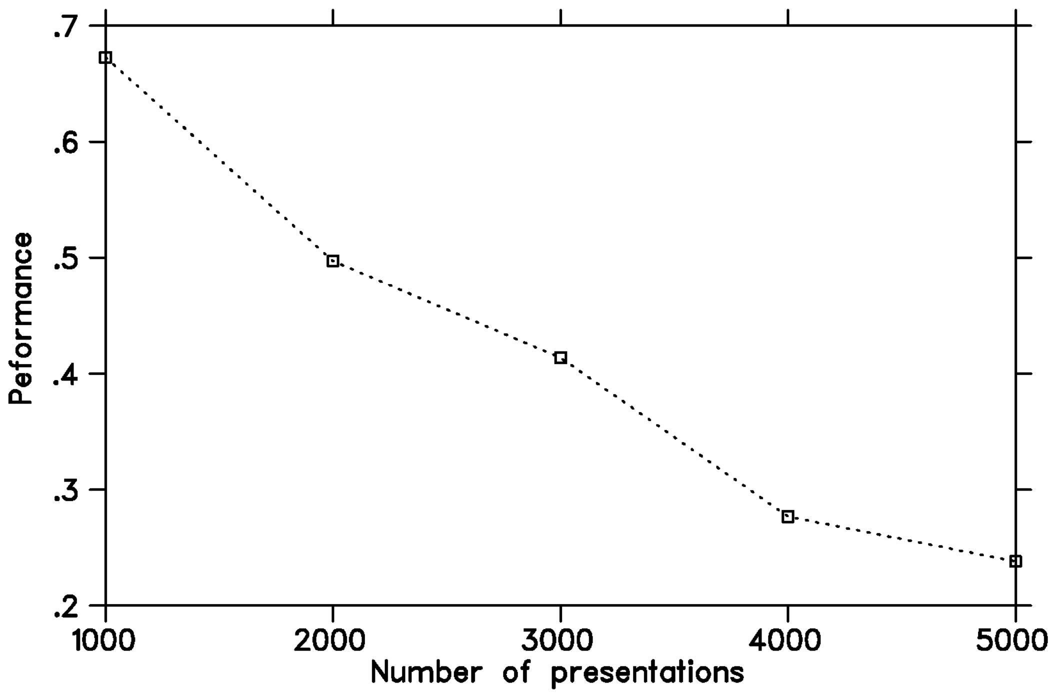


Figure 5.7: Performance of the network at the beginning of the fourth sub-sequence (see Figure 5.6), as a function of the length of the third sub-sequence. The decreasing performance is consistent with the fact that the longer the third sub-sequence is, the more the memories related to the second sub-sequence are forgotten.

5.2 Summary

We show that our network is able to play the 'scissors, paper, stone' game. The increasing performance of the network during the course of the game goes together with the amount of information encoded by the learning rule in the synaptic matrix of the network. The game is played with different sequences and the theoretical, maximum performance is compared with the actual performance. The actual performance of the network well approaches the theoretical one with the given traning set.

Chapter 6

Conclusion

We have showed how a Hopfield-like network can learn in unsupervised way temporal sequences of stimuli consisting of patterns of activity of a set of neurons (the input layer). When presented with a sequence of stimuli, a Hebbian learning rule modifies the synaptic matrix. The learning rule creates attractors corresponding to the patterns of activity elicited by the external inputs. Additionally, the learning rule creates projection from the neurons participating in the pattern of activity of one attractor to the neurons participating in the pattern of activity of the attractors following the first one. The number of projection between attractors is roughly proportional to the frequency by which patterns follows each other in the input sequence.
In the presence of noise, the attractors are metastable. We showed that the transition probabilities between attractors depend on relative frequencies of presentation of the stimuli and so on the number of projections between neurons participating in the activity pattern of different attractors. This finding represents one step forward in the understanding of the mechanisms by which brains process temporal stimuli. Our results complement the research done by various researchers in this field, many of which were mentioned in the previous chapters. The dependence of the transition probabilities from the correlations of the attractors and, indirectly, from the temporal correlations of the presented stimuli, was only supposed. We carefully investigated these phenomena and showed under which conditions the network dynamics can better replicate the statistics of the sequence used in the training. The dependence on the level of noise could indicate a possible important role of fluctuations in real neural circuits. The same principles might more generally apply to other kinds of network, as, for example, gene or metabolic networks. The formal similarities between neural networks and gene networks, schematically shown in Table 6.1. Cell cycle, determination, differentiation, and the so-called cell memory can be easily modeled with the kind of net-

Neural networks	Genetic networks
Neurons	Genes
Dendrites	Regulatory regions
Synapses	Operators
Axons	Operons
Neurotransmitters	Regulatory proteins
Attractors	Cell memory
Population coding	Combinatorial control
Trained by experience	Trained by evolution

Table 6.1: Formal similarities between neural and gene networks.

work we employed. The fact the an optimal level of noise exists that optimize the performance suggests that natural networks might be tuned to correctly work in some range of noise. This conjecture might be worth some serious thought.

The network can only work for pattern sequences generated by Markov processes. This is because the transition probabilities at any point in time only depend on the present state of the network, that is, on its present activity pattern. One way to allow the network to learn and generate patterns sequences generated by non-Markov processes is to make the state of the network at any point in time dependent on more than one pattern. If the network state is made dependent on several input stimuli, the network can process sequences generated by non-Markov processes.

In our approach, the network state is incrementally built by mixing the present state of the network with the pattern of activity of the input layer. This mixing mechanism allows to slowly forget old input patterns. A certain fraction of the active neurons code for the latest pattern presented to the network, and a decreasing number of neurons for patterns presented earlier in time. The encoding layer together with the buffer network provides for this kind of pre-processing of the input pattern sequence. The encoding mechanism is good enough to show that the kind of preprocessing we devised is sufficient to allow the network to learn and generate stimuli sequences generated by non-Markov processes. Unfortunately, its implementation does not meet some basic requirements of simplicity. In particular, the buffer network was introduced to provide a functionality that we were not able to generate otherwise. A better approach could use the delay activity of the network as a memory of the pattern of activity elicited by the previous stimulus. So, instead of combining the inputs from the buffer network and the input layer, one would combine the *recurrent* input of the encoding layer (of course, a fully connected network should be used in place of a simple neuron layer) and the input from the input layer. A similar approach failed, as explained in Section 4.1.1. However, its simplicity might look appealing enough to make someone go on investigating this possible solution.

6.1 Echoed (liquid) states

The approach that most resembles our network is the one described by Jaeger [5] (A similar model has been developed by Maass and colleagues [41]). Jaeger uses a large recurrent neural network as a dynamics "reservoir" to store information about the temporal pattern sequence presented to the network. Output units tap from this reservoir. The network-to-output connection weights between the reservoir and the output units are trained using highly efficient linear regression algorithms. Our model differs from Jaeger's network in three ways:

Learning The learning procedure used by Jaeger is supervised. This means that the weights from the reservoir to the output units are adjusted to minimize the error between the trajectory described by the output units and a given trajectory. The need for supervision does not allow one to model, for example, the experimental findings of Miyashita. The phenomena described by Miyashita occur 'automatically', without the necessity of an explicit comparison between the output of the network and a target output. Our model extracts, in an unsupervised manner, the transition probabilities from a temporal sequence of patterns, without the need of an external target function.

Generation Jaeger's model is not able to autonomously generate the next point of a learned trajectory. The values of the output units are used, after normalization, as probabilities for the generation of the next pattern. An external system 'tosses the coin' and selects the next pattern. Our model does not need such a system: the dynamics of the network itself generates the next pattern of activity. The transition probabilities do not need to be read out, they are embedded in the synaptic matrix of the network. In a sense, the network dynamics reads out these information and produces the transitions.

Inter-stimulus time Probably, the most significative difference lies in the sensitivity to inter-stimulus times. Given a sequence of patterns, our model will produce the same results independently of the time interval between subsequent patterns. This is possible because of the buffer

network, whose fast learning synapses allow it to keep a memory of the last pattern of activity of the encoding network. The buffer is a Hopfield network, with a Hebbian learning rule ($q_+ = 1$, $q_- = q_\times = 0$, see Section 2.2.2) and the Glauber's dynamics with a very low temperature (high β) for the neuron update. The old memory in the buffer is updated every time a new pattern is presented to the network. The presentation of a new pattern does not only trigger the update of the memory of the buffer, but also the generation of a new pattern of activity by the encoding network.
Jaeger's reservoir consists of a set of integrate-and-fire neurons with different time constants, that are randomly connected to each other. The activity produced by an input pattern reverberates in the reservoir. These reverberations are a function of a part of the pattern sequence. The patterns influencing the reverberations of the reservoir belong to a window in time, whose width depends, in some unknown way, on the number of neurons composing the network, on their connectivity, and on their time constants. The pattern of activity of the reservoir depends on the time intervals between subsequent patterns. In this way, two identical sequences of patterns that only differ in the inter-spike intervals, will produce two completely different activity patterns reverberating in the reservoir. In order to generalize to any inter-spike interval, the readout should be trained with all possible combinations of intervals (the number would depend on the temporal resolution of the readout), which is, of course, impractical. On the other hand, such a behavior might be desired in a task that requires the recognition of the patterns *and* the inter-spike intervals. It must be noted that these considerations are only valid for a recognition task. When the patterns sequences are generated, both networks ignore the information about the inter-stimulus intervals. This information is not stored in the synaptic matrix of our network, or in the reservoir of Jaeger's network, which has fixed synaptic weights.

Our network uses a learning rule that is local in time and space. This makes it suitable for the modeling of cortical circuits and their functionalities. The ability to ignore the inter-spike intervals is consistent with the generalization abilities we normally experience. Brains can learn, recognize, and generate stimuli sequences (e.g., sequences of images) even if the temporal scale is changed. Likewise, our network ignores any information regarding the inter-stimulus intervals. Jaeger's network is less suitable for the modeling of the brain functionalities, but it can be employed in applications that require temporal precision. The functionalities of the buffer network and the reservoir almost completely overlap, being the most notable difference the use we make of stable attractors to store the memory of the previous pattern of activity of the network. This difference accounts for the different ways of dealing with time intervals during the recognition task. The generation of patterns is accomplished in a more elegant way by our implementation, wtich does not need an external system performing a random draw to choose the next pattern. Again, this detail makes our network a more plausible model of the processing of temporal sequences in natural neuronal circuits.

6.2 Outlook

A recent paper by Orlov et al. [60] suggests a possible extension of our model. In a task involving recalling the order in which some images had been seen, the Macaque monkeys' most common error was touching the distractor image (1 distractor out of 4 images) when it had the same ordinal position as the correct image. This finding suggests that monkeys associate images to the ordinal number of appearance in a sequence. The neural mechanisms underlying this phenomenon are not yet clear. A possible explanation involves the independent activation of a population of neurons selective to the ordinal position of the image presented. The learning rule would associate in the same attractor this population of neurons to the population active upon the presentation of the image. After learning is completed, the presentation of an image would then elicit the activity of the neurons selective for that image, which would then provoke a partial activation of the neurons selective to the next image *and* those selective to the ordinal position of the next image. The neurons selective to an ordinal position are in common with all attractors related to images occurred in the same ordinal position. This might explain why, when recalling the order of a given set of images, monkeys often mistake an image with another image that occurred, during the training, in the same ordinal position. Work on this hypothesis might represent a way to extend the model we developed.

Along the same line, the same mechanism described above could be extended to associate neurons selective to external stimuli with neurons representing 'internal states'. In this way, our model could find its way to become a sort of 'behavioral engine', where the sequence of internal state (metastable attractors) is both influenced by external stimuli (sensorial inputs) and internal states. Examples of internal states are the level of glucose in the blood, the need for sleep, or the battery charge level. Such states can provide stimuli to the network just like external stimuli. For example, the stimulus corresponding to some kind of food might be associated to an increased level of glucose. Seeing the same kind of food again and again would create a strong association between it and its nutritional properties, which could accordingly modify the behavior of an agent. The behavior could be implemented by a system recognizing the state of the network, and linking it to behavioral or motor patterns.

A weak external input was shown to influence the transition probabilities. This suggests a way to create 'modalities' in the dynamics of the network. A modality would correspond to a weak input to a subset of the neurons. The transition probabilities would in this way be altered to provide a specialized behavioral response to external stimuli and internal states, such as, for example, emotions (happiness, fear, etc.).

A major improvements would come from mapping the whole network to a network of integrate-and-fire neurons. As it has been explained in Section 2.2, the variability of the interspike intervals provide for the stochasticity needed by the learning process, without the need to invoke an external mechanism. Integrate-and-fire neurons and binary synapses represent an excellent choice for aVLSI implementation [61]. This technology is leading us toward the design of very small, lightweight, and low-power-consumption devices capable of real-time computation. The principles for the construction of such devices are inspired by brain function. The fact brains show a performance not yet attained by any artificial device, justifies these efforts both from the point of view of Science and technological advancement.

Appendix A

Code

All simulation were written in Java, run using the IBM Java2 virtual machine version 1.3 for Linux on x86 processors, and Matlab[1]. We made extensive use of the java scripting language *Pnuts*[2]. We also used the Open Source Libraries for High Performance Scientific and Technical Computing in Java (COLT)[3]. Data were usually analyzed using Matlab. Figures where produced using the plotting package PHYSICA[4] and Matlab.

A.1 Variables

This is a list of some variables used in the following sections. Several variables have been omitted, whose name is usually self explanatory.

`beta` float; pseudo-temperature of the network.

`externalInput` 1D float array; array of the external synaptic inputs to the neurons.

`fields` 1D float vector; vector of synaptic input to the neurons of the network.

`globalActivity` float; fraction of active neurons of the network.

`inhibition` float; dynamically adjusted inhibition to the neurons.

`minInhibition` float; minimum inhibition that the netowrk is allowed to have.

[1] *The MathWorks, Inc.*

[2] http://javacenter.sun.co.jp/pnuts/index.html

[3] http://tilde-hoschek.home.cern.ch/ hoschek/colt/index.htm

[4] http://www.triumf.ca/people/chuma/physica/homepage.html

`networkState` 1D byte vector; vector of the activities of all neurons. Elements can be either 1 or 0.

`numberOfNeurons` integer; number of neurons conmposing the network.

`weightMatrix` 2D byte matrix; synaptic matrix of the network.

A.2 Field

Method computing the fields (synaptic inputs) to the neurons of the network.

```
/**
 * Return fields for the actual network state. External field
 * is not considered.
 * @return float[] Fields to the neurons
 */
  public float[] getFields() {
    float[][] weightMatrix = network.getWeightMatrix();
    byte[] networkState = network.getNetworkState();
    float[] fields = new float[numberOfNodes];
    for (int i = 0; i < numberOfNodes; i++) {
        for (int j = 0; j < numberOfNodes; j++) {
            fields[i] += weightMatrix[i][j] * networkState[j];
        }
        fields[i] /= numberOfNodes;
    }
    return fields;
}
```

A.3 Glauber dynamics

Method updating all neurons in random order. The order is randomized each time the method is called. The method `adaptInhibition` is described in Section A.4.

```
/*
 * Update all neurons of the network in random order.
 * The order is randomized each time this method is called.
 * ial is a cern.colt.list.IntArrayList. It must be initialized
 * with the set of integers 1...N, where N is the number
 * of neurons. Its method shuffle() makes a random
 * permutation of the list of integers with which it was initialized.
 */
    public void evolve() {
        float field; // Synaptic input
        ial.shuffle(); // Randomize neurons update order
        int randomNeuron;
        for (int i = 0; i < numberOfNeurons; i++) {
            randomNeuron = ial.get(i);
            field = 0.0f; // reset synaptic input
            for (int j = 0; j < numberOfNeurons; j++) {
                field += weightMatrix[randomNeuron][j] *
                 networkState[j];
            }
            field /= numberOfNeurons;
            field += externalInput[randomNeuron];
            // Glauber dynamics
            float prob = (float) (1.0 / (1.0 + Math.exp(-2.0 *
             (field - inhibition) * beta)));
            networkState[randomNeuron] = (mt.nextFloat() < prob) ?
             (byte) 1 : (byte) 0;
            // Call the method implementing the dynamic inhibition
            adaptInhibitionToActivity();
        }
    }
```

A.4 Dynamic inhibition

Method adapting the ihhibition to all neurons of the network to the level of activity of the network.

```
/**
 * Set inhibition to a value dependent on the global
 * network activity.
 * Inhibition I has the form
 * I=slope*(activity-intercept) where the intercept is the
 * intersection point of the inhibition curve with the I=0 axis.
 * ‘activity’ and ‘intercept’ correspond, in the text, to
 * ‘s_0’ and ‘s_1’ in the text.
 */
public void setInhibition() {
    float globalActivity = 0.0f;
    // Compute activity of the network
    for (int i = 0; i < numberOfNodes; i++) {
        globalActivity += networkState[i];
    }
    globalActivity /= numberOfNodes;
    float targetInhibition = slope * (globalActivity - intercept);
    float newInhibition = inhibition+
      0.02f*(targetInhibition-inhibition); // 0.02
    if (newInhibition >= minInhibition)
        setInhibition(newInhibition);
    /** Avoid a too low inhibition: if the network activity
     * is too low, then it should go to zero activity.
     * This can happen when the network does cannot
     * recognize a pattern.
     */
     else setInhibition(minInhibition);
}
```

A.5 Overlaps

Method returning the overlaps of the present network state with all the patterns given in the rows of the input matrix.

```
/**
 * Return an array containing the overlaps of the network
 * state with the given patterns.
 * @return float[] Overlaps
 * @param patterns byte[][] Patterns (in the rows)
 */
public float[] observeOverlaps(byte[][] patterns) {
    int numberOfPatterns = patterns.length;
    float[] overlaps = new float[numberOfPatterns];
    for (int m = 0; m < numberOfPatterns; m++) {
        overlaps[m] = observeOverlap(patterns[m]);
    }
    return overlaps;
}
```

A.6 Pattern with maximum overlap

Method returning the index of the pattern having maximum overlap with the present network state. Patterns are provided as rows of the argument matrix.

```
/**
 * Return index of the pattern whose overlap with the network
 * state is maximal.
 * @return int Pattern number
 * @param patterns Patterns to be used in calculating overlaps
 */
public int patternWithMaxOverlap(byte[][] patterns) {
    int numberOfPatterns = patterns.length;
    float overlap;
    int winningPattern = -1; // lower than any possible value
    float maxOverlap = -1;   // lower than any possible value
    for (int m = 0; m < numberOfPatterns; m++) {
        overlap = 0.0f;
        for (int i = 0; i < numberOfNodes; i++) {
            overlap += patterns[m][i] * networkState[i];
        }
        overlap /= numberOfNodes;
        if (overlap > maxOverlap) {
            maxOverlap = overlap;
            winningPattern = m;
        }
    }
    return winningPattern;
}
```

A.7 Compute transition probabilities

Method computing the transition probabilities for a network, usually after the training phase is finished.
Note: This method describes a way to compute the transition probability matrix for a network. The network state is never reset during the computation. In some cases (see Section 3.1.3), to uniformly sample all transition probabilities, it is better to set the network state to a new starting state after a transition has occured.

```
/**
 * Compute the transition probabilites. All neurons are updated
 * in random order once; after that, the index of the pattern
 * having maximum overlap with the network state is calculated.
 * The corresponding element of the transition probability matrix
 * is incremented by 1. At the end, the matrix is normalized so
 * that the cumulative sum of all rows equals 1.
 * The whole process is repeated a given number of times.
 * @return ale.nn.FloatMatrix
 * @param timesteps int Number of times the network update
 * has to be done
 */
public FloatMatrix computeTransitionProbabilities(int timesteps) {
  int winningPattern = network.patternWithMaxOverlap(patterns);
  int oldWinningPattern = winningPattern;
  float[][] transitionProbabilityMatrix =
   new float[numberOfPatterns][numberOfPatterns];
  for (int t = 0; t < timesteps; t++) {
      network.evolve(); // update all neurons once
      oldWinningPattern = winningPattern;
      winningPattern = network.patternWithMaxOverlap(patterns);
      if (winningPattern != oldWinningPattern) {
      // a transition occured
      transitionProbabilityMatrix[oldWinningPattern]
       [winningPattern] += 1.0f;
      } else {
      // the network is still sitting in the same attractor
     transitionProbabilityMatrix[winningPattern]
      [winningPattern] += 1.0f;
      }
  }
  // normalization of the transitionProbabilityMatrix
  float[][] normalizedTransitionProbabilityMatrix =
  new float[numberOfPatterns][numberOfPatterns];
  for (int irow = 0; irow < numberOfPatterns; irow++) {
    float sum = 0.0f;
    for (int icolumn = 0; icolumn < numberOfPatterns; icolumn++)
      sum += transitionProbabilityMatrix[irow][icolumn];
      if (sum != 0.0f) {
        float normalisation = 1.0f/sum;
        for (int icolumn = 0; icolumn < numberOfPatterns; icolumn++)
```

```
          normalizedTransitionProbabilityMatrix[irow][icolumn] =
          transitionProbabilityMatrix[irow][icolumn] * normalisation;
       } else {
          for (int icolumn = 0; icolumn < numberOfPatterns; icolumn++)
          normalizedTransitionProbabilityMatrix[irow][icolumn] = 0.0f;
       }
   }
   return new FloatMatrix(normalizedTransitionProbabilityMatrix);
}
```

A.8 Online learning

Method modifying the synaptic matrix using a Hebbian-like learning rule.

```
/**
 * Stochastically modify the synaptic weights using a
 * Hebbian-like learning rule.
 * @param previousNetworkState byte[] Previous state of
 * the network. The learning rule algorithms can potentiate
 * the synapses connecting neurons active in this vector
 * and neurons which are active in the present state of
 * the network.
 * @param probability float Probability of potentiating a
 * synapse connecting two active neurons
 * @param weightMatrix Weight matrix
 * @param activity float Mean global activity of the network
 * @param lambdas Lambda_f and lambda_b
 */
public stochasticallyUpdateWeightMatrix(
  byte[] previousNetworkState, float probability, float activity,
  float[][] weightMatrix, float[] lambdas) {
  /** The probability of potentiating a synapses between
   *   2 active neurons is q11*activity^2
   */
   float q11 = probability;
   /**
    * the probability of depressing a synapses connecting
    * 2 neurons with different activity is
    * 2f(1-f)*probability_of_depression (f is the activity of
    * the network).
    * The probabilty of depression must be chosen to
    * equilibrate the number of potentiations and depressions.
    */
    float q10 = (activity * q11) / (2 * (1 - activity));
    byte[] networkState = network.getNetworkState();
    int i, j;
    for (i = 0; i < numberOfNodes; i++) {
      for (j = 0; j< numberOfNodes; j++) {

        if (lambdas[0]!=0 && previousNetworkState[i] *
         networkState[j] >= 1 && weightMatrix[j][i] == 0 &&
          i != j) {
          if (mt.nextFloat() < (q11 * lambdas[0])) {
           weightMatrix[j][i] = 1;
          }
        }

        if (lambdas[1]!=0 && previousNetworkState[j] *
         networkState[i] >= 1 && weightMatrix[j][i] == 0
          && i != j) {
          if (mt.nextFloat() < (q11 * lambdas[1])) {
           weightMatrix[j][i] = 1;
           }
         }

        if (weightMatrix[j][i] == 0 && networkState[i] *
         networkState[j] >= 1 && i != j) {
           if (mt.nextFloat() < q11) {
          weightMatrix[j][i] = 1;
           }
         } else if (weightMatrix[j][i] >= 1 &&
            networkState[i] == 1
          && networkState[j] == 0 &&  i != j) {
           if (mt.nextFloat() < q10) {
            weightMatrix[j][i] = 0;
           }
         }
      }
   }
}
```

A.9 Fast learning

Method teaching the network a Markov process by computing the probability for each synapse to be potentiated or depressed, without having to present the network a pattern sequence generated by the markov process.

```
/* Create weight matrix calculating the theorical probability
 * for each synapse to be potentiated or depressed.
 * @param patterns byte[][] Patterns to be taught in the rows.
 * @param probability float Probability of potentiating a
 * synapse connecting two active neurons
 * @param lambda float[][] Lambda_f and lambda_b
 * @param linkMatrix float[][] Matrix with links between patterns
 * (Markov matrix)
 * @param activity float Activity of the network
 */
public void stochasticallyComputeWeightMatrix(byte[][] patterns,
 float probability, float[] lambda, float[][] linkMatrix, float
activity) {
  /** The probability of potentiating a synapses between
   * 2 active neurons is q11*activity^2
   */
  float q11 = probability;
  /**
   * The probability q10 of depressing a synapses connecting 2 neurons
   * with different activity is 2f(1-f)*probability_of_depression
   * (f is the activity of the network).
   * The probabilty of depression must be chosen to equilibrate
   * the number of potentiations and depressions.
   */
   float q10 = 2.0f*(activity * q11) / (2 * (1 - activity));
   float p; // probability for a synapse to be 1
   float q; // probability for a synapse to be 0
   for (int irow=0; irow<patterns[0].length; irow++) {
     for (int icolumn=0; icolumn<patterns[0].length; icolumn++) {
       p = 0.0f;
       q = 0.0f;
       for (int mu=0; mu<patterns.length; mu++) {
         // compute p
         p += patterns[mu][irow]*patterns[mu][icolumn]*
          q11*frequencies[mu];
```

```
        for (int nu=0; nu<patterns.length; nu++) {
          if (nu!=mu) {
            p += patterns[mu][irow]*patterns[nu][icolumn]*
             linkMatrix[nu][mu]*lambda[0]*frequencies[mu]*q11;
            p += patterns[nu][irow]*patterns[mu][icolumn]*
             linkMatrix[nu][mu]*lambda[1]*frequencies[nu]*q11;
          }
        }
        // compute q
        q += (1-patterns[mu][irow])*patterns[mu][icolumn]*
         q10*frequencies[mu];
        q += (1-patterns[mu][icolumn])*patterns[mu][irow]*
         q10*frequencies[mu];
       }
      weightMatrix[irow][icolumn] =
        (mt.nextFloat()<(p/(p+q)))? 1 : 0;
     }
   }
}
```

A.10 Next state of a Markov chain

Method returning the next state of a Markov chain. It is used to generated pattern sequences.

```
/* Return the next state in a Markov chain
 * @param m float[][] Markov matrix (sums of columns are
 * normalized to 1)
 * @param start int Starting state
 *
 * mt: random number generator
 */
public int getNextState(float[][] m, int start) {
    float f = mt.nextFloat();
    int index = 0;
    float sum = 0.0f;
    for (int i =0; i<mm[start].length; i++) {
        sum += mm[start][i];
        if (f<sum) {
            index = i;
                break;
        }
    }
    return index;
}
```

A.11 Measurement of the transition probabilities in the non-Markov case

This method computes the transition probability matrix from a sequence of integers. Integers correspond to the pattern with which the network had maximum overlap when overlaps were measured. This method is used when measuring the performance of the network in learning sequences generated by a non-Markov process. In the case of Markov processes, the code in Section A.7 is usually used.

```
/** Return the Markov matrix extracted from the sequence . The
 * transition probabilities calculated are between the states of
 * subsequences of states in the 2D integer matrix 'states'.
 * @param sequence int[] Sequence (pattern sequence)
 * @return float[][] Transition probability matrix
 */
public float[][] sequenceToMatrix(int[] sequence) {
  public static int[][] states =
  {{0},
   {1},
   {2},
   {3},
   {0,1},
   {3,1}};
  // tp: transition probabilities
  float[][] tp = new float[states.length][states.length];
   int[] fromState;
   int[] toState;
   int fromStateLength;
   int toStateLength;
   for (int i=0; i<states.length; i++) {
     fromStateLength = states[i].length;
     fromState = new int[fromStateLength];
     for (int j=0; j<states.length; j++) {
       toStateLength = states[j].length;
       toState = new int[toStateLength];
       for (int k=0; k<sequence.length; k++) {
         if ((k+fromStateLength+toStateLength)>
          sequence.length) break;
         for (int ifrom=0; ifrom<fromStateLength; ifrom++) {
           fromState[ifrom] = sequence[k+ifrom];
          }
          for (int ito=0; ito<toStateLength; ito++) {
            toState[ito] = sequence[k+fromStateLength+ito];
          }
          // the helper function match can be found at
          // the end of this section
          if (match(fromState, states[i]) &&
           match(toState, states[j])) {
            tp[i][j]++;
          }
       }
     }
   }
   /* Normalization */
   float sum;
   for (int i=0; i<tp.length; i++) {
     sum = 0.0f;
       for (int j=0; j<tp[i].length; j++) {
         sum += tp[i][j];
       }
       for (int j=0; j<tp[i].length; j++) {
         tp[i][j] /= sum;
       }
   }
   return tp;
}
/* Helper function: return true if the input
 * arrays are the same */
public boolean match(int[] a, int[] b) {
  boolean output = true;
    for (int i=0; i<a.length; i++) {
      output = output && (a[i] == b[i]);
    }
    return output;
}
```

A.ACb11eMeasurement of the transition probabilities in the non-Markov case

A.12 Encoding network

```
clear; rehash
activity = 0.1;
% Number of neurons
N = 500;
% Number of active neurons
fN = activity*N;
% Total number of timesteps
totalT = 3000;
R = randperm(N);
% Weight matrix of the buffer network
wm = zeros(N);
inhibition = 0.05; % starting inhibition of the buffer
% Total number of timesteps The number of timesteps during which an
% external input is present (200) is 1/2 of the time constant of the
% synapses between the buffer network and the encoding network. This
% to avoid the encoding layer to see the changing state of the buffer
% before all neurons are updated in the encoding layer.
inputT = [[1,200];[1000,1200];[2000,2200]];
% Input over threshold
inputOver = zeros(3,N);
% Prepare input over threshold
for k = 1:size(inputOver,1)
  inputOver(k,fN/2*(k-1)+1:fN/2*k) = 3;
end
inputSub = zeros(3,N); % Input below threshold
% Prepare input over threshold
for k = 1:size(inputSub,1)
  r = randperm(size(inputSub,2));
  inputSub(k,r(1:size(inputSub,2)/2)) = 1.5;
end
% Prepare input to the encoding network
input = zeros(totalT,N);
for k = 1:size(input,1)
  for p = 1:size(inputT,1)
    if (k>=inputT(p,1) & k<=inputT(p,2))
      input(k,:) = input(k,:)+inputOver(p,:);
      input(k,:) = input(k,:)+inputSub(p,:);
    end
  end
```

```
end
% State of buffer network
buffer = zeros(1,N);
r = randperm(size(buffer,2));
% Initializes network
buffer(1,r(1:fN)) = 1;
% Actual input from buffer network to
% the encoding network
feltBuffer = buffer;
% Save initialization state of the buffer
startBuffer = buffer;
% Time constant of the neurons of the
% encoding layer
ty = 40; % tau_e
% Time constant of the synapses between
% the buffer network and the encoding layer
tz = 400; % tau_b
% State of the encoding layer
state = zeros(1,N);
% Helper variables for monitor output
output = zeros(size(input));
output2 = zeros(size(input));
outputi = zeros(1,size(input,1));

% Main loop
for t=1:size(input,1)
  output(t,:) = state; % Log
  output2(t,:) = feltBuffer;% Log
  outputi(t) = inhibition;% Log

  feltBuffer = feltBuffer+(1/tz)*(buffer-feltBuffer);
  % Field (synaptic input) to the encoding layer
  fields = input(t,:)+feltBuffer(R)*1.5;
  % R is a random permutation
  % active neurons
  f = fields>2;
  state = state+(1/ty)*(f-state);
  % Buffer receives input from encoding layer only if state>0.9
  [buffer,wm,inhibition] = evolveBuffer(buffer,state>0.9
      ,wm,inhibition);
end
```

```
% Plot data

subplot(5,1,1); imagesc(output)
title('State dynamical system')
subplot(5,1,2); imagesc(output2)
title('Buffer dynamical system')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Repeat everything using an algorithm and not
% difference equations

state = zeros(1,N);
buffer = startBuffer;
output3 = zeros(size(inputOver,1),N);
output4 = zeros(size(inputOver,1),N);

for k=1:size(inputOver,1)
  fields = inputOver(k,:)+inputSub(k,:)+buffer(R)*1.5;
  state = fields>2;
  buffer = state;
  output3(k,:) = state;
end

subplot(5,1,3); imagesc(output3)
title('State algorithm')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Compares the two methods: if output5-output3 are all zeros,
% the two methods are equivalent

output5 = zeros(3,N);
output5(1,:) = output(199,:)>0.8;
output5(2,:) = output(1199,:)>0.8;
output5(3,:) = output(2199,:)>0.8;
subplot(5,1,1); imagesc(output5)
title('State dynamical system: detail')

subplot(5,1,5); imagesc(output5-output3)  % deve essere zero
```

```
title('Difference dynamical system - algorithm')
```

A.13 Buffer network

```
function [buffer,wm,inhibition] =
  evolveBuffer(presentBuffer,input,wm,inhibition)
  state = presentBuffer;
  N = length(presentBuffer); % Number of neurons
  for t=1:3
    wm = state'*state; % one-shot learning
    fields = (wm*state')'/N+input; % recurrent+external inut
    state = fields-inhibition>0; % Heaviside function
    % very simple dynamic inhibition
    activity = sum(state)/N;
    if (activity>0.15)
      inhibition = 0.5;
    else
      inhibition = 0.05;
    end
  end
  buffer = state;
```

Index

Glossary

A

activity Fraction of active neurons in a set of neurons. Activity is a real number between 0 and 1.

activity pattern Configuration of the activities of the neurons composing a network at any point in time. Neurons can be either active or quiescent (non-active).

attractor Sets to which all nearby trajectories in the variable space converge. A system which is moved away from one of its attractors will tend to go back to it. For a neural network, the variable space is the space of the activity patterns.

aVLSI Analog Very Large-Scale Integration: VLSI is the current level of computer microchip miniaturization; Analog is meant as 'continually changing', as opposed to the binary behavior of digital computing.

G

global inhibition Inhibition provided to all neurons of a network. It is the same for all network neurons and is used to limit the activity of a network.

I

inhibition Negative synaptic input to a neuron. It is usually provided by a single or a population of inhibitory neurons.

input pattern Configuration of the synaptic input provided to the neurons of a network.

L

learning phase A part in the process of the employment of a network distinguished from the part in which the network is used to generate pattern sequences. During this part, stimuli sequences elicit in a network patterns of activity that are used, in combination with a learning rule, to modify the synaptic matrix.

N

network state See also **activity pattern**. Used as synonymous of 'activity pattern of the network'.

P

palimpsest Property exhibited by neural networks for which old stimuli are forgotten to make room for the most recent ones. The palimpsest property depends from the learning rule used.

S

spatiotemporal patterns Activity pattern that changes in time. Also used for a sequence of activity patterns.

state Configuration of the activity of a set of neurons or network.

state of the network See **network state**.

stimulus An agent that influences the activity of a set of neurons. Normally, stimuli are completely identified with the pattern of activity they elicit in the network or in the input layer.

stochastic synapses Synapse whose state (potentiated or depressed) is a stochastic function of the activity of the pre- and postsynaptic neuron. This means that, given the activity of the pre- and postsynaptic neurons, only a probability can be assigned to the state of synapse.

sub-threshold Said of a synaptic input to a neuron not capable of making a neuron go to an active state.

super-threshold Said of a synaptic input to a neuron capable of making a neuron go to an active state.

synaptic matrix Bidimensional matrix containing the weights of the synaptic links between any two neurons of a neural network. The columns of the matrix represent the axons of the neurons, the rows the dendrites. FOr example, the weight at row x and column y is the weight connecting neuron y to neuron x.

synaptic strengths See synaptic matrix.

T

transition probabilities Probability for the passage of a network state from an activity configuration of its neurons that has maximum overlap with a pattern, to another configuration that has maximum overlap with another pattern.

W

weight matrix See synaptic matrix.

Bibliography

[1] H. Sompolinsky and I. Kanter. Temporal association in asymmetric neural netoworks. *Physical Review Letters*, 57(22):2861–2864, 1986.

[2] J. Buhmann and K. Schulten. Noise-driven temporal association in neural networks. *Europhysics Letters*, 4(10):1205–1209, 1987.

[3] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[4] J. J. Hopfield and C. D. Brody. What is a moment? transient synchrony as a collective mechanism for spatiotemporal integration. *Proceedings of the National Academy of Sciences, USA*, 98(3):1282–1287, January 2001.

[5] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD 148, Fraunhofer Institute for Autonomous Intelligent Systems, December 2001.

[6] R. Heath. Can people predict chaotic sequences? *Nonlinear Dynamics, Psychology, & Life Sciences*, 6(1), January 2002.

[7] M. Griniasty, M. V. Tsodyks, and D. J. Amit. Conversion of temporal correlations between stimuli to spatial correlations between attractors. *Neural Computation*, 5:1–17, 1993.

[8] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 1982.

[9] Y. Miyashita and H. S. Chang. Neuronal correlate of pictorial short-term memory in the primate temporal cortex. *Nature*, 331:68–70, January 1988.

[10] Y. Miyashita. Neuronal correlate of visual associative long-term memory in the primate temporal cortex. *Nature*, 335(6193):817–820, October 1988.

[11] V. Yakovlev, S. Fusi, E. Berman, and E. Zohary. Inter-trial neuronal activity in inferior temporal cortex: a putative vehicle to generate long-term visual associations. *Nature neuroscience*, 1(4):310–317, August 1998.

[12] D. J. Amit and M. V. Tsodyks. Quantitative study of attractor neural network retrieving at low spike rates i: Substrate-spikes, rates and neuronal gain. *Network: Computation in neural systems*, 2:259–273, 1991.

[13] D. J. Amit and M. V. Tsodyks. Quantitative study of attractor neural network retrieving at low spike rates ii: Low-rate retrieval in symmetric networks. *Network: Computation in neural systems*, 2:275–294, 1991.

[14] M. S. Bartlett and T. J. Sejnowski. Learning viewpoint invariant face representations from visual experience in an attractor network. *Network: Computation in neural systems*, 9(3):399–417, 1998.

[15] D. R. Cox and H. D. Miller. *The theory of stochastic processes.* Chapman & Hall, 1967.

[16] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation.* Santa Fe Institute Studies in the Sciences of Complexity. Perseus Books, 1991.

[17] D. Kleinfeld. Sequential state generation by model neural networks. *Proceedings of the National Academy of Sciences, USA*, 83(24):9469–9473, December 1986.

[18] P. Peretto and J. J. Niez. Collective properties of neuronal networks. In E. Bienenstock and F. Fogelman-Soulie, editors, *Disordered systems and biological organisation.* Springer-Verlag, 1986.

[19] I. Nebenzahl. Recall of associated memories. *Journal of Mathematical Biology*, 25:511–519, 1987.

[20] S. Dehaene, J. P. Changeux, and J. P. Nadal. Neural networks that learn temporal sequences by selection. *Proceedings of the National Academy of Sciences, USA*, 84:2727–2731, 1987.

[21] H. Nishimori and T. Nakamura. Retrieval of spatio-temporal sequence in asynchronous neural network. *Physical Review A*, 41(6):3346–3354, March 1990.

[22] H. Gutfreund and M. Mezard. Processing of temporal sequences in neural networks. *Physical Review Letters*, 61(2):235–238, 1988.

[23] I. Guyon, L. Personnaz, J. P. Nadal, and G. Dreyfus. Storage and retrieval of complex sequences in neural networks. *Physical Review A*, 38(12):6365–6372, December 1988.

[24] R. Kühn, J. L. van Hemmen, and U. Riedel. Complex temporal association in neural networks. *Journal of Physics A: Math. Gen.*, 22:3123–3135, 1989.

[25] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages and computation.* Addison Wesley, 1979.

[26] K. S. Fu. *Syntactic patterns recognition and applications.* Prentice-Hall, 1982.

[27] J. L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–235, 1991.

[28] J. B. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7:227–252, 1991.

[29] C. L. Giles, C. B. Miller, D. Chen, G. Z. Sun, H. H. Chen, and Y. C. Lee. Extracting and learning an unknown grammar with recurrent neural networks. In J. E. Moody, S.J. Hanson, and R.P Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 317–324, 1992.

[30] C. L. Giles and C. W. Omlin. Learning, representation, and synthesis of discrete dynamical systems in continuous recurrent neural networks. In *Proceedings of the IEEE Workshop on Architectures for Semiotic Modeling and Situation Analysis in Large Complex Systems*, 1995.

[31] S. Lawrence, S. Fong, and C. Lee. "natural language grammatical inference: A comparison of recurrent neural networks and machine learning methods. In S. Wermter, E. Riloff, and G. Scheler, editors, *Symbolic, Connectionist, and Statistical Approaches to Learning for Natural Language Processing*, pages 33–47. Springer Verlag, 1996.

[32] S. Lawrence, C. L.Giles, and S. Fong. Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):126–140, 2000.

[33] R. F. Hadley and V. C. Cardei. Language acquisition from sparse input without error feedback. *Neural Networks*, 12:217–235, 1999.

[34] S. Das, C. L. Giles, and G. Z. Sun. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *The Fourteenth Annual Conference of the Cognitive Science Society*, pages 791–795. Morgan Kauffman, 1992.

[35] P. Rodriguez, J. Wiles, and J L. Elman. A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40, 1999.

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

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

[38] S. C. Kremer. Spatio-temporal connectionist networks: A taxonomy and review. *Neural Computation*, 13(2):249–306, February 2001.

[39] T. Natschläger, W.ăMaass, and A.ăZador. Efficient temporal processing with biologically realistic dynamic synapses. *Network: Computation in Neural Systems*, 12:75–87, 2001.

[40] T.ăNatschläger and B.ăRuf. Spatial and temporal pattern analysis via spiking neurons. *Network: Computation in Neural Systems*, 9(3):319–332, 1998.

[41] W.ăMaass, T.ăNatschläger, , and H.ăMarkram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[42] H. Jaeger. Beautiful beasts: Recurrent neural networks. Colloquium of the Institute of Neuroinformatics Uni/ETH Zurich, 2002. http://www.ais.fraunhofer.de/INDY.

[43] Z. Ghahramani. An introduction to hidden markov models and bayesian networks. *International Journal of Pattern Recognition and Artificial INtelligence*, 15(1):9–42, 2001.

[44] Y. Bengio. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162, 1999.

[45] D. Kleinfeld and H. Sompolinsky. Associative neural network model for the generation of temporal patterns. *Biophysical Journal*, 54:1039–1051, 1988.

[46] N. Brunel. Hebbian learning of context in recurrent neural networks. *Neural Computation*, 8:1677–1710, 1996.

[47] D. J. Amit. *Modeling brain function.* Cambridge University Press, New York, 1989.

[48] N. Fedoroff and W. Fontana. Small numbers of small molecules. *Science*, 297:1129–1131, August 2002.

[49] S. Fusi. Hebbian spike-driven synaptic plasticity for learning patterns of mean firing rates. *Biological Cybernetics*, 82:459–470, 2002.

[50] C. C. Petersen, R. C. Malenka, R. A. Nicoll, and J. J. Hopfield. All-or-none potentiation at ca3-ca1 synapses. *Proceedings of the National Academy of Sciences, USA*, 95(8):4732–4737, April 1998.

[51] W. R. Softy and C. Koch. The highly irregular firing of cortical cells is inconsistent with temporal intergation of random epsps. *Journal of neuroscience*, 1:334–350, 1993.

[52] P. L. Meyer. *Introductory probability and statistical applications.* Addison-Wesley, Reading, MA, 1965.

[53] D. J. Amit and S. Fusi. Learning in neural networks with material synapses. *Neural Computation*, 6:957–982, 1994.

[54] N. Brunel, F. Carusi, and S. Fusi. Slow stochastic Hebbian learning of classes of stimuli in a recurrent neural network. *Network*, 9:123–152, 1998.

[55] Y. Miyashita. Inferior temporal cortex: where visual perception meets memory. *Annual Review of Neurosciences*, 16:245–263, 1993.

[56] H. Jaeger. Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology, 2002.

[57] D. R. Hofstadter and D. C. Dennet. *The mind's I: Fantasies and reflections on self and soul.* Basic Book, Inc., New York, 1981.

[58] Wikipedia. http://www.wikipedia.com/.

[59] Gamers.com. http://www.gamers.com/game/75123/.

[60] T. Orlov, V. Yakovlev, D. Amit, S. Hochstein, and E. Zohary. Serial memory strategies in macaque monkeys: Behavioral and theoretical aspects. *Cerebral cortex*, 12:306–317, March 2002.

[61] S. Fusi, M. Annunziato, D. Badoni, A. Salamon, and D.J. Amit. Spike-driven synaptic plasticity: theory, simulation, vlsi implementation. *Neural Computation*, 12:2227–2258, 2000.

Curriculum Vitæ

General Information

Name: Alessandro Usseglio Viretta
Date of birth: 15 Sept. 1970
Nationality: Italian

Education

October 1998-Present PhD student at the Institute for Neuroinformatics, Center for Neurosciences, University and Institute of Technology (ETH) of Zurich. **April 1997 to September 1998** PhD student at ETH Zurich, Lab for electron microscopy, under the supervision of Prof. Peter Schurtenberger. **July 1996** Graduated *Summa cum Laude* in Physics. **April 1993 to September 1993** Student at the Ruprecht-Karls-Universität in Heidelberg (ERASMUS exchange program). **September 1989** Physics student at the University of Torino (Italy).

Experience

October 1996 to April 1997 Scientific assistant (employed by the Institut für Kernphysik-Frankfurt). Working at CERN (Geneva). **July 1995 to September 1996** Technical student at CERN. Prepared my diploma thesis working on a prototype of particle detector. **March 1995 to June 1995** Lab assistant at the University of Torino. **July 1994 to September 1994** Summer student at CERN.