UNIVERSIDADE DO ALGARVE

Faculdade de Ciências Humanas e Sociais

# Self-Organized Sequence Processing in Recurrent Neural Networks with Multiple Interacting Plasticity Mechanisms

Renato Duarte

Mestrado em Neurociências Cognitivas e Neuropsicologia

Especialização em Neuropsicologia

2011

Universidade do Algarve
Faculdade de Ciências Humanas e Sociais

# Self-organized Sequence Processing in Recurrent Neural Networks with Multiple Interacting Plasticity Mechanisms

*Autor:*
Renato Duarte

*Dissertação Orientada por:*
Prof. Dr. Karl Magnus Petersson

Mestrado em Neurociências Cognitivas e Neuropsicologia
Especialização em Neuropsicologia
2011

# Contents

# Abbreviations

| | |
|---|---|
| 5HTR | 5-Hydroxytryptamine (Serotonin) Receptor |
| AI | Artificial Intelligence |
| AMPA | $\alpha$-Amino-3-hidroxi-5-methylisoxazole-4-propionic Acid (Glutamate Receptor) |
| ANN | Artificial Neural Network |
| APRL | Atiya-Parlos Recurrent Learning |
| BP | BackPropagation |
| BPDC | BackPropagation Decorrelation |
| BPTT | BackPropagation Through Time |
| CaMKII | $Ca^{2+}$/Calmodulin-dependent Protein Kinase II |
| EPSC | Excitatory Post-Synaptic Current |
| FFN | Feed-Forward Network |
| IP | Intrinsic Plasticity |
| LTD | Long-Term Depression |
| LTP | Long-Term Potentiation |
| mAChR | Metabotropic Acetilcholine Receptor |
| MAPK | Mitogen-activated Protein Kinase |
| mGluR | Metabotropic Glutamate Receptor |
| MLP | Multi-Layer Perceptron |
| NMDA | N-Methil-D-Aspartate (Glutamate Receptor) |
| nNOS | Neuronal Nitric Oxide Synthase |

| | |
|---|---|
| PK | Protein Kinase (A, C and G) |
| RC | Reservoir Computing |
| RNN | Recurrent Neural Network |
| RTRL | Real-Time Recurrent Learning |
| SN | Synaptic Normalization |
| SOM | Self-Organizing Maps |
| STDP | Spike-Timing-Dependent Plasticity |
| VGIC | Voltage-Gated Ion Channel |

# List of Figures

# Abstract

The highly recurrent connectivity encountered in the neocortical circuitry makes recurrent neural network (RNN) models highly suitable when investigating the computational properties of biologically inspired model neurodynamics. The recent reservoir computing (RC) models, an extension of the RNN paradigm, provide a framework for state-dependent computations, where information is encoded in the form of state-space trajectories, which is similar to recent findings in neurobiology. Over the past few years, several attempts have been made to endow these network models with adaptive mechanisms, capable of mimicking the various neural plasticity mechanisms known to exist in the brain and to play a fundamental role in shaping the dynamics and information processing capabilities of the underlying neural networks.

In this thesis, we analyze the dynamic properties of a simple reservoir computer model, with self-organizing plasticity mechanisms operating concomitantly. We investigate how different combinations of three forms of biologically inspired adaptive mechanisms shape the reservoir's dynamic properties and their effectiveness in acquiring an internal representation of structured symbol sequences. We demonstrate, replicating previous work, that only combined do these mechanisms allow the dynamic reservoir networks to achieve an input separation that outperforms static (i.e., without plasticity) reservoir networks. We further assess how the symbol sequences are internally represented in different network settings. All reservoir networks are shown to reflect the input structure in their state dynamics, but plasticity is clearly beneficial by modifying network parameters, increasing the network's ability to 'learn' the temporal structure of the input sequences.

Keywords: *Recurrent Neural Networks, Reservoir Computing, Plasticity, Time Series, Sequence Processing, Self-Organization*

# Resumo

As redes neuronais encontradas no neocórtex são dos sistemas mais complexos conhecidos tanto a nível estrutural como a nível funcional. Na generalidade, apresentam padrões de conectividade altamente recorrentes, tornando os modelos de redes artificiais recorrentes os mais apropriados para a sua simulação em estudos teóricos. Estes modelos permitem analizar as propriedades emergentes do padrão de conectividade, nomeadamente a capacidade de processar naturalmente conteúdo espacio-temporal, que se tem vindo a revelar crucial no processamento de informação no cérebro.

Dados empíricos recentes, obtidos maioritariamente nas áreas sensoriais primárias do córtex (onde o estímulo externo está prontamente acessível e é controlável), sugerem que a informação é largamente representada e processada, localmente, sob a forma de trajectórias neurais, lábeis e transientes (cuja duração depende da tarefa), evoluíndo no espaço de activação da rede neuronal, de forma reprodutível e específica para os sinais do meio que as originaram. Essas trajectórias podem depois, teoricamente, ser "lidas" por determinados neurónios cuja localização no circuito lhes permita distribuir essa informação para o próximo módulo de processamento.

Tendo em conta que existe uma certa regularidade nos circuitos neuronais e que as mesmas redes são capazes de processar informação muito diversa de forma inespecífica, há claramente princípios que regem esses circuitos, tornando-se fundamental e urgente encontrar modelos genéricos que permitam tirar partido dessa universalidade. A tremenda complexidade do sistema requer um certo grau de abstracção, embora não seja ainda claro ou consensual quais os detalhes biológicos pertinentes para a computação e quais os que podem ser descartados.

Os modelos mais comuns de redes neurais artificiais tornam-se inadequados quando o objectivo é modelar realisticamente microcircuitos neuronais genéricos. As tradicionais redes *feed-forward* constituem bons modelos para as "linhas de transmissão" do sistema nervoso, por exemplo, e as redes associativas, baseadas em *atractores* podem ser relevantes para a análise de determinados processos cognitivos que requerem maior estabilidade da in-

formação codificada (como memória a longo prazo), no entanto, são modelos insuficientes, imprecisos e incapazes de processar em tempo real, sinais não lineares contendo dependências temporais. Mais adequados são os modelos recorrentes, no entanto os custos computacionais envolvidos nos algoritmos de treino de muitos destes modelos tornam proibitivo o seu uso.

Uma abordagem recente para treinar redes neurais recorrentes de forma não supervisionada, denominada "computação em reservatório", surgiu, por um lado da área da neurobiologia (sob o nome de *liquid state machine* (LSM)) como um sistema capaz de processar em tempo real sinais não lineares, altamente variáveis e, por outro, da área da engenharia (sob o nome de *echo state network* (ESN)), pela necessidade de usar redes neurais recorrentes numa série de aplicações que envolvem processamento temporal. Esta abordagem apresenta-se actualmente como o paradigma dominante nesta área e, devido à sua simplicidade e facilidade de implementação tem atraído a atenção de diversos domínios da ciência nos últimos anos e diversos modelos, com complexidade variada e direccionados para diferentes propósitos, têm vindo a surgir como extensões do paradigma inicial.

A ideia consiste em construir um grande reservatório de neurónios artificiais (pelo menos 100) e excitá-lo passivamente com um sinal (uma sequência temporal). O reservatório funciona assim como um filtro não linear. As conexões internas do reservatório mantêm-se sem treino, sendo o treino das ligações do reservatório a um neurónio de leitura externo o único procedimento supervisionado. Este treino consiste apenas na resolução de um sistema de equações lineares, o que pode ser feito com métodos simples de regressão linear.

Para além de procurar melhorar a qualidade e estabelecer regras para a construção dos reservatórios (que são largamente construídos ao acaso, por tentativa e erro), uma das questões de mais intensa investigação neste domínio, é a criação de mecanismos de adaptação e aprendizagem não supervisionada, procurando mimetizar os mecanismos de plasticidade encontrados no cérebro.

De facto, uma das mais importantes características das redes biológicas é a sua adaptabilidade. Vários mecanismos de plasticidade operam em si-

multâneo para moldar as propriedades estruturais e funcionais dos seus componentes em resposta à actividade e experiência. Como tal, o estudo da neuroplasticidade é fulcral quer do ponto de vista da neurociência computacional, quer do ponto de vista da engenharia e inteligência artificial. Tradicionalmente, os modelos de plasticidade em redes neurais focavam-se maioritariamente na sinapse, em fenómenos como potenciação e depressão a longo prazo. Nos últimos anos têm surgido várias implementações de outros mecanismos de plasticidade, em paralelo com os avaços da neurobiologia, para treino dos reservatórios, como plasticidade intrínseca e outros mecanismos homeostáticos. No entanto, muitas tentativas não têm atingido o sucesso esperado, particularmente quando se procura modelar vários mecanismos operando em sincronia (como é sabido acontecer nas redes biológicas).

Nesta tese analisamos as propriedades dinâmicas de um modelo de reservatório previamente proposto, composto por simples neurónios cujo estado de activação é binário e/ou linear (excitatórios e inibitórios), em que a população excitatória é dotada de 3 mecanismos de plasticidade operando em simultâneo: *spike-timing-dependent plasticity* (STDP) e normalização sináptica alteram os pesos das sinapses e, simultaneamente, um mecanismo de plasticidade intrínseca (IP) faz variar os limiares de excitação dos neurónios, de forma a distribuir a actividade de forma idêntica por todos os neurónios da rede. Dessa forma, o reservatório auto-organiza-se em resposta ao sinal que recebe, neste caso sequências de símbolos desenhadas para testar as características de memória, dependência de contexto temporal e capacidade de representar os diferentes símbolos em diferentes trajectórias neurais (padrões de actividade). Estas redes são então comparadas com redes equivalentes mas estáticas (sem plasticidade), sendo claramente demonstrada a vantagem da presença da plasticidade.

Ao longo da tese são analisadas diferentes propriedades destas redes, como a separação (determinada por análise de componentes principais e clustering efectuados à matriz de activações), a aproximação ou performance do mecanismo de leitura, a criticalidade da dinâmica da rede (como medida de estabilidade, determinada como resposta a perturbações externas), bem como as várias alterações introduzidas pelos diferentes mecanismos de plasticidade, individualmente e em conjunto. É demonstrado que as redes dotadas de plasticidade são claramente superiores na capacidade de separar os diferentes símbolos apresentados em diferentes trajectórias neurais. Para além disso, há uma clara vantagem no uso dos três mecanismos em simultâneo, sendo que a ausência de qualquer um deles, ou o seu isolamento, conduzem a rede a desenvolver um comportamento mais instável, com impacto em diversos parâmetros relevantes.

`Palavras-Chave:` *Redes Neurais Recorrentes, Computação em Reservatório, Neuroplasticidade, Processamento de Sequências, Séries Temporais, Auto-Organização*

# 1 Introduction

*I don't have any solution, but I certainly admire the problem.*

– Ashleigh Brilliant

The human brain is a remarkably complex dynamical system, composed of approximately a hundred billion individual neurons, each of which connected to another thousand to ten thousand neurons, all embedded in a mesh of glia cells (which also play an active functional role pertaining to the system's computations [31, 118]). While a lot is known about the basic components of the nervous system, significant insights into how they come together, and which operational constraints allow for the emergence of complex, coherent activity is a tremendous challenge and is widely acknowledged as one of the major tasks facing modern science.

The reasons to pursue this goal are obvious: the computations performed result in nothing less than human cognition and behavior, thus outperforming any current engineered artificial or machine learning system; the application of principles guiding neuronal information processing to computing and robotics holds the promise of revolutionizing technology; from an applied point of view, a rational approach to psychiatric and neurological illness requires a thorough understanding of the system's components and their interactions.

This endeavor has become more urgent in the past few years, with the emergence of new experimental methods that not only provide rich datasets for network analysis, but also allow for a precise manipulation and control of their components, a control that is only precluded by a lack of understanding of the operational principles involved.

The work presented in this thesis holds no promise of originality. It mainly reviews and recreates a previously presented neural network architecture (a self-organizing recurrent neural network), whose dynamics and organizational principles are guided by simple plasticity rules, operating synergistically.

The thesis is organized as follows:

**Chapter** 2 – This chapter presents an overview of the fundamental biological background that serves as a basis for the construction of the models used throughout the thesis. These include a general explanation of neuronal networks and their organization and dynamics; neuroplasticity, focusing on the empirical data that supports the 3 adaptive mechanisms used in the model, and how they operate in neurobiology; and the processing of spatiotemporal information as evolving neural trajectories.

**Chapter** 3 – This chapter focuses on network models and their evolution over the past 50 years or so. It starts with a brief historical review, outlining the most relevant contributions to the field, stemming both from neuroscience and artificial intelligence (AI). Then the most prominent models and their characteristics are described, starting with the traditional feed-forward networks (FFNs) to the currently dominant paradigms used both in computational neuroscience and machine learning applications.

**Chapter** 4 – This chapter contains a description of the models and analysis methods used throughout the thesis, their formalization and implementation (included, for completeness in appendices B and C).

**Chapter** 5 – This chapter describes and analyzes the relevant results obtained from the simulation studies conducted, starting from the specification of the appropriate parameters (tuned to the highest readout performances) and using those parameters to analyze the properties of dynamic versus static reservoirs in different settings.

# 2
# Biological Inspiration

*Science has shown you that 'you', your joys and your sorrows, your memories and your ambitions, your sense of identity and free will, are in fact no more than the behavior of a vast assembly of nerve cells and their associated molecules. As Lewis Carroll's Alice might have phrased it: 'You're nothing but a pack of neurons.'*

– Sir Francis Crick

## 2.1 Neurons and Neuronal Networks

Understanding neural dynamics is an incredible challenge due to the staggering complexity and diversity encountered at every level of description of the brain: from the dynamic gene regulation to cognition and behavior, several layers of adaptable systems operate upon one another, making it extremely difficult to discern which biological facts are essential to the system's operation and which can be regarded as artifacts of biological evolution.

Each individual neuron is unique and presents, on its own, a complex, dynamic range of behaviors, determined by its specific cellular and synaptic properties [97, 96]:

**Cellular properties** — include spike threshold (voltage at which a spike is generated); excitability (number or frequency of spikes evoked by an input); spike frequency adaptation (reduction in excitability during repetitive activity); post-inhibitory rebound (increased excitability after inhibition is removed); plateau potentials (sustained spiking that outlasts the input that triggered it).

**Synaptic properties** — include input sign (excitatory or inhibitory), amplitude and time-course; activity-dependent properties (section 2.2).

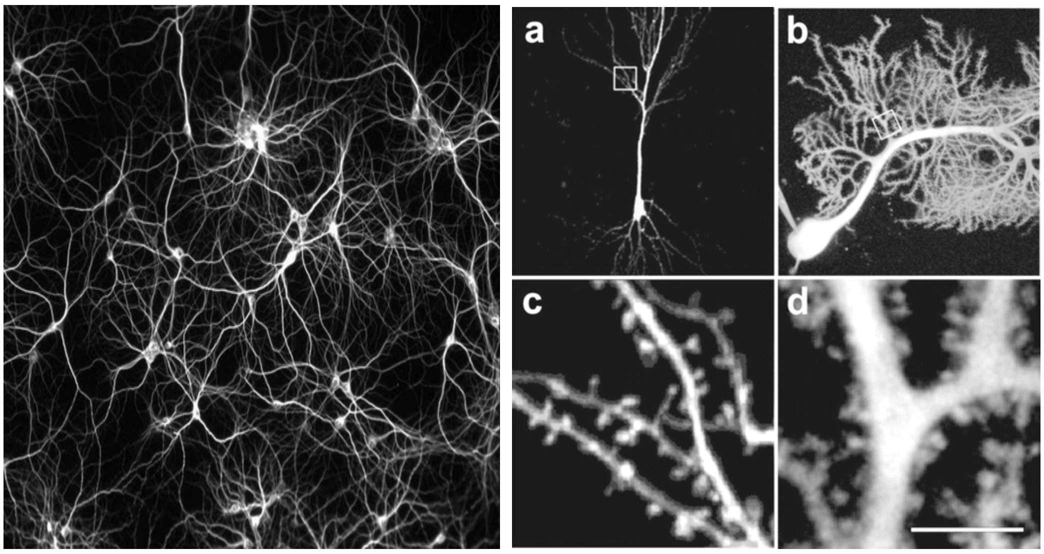


**Figure 2.1:** Left: Dissociated culture of hippocampal neurons. Right: Dendritic spines in a hippocampal CA1 neuron filled with calcein (a, c) and in a Purkinje neuron loaded with fluorescein dextran (b, d). Images obtained by 2-photon laser scanning microscopy ([91]).

All these properties are in turn influenced by the dynamics of their molecular substrates: with over 143 genes coding for voltage-gated ion channels (VGICs) [137]; over 200 substances acting as neurotransmitters [97]; and the even greater diversity introduced by post-translational modifications, alternative RNA splicing and the varying combinations of channel subunits. Even with the simplification of a binary representation (active or inactive neurons), the number of possible network configurations exceeds the 'absolute number of elementary particles in the universe' [106].

Nonetheless, the neocortex has some discernible universal properties, both structurally and functionally. It is organized in a hierarchical and modular architecture. Each module (the hypercolumn) displays a general six-layer architecture [87, 109] and a certain regularity in connectivity between layers (although the connectivity, at the local circuit level, appears to be stochastically guided rather than follow a specific ‘plan’). These circuits also appear not to be task-specific, i.e., the same ‘generic’ microcircuit can perform very diverse computations [76]. One should strive to take advantage of these universal properties and try to deduce the laws governing both the structural organization and the functional relations, either by theoretical abstraction and modeling or by carefully designed experimentation.

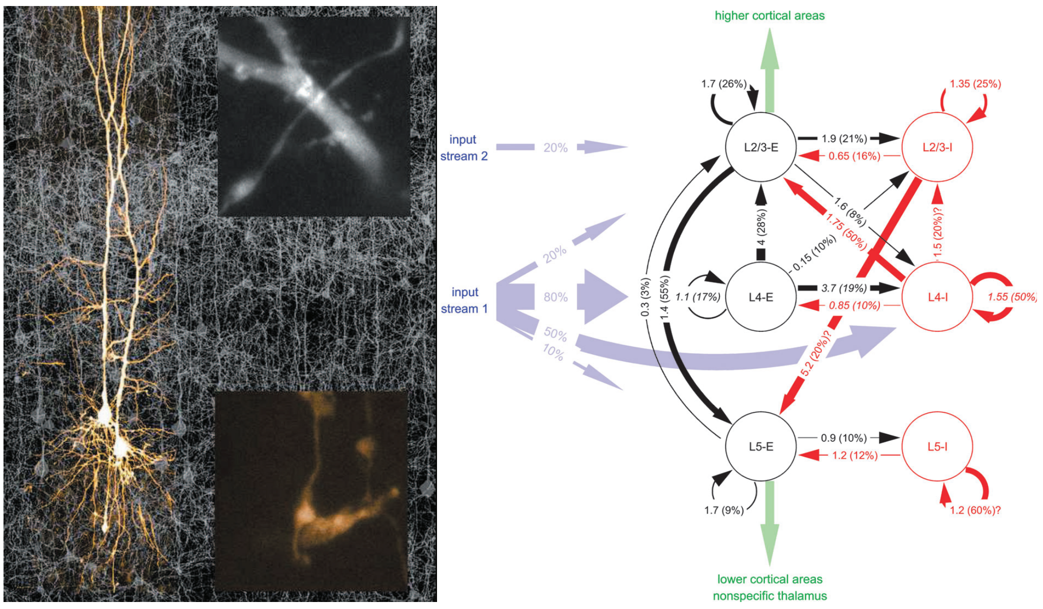


**Figure 2.2:** Left: Two confocally reconstructed neurons overlaid on the other pyramidal neurons in the rat somatosensory cortex. The upper right image shows an axon passing by a dendrite without forming a synaptic connection, which is verified by paired recordings. The lower right image shows an axo-dendritic synapse. Image reproduced from [62]. Right: General lamina-specific cortical microcircuit template used for simulation studies in [51]. Numbers in the arrows denote connection strengths and connection probabilities based on various sources of experimental data. Each layer consists of an excitatory and an inhibitory population.

## 2.2 Neuroplasticity

Neuronal networks are extremely versatile. The structural and functional properties of network components are the subject of activity-dependent adaptation. Structural modifications refine network topologies and connectivity patterns, either by selective synaptic generation and pruning [119, 115, 20], or by targeted neurogenesis [63, 25] (albeit limited in adult systems). Functional changes allow for purposeful computations to take place by adjusting synaptic strengths and intrinsic physiological properties.

One of the primary sites of functional adaptation in the nervous system is the synapse. Synapses reveal a great morphological and molecular diversity, with several of its components known to be subjected to activity-dependent modulation (number and probability of neurotransmitter release, receptor type, number and density of receptors, channel conductance and kinetics, etc.) [96]. The duration of changes in synaptic strength can span a wide range of timescales, from a few milliseconds to several hours, days or even longer. It is thus not surprising that synapse-specific Hebbian plasticity[1] [44] (long-term potentiation (LTP) and long-term depression (LTD)) became the most well known and established basis of learning and memory in the brain (for reviews, see [8, 72, 68]).

But, over the past few years, additional forms of plasticity have been extensively documented (see [89, 1]) and it is now established that many different plasticity mechanisms operate synchronously in the mammalian cortex (Figure 2.3). The synergy between these different mechanisms has proven to be crucial in ensuring stability by coordinating and balancing neural activity and preventing any individual change from having a deleterious effect.

### 2.2.1 Spike-Timing-Dependent Plasticity

It has long been demonstrated that repetitive synaptic activity can induce a persistent, long-lasting increase or decrease in synaptic efficacy (usually

[1]In honor of Donald Hebb (see section 3.1), activity-dependent synaptic plasticity that depends on pre- and post-synaptic activity is often called Hebbian plasticity or Hebbian learning.

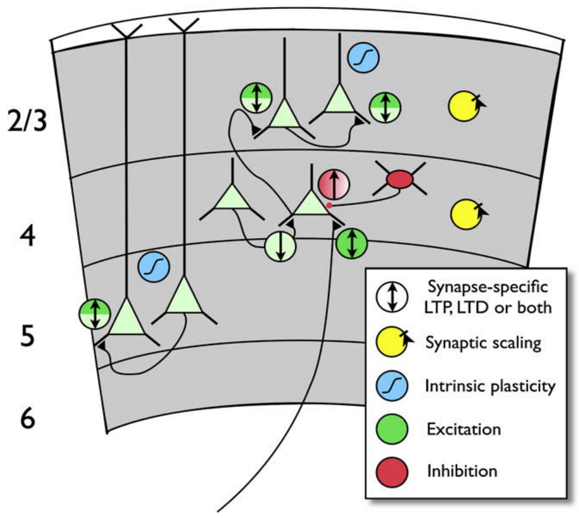


**Figure 2.3:** Several known plasticity mechanisms in sensory cortical circuits. Circles with arrows indicate LTP (up), LTD (down), or both, in excitatory (green) and inhibitory (red) synapses. Excitatory and inhibitory synapses in L2/3 and L4 exhibit synaptic scaling. IP has been demonstrated in L2/3 and L5 pyramidal neurons. The axons entering L4 symbolize thalamic input. Figure reproduced from [89].

determined as changes in the average amplitude and latency of excitatory postsynaptic potentials (EPSPs)), a phenomenon known as long-term potentiation (LTP) and long-term depression (LTD). Such synaptic plasticity can provide a cellular mechanism for experience-dependent modification of developing neural circuits [39, 114, 70] and for learning and memory in the adult brain [81]. LTP can enforce associative responses to pre-synaptic firing patterns that are temporally linked to postsynaptic firing, while LTD can facilitate the unlearning of such associations, when pre- and postsynaptic firing are not consistent [120].

Experimental protocols used to induce LTP and LTD *in vitro* traditionally involved high- or low-frequency stimulation, applied to the entire presynaptic axon tract. But, despite their convenience, these stimulation protocols are not physiologically realistic, as they usually involve a thorough culture preparation (for example, preincubation with tetrodotoxin or perfusion with zero $Mg^{2+}$ solution during stimulation [12]). Besides, cortical neurons seldom fire at the rates used in these protocols [46].

Over the past decade or so, thinking about how synaptic plasticity is induced *in vivo* shifted radically, with the discovery that both LTP and LTD can be induced at low-frequency, depending only on the precise timing between pre- and postsynaptic firing. In mammalian pyramidal neurons, it has been shown that if a pre-synaptic spike precedes a postsynaptic spike within a narrow time window ($\leq$ 40 ms) this results in LTP; the reverse order leads to LTD [79, 12, 9]. This means that the relative timing between pre- and postsynaptic spiking is a factor in determining the direction and extent to which those changes occur (Figure 2.4). The term spike-timing-dependent plasticity (STDP) was then introduced, referring to this observation that the precise timing of spikes affects the sign and magnitude of synaptic plasticity.

The discovery of STDP led to a quantitative formulation of Hebbian plasticity rules, representing a major advance over previous means of studying synaptic plasticity. As a consequence, the quality and physiological plausibility of experimental *in vitro* stimulation protocols improved [1], along with the development of timing-based theoretical models and learning rules.

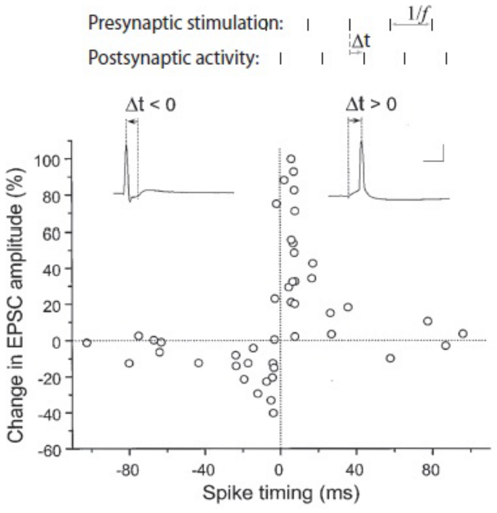


**Figure 2.4:** Critical time window for the induction of synaptic potentiation and depression. The precise spike timing determines the size and magnitude of EPSC change. Figure reproduced from [12]

STDP is often interpreted as the most comprehensive learning rule for a synapse, determining how individual synapses participate in circuit function by introducing competition: groups of synapses that are effective at rapidly generating postsynaptic spikes are strengthened by STDP, making them even more effective at controlling the timing of postsynaptic spikes. This competition assures that only the synapses that consistently take part in eliciting a postsynaptic response are maintained and strengthened, thus creating input selectivity and a primary form of memory in neuronal circuits.

Functionally, STDP allows neurons to detect which of their afferent inputs are relevant for eliciting a spike (the ones that arrive within the time window over which the neuron integrates its inputs) and which are the result of random fluctuations. The former will be strengthened, while the latter weakened [33, 1]. This makes obvious sense: given the high connectivity involved, random 'coincidences' are bound to occur.

The precise mechanisms that make synaptic plasticity sensitive to spike timings are not entirely understood, but seem to depend on a complex interplay between NMDA receptor activation and the timing of backpropagating spikes [1]. LTP and LTD are known to be mediated by NMDA receptors. Recent studies showed that the underlying biological pathways for each of them are mechanistically distinct and may, in fact, involve different subpopulations of NMDA receptors, activating different signaling cascades [110, 111].

Ultimately, a precise mechanistic explication of STDP will need to take into consideration the specific synapse involved, its type and location and the modulatory changes it is subjected to.

Nevertheless, the introduction of sensitivity to spike timing into Hebbian plasticity has a host of interesting implications as a learning mechanism for generating neuronal responses selective to input timing, order and sequence, making STDP the most relevant synaptic plasticity mechanism for studying temporal processing in neuronal circuits. Also, given that the synaptic increments or decrements are small (*infinitesimal learning*), the impact of stimulus history can be accounted for by this learning rule, as it takes some time to build up a 'noticeable' effect.

### 2.2.2 Intrinsic Plasticity

Studies of neuroplasticity usually focus on synaptic plasticity and, until recently, activity-dependent modifications were ascribed exclusively to synaptic changes. But, it is now clear that experience produces enduring alterations in the neuron's intrinsic excitability, directly affecting its input-output function and, consequently, network behavior (Figure 2.5). Empirical evidence for this phenomenon has been observed in many different species, in distinct brain areas and for a great variety of neurons [138, 24, 21, 78, 14]. It is usually measured as a change in the neuron's frequency-current (f-I) function, manifested as alterations in the spike threshold, spike accommodation[2] and the amplitude of burst-evoked afterhyperpolarization[3], resulting from changes in the function and responsiveness of VGICs [24, 138].

The precise mechanisms are not entirely known, but several possibilities have been referred in the literature (Table 2.1). VGICs participate in various different signaling pathways and their distribution and density throughout the membrane is not uniform nor static, so the mechanisms underlying intrinsic plasticity (IP) are likely to be diverse and to depend on an interaction of several components.

In some reported cases, training-induced IP was accompanied by persistent changes in synaptic strength within the same neuron or in related portions of a local circuit and it was positively correlated with some measure of learning [14, 138], leading to the speculation that it might create a hyperexcitable state responsible for promoting the consolidation of memory by facilitating synaptic plasticity. It is thus often called a "metaplastic" phenomenon.

The adjustments made by IP, together with scaling phenomena (subsection 2.2.3), help the neuron to regulate the transduction of its synaptic inputs homeostatically, while maintaining its relative responsiveness in periods of intense activity.

[2]The cessation of spike firing despite constant depolarization above threshold[138]
[3]Transient phase of hyperpolarization after an activity burst (such as epileptiform activity), inhibiting the neuron from firing for a while

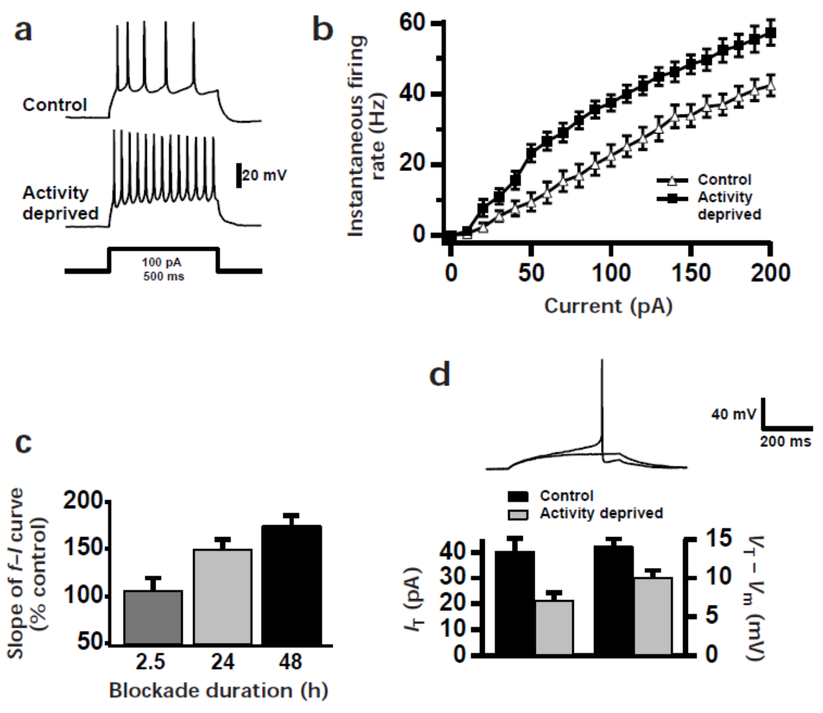


**Figure 2.5:** Chronic activity blockade of pyramidal neurons changed their firing frequency and lowered their threshold. (a) Spike trains evoked by somatic current injection. (b) Average f-I curves, plotting the instantaneous firing frequency versus the amplitude of injected current. (c) The effect size depends on the duration of the blockade. (d) Reduction in spike threshold: $I_T$ is the minimum current necessary to elicit a spike; $V_T$ is the threshold potential and $V_m$ the resting potential. Figures reproduced from [24]

| **Receptors / Ion Channels** | **Effectors** | **Enzymes** | **Functional Targets** |
|---|---|---|---|
| Voltage-Gated $Ca^{2+}$ channels | $Ca^{2+}$ | PKC | $K^+$ channels (several different types) |
| NMDARs | G-Proteins | CaMKII | $Na^+$ channels |
| mGluRs | | Adenylyl Cyclase | $Ca^{2+}$ channels |
| mAChRs | | Guanylyl Cyclase | Others |
| 5HTRs | | nNOS, PKA, PKG, MAPK … | |

**Table 2.1:** Potential molecular signals and substrates involved in neuronal intrinsic plasticity. Table reproduced from [138].

### 2.2.3 Synaptic Scaling

Neocortical neurons can detect changes in their own average firing rates through a set of calcium dependent sensors involved in the regulation of functional glutamate receptor synthesis and trafficking [69, 125, 126]. In so doing, these neurons maintain the average firing rates by distributing synaptic strengths, ensuring that neuronal activity does not saturate. This scaling phenomenon involves a proportional regulation of all of a neuron's synapses, allowing the neuron to preserve its synapse-specific differences established by Hebbian plasticity, that presumably encode information, thus ensuring that neuronal activity remains within a functional range.

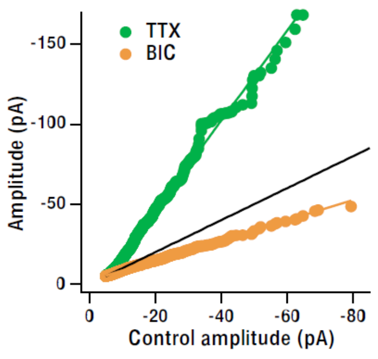


**Figure 2.6:** Variation in quantal amplitudes of mEPSCs recorded from cortical pyramidal neurons in cultures that experience normal levels of spontaneous activity (control amplitude) plotted against amplitudes in sister cultures in which activity was either blocked with tetrodotoxin (TTX) or enhanced with bicuculline (BIC) (by blocking inhibition), for 2 days. Activity blockade scales up mEPSC amplitude, whereas enhancement scales it down. The fact that, in both cases, the plots are well fit by straight lines indicates that the scaling is multiplicative. Figures reproduced from [1], adapted from [124].

This phenomenon has been observed in cultured networks of neocortical [124], hippocampal [69] and spinal cord [93] neurons, where pharmacological manipulation caused a multiplicative increase or decrease in mEPSCs (Figure 2.6). Increases in synaptic strength during periods of low activity have been associated with a decrease in the turnover rate[4] of AMPA receptors (Figure 2.7). NMDA-receptor-mediated glutamatergic synaptic transmission has also been linked to synaptic scaling [134], which implies that the scaling phenomenon may have an impact on Hebbian mechanisms. If neurons scale down NMDA receptor currents in response to enhanced activity, this may make it harder to evoke LTP and easier to evoke LTD.

[4]Ratio between insertion and removal rates.

Activity stabilization during developmental processes, for example, have been partially ascribed to synaptic scaling [126], with the readjustment of a neuron's synaptic strengths as a fundamental mechanism to deal with the increasing number of inputs the neuron receives during synaptogenesis.

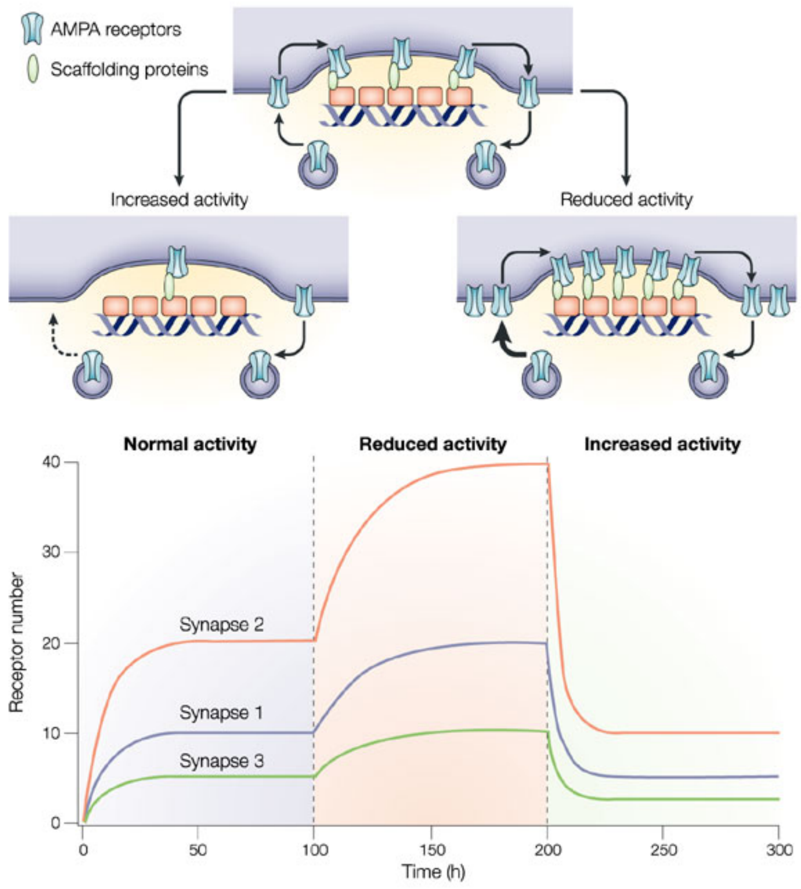


**Figure 2.7:** Synaptic scaling is accompanied by changes in the accumulation of AMPA receptors at synaptic sites, as well as the turnover of scaffolding proteins that tether AMPA receptors to the cytoskeleton. One possible mechanism for such changes is that activity targets the rate of constitutive receptor insertion, so that increased activity reduces this rate (dashed arrow) whereas reduced activity increases this rate (wide arrow). The number of receptors at the synapse is the result of a dynamic equilibrium between insertion and removal rates. If the insertion rate doubles (reduced activity), each synapse will reach a new steady-state level of receptor accumulation that is double the initial value at that synapse. Conversely, if the insertion rate is cut in half (increased activity), each synapse will end up with half the initial number of receptors. Figure and results reproduced from [127].

## 2.3 Temporal Processing and Transient Dynamics

All natural stimuli have an inherent spatiotemporal structure. Sensory perception depends critically on the ability of the sensory ‘modules’ to represent and process those spatial and temporal features. The distinction between spatial and temporal content is relevant, since they are likely to be processed in fundamentally different ways [17], for which no general computational framework exists [18].

The dominant approach to systems neurobiology has been to correlate spiking patterns (number and timing) in defined neuronal populations with specific aspects of perception and behavior. It is now clear that this is not a well constrained problem and there is a need to dig deeper in order to understand how neuronal networks extract relevant information about a given stimulus from the spatial and temporal structure of the complex spiking patterns elicited in response to said stimulus. Analysis and modeling have been largely biased towards the spatial components of stimuli, focusing on the responses of neuronal populations defined mainly by the location of sensory afferents and on the ability of such networks to develop selective responses to the spatial aspects of stimuli. This approach led to the discovery of place codes and maps in the early sensory stages (retinotopy [82], somatotopy [94], orientation selectivity (Figure 2.8) [113, 112, 19], etc.).

But neuronal responses are largely influenced and depend on the temporal features of stimuli as well [22, 23]. Sensory information about a constantly changing environment has to be gathered and integrated over different time scales, in order to allow for the construction of a stable percept [49]. Temporal information is crucial, for example, to [18]:

**Visual System** – object motion direction and velocity and intervals between sensory events (Figure 2.10) [90].

**Somatosensory System** – motion detection; object and texture discrimination [86].

**Auditory System** – spectral and temporal structure of vocalizations, auditory context and speech processing [7, 16].

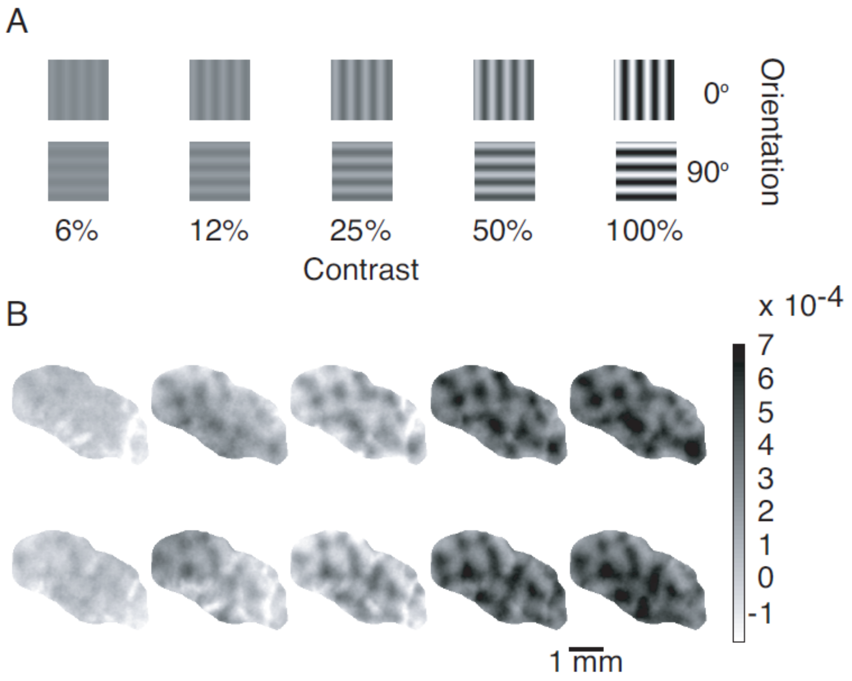


**Figure 2.8:** Examples of stimuli and optical responses recorded from the cat's visual cortex for two orientations and five contrasts. A: A subset of the stimuli used, which included four or more orientations and in some cases additional contrasts. B: Optical responses, scaled by subtracting response to blank screen, and dividing by maximal recorded response. Dark shading represents activation. Figures reproduced from [19].

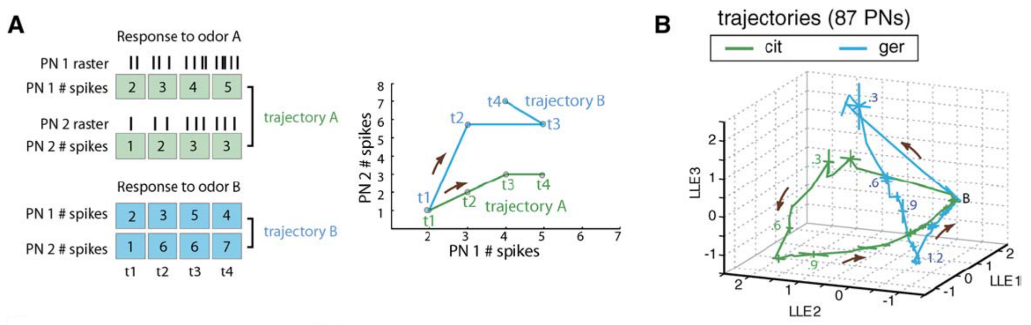


**Figure 2.9:** Representation of the responses evoked by 2 odors (citral and geraniol) in populations of projection neurons in the locust antennal lobe. A: Responses from 87 PNs are divided in 50 ms bins and the spikes in each bin counted. The resulting consecutive $87D$ vectors are then linked, generating trajectories. B: By dimensionality reduction, the trajectories can be plotted in $3D$ space. Procedure and figures reproduced from [15].

**Olfactory System** – odor identity and concentration (Figure 2.9) [15].

Recent models of cortical activity reflect the fact that temporal and spatial processing are inextricably linked. These state-dependent network models are capable of encoding information in the form of evolving neural trajectories (through the network's state-space) and suggest that spatiotemporal computations emerge naturally from the inherent dynamics of cortical networks.

The notions of state-dependency and state-space stem from dynamical systems theory, an approach that basically allows one to study these complex systems without knowing all the details that govern their evolution. A dynamical system can be described by a set of variables that represent its state at any given point (the state variables) as well as the laws that describe the evolution of the state variables with time (the state equations) [53]. The state of the system is, in essence, an explicit account of the values of its internal components, in this case, the activation state of the neurons in the circuit. The set of all possible states of the system is called the state-space, where each state corresponds to a unique point. The system's evolution with time, creates a trajectory through this state-space.

The traditional hypothesis in neurodynamics is that of stable attractor states as supporting neural computation, i.e., upon some input signal, network activity gradually converges to some point in state-space (in which neuronal firing rates remain constant for a given period). This concept should not be disregarded (attractor states are known to play a role in higher-order cognitive processes [18]), but it is unlikely to contribute to sensory processing in real-time and in the presence of ever-changing sensory information.

The general framework discussed here, emphasizes the idea of transient dynamics, where computation is carried out over time without any need for stable states. The system never attains a true steady-state but, instead, is in a constant state of adaptive self-organization, a state that is inherently transient [26, 49].

> "Brain activity is characterized by a high-dimensional chaotic ground state from which transient spatio-temporal patterns briefly

emerge (...) It is the transient dynamics, rather than any state eventually reached that is at the heart of the computational process." [26]

This view is consistent with experimental data and well suited to explain the most complex forms of sensory processing. The measured responses from a neuronal population are known to be the result of a complex interplay between stimulus characteristics and the network's internal state [4, 18]. If driven by ongoing external stimulation, network activity will evolve in a complex neural trajectory (Figure 2.9). Different stimuli elicit different trajectories, the characteristics of which are driven both by the stimulus structure and by the internally generated dynamics (resulting from the highly recurrent connectivity of the circuit) [18]. Even when the stimulus is static, the trajectories that represent the network's activity continue to dynamically evolve [15, 100], meaning that further stimulus characteristics are being processed (like intensity, or duration). Besides, the succession of states visited by the system is stable in the presence of noise and reliable, even in the face of small variations of the initial conditions [100].

Theoretically, the information contained in the local circuit's trajectories can be extracted by read-out neurons, located at specific sites to serve as relays, conveying the information to the downstream circuitry to which they project. The higher the dimensionality of the input space to these neurons, the easier it is for them to decode the activity patterns [18]. Although this can easily be achieved in theoretical studies (by finding the read-out synaptic weights through an appropriate learning rule), evidence for this phenomenon *in vivo* are still lacking.

State-dependent network models (see subsection 3.3.2), provide the key to begin to understand cortical function and how information is reliably represented, extracted and conveyed in cortical networks.

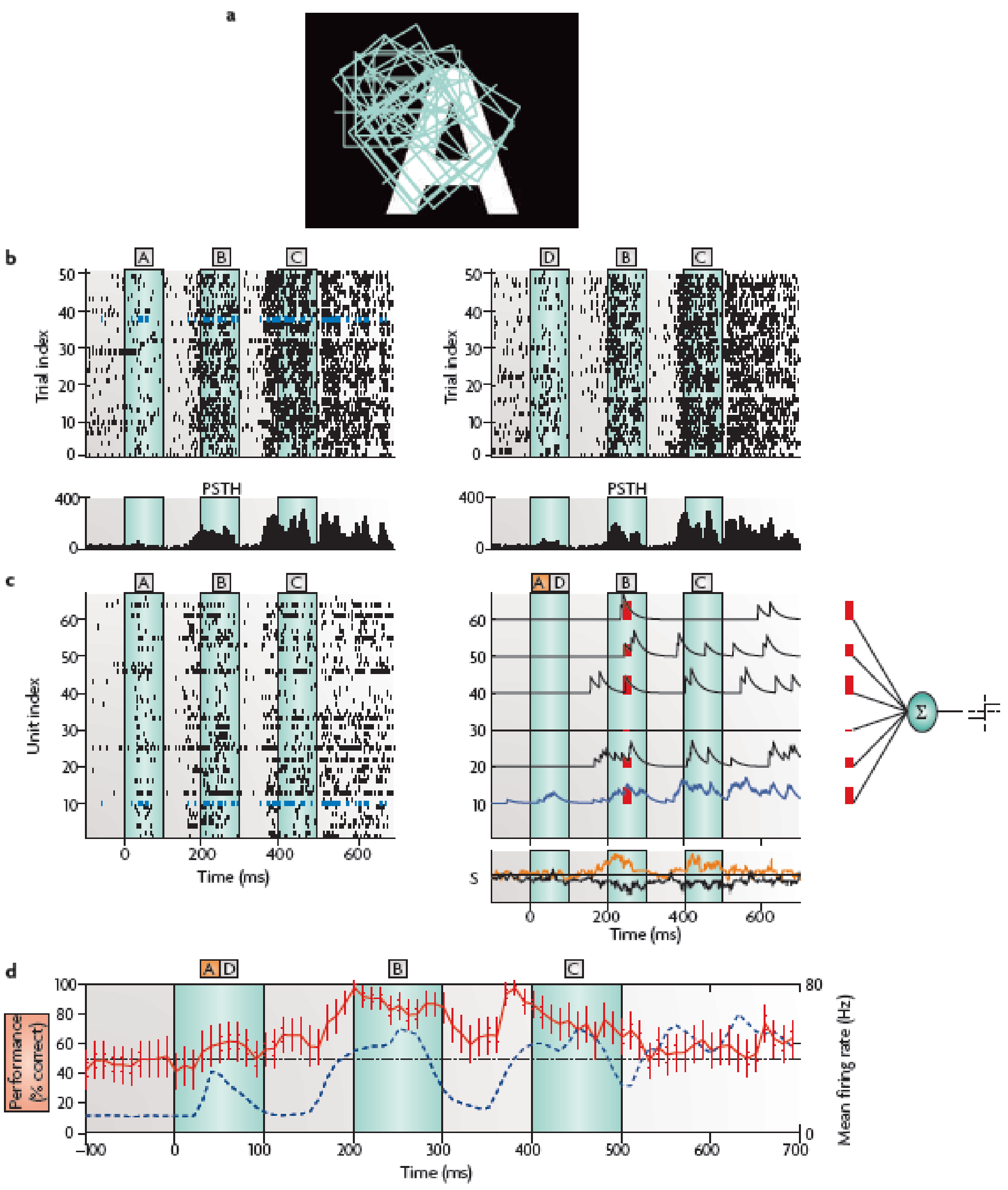


**Figure 2.10:** Population responses of neurons in the cat visual cortex upon presentation of sequences of symbols. **a**: Example stimulus, superimposed with the receptive fields of the recorded neurons. **b**: Raster plot and PSTH from one neuron over 50 trials, upon presentation of the sequences A, B, C (left) and D, B, C (right). **c**: Responses recorded from 64 neurons in trial 38 with sequence A, B, C and the readout mechanism used to decode information from those 64 spike trains. Each spike train was low-pass filtered with an exponential and sent to a linear classifier. **d**: Classifier's performance at various points in time and average firing rates. The red trace shows the percentage of trials in which the readout correctly classified the initial symbol as being A or D. The classifier was able to readout this information for up to 700$ms$ after stimulus presentation. Figures reproduced from [18], adapted from [90].

# 3 Neural Network Models

*Swiftly the head-mass becomes an enchanted loom where millions of flashing shuttles weave a dissolving pattern, always a meaningful pattern though never an abiding one; a shifting harmony of subpatterns.*

– Sir Charles Sherrington

## 3.1 Historical Notes

Artificial neural networks (ANNs) attempt to capture the essential computations taking place in the dense, intricate biological neural networks, while providing a much sought symbolic formalism for their analysis. When the first models emerged, the prevailing belief was that of elementary logic as the foundation of neural computations. Guided by this belief, and following the ideas of Alan Turing [123], McCulloch and Pitts designed and built a primitive ANN, using simplified 'binary' neurons [83]. Each neuron implements a simple threshold operation, comparing the weighted sum of all its afferent connections to a given threshold and providing a binary output (Figure 3.1). It is generally agreed that their work gave rise to the disciplines of neural networks and AI.

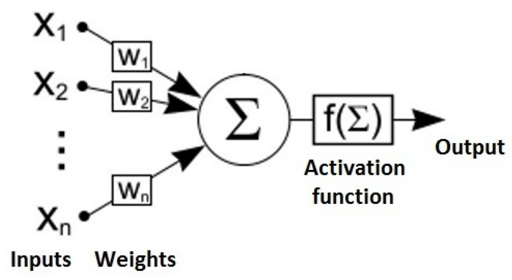


**Figure 3.1:** Schematic representation of the McCulloch-Pitts model neuron, the first mathematical model of an artificial neuron.

In 1949 Donald O. Hebb published *The Organization of Behavior*, where the first concrete proposal of self-organization in the brain as supporting cognition and behavior was introduced, stating that specific synaptic changes may underlie learning. He postulated that the effectiveness of a synapse between two neurons increases by repeated co-activation:

> "Let us assume then that the persistence or repetition of a reverberatory activity (or trace) tends to induce lasting cellular changes that add to its stability. The assumption can be precisely stated as follows: when an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place on one or both cells so that A's efficiency as one of the cells firing B is increased."[44]

In 1958, Frank Rosenblatt proposed a novel method of supervised learning for pattern recognition, the *Perceptron Convergence Theorem* [104]. The Perceptron was the first 'practical' ANN. His work was based on the premises that the connections in the brain do not initially allow useful computations to be performed; order ought to be brought into the system by learning. The Perceptron was a linear classifier with a simple input-output relationship (feed-forward mapping) composed of McCulloch-Pitts neurons [101]. Given their simplicity, work on the perceptrons grew rapidly and new training algorithms were soon developed.

In the meantime, neuron models evolved to deal with real-valued inputs and outputs and the threshold activation functions (used in the perceptron) were also replaced by linear input-output mapping (Adaline[1]) or non-linear functions (like the sigmoid, used in multi-layer perceptrons (MLPs)).

In the 1970s, *Self-Organizing Maps (SOM)* were introduced by Teuveo Kohonen, using competitive learning and motivated by the topographic organization of the brain's sensory system. These models introduced the "distance neurons", where the activation function was applied to the distance between the weights and inputs ($\| X - W \|$) (Figure 3.2B), instead of their inner product ($< X, W >$) (Figure 3.2A). Although there is no real difference in computational power, the neighborhood functions allowed these networks to preserve the topological properties of the input space, making them useful for visualizing low-dimensional views of high-dimensional data.

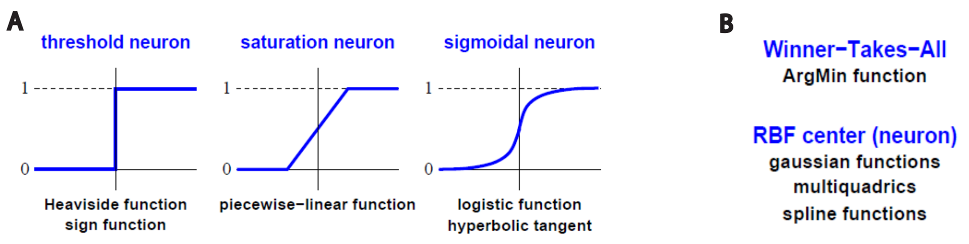


**Figure 3.2:** Variants of neuron models with different activation functions. Figures reproduced from [98].

In 1982, John Hopfield presented a paper in which he stated that the approach to AI should not be to purely imitate the human brain, but to use its concepts to build machines that could solve dynamic problems [47]. He introduced one of the first types of recurrent connectivity in ANNs (section 3.3). The Hopfield networks were controlled by constructing very specific topologies with symmetrical weights and were operated by applying a fixed input pattern. Under certain conditions (and after a sufficient time interval), the network 'settles'. This means that the system's output has converged to some activation pattern which is considered the result of the computation. This result is some association made by the system in response to the input, hence the name associative network. The process of arriving at the output is called relaxation dynamics, and can be expressed with the help of an energy

[1]Adaline (Adaptive Linear Elements), was the first trainable neural network to be applied to real-world problems

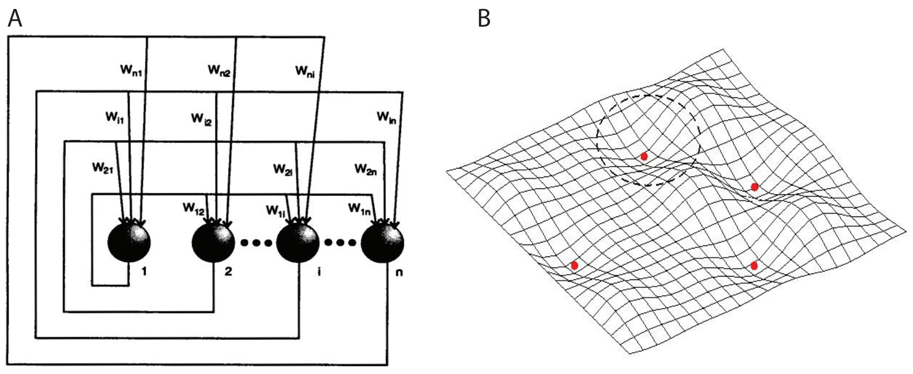


**Figure 3.3:** The Hopfield network. A: Schematic representation of the network topology (crossbar representation); B: Schematic landscape in a fictitious network space, the presence of point attractors is highlighted by the red dots.

equation. The local minimum energy states are the network ‘attractors’, towards which the network converges (Figure 3.3).

These networks have the property of *content-addressable memory*, which is described by an appropriate flow through the system’s state-space, i.e., the network retrieves a memory from a given input if the activity generated by that input converges toward the local attractor that represents the original pattern. So, after learning a set of patterns, the network could retrieve a complete pattern given only a sub-part or a noisy version of it as input. The Hopfield network model was later on extended to neurons that operate stochastically (borrowing theory from statistical mechanics), called the *Boltzmann machine* [45].

Since then, many other models have been proposed, like Vapnik’s *Support Vector Machine*, and previous models improved, particularly during the 1990’s. These new network models, along with advances in neurobiology, gradually began to focus on time, temporal context and temporal sequence processing [2, 98].

The inclusion of temporal information in network models is central both from a neuroscience (section 2.3) and from an engineering perspective (given that many real-world tasks are temporal by nature).

## 3.2 Traditional Neural Networks

The classic approach is to use multi-layered, feed-forward network topologies (FFNs), with unidirectional information flow (Figure 3.4), propagating the input via weighted connections, through internal (hidden) layers of *neural amplifiers* to an output node (or layer), thus functioning as a universal approximator for any spatially finite input-output relationship [50]. The

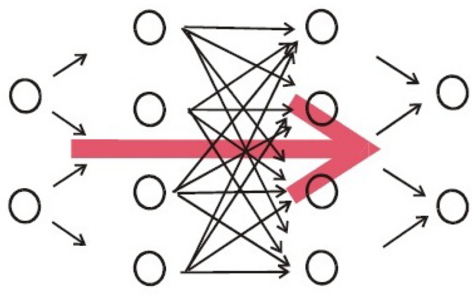


**Figure 3.4:** Schematic representation of the Feed-Forward Network topology

weighted connections between the layers are trained in a supervised fashion, using a form of error backpropagation[2]. Since the model requires full connectivity between the layers, the number of connections increases with the square of the number of nodes, causing the computational efficiency of the backpropagation algorithm to rapidly decrease. This procedure is commonly used for static pattern recognition or classification (non-temporal tasks). The lack of feedback makes each input pattern independent and, upon presentation of a new input, all information about a previous one is lost. While this is beneficial in some cases, it is clearly unsuitable for realistic neuronal modeling, where temporal information is crucial (section 2.3).

Several attempts have been made to extend FFNs to temporal data processing [30, 29], by transforming the temporal problem into a spatial problem. However, this requires unfolding layers (the number of which is dependent upon the input length), becoming prohibitive in terms of computational costs

[2]The network's output in response to a given input is compared to a target value and their difference is then used to modify the weights, in order to minimize the output error below a certain threshold.

and processing time. Besides, it is clearly not close to the “biological solution” to the problem.

## 3.3 Recurrent Neural Networks

The obvious solution for the temporal coding problem is to endow the network with recurrent connections (Figure 3.5). The neurons can remain active after the input presentation, which confers the network a *self-sustaining temporal activation* [71]. By representing temporal sequences as a flow of network

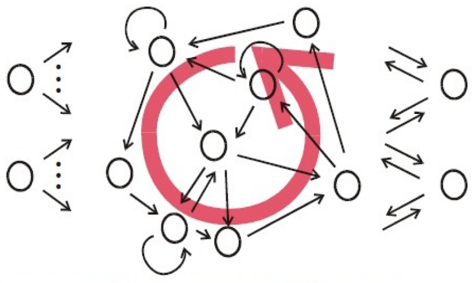


**Figure 3.5:** Schematic representation of the Recurrent Network topology. The network contains directed cycles that feed its activity back serving as additional inputs, radically transforming the system into a, potentially very complex, dynamical system.

activity and not as an additional spatial dimension, recurrent neural networks (RNNs) can handle temporal information directly and naturally. Other capabilities include an *associative memory* [47, 103, 48, 84] and *optimization capacity* [13].

The overall advantages of RNNs can be summarized as [129]:

- Robust
- Capable of learning by example (generalization)
- Capable of modeling highly nonlinear systems (for which concrete, tractable models are hard to obtain)
- Inherently capable of temporal processing

From a computational neuroscience perspective, RNNs constitute a good model, given that biological brain modules almost universally exhibit recurrent connection pathways. But, although their potential is widely acknowledged, their application has remained limited, because of several constrains [32, 129, 71]:

**High computational costs** – the traditional gradient-descent based methods[3] are difficult to employ as a training procedure and many update cycles are required to generate any single update parameter. Besides, their optimization requires substantial skills and expertise to be successful.

**Limited number of available learning rules** – dependencies requiring long-range memory are inherently hard to learn [10].

**Slow convergence rates** – the network's dynamics goes through bifurcations[4], making it difficult to guarantee convergence.

## 3.3.1 Evolution of the paradigm

The first algorithms used to train RNNs (and also the most common in FFNs) were based on the iterative update of all weight values according to an estimated error gradient [40, 105, 71]: $\frac{\partial E}{\partial W}$, in order to minimize $E = E(y, y_{target})$.

Based on this procedure, the first proposed methods for RNNs were back-propagation through time (BPTT) [135] and real time recurrent learning (RTRL) [136]. BPTT is based on the transformation of the RNN into a FFN, where classical BP algorithms can be applied. At every time step of the training sequence, a new layer is constructed as a copy of the sets of internal units in the RNN. The connections between units are transformed into connections between layers. Each layer also receives connections from

[3]Also known as steepest descent, it's a first order optimization algorithm that finds the local minima of a given function and uses it to iteratively reduce the training error

[4]A bifurcation occurs when a small change made to the parameter values (the bifurcation parameters) of a system causes a sudden 'qualitative' or topological change in its behavior

the input at the respective time. The weights are subsequently updated using classical backpropagation. This method has a runtime complexity[5] of $O(N_x^2)$, per weight update, per time step, for a single output.

RTRL consists on the computation of the exact error gradient in a recursive way. The derivatives of the states with respect to the weights are computed first, using the results of a time step $k$ to get the derivatives at $k+1$. The procedure computes the gradient through forward recursion in time. Hence, one advantage of this algorithm is that it can be used online, so that the weights are adapted as the new training patterns are introduced. The runtime complexity of RTRL is $O(N_x^4)$.

A major change in RNN algorithms was the Atiya-Parlos recurrent learning (APRL) method [5], proposing a fundamentally different approach. In APRL, training is treated as a constraint optimization problem [40]. The error gradient is computed with respect to the neuron's activation states, instead of the connection weights ($\frac{\partial E}{\partial x}$). Once the desired change to the states is known, a proportional change to the weights is computed. So, the weights are gradually adapted to move $x$ into the desired directions [71]. This algorithm partially solved some of the problems with RNN training, allowing a much faster convergence to the solution.

As an extension and refinement of the APRL, the Backpropagation-Decorrelation (BPDC) algorithm was proposed as an online iterative training method [116, 117]. It resulted from a careful analysis of the weight dynamics of the APRL algorithm, that demonstrated that the input weights and the internal weights change at a different pace. APRL decouples the RNN into a fast adapting output ($W_{in}$ changes rapidly) and a slowly adapting reservoir ($W$ changes slowly) [71]. The BPDC algorithm is capable of tracking fast-changing signals, but quickly forgets previously seen data. The great novelty introduced by this algorithm was the fact that the focus of training moved from the entire network towards the output weights (Figure 3.7). This method emerged almost simultaneously with the echo state network (ESN) approach and is thus frequently included in the domain of reservoir

[5]Used to describe the limiting behavior of a function when the argument tends towards a particular value or infinity.

computing (subsection 3.3.2).

So, in summary, the evolution of training algorithms for RNNs was:

- Classical BP
- BPTT/RTRL
- BPDC
- ESN/LSM

### 3.3.2 Reservoir Computing

A fundamentally different approach to use RNNs has been proposed independently by Herbert Jaeger, under the name of *Echo State Networks* [54] and by Wolfgang Maas, under the name of *Liquid State Machines* [73].

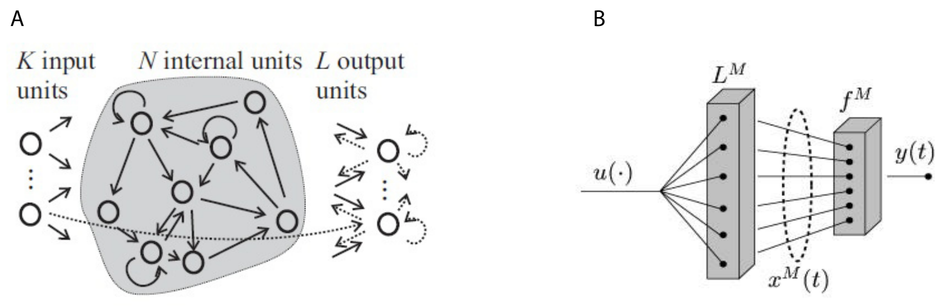
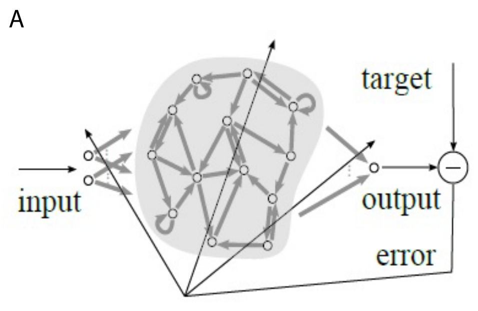


**Figure 3.6:** Schematic representation of the Echo State Network (A) and the Liquid State Machine (B), as originally depicted by their authors [54, 73].

The two models stem from different perspectives. LSMs have their origin in neurobiology and are intended to serve as models of generic cortical microcircuits, whereas ESNs were aimed at solving engineering problems. Despite their differences, they are very similar, on an abstract level, which led to the introduction of the term *Reservoir Computing* [131, 107] as a general term for this kind of approaches and their subsequent variants.

The main idea is to use random, sparsely connected and untrained RNNs as an excitable medium (the reservoir), composed of a large set of neurons



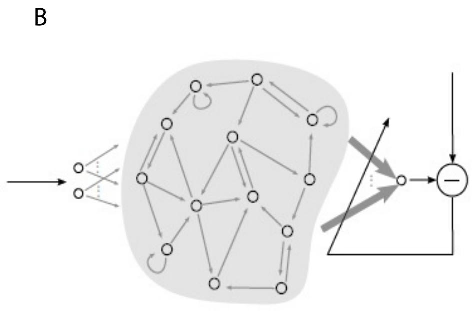


**Figure 3.7:** Contrast between the traditional gradient-descent based methods for training RNNs, where all connection weights are adapted during training (A) and the RC approach, where the only training required is to the output weights (B). Illustration reproduced from [71].

(typically $> 100$) to give a “rich set of dynamics to combine from”[54]. The reservoir is passively excited by the input data, which is then stored in the reverberating activity patterns. Those patterns are then fed to a memoryless readout neuron (or layer).

In this setup there is only a need to train the output weights in a supervised fashion, from the reservoir to the readout, and this can be done using very simple regression techniques. Once the readout has been sufficiently trained, it is robust against noise in the input data, displaying very good generalization capabilities [74, 34]. The reservoir is left untrained, functioning as a nonlinear filter.

This is a simple, but powerful “black-box” approach, that has attracted a lot of attention in the last couple of years, from very diverse fields of research [56, 131, 57, 58, 130], and constitutes the dominant paradigm for RNN modeling. Besides, this type of model is extremely well suited for neuro-inspired modeling, given that the reservoir can be used independently of the exact nature of its constituents, thus allowing several studies to be performed using increasingly accurate neuronal models [51, 75].

Nonetheless, the RC approach has been criticized mainly for not being principled enough, i.e., for not having a generally agreed upon ‘recipe’ that can be used regardless of the intended application [102, 99]:

- The choice of parameters (like connectivity parameters) is a crucial step in constructing the reservoir, but there are no clear rules for it, and it is usually achieved through a series of random attempts, relying mostly on trial and error.

- The random connectivity and internal weight structure is unlikely to be optimal and does not provide a clear insight into the reservoir's dynamic organization, contravening a deeper theoretical investigation.
- It is difficult to specify the properties of the reservoir responsible for its success because the exact function of some parameters is still poorly understood.

There have been several attempts to address these issues and devise what parameters would constitute a 'good' reservoir for a given application [95, 102, 58], regularize its construction to improve its implementability [27, 77], extend its stability by incorporating subreservoirs [60, 129], pre-training the reservoir in a supervised manner, for example, with evolutionary algorithms [61, 52] or reinforcement learning, etc. The mainstream research in the field is currently directed at improving reservoir design and adaptation characteristics, while attempting to understanding the effect those changes have on the reservoir's performance and dynamics [71].

### Plasticity

Many extensions of the original networks have been proposed to incorporate adaptation through plasticity, in a supervised or unsupervised manner. Because biological brains are endowed with several plasticity mechanisms (section 2.2), improving reservoirs with local adaptation rules is a natural strategy. The first attempts to do so focused on classical Hebbian [44] or anti-Hebbian learning [55], but obtained no success. Introducing modifications to anti-Hebbian learning, the so-called anti-Oja rule was shown to slightly improve reservoir's performance [6].

Liquid state machines (LSMs) have already incorporated realistic synaptic models with dynamic adaptation, based on empirical data [80], in their original appearance [73]. Conferring STDP to the synapse models in LSMs was shown to improve the separation capabilities (subsection 4.2.1) for real-world speech data, but not for random inputs [92].

The addition of IP learning rules has attracted a lot of attention in the RC community. A biologically inspired and computationally efficient IP learning rule to adjust neuron's threshold and gain was originally proposed in [121] and shown to improve the input-specific information encoding in the reservoir. This IP rule is local in time and space and attempts to optimize information transmission by regulating the neuron's output distribution [117]. It is based on 3 fundamental principles [133]:

- Information Maximization, i.e., the neuron's output should contain as much information on the input as possible
- Each neuron has to keep its average output at a certain level, because the energy available is limited
- Adaptation is realized in the neuron's intrinsic properties and not in the connection weights

Of great interest to computational neuroscience is to understand how different plasticity mechanisms observed in biological brains (section 2.2) interact and what impact they have, operating both individually and synergistically, on the reservoir's quality. The synergy of IP with Hebbian synaptic plasticity was shown to result in a better specialization of the neuron in finding heavy-tailed directions in the input and on the "bars" problem[6] (Figure 3.8) [122]. Combinations of STDP and IP in a LSM-like reservoir were shown to be more robust to perturbations and to achieve a better short-term memory and prediction performance [66]. An extension of this work, including a third plasticity mechanism, synaptic normalization, was also shown to outperform static reservoirs [64]. Besides, it was shown that only the synergistic combination of all mechanisms was able to maintain the performance advantages and overall reservoir quality. Further work on synaptic plasticity for reservoir computing includes the introduction of a STDP modulated rein-

[6]A common nonlinear independent component analysis problem

forcement signal as a reward-based feedback, mimicking biological dopamine reward modulation [67], among other approaches.

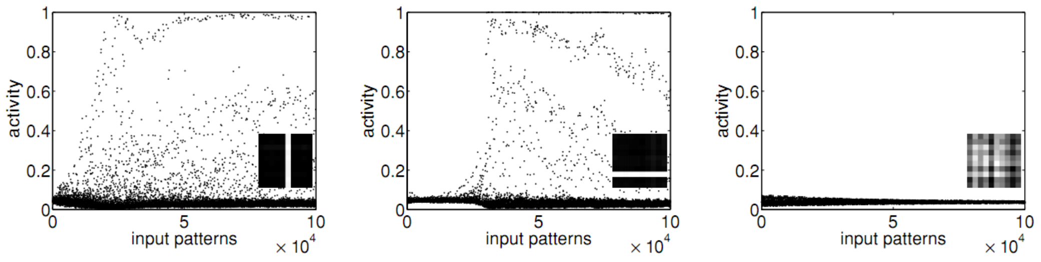


**Figure 3.8:** Discovery of single bars in the "bars" problem with IP rules. Left plot: fast IP ($\eta_{IP} = 0.01$,$\eta_{Hebb} = 0.001$). Center plot: slow IP ($\eta_{IP} = 0.001$,$\eta_{Hebb} = 0.01$). Right plot: no IP. Without IP, no bar is discovered. Figures reproduced from [122].

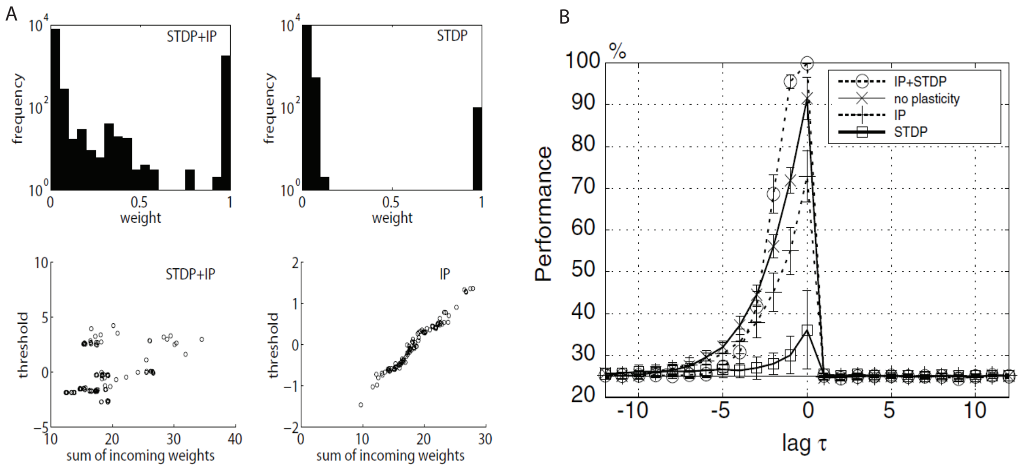


**Figure 3.9:** Impact of different combinations of plasticity in the network's structure (A) and on the fading memory property (B). Figures reproduced from [66].

# 4 Methods

*By three methods we may learn wisdom: first, by reflection, which is noblest; second, by imitation, which is easiest; and third, by experience, which is the most bitter.*

– Confucius

## 4.1 Network Definition

The network used throughout this thesis is a self-organizing RNN, as originally proposed by [64, 65]. This network is composed of $N_E$ excitatory and $N_I$ inhibitory neurons ($N_I = 0.2 \times N_E$), consistent with the known average neocortical distribution.

Neurons are connected by weighted synapses (with $W_{ij}$ representing the synapse from neuron j to neuron i), whose initial values are randomly drawn from the interval $[0, 1]$ and normalized so that the sum of incoming connection strengths to a neuron is constant($\sum_j W_{ij} = 1$). Connectivity in the excitatory population ($W^{EE}$) is sparse with a mean number of total connections a neuron is allowed to establish defined by $\lambda^W$. There's full connectivity between inhibitory and excitatory populations ($W^{EI}$ and $W^{IE}$) and no connections among inhibitory units (Figure 4.1).

The units' thresholds ($T^E$ and $T^I$) are random values initially drawn from a uniform distribution in the interval $[0, T^E_{max}]$ and $[0, T^I_{max}]$, for excitatory and inhibitory units, respectively.

The external input is composed of sequences of symbols ($U(t)$), represented as input vectors, whose structure the network is expected to acquire in an unsupervised manner. Each symbol activates a set of $N_U$ input units ($N_U = 0.05 \times N_E$), meaning that when that particular symbol is present, the $N_U$ units responsive to it will receive a positive external input ($v_i^U = 1$). Each set of $N_U$ units is responsive to one particular input symbol. The sequences are composed of six different symbols ($a$, $b$, $c$, $d$, $e$ and $f$) whose order can be random or structured. In the structured form, these symbols are organized in two "words" ($a$, $n \times b$, $c$ and $e$, $n \times d$, $f$), with $n$ determining the word length.

The network evolves in discrete time steps, according to:

$$x_i(t+1) = f(\sum_{j=1}^{N_E} W_{ij}^{EE} x_j(t) - \sum_{k=1}^{N_I} W_{ik}^{EI} y_k(t) + v_i^U - T_i^E(t)) \tag{4.1}$$

$$y_i(t+1) = f(\sum_{j=1}^{N_E} W_{ij}^{IE} x_j(t) - T_i^I) \tag{4.2}$$

for the state of the excitatory (Equation 4.1) and inhibitory (Equation 4.2) units, respectively. $f(.)$ refers to the transfer function (subsection 4.1.2).

An internal state is also defined, based on the network's activity on the previous time step (the network's recurrent drive):

$$x_i'(t+1) = f(\sum_{j=1}^{N_E} W_{ij}^{EE} x_j(t) - \sum_{k=1}^{N_I} W_{ik}^{EI} y_k(t) - T_i^E(t)) \tag{4.3}$$

Most of the analysis will be based on this internal state.

## 4.1.1 Plasticity

Three plasticity mechanisms operate in parallel to shape the network's dynamics, endowing these networks with self-organizational capabilities. STDP and SN dynamically regulate the synaptic strengths of the internal weights ($W^{EE}$) and IP regulates the excitatory units' thresholds, so as to spread the

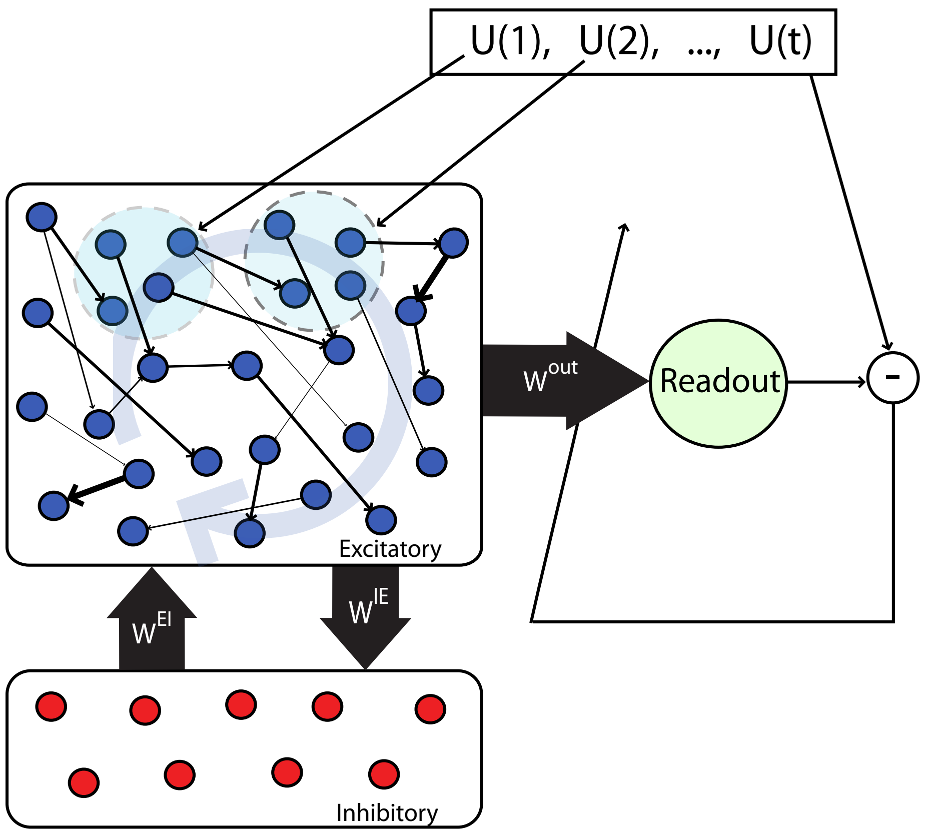


**Figure 4.1:** Schematic representation of the Self-Organizing RNN[64, 65], with a reservoir of excitatory units (blue), and a smaller population of inhibitory units (red). The internal connection weights in the excitatory population (depicted by the arrows, the width of which represents different strengths) are subjected to STDP and SN. The connections between the two populations and between the reservoir and the readout are depicted by the big arrows (indicating full connectivity). Each input population (light blue) receives external input in the presence of the particular symbol it is responsive to. The excitatory units' thresholds vary according to the IP rule.

network activity evenly across all units.

**Spike-timing-dependent plasticity (STDP)** small, fixed increments / decrements (adaptation/learning rate: $\eta_{STDP} = 0.001$) change the synaptic strengths of pre-existing connections in the excitatory population. This is done in a temporally asymmetric way. When unit $i$ is active in the time step following activation of unit $j$, their connection ($W_{ij}^{EE}$) is strengthened (increased) and when unit $i$ is active in the time step preceding the activation of unit $j$, their connection is weakened (decreased) by the same amount:

$$\Delta W_{ij}^{EE}(t) = \eta_{STDP}(x_i(t)x_j(t-1) - x_i(t-1)x_j(t)) \tag{4.4}$$

**Synaptic normalization (SN)** rescales the $W^{EE}$ matrix on every time step to prevent STDP from causing uncontrollable growth in the synaptic strengths. This rule keeps those synaptic strengths bounded, without destroying the relative distribution of weights:

$$W_{ij}^{EE}(t)/\sum_j W_{ij}^{EE}(t) = W_{ij}^{EE}(t) \tag{4.5}$$

**Intrinsic plasticity (IP)** distributes the network activity evenly throughout the neurons, by regulating their responsiveness, i.e., their firing thresholds, in small, fixed amounts ($\eta_{IP} = 0.001$). On average, each neuron will tend to fire with $H_{IP}$ firing rate. The threshold of a neuron that has just been active is decreased, while the threshold of an inactive unit is increased by the same amount:

$$T_i^E(t+1) = T_i^E(t) + \eta_{IP}(x_i(t) - H_{IP}) \tag{4.6}$$

The implementation of the described model and all the simulations are performed in Matlab (appendices B and C).

### 4.1.2 Transfer Functions

Network behavior also depends on the neuron's input-output function (transfer function). Throughout this thesis, 2 different transfer functions will be used (Figure 4.2):

**Heaviside step function** This function acts as a mathematical 'on/off' switch. The output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value (or zero):

$$\Theta(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

**Piecewise linear function (clipped linear)** This function is defined in 3 'pieces', with a linear portion in $[0, 1/m]$, where the output activity is proportional to the input. $m$ controls the slope of the linear portion:

$$\varphi(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ mx & \text{if } 0 < x < 1/m \\ 1 & \text{if } x > 1/m \end{cases}$$

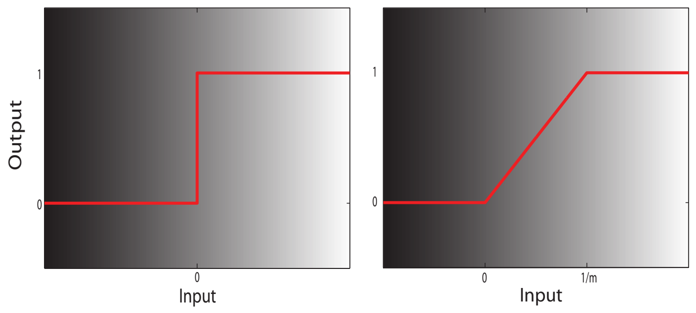


**Figure 4.2:** Graphical representation of the transfer functions used. The depicted clipped linear has a slope ($m$) of 1.

The clipped linear is a softened version of the hard limiter (step function), where the transition to clipping is not so abrupt. True hard limits are seldom seen in biological neuron's behavior, where there are always graded transition regions [41].

## 4.2 Analysis Methods

To determine the dynamic characteristics of these kind of reservoirs (subsection 3.3.2) two main properties ought to be assessed [71, 73]:

**Separation property** – different inputs result in clearly separable state-space trajectories, i.e., the distance between different network states is caused by different inputs.

**Approximation property** – the readout is capable of producing a desired output based only on the internal network states.

Other desirable features to be assessed are the existence of small pairwise correlations of the reservoir activations and a subcritical network behavior.

### 4.2.1 Separation Property

To correctly ascertain whether the differences in the network's internal states are caused by different inputs, we perform *agglomerative hierarchical clustering* and *principal component analysis* (Appendix A) on the internal network activations ($x'(t)$), for a given training period.

For the clustering procedure, the network's excitatory activity pattern at each time step is considered a point in $N^E$ - dimensional space. Each of these points is then considered the center of its own cluster and the Euclidean distance between these centers is computed. The resulting clusters are hierarchically merged, until only the last $n+2$ ("word" length) clusters remain.

The PCA procedure is used to visualize the network's state representations in the presence of the different input conditions, projected along the axis of the first PCs.

### 4.2.2 Approximation Property

To determine the readout "quality", a normalized performance measure is used. Given the nature of the input sequences, this measure will determine the capacity of the readout to correctly predict the immediately subsequent input in the sequence, based only on the network's internal state (*one-step prediction performance*).

For that purpose, a time series of activations is collected ($X \in \mathbb{R}^{N^E \times T}$) over a training period $(1, \dots, T)$ and used, together with the target output

($Y_{target} \in \mathbb{R}^{N_y \times T}$) to find the readout weights ($W_{out}$). Finding the appropriate weights is solving a system of linear equations:

$$W_{out}X = P_X(Y_{target})$$

where $P_X$ is the orthogonal projection onto the space spanned by the columns of X.

To do so, we use the Moore-Penrose pseudo-inverse method to find the least square estimate, i.e., to minimize the squared difference between the readout output and the target output. This is the only supervised training procedure employed:

$$W_{out} = Y_{target}X^{\dagger}$$

The trained $W_{out}$ are then used to assess the readout performance on a second time series, where the readout output depends exclusively on the internal network states. The absolute difference between target and readout output is then normalized to provide a measure of performance.

### 4.2.3 Criticality

Criticality will be assessed through a perturbation analysis. For every state $x(t)$, an external perturbation is introduced to a randomly chosen excitatory neuron (altering its state from active to inactive, or vice-versa). This perturbation creates an altered state $\tilde{x}(t)$, with a Hamming Distance (Appendix A) of 1 from the normal state ($d(t) = 1$). Both perturbed and normal states evolve according to Equation 4.1, in order to determine how the network handles the perturbation (by calculating the distance at time $t+1$). The procedure is repeated throughout the entire series and the average distance is determined ($\bar{d}(t+1)$). If $\bar{d}(t+1) > 1$ the network amplifies the perturbations and is said to operate at a supercritical regime; if $\bar{d}(t+1) < 1$ the network displays self-correcting behavior, operating at a subcritical dynamic regime (the desirable one); if $\bar{d}(t+1) \simeq 1$ (Figure 4.3 (red line)), the network operates at a critical regime. It has been argued that reservoirs achieve optimal computational performance in this critical regime, operating *at the edge of* *chaos* [11, 88].

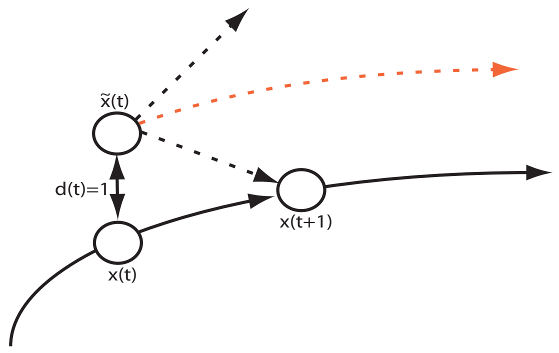


**Figure 4.3:** Schematic representation of the perturbation analysis. In the time step following the perturbation, the network can either amplify or smother it. This constitutes a measure of criticality of network's dynamics.

# 5 Results

*A man of science is responsible for the accuracy of his observations and of his inferences, not for the results which may follow therefrom.*

– Arthur Keith

## 5.1 Initial Parameters

In the following experiments, we tweak some of the initial variables that are prone to have a significant impact on the network's dynamics and prediction performance. For that purpose, we consider the network architecture presented in section 4.1, with binary logic gates (Heaviside step transfer function): $x(t) \in \{0,1\}^{N^E}$ and $y(t) \in \{0,1\}^{N^I}$. We assess several parameter settings for both dynamic and static reservoirs, in the presence of structured inputs with $n = 8$. Dynamic reservoirs are shaped by plasticity for 50.000 time steps, after which plasticity is turned off and the analysis will then be based on the final, post-plasticity values of $T^E$ and $W^{EE}$. The first 5.000 steps of the analysis phase are discarded (as they may be contaminated by initial transients) and the analysis is performed on the subsequent 5.000 steps. For both dynamic and static reservoirs, the choice of initial parameters is directed at achieving the best performance. The results of this study will lead to the choice of the optimal initial conditions that will be used in the following simulation studies.

### 5.1.1 Initial Threshold Values

The presence or absence of a spike for each time step is dependent upon the comparison of the total drive the neuron receives with its threshold. As previously mentioned (section 4.1), the initial threshold values for both inhibitory and excitatory neurons are drawn from a uniform distribution in $[0, T^I_{max}]$ and $[0, T^E_{max}]$, respectively. So, the choice of $T^E_{max}$ and $T^I_{max}$ has a great impact on the neuron's responsiveness and needs to be carefully assessed. In this section, we consider the impact of 32 different combinations of $T^E_{max}$ and $T^I_{max}$ on:

**Performance** – one-step prediction performance

**Firing Rate** – mean number of firing neurons per time step

All the results depict the averages and standard deviations over 10 simulations per condition for networks with 100 (Figure 5.1) or 200 (Figure 5.2) excitatory neurons.

Several combinations of $T^E_{max}$ and $T^I_{max}$ are acceptable. In both cases, the choice will have performance as the dominant factor. It is noticeable that dynamic reservoirs appear to outperform static ones in every setting and they all present firing rates within the target range.

In the case of static reservoir, we choose $T^E_{max} = 0.75$ and $T^I_{max} = 0.8$, it is the best performing configuration for $N^E = 100$ and among the best for $N^E = 200$. Also, with these values, the firing rates are, on average, close to the target rate of 0.1.

For the dynamic reservoirs, we choose $T^E_{max} = 0.5$ and $T^I_{max} = 1.4$. In this case several other combinations achieve high performances, so any one of those combinations would be acceptable.

### 5.1.2 Sparsity of the $W^{EE}$ matrix

In the construction of the reservoirs, the internal connectivity among excitatory neurons is sparse. A mean absolute value ($\lambda^W$) is defined as an in/out-degree, i.e., each neuron receives, on average, $\lambda^W$ connections from all

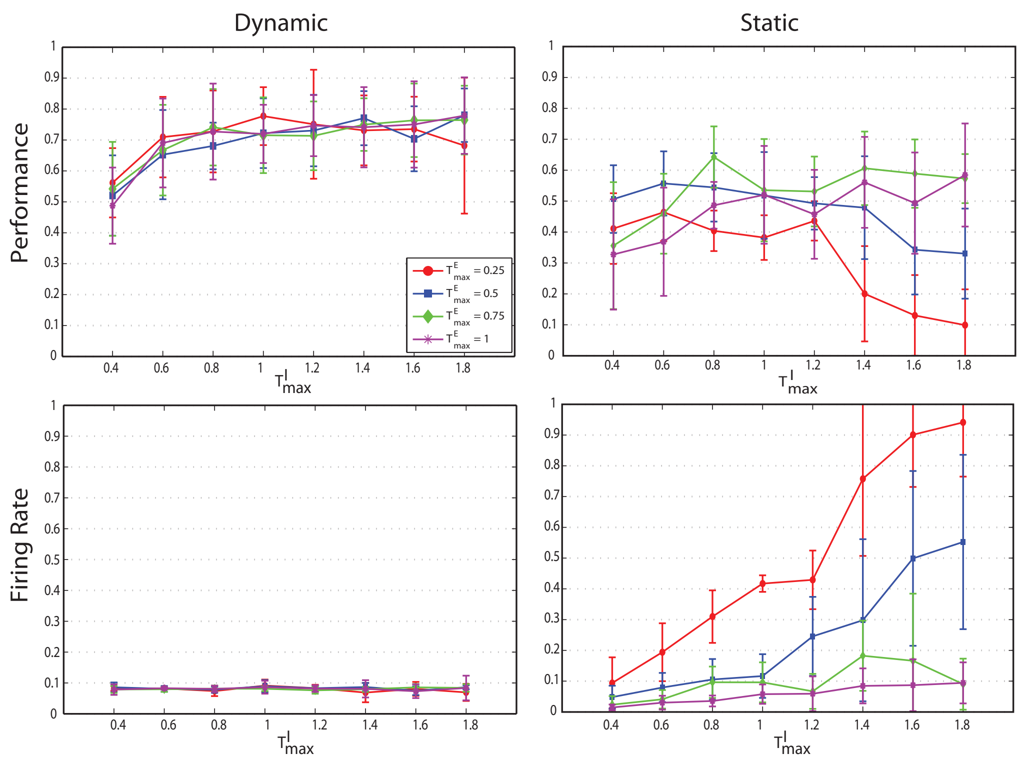


**Figure 5.1:** Impact of different initial threshold values on network's performance and firing rates. The results depict the means and standard deviations obtained over 10 simulations, for networks with $N^E = 100$.

other neurons in the reservoir and establishes, on average, $\lambda^W$ connections with other neurons. In this section we investigate the impact of different sparsity indices on network's performance (Figure 5.3A). This is an important parameter, given that sparsity is retained throughout plasticity, i.e., no new connections are allowed to be established. So, this parameter determines the general connectivity in the reservoir. It's important to point out that this index ought to have different impacts on networks of different sizes. Networks with 200 neurons with $\lambda^W = 10$ establish 5% of all possible connections, whereas in networks with 800 neurons this same index corresponds to a 1,25% connectivity.

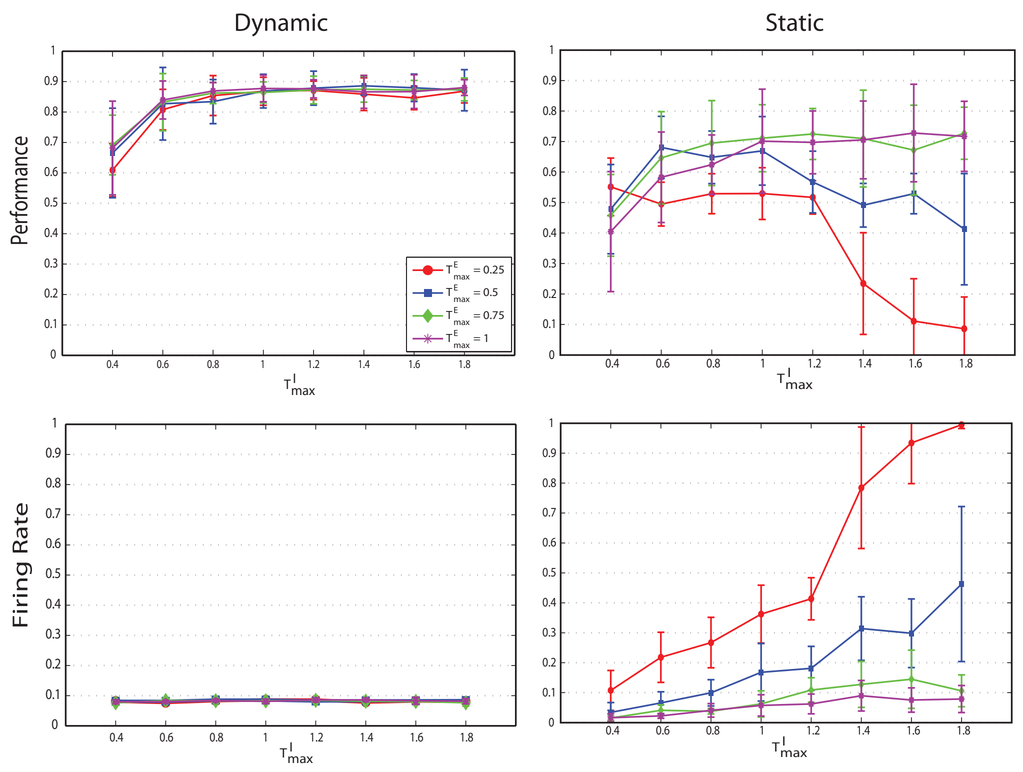


**Figure 5.2:** Impact of different initial threshold values on network's performance and firing rates. The results depict the means and standard deviations obtained over 10 simulations, for networks with $N^E = 200$.

### 5.1.3 Target Firing Rate

In dynamic reservoirs, the IP rule spreads the activity evenly across the neurons, constrained by a target firing rate (see section 4.1). This rate is defined as the mean proportion of firing neurons per time step. The results (Figure 5.3B) indicate that dynamic reservoirs obtain the highest performance for a rate of $20 - 25$ firing units per time step. For a rate of 25, the performance results are more consistent, with smaller standard deviations over the 10 simulations. However, the greatest absolute performances of all settings are achieved for 20 firing units per time step, where 3 of the 10 simulations obtained a performance greater than 0.92.

Given that the considered networks were made up of 200 neurons, 10 of which are input neurons, a firing rate of 20 corresponds to $2 \times N^U/N^E$.

Obviously, setting the firing rate to this value will have a different impact on networks of different sizes.

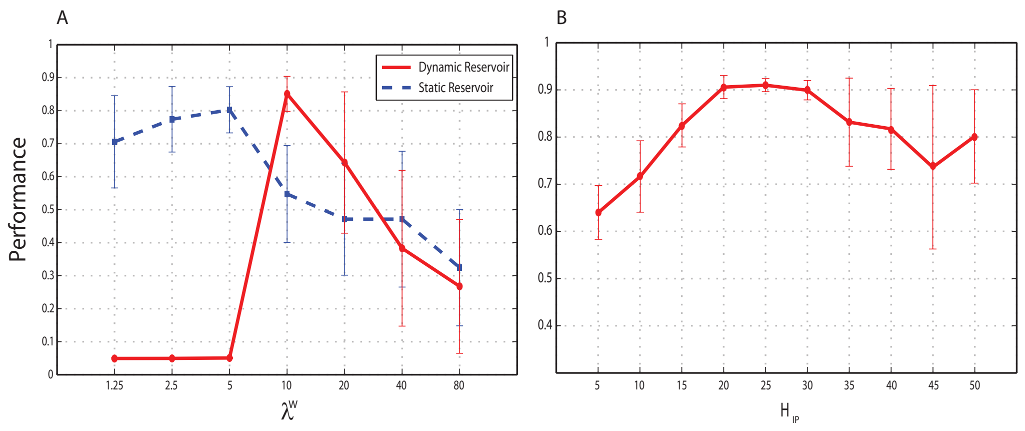


**Figure 5.3:** Impact of $W^{EE}$ sparsity (A) and target firing rate (B) on network's performance. For dynamic reservoirs $\lambda^W = 10$ clearly achieves the highest performances. Static reservoirs perform best with sparser connectivity indices. A target firing rate of $20 - 25$ firing units per time step achieves the highest performances. Depicted are the means and standard deviations over 10 simulations per condition, for networks with $N^E = 200$.

## 5.2 Changes Introduced by Plasticity

During the learning stage in dynamic reservoirs, the combination of the 3 plasticity mechanisms shapes the network's parameters to better represent and separate the input. The reservoir's internal connectivity matrix ($W^{EE}$), initially distributed over small values (most of them between 0 and $0,3$ (Figure 5.4)), becomes selectively strengthened by STDP, with a few synapses reaching values above $0,9$, although the majority becomes saturated around small values. This indicates a certain degree of specialization, with some connections being preferably selected over the others (as intended).

As explicitly enforced by the model, SN constrains the weight values to $[0,1]$, thus inhibiting the changes introduced by STDP from having a deleterious effect on the network's connectivity by causing uncontrolled growth or decay in weight values. When SN is not present, the network shows synchronous, periodic activity bursts with almost every unit firing simultaneously (Figure 5.7 B and E). In between these bursts, most neurons are

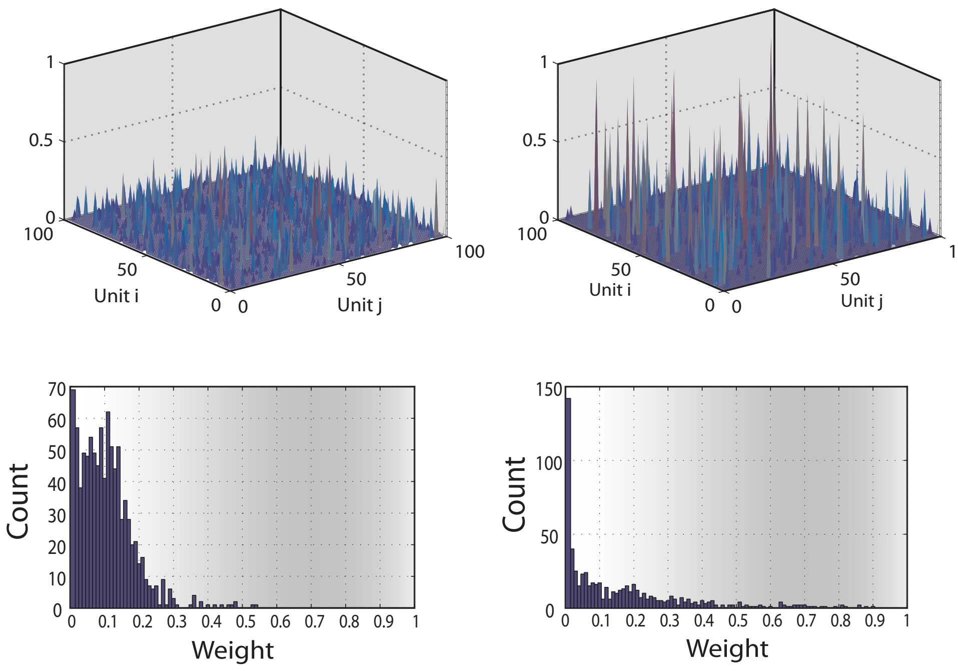


**Figure 5.4:** Variation in the internal connection weights matrix ($W^{EE}$) before (left) and after (right) 50.000 steps of plasticity. For the purpose of good visualization, we used an example network composed of 100 neurons. The top row depicts the weights matrices as surface plots, the bottom row includes weight distribution histograms.

silent.

The introduction of IP effectively spreads the spiking activity evenly across the network by readjusting the unit's thresholds in every iteration. When IP is not present, the distribution of neural firing rates becomes spread over a larger interval (Figures 5.5, 2.5).

Another relevant measure, pertaining to the internal connection weights, is the *spectral radius* ($\rho$), i.e., the largest absolute eigenvalue. This is an important parameter in the context of ESNs [54, 71]. For the ESN principle to work, the reservoir must have the *echo states property*[1]. This property is assured only if the reservoir's internal weights matrix satisfies certain algebraic conditions (see [54], for the full mathematical demonstration). One such con-

[1]The effect of a previous state and a previous input on a future state vanishes gradually as time passes[71].

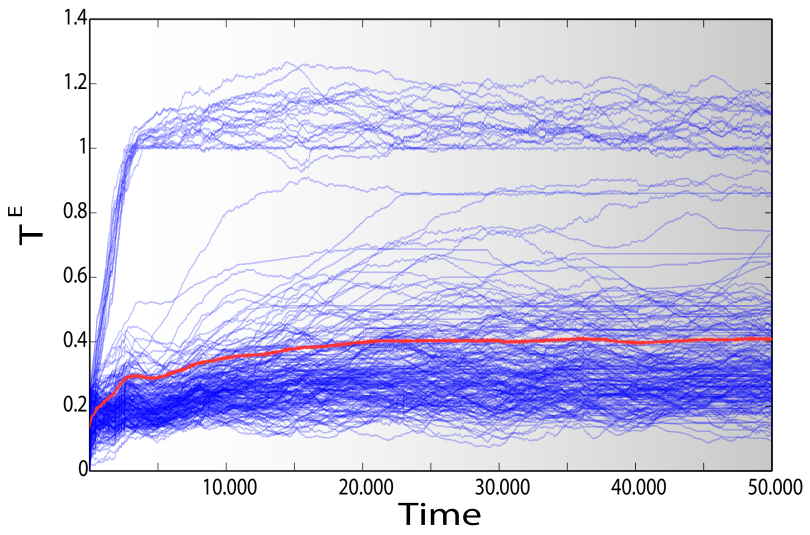


**Figure 5.5:** Variation in the excitatory unit's threshold values for an example network with 200 neurons, during 50.000 steps of plasticity. The red line indicates the mean.

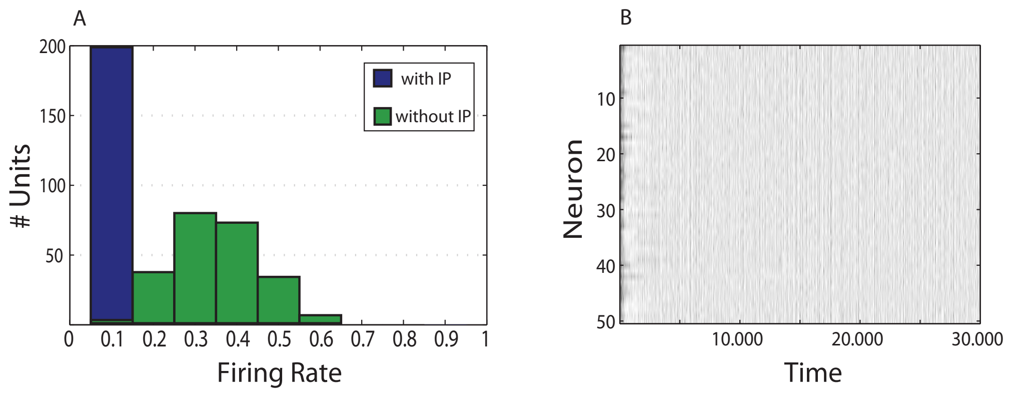


**Figure 5.6:** In the presence of IP, the network's activity is successfully spread (B), with all neurons firing with the desired target rate (A). This homeostatic distribution of activity allows the network to develop a "healthier" dynamics, without excessive activity.

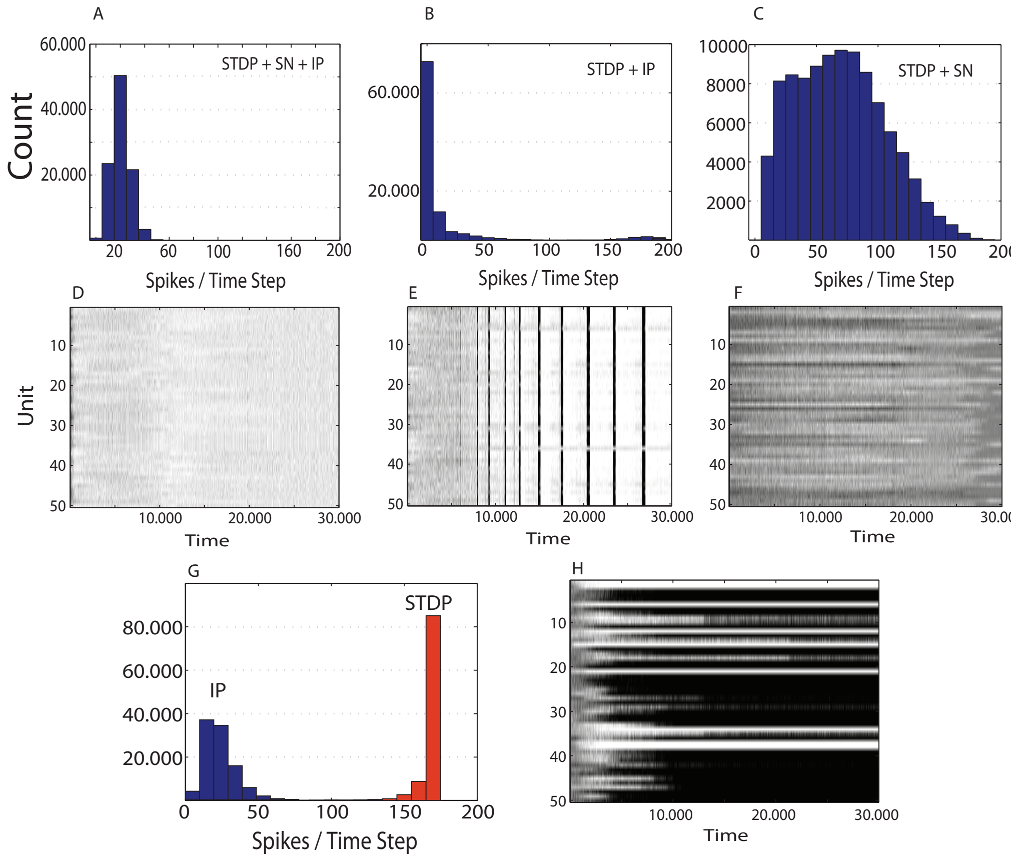


**Figure 5.7:** Effects introduced by the different plasticity mechanisms in the reservoir's activity. A, B, C and G show the distribution of spiking activity per time step throughout 65, 000 steps (50, 000 of which have plasticity active) for different combinations of mechanisms. D, E, F and H display the corresponding activity patterns for 50 randomly chosen reservoir units, during the first 30, 000 plasticity steps. H corresponds to the activity in the presence of STDP alone. The networks in these experiments were presented with an unstructured (random) input sequence. The combination of all 3 mechanisms is clearly beneficial in maintaining desirable activity levels (A, D).

dition (necessary, but not sufficient) is $\rho < 1$. Usually, in the construction of ESNs, the internal weights matrix is scaled to assure that $\rho < 1$.

The optimal value of $\rho$ depends on the amount of memory required by a given task. It has been experimentally demonstrated that ESNs perform better for higher values of $\rho$ (around 0.8) [128].

In our case, no caution was taken with this parameter in the construction of the reservoir. Nonetheless, both static and dynamic reservoirs have $\rho < 1$ (Figure 5.8). However, plasticity seems to distribute the eigenvalue spectra more widely, which can signify an adjustment to the task's memory requirements.

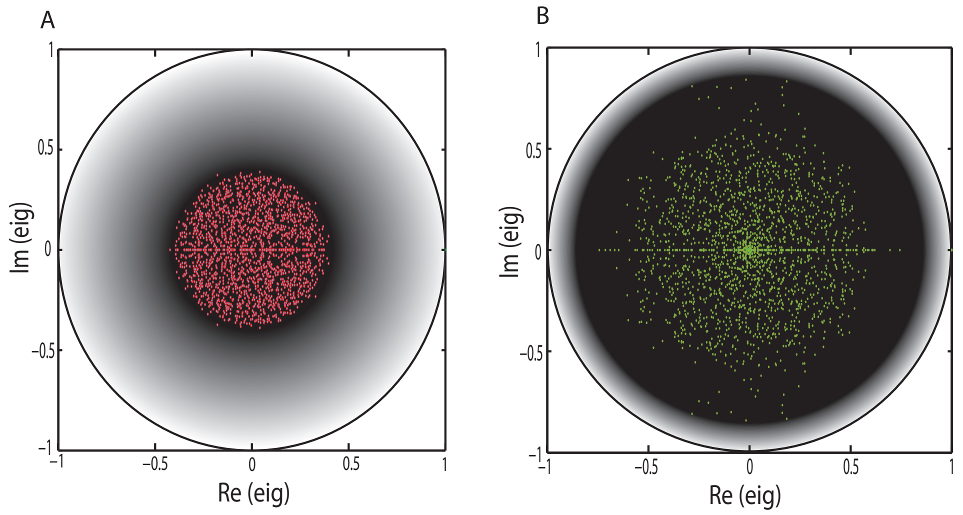


**Figure 5.8:** Eigenvalue spectra for 10 simulations of static (A) and plastic (B) reservoirs. Static reservoirs display a much more contractive spectral radius.

## 5.2.1 Learning Interval

The results discussed so far in this section, were based on the changes in network's parameters introduced by having plasticity mechanisms active for 50.000 time steps. But the size of this learning phase, i.e., the number of time steps in which plasticity is active, needs to be accounted for as it is bound to have an impact.

As the results demonstrate (Figure 5.9), longer learning phases contribute to better performances and to stabilize network's dynamics (measured here as the network's response to external perturbations (see subsection 4.2.3)). The impact on these parameters is more significant for durations of up to 80.000. Learning phases longer than that, display smaller variations.

The impact on average firing rates is not as noticeable, with all settings displaying some variation between 0.06 and 0.1, but rarely actually achieving the target rate.

Setting the number of learning steps to 50.000 appears to be a good choice, allowing these networks to settle into a subcritical dynamic regime and achieve good prediction performances.

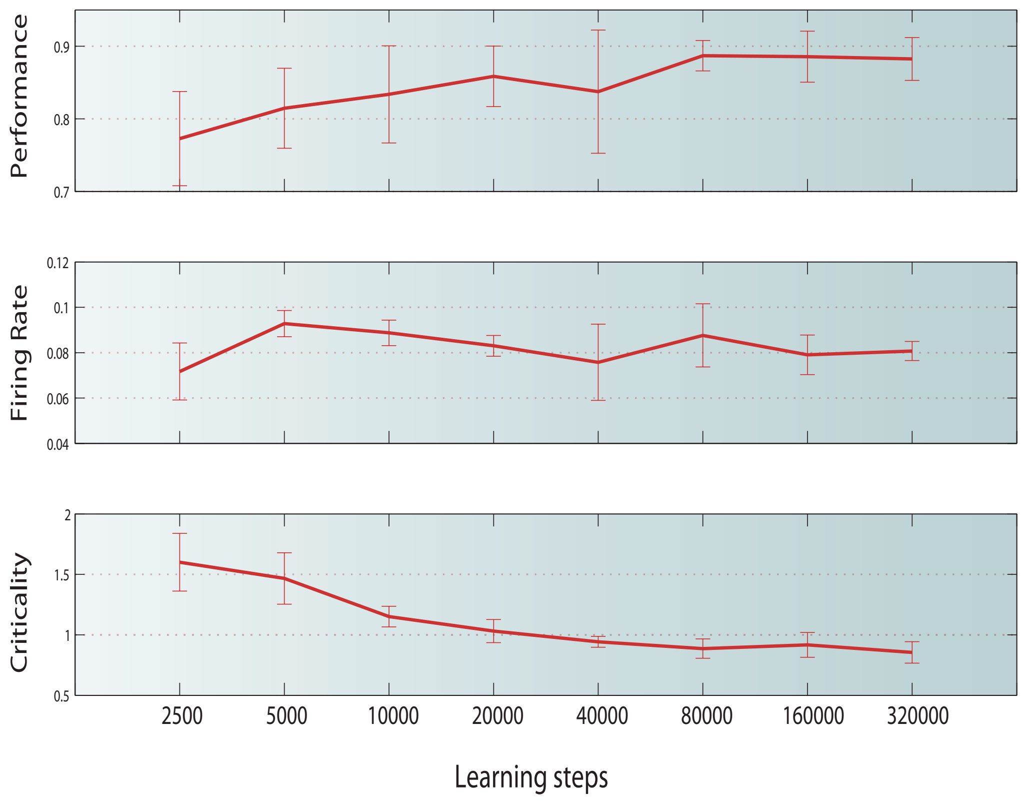


**Figure 5.9:** Impact of the duration of the learning phase in dynamic reservoirs shaped by all 3 plasticity mechanisms in performance, mean firing rates and criticality. The results depict the means and standard deviations obtained from 10 simulations of networks with 200 neurons.

## 5.3 Changing transfer function

Modifying the neuron's transfer function will significantly impact network behavior. In this section, we simulate example networks with $N^E = 200$ in the presence of a structured input ($n = 8$), using the two transfer functions described in subsection 4.1.2. The clipped linear function used in these simulations has its slope set to 1 (see performance results below (subsection 5.4.2)). In both cases, the networks have their initial parameters set according to the results in section 5.1.

Note that the activation state of neurons with a clipped linear transfer function is real-valued, varying in the interval $[0, 1]$ and not binary, so these neurons cannot be seen as spiking neurons, hence the notation in the histograms (Figures 5.10 and 5.11). However, the plotted values (activity per time step and normalized activity) are calculated in the same manner as spikes per time step and firing rate, respectively.

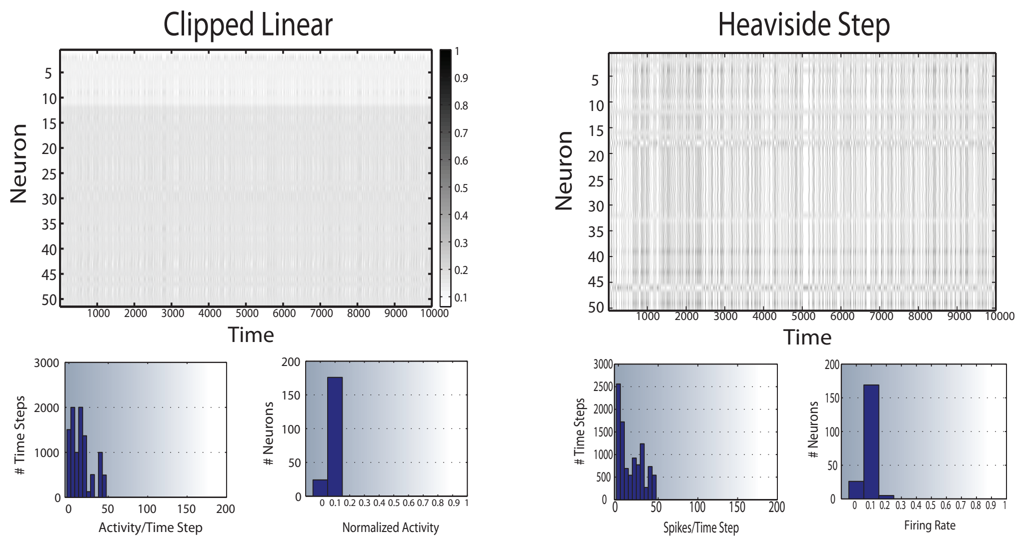


**Figure 5.10:** Comparison between network activity in 2 dynamic reservoirs with different transfer functions. The first row depicts a snapshot of the activity of 50 randomly chosen neurons. The bottom row contains activity distribution histograms.

Despite their differences, both types of dynamic reservoirs develop into similar activity patterns (given the constraints enforced by plasticity)(Figure 5.10). The bigger differences are in the static reservoir case. Without plasticity

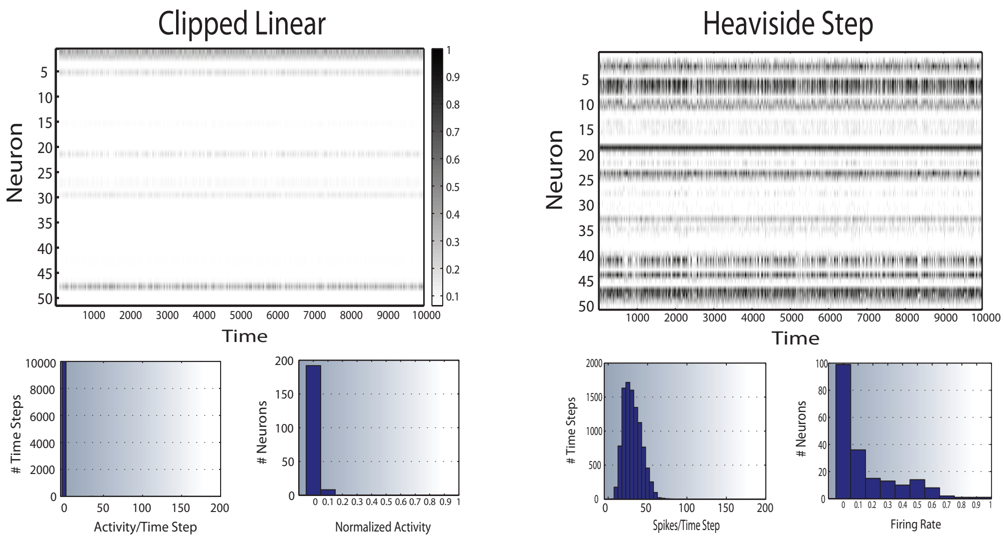


**Figure 5.11:** Comparison between network activity in 2 static reservoirs with different transfer functions. The first row depicts a snapshot of the activity of 50 randomly chosen neurons. The bottom row contains activity distribution histograms.

shaping network's parameters, the clipped linear network shows very low activity, with most neurons silent throughout simulation (Figure 5.11).

Another interesting difference between the results obtained for reservoirs with the two transfer functions lies in the modifications enforced by the IP rule in the unit's thresholds. The other plasticity mechanisms operate similarly for both, but IP, in the clipped linear case, lowers the thresholds of most units to 0, or even to negative values in some cases, while raising a few unit's thresholds to values close to 1 (Figure 5.12).

The threshold determines whether the pre-activation state of each neuron is positive or negative (Equation 4.1). In 'binary' neurons, this is translated into firing (1) or not (0), whereas in the clipped linear case, it determines whether the neuron's output is 0, or $\neq 0$ (in which case, the output is a value in $[0, 1]$). Hence the differences in the observed variation in threshold values. In both cases, the objective is to obtain an average neuronal activity of 0.1, but the variation in threshold values leading to the completion of this objective varies depending on the transfer function that determines the value of each variable at each time step.

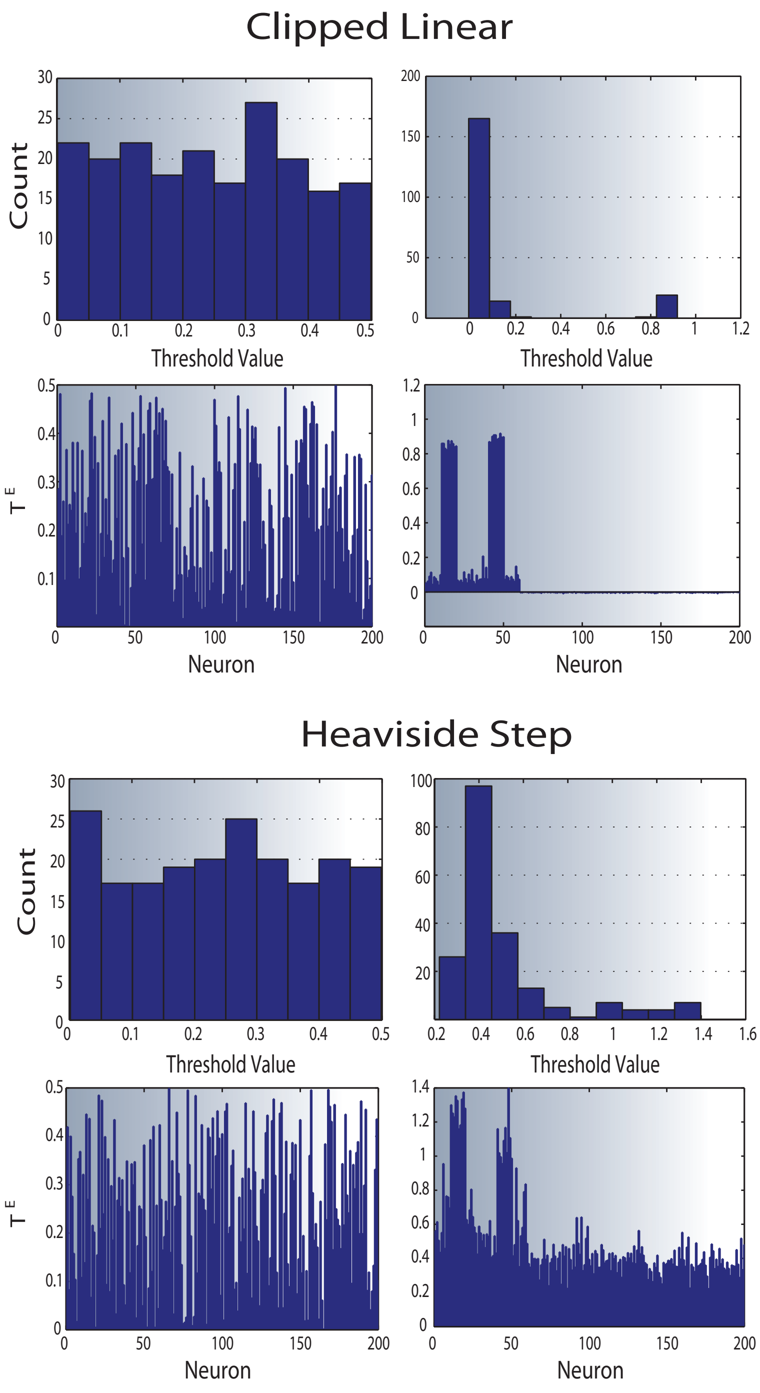


**Figure 5.12:** Variations in the unit's thresholds introduced by plasticity. Initially distributed similarly, the effect of IP on the unit's thresholds is markedly different for the two types of networks. Depicted are the threshold distribution histogram (pre- and post-plasticity), in the top row, and the individual neuron's threshold values (pre- and post-plasticity).

## 5.4 Prediction Performance

In this section we assess prediction performances, i.e., the ability of a trained readout to correctly predict the next input in the sequence, using only the network's internal states (see subsection 4.2.2).

To do so, the network activity patterns must reflect the temporal structure of its input sequence. As described in section 4.1, the structured input sequences are composed of random alternations of two "words" with $n+2$ letters each ($a$, $n\times b$, $c$ and $e$, $n\times d$, $f$). In order to allow the readout mechanism to predict the next symbol in the sequence, network activity must reflect how many repetitions of $b$ or $d$ it has been presented with, i.e., identical inputs ($b$ and $d$) have to be mapped onto distinct internal representations, based only on temporal context (recent input history).

The word length ($n$) can be seen as a measure of task difficulty. The longer the symbol sequence, the harder it is for the network to retain this information in memory. Also, because the order of the words is randomized, the prediction of the first letter of each word in the sequence is random. These effects become apparent if we plot the prediction errors (Figure 5.13). For small values of $n$, the only significant errors occur in the first letters of each word and in the first repetition of $b$ or $d$ (in dynamic reservoirs). Increasing task difficulty, also increases the error, particularly in the last symbols of the sequences, because it becomes harder for the network to retain the full sequence information. Memory for how many repetitions were presented fades gradually, leading to the steady increases in prediction errors.

For the largest word lengths (last row in the figure), the differences between dynamic and static reservoirs fade (similar errors occur) and it becomes difficult to discern the two results.

### 5.4.1 Network size

Mathematical studies demonstrate that a robust separation of trajectories is directly related to the dimensionality of the state space [18]. The higher the dimensionality of the state space, the easier the linear separation

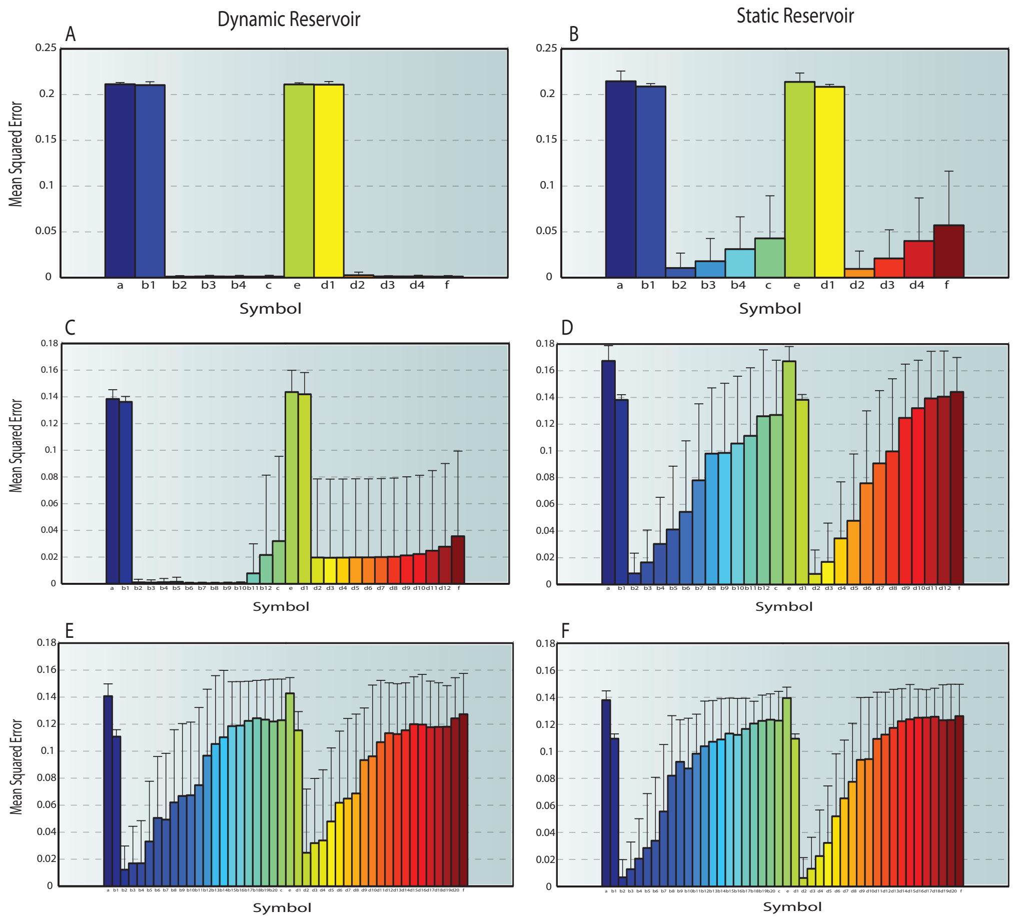


**Figure 5.13:** Root mean squared error for each input symbol in the sequences, obtained from 10 simulations of networks with $N^E = 800$ and word lengths of 4 (A, B), 12 (C, D) and 20 (E, F). These networks have a memory of input history that fades with the increase in sequence length, leading to ever greater prediction errors. The presence of plasticity leads to more accurate predictions.

of trajectories becomes.

Given that in the current model, the state space is $N^E - \textit{dimensional}$ (because each neuron in the reservoir is described by a single state variable), increasing $N^E$ leads to a better separation of the trajectories of active states created in response to the input sequences, making it easier for the readout mechanism to discern the different symbols in the sequence (Figure 5.14). Regardless of being dynamic or static, all reservoirs achieve increasing performance results with increasing $N^E$.

Dynamic reservoirs display a clear performance advantage, that is consistent across the different network sizes. This difference does, however, diminish in the larger networks.

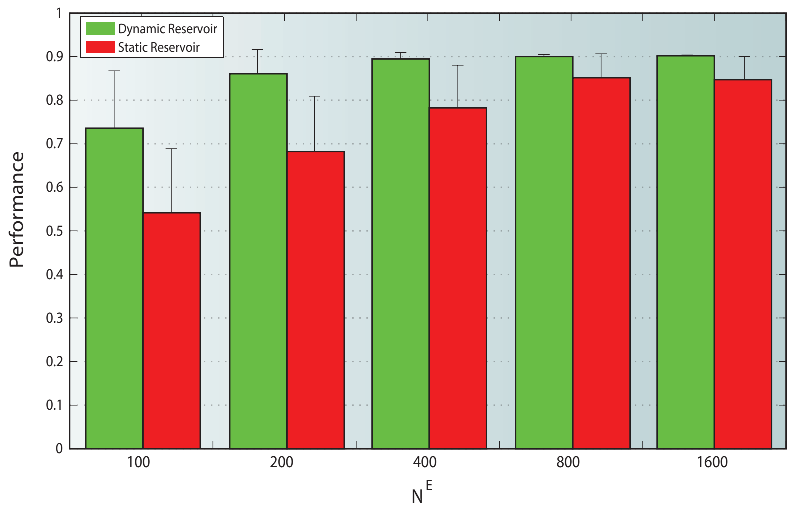


**Figure 5.14:** Impact of network size on performance. Depicted are the mean performances obtained from 100 simulations of networks with $N^E = 100$ and 200, and 10 simulations of networks with $N^E = 400$, 800 and 1600 in a task with $n = 8$. For the same task difficulty, larger networks always outperform smaller ones.

Increasing the dimensionality of the state space also has an impact on the network's memory capacity (Figure 5.15). In the smallest networks ($N^E = 100$), the peak performance corresponds to the shortest word length, decreasing from then on. Increasing network size, augments the memory span, with the best performances of dynamic reservoirs being achieved at longer sequence lengths. In the higher-dimensional case analyzed ($N^E = 800$), the top performance is achieved at $n = 14$, and only begins to decline from that point on. So, the ability to retain information about sequence history for long sequences correlates with the dimensionality of the state space to which those sequences are projected.

### 5.4.2 Transfer Function

All the performance results thus far discussed have been acquired using networks of 'binary' neurons (Heaviside transfer function). Now we analyze

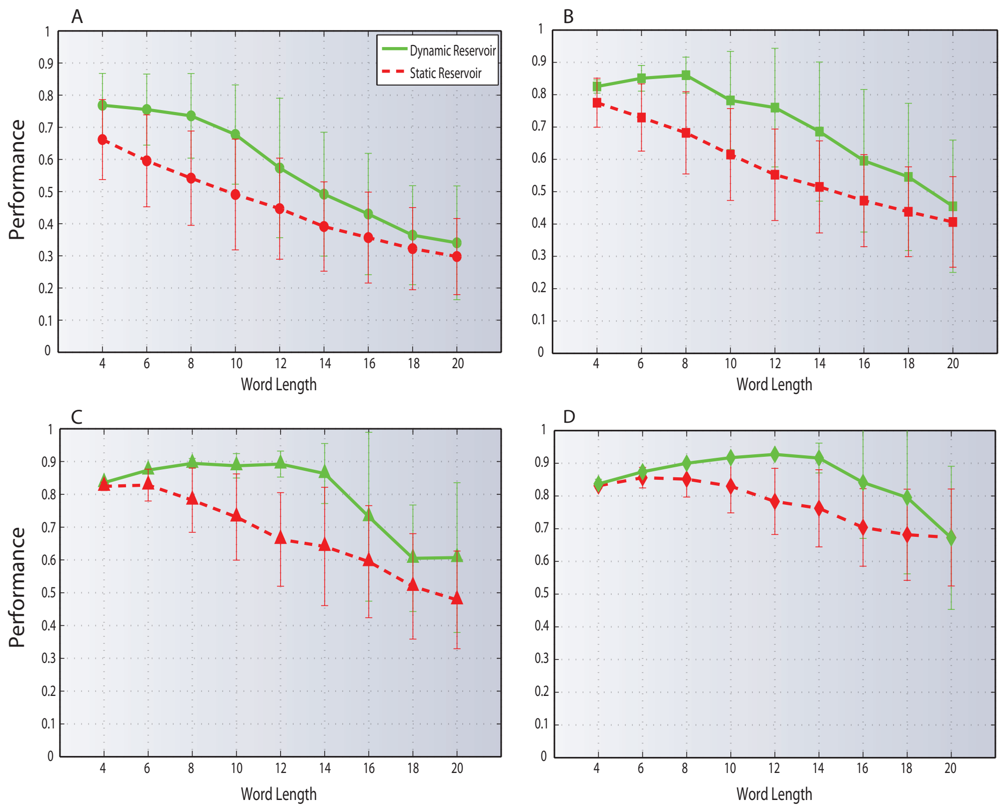


**Figure 5.15:** Mean performances (and standard deviations) obtained from 100 simulations of networks with $N^E = 100$ (A) and $N^E = 200$ (B) and 10 simulations of networks with $N^E = 400$ (C) and $N^E = 800$ (D).

the impact of changing the neuron's transfer function to clipped linear. We start the analysis by assessing the impact of the function's slope in prediction performance. For that purpose, we simulate 10 networks with $N^E = 200$ in the presence of a structured input with $n = 8$.

As the slope increases, the transformation that occurs in each neuron will approximate a step function (Figure 5.16). The results obtained never significantly deviate from those of the step function. There are, however two interesting results. With a slope of 1, dynamic reservoirs achieve a performance above average (0.9), and this result is very consistent. In fact, there is no standard deviations, all 10 simulations achieved a performance of exactly 0.90. In the case of static reservoirs, a similar phenomenon occurs at a slope of 5, although in this case there is some variation among simulations.

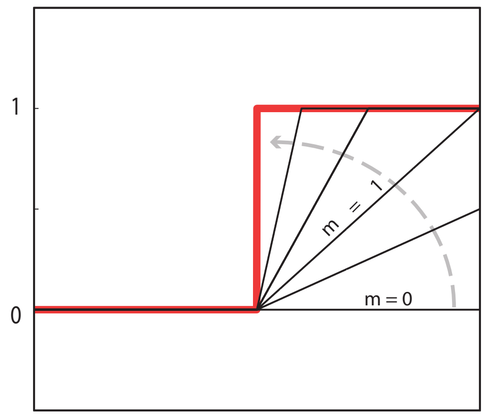


**Figure 5.16:** Changing the slope of the clipped linear transfer function, to approximate the step function (red).

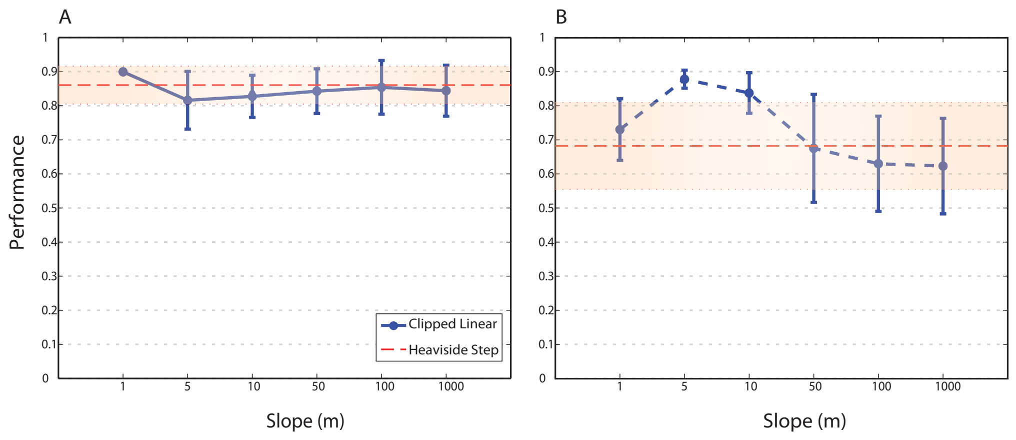


**Figure 5.17:** Prediction performance in networks of linear neurons (clipped linear transfer function) obtained from 10 simulations per condition for dynamic (A) and static (B) networks with 200 neurons, in a task with $n = 8$. As the slope increases (approximating the step function), performance results also get closer to the results obtained for the equivalent networks with 'binary' neurons (shaded area).

With the clipped linear as the neuron's transfer function, the network's active states are likely to contain more information about the input that caused them as well as the sequence history. Given the results presented in Figure 5.17 and section 5.3, we decided to further assess the performance and memory capacity of these networks with the slope of the transfer function set to 1 in dynamic reservoirs and 5 in static reservoirs.

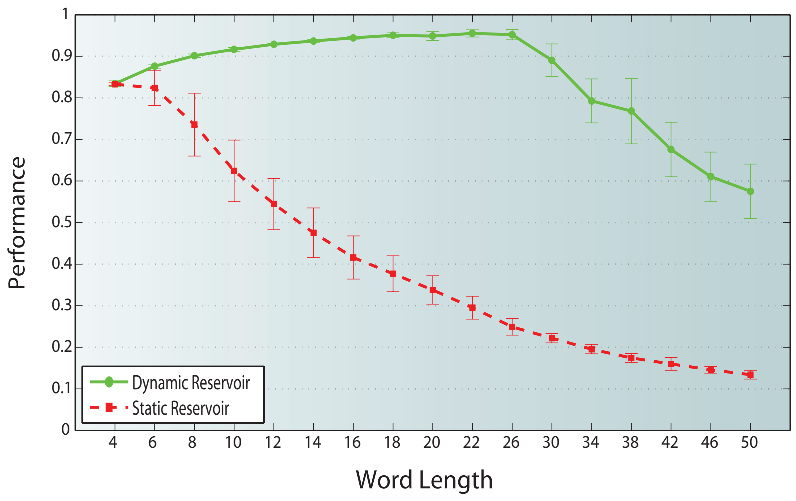


**Figure 5.18:** Prediction performance (means and standard deviations) obtained from 10 simulations per condition for networks with $N^E = 200$ and clipped linear as the neuron's transfer function. Dynamic reservoirs achieve very high performances, and performance only starts to decline for word lengths 26.

The results show an unequivocal advantage of dynamic reservoirs with linear neurons over even the best performing networks of binary neurons, in both performance and memory capacity (Figures 5.18 and 5.19). Shaped by plasticity, these networks achieve amazing performance results, maintaining a consistent peak performance above 0.95 for sequence lengths of up to 26. Prediction performance only begins to decline for $n \geq 30$.

This is not the case for the static reservoirs, that are only capable of outperforming the corresponding 'binary' networks for the smaller sequence lengths (up to 12), and whose performance rapidly declines.

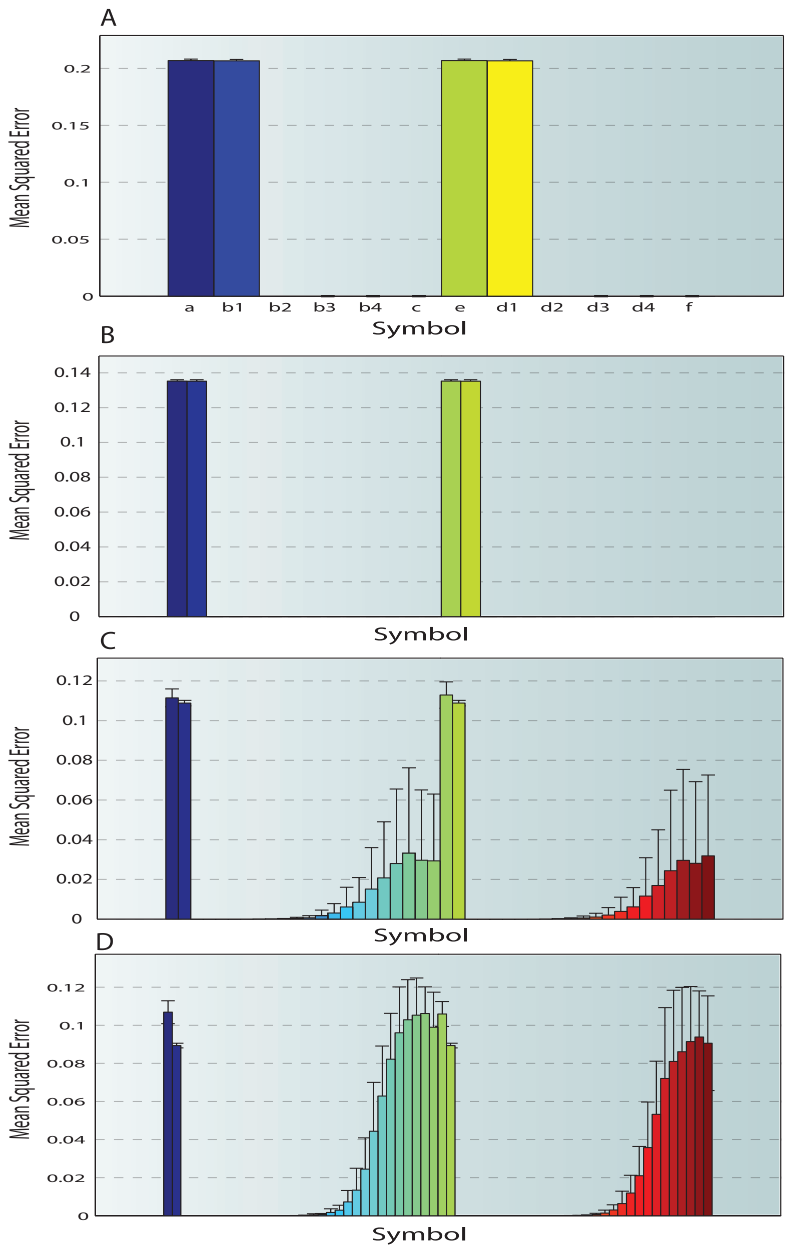


**Figure 5.19:** Mean squared error associated with each symbol in the sequence, in dynamic reservoirs with 'linear' neurons and $N^E = 200$, for word lengths of 4(A), 12(B), 20(C) and 30(D). Readout predictions are extremely accurate even for larger sequences, when compared to the best performing dynamic reservoirs with 'binary' neurons (Figure 5.13).

## 5.5 Internal Representations

To assess whether the networks acquire the input structure, i.e., whether different input conditions are reflected in spatially separable trajectories, we performed agglomerative clustering and PCA on the internal network states ($x'$), according to the description in subsection 4.2.1. For the clustering algorithm, 10.000 steps of network activity (networks with $N^E = 200$ and initial parameters set according to section 5.1) were considered (after 50.000 steps of plasticity, in the case of dynamic reservoirs). The clustering stops when the last 20 clusters are determined, corresponding to the 20 input symbols (given that in these experiments $n = 8$). The objective of this analysis is to assess how the network's state-space trajectories reflect the input structure. We analyze static and dynamic reservoirs of 'binary' neurons, as well as a dynamic reservoir of 'linear' neurons.

A result that is common to all reservoir types analyzed is the overlap between the representations of the first symbols of each sequence ($a$ and $e$). As explained in the previous section, the two sequences are randomly ordered, so these symbols are equally likely to appear. Network activity reflects that fact, the trajectories of active states generated in response to these symbols are similar and the network cannot discern between the two. Despite being grouped together in the same activity clusters, these representations do not settle into a single cluster. They are distributed throughout 4 different clusters, in static reservoirs (Figure 5.20)) and in 2 different clusters, in both dynamic reservoirs (Figures 5.21 and 5.22).

The active states generated in static reservoirs are distributed in fewer clusters than their dynamic counterparts, with several input symbols generating similar activity patterns (as many as 9 input symbols are grouped in the same cluster). Nonetheless, this distribution is not random and clearly reflects the input structure, with each 'word' being almost totally grouped in single clusters (Figure 5.20). Cluster 2 contains the state representations of $d1$ to $f$ (almost the entire second word) and cluster 9 contains the state representations of $b3$ to $c$ (almost the entre first word). Given the spatial proximity between the activity vectors generated in response to each individual symbol, discerning between them is harder than in the case of dynamic reservoirs, thus explaining the difference in readout prediction performance.

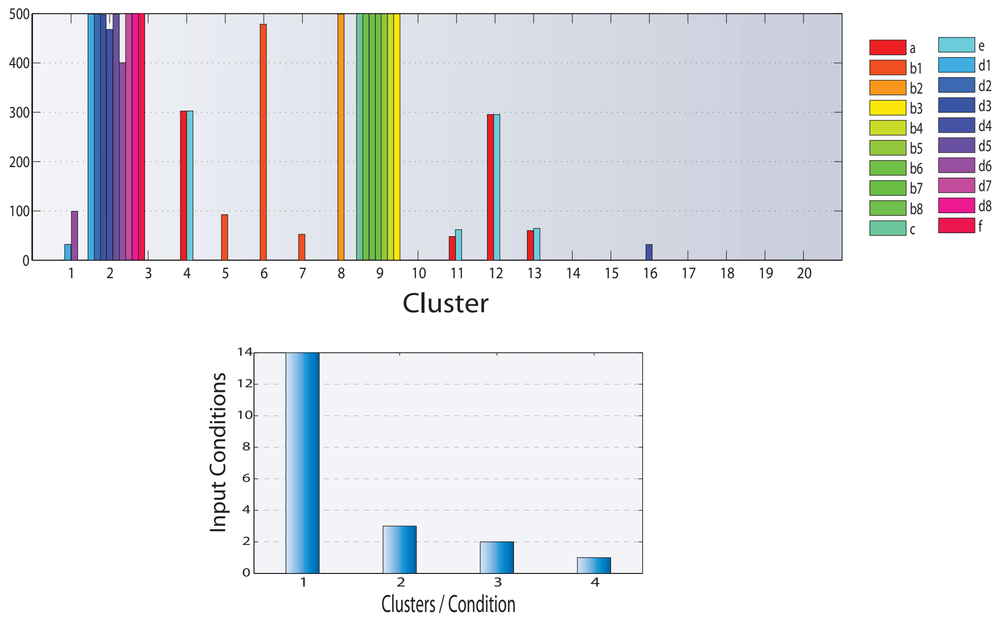


**Figure 5.20:** Results of cluster analysis in static reservoirs. 2 of the clusters contain several input conditions, corresponding to some of the repetitions of $b$ or $d$. The network is, however, capable of discerning between the 2 different sequences (words). Network activity in response to 14 of the 20 input symbols converges to a single cluster. The remaining 6 symbols create trajectories that are spread into 2, 3 or 4 activity patterns.

Plasticity fundamentally changes network behavior and, consequently, the state representations of the input conditions. Dynamic reservoirs are better capable of mapping individual input symbols to discernible activity patterns (Figure 5.21). Several symbols generate distinct, unique network responses ($b3$, $d3$, $d5$, $d6$, $d7$, $d8$ and $f$); other symbols generate patterns that, although discernible, can be grouped in 2 different clusters ($d1$, $d4$, $b1$ and $b2$), while the remaining representations overlap in the same cluster. Activity cluster 4 contains the state representations of the last 6 symbols of the first sequence ($b4$ to $c$). Interestingly, clumped in this same cluster are also representations of symbols $a$ and $e$.

Dynamic reservoirs display different characteristics and, apparently different 'methods' of representing the input, depending on the neurons' transfer

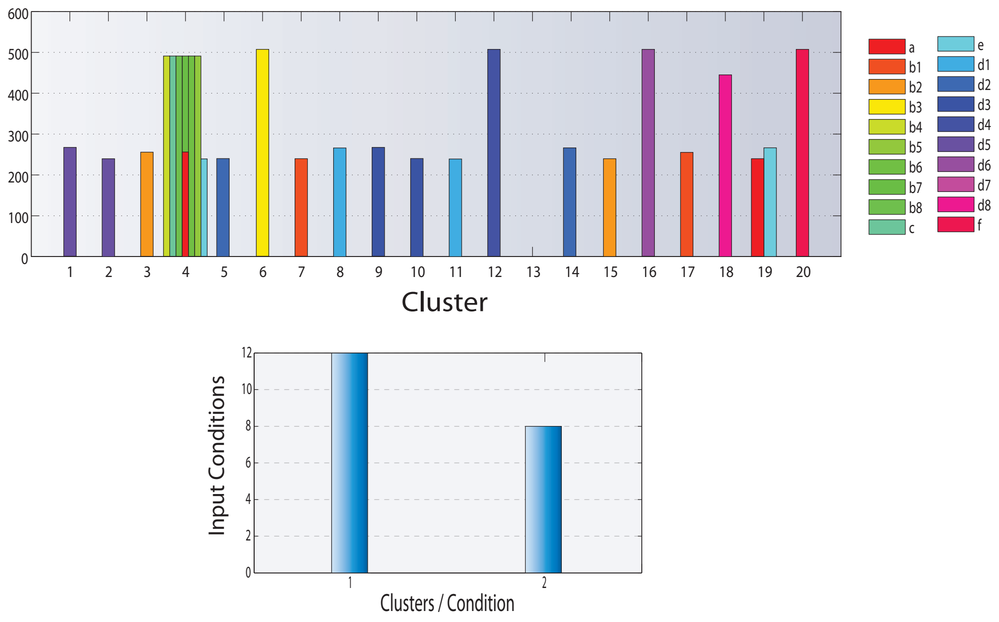


**Figure 5.21:** Results of the cluster analysis in dynamic reservoirs. Cluster 4 contains almost all repetitions of $b$, but $a$ and $e$ are also placed in that same cluster. Most symbols' representations fall within discernible activity patterns (single cluster), particularly the symbols of the second 'word'. 8 of the 20 symbols are projected to 2 different activity clusters.

function used. Whereas in the previous cases analyzed, the state representations appear to be grouped according to sequence (the word to which each symbol belongs), in the case of clipped linear neurons, the network appears to code for temporal order, i.e., the state representations of the symbols are grouped, not according to the word they belong to, but according to their position within the word (Figure 5.22). Cluster 2 contains the representations of $b6$ and $d6$, cluster 10 contains the representations of $b5$ and $d5$ and cluster 1 contains the representations of the final portions of both words ($b7$, $b8$, $c$, $d7$, $d8$ and $f$). This is an interesting result and it may underly the outstanding performance results obtained with these networks. Although the symbol representations are not entirely separated, this method of grouping network activity may facilitate linear separation and increase the memory span.

The only symbol that generates a distinct and unique representation is

$d3$ (cluster 5). $b3$, $b4$, $d4$, $d2$ and $b1$ generate representations that fall into 2 different cluster, but in close spatial proximity, it is likely that, if we analyzed longer time series of activations, the network would ‘settle’ into a single state representation for each of those symbols. The only ‘confusion’ is in the representations of the first and second repetitions of $b$ and in $a$ and $e$ (for the reasons previously outlined).

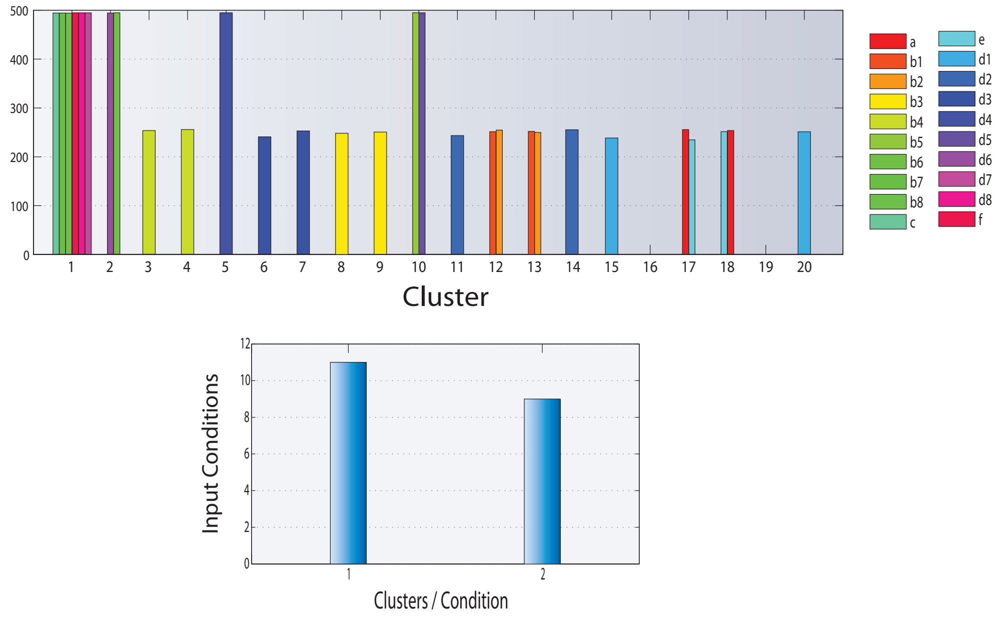


**Figure 5.22:** Results of the cluster analysis in dynamic reservoirs made up of ‘linear’ neurons. Cluster 4 contains almost all repetitions of $b$, but $a$ and $e$ are also placed in that same cluster. Most symbols’ representations fall within discernible activity patterns (single cluster), particularly the symbols of the second ‘word’. 8 of the 20 symbols are projected to 2 different activity clusters.

In any case, sequence history is clearly reflected in network activity, either accounting for which of the 2 sequences is present, or which position each symbol occupies in the sequence.

The results of PCA, performed on 50.000 steps of network activity, can complement the previous observations. By scattering the state representations of each input symbol along the projection space of the first principal components, the differences between static and dynamic reservoirs become clearer. In static reservoirs there is a significant overlap between these different network states (Figure 5.23). Each input symbol produces a cloud of network states depicting the existence of several distinct internal representations that are not, in most cases, well separated from the other symbols in the sequence. In accordance with the results of the cluster analysis, the representations of the two symbol sequences projected in PCA space are distinct (albeit overlapping).

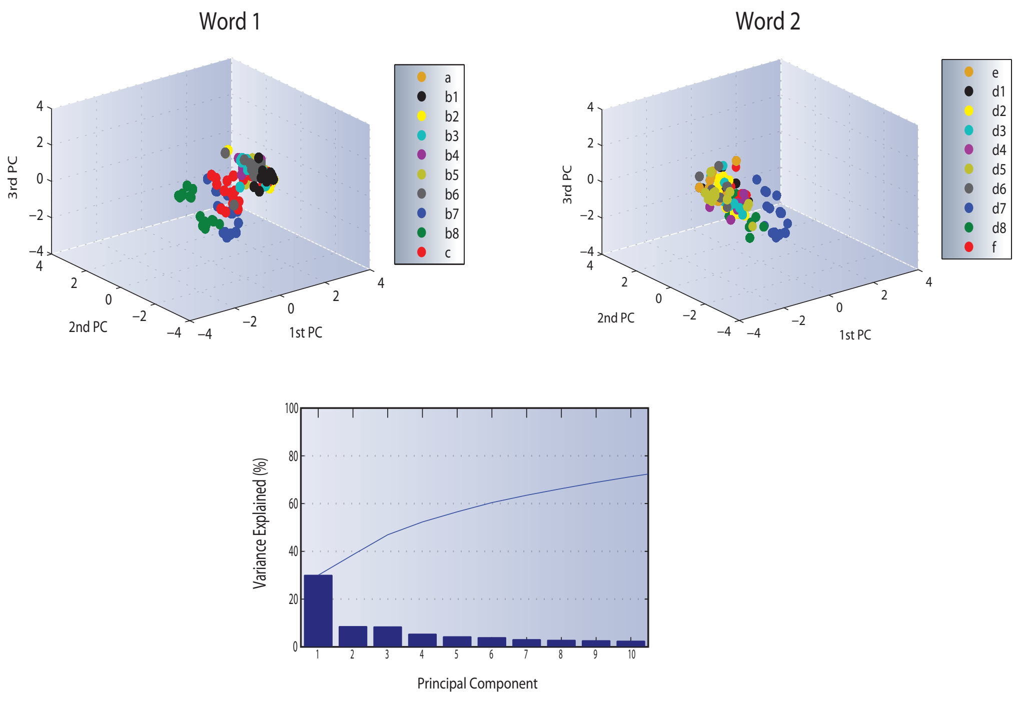


**Figure 5.23:** Results of PCA performed on static reservoirs. Top row: Projection space of the first 3 principal components. There is a substantial overlap between the network states created in response to each input symbol, when projected to the PCA space. The representations of the two symbol sequences are distinct from each other. Bottom row: amount of variance explained by the first few PCs.

Dynamic reservoirs, on the other hand, learn stabler representations of the individual input symbols, although no individual symbol representation forms a single cluster in PCA space (Figure 5.25). Particularly well separated are the last symbols of each sequence, $b8$, $d8$, $c$ and $f$, whose state representations are scattered further away from the remaining symbols' representations.

There is, in both symbol sequences, a tighter 'central' cluster of active states, corresponding to the first few symbols (particularly $a$, $b1$ and $e$, $d1$). This result is consistent with the greater readout prediction errors for these first symbols (Figure 5.13).

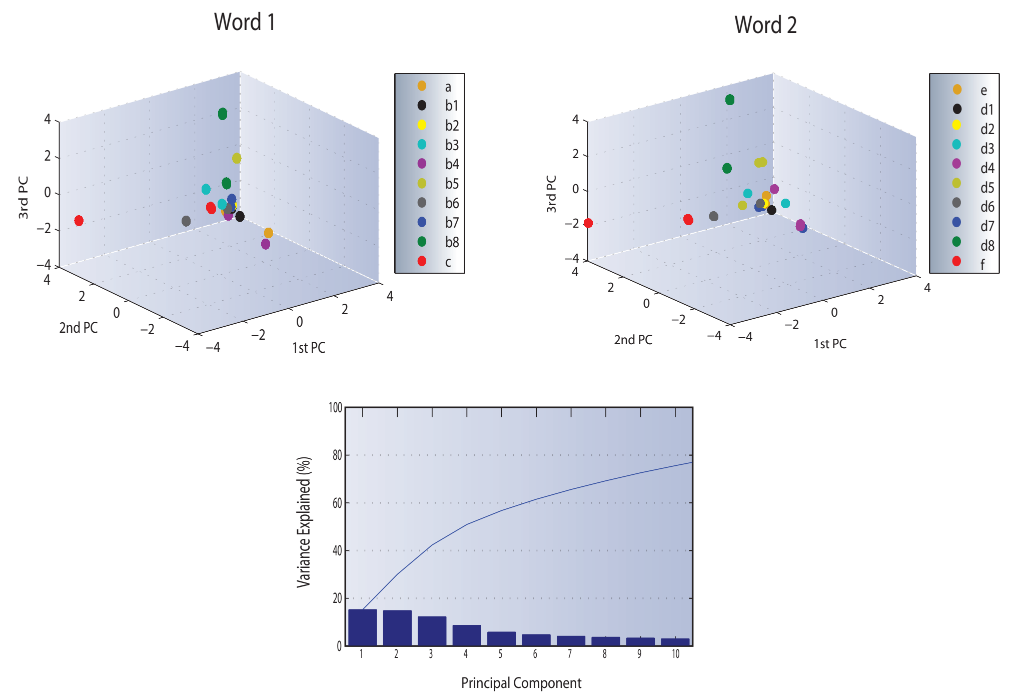


**Figure 5.24:** Results of PCA performed on dynamic reservoirs. Top row: Projection space of the first 3 principal components. The majority of the input symbols creates discernible and well separated network states. There is a significant overlap in 'central' cluster, but some symbols' representations are distributed further from it, facilitating linear separation. Bottom row: amount of variance explained by the first few PCs. The first 10 PCs account for over 70% of the variance in the data.

As previously highlighted in the results of the cluster analysis, dynamic reservoirs whose neurons' transfer function is the clipped linear, develop timing-based representations, effectively 'counting' the symbol positions within the words. This effect is even clearer when the state representations are projected onto the space spanned by the first 3 PCs. The two symbol sequences are indistinguishable from each other, but the symbols' positions are represented by similar active states in both words (identical state space trajectories). The distinct representations of each individual symbol are also closely clustered, allowing a much better linear separation. As in the previous cases, the greatest overlap exists in the representations of the first symbols ($a$, $b1$ and $e$, $d1$). Also, the organization of network states is more ordered, as suggested by the amount of variance explained by the first PCs. The first 8 PCs account for almost 100% of the variation in the data. In fact, just the first 3 PCs encompass as much of the sample variance as the first 10 in the reservoirs of the 'binary' type.

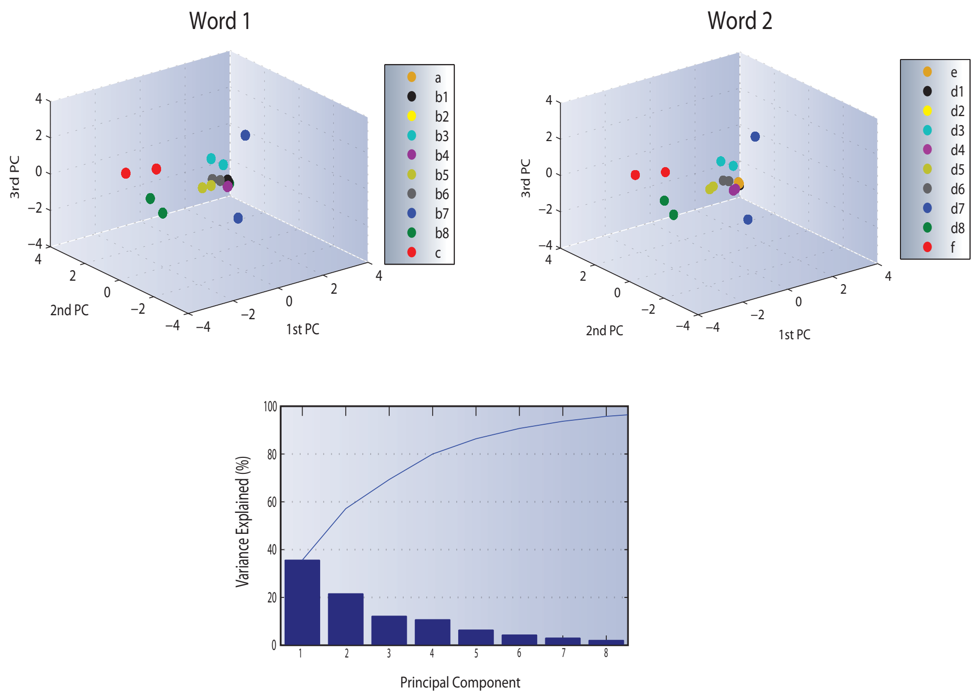


**Figure 5.25:** Results of PCA performed on dynamic reservoirs made up of 'linear' neurons. Top row: Projection space of the first 3 principal components. Input symbols create well separated network states, some overlap is present mainly in the states corresponding to the symbols $a$, $b1$, $e$ and $d1$. Bottom row: amount of variance explained by the first few PCs. The first 8 PCs capture almost 100% of the variance.

# 6 Discussion

*The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' (I found it!) but 'That's funny'*

– Isaac Asimov

Understanding information processing in the neocortical circuitry and the laws that govern it constitutes a tremendous task. Ubiquitous variability and complexity hinder the analysis, but the essence of the brain's mechanisms can be accounted for with the help of simplified model systems.

New theoretical frameworks are emerging that can explain the generalized ability of neural circuits to process, classify and discriminate stimuli in real time. This ability is the result of an interaction between the external stimulus and the network's internal state. It is this state dependency that allows neural networks to encode time and spatiotemporal structure (in the form of evolving state-space trajectories). This implies that network activity patterns at any given time, reflect, not only the current input, but a nonlinear combination of recent stimulus history (a fading memory).

According to this general framework, downstream 'read-out' neurons can learn to decode the trajectories by appropriately adjusting the synaptic connections they receive from the circuit's neurons. Different trajectories of active states encode different features of the stimulus structure.

An extremely relevant issue that needs to be accounted for is the fact that the internal state of a cortical circuit is dynamically shaped by a host of plasticity mechanisms, subserving particular functions, and operating at different spatial (from the individual synapse to the entire network) and temporal scales (from hundreds of milliseconds to several days or longer). These mechanisms shape the state-space trajectories, optimizing the brain's computational efficiency. Although diverse, the set of plasticity mechanisms is finite and great progress has been made in identifying them and deciphering how they operate. A lot is known about how these mechanisms operate individually, but understanding their interaction has only recently come into focus.

Throughout this thesis, we have analyzed the properties of simple reservoir networks, built according to the general principles of reservoir computing (state-dependent models), while they evolved through the combined effect of three plasticity mechanisms (STDP, SN and IP) in response to external input sequences designed to test the reservoir's memory capacity and the dependence of the internal representations on temporal context.

Plasticity was shown to be clearly advantageous in improving the network's capacity to separate state-space trajectories generated in response to the individual symbols, thus improving the readout prediction performances, and in stabilizing network activity towards a 'healthy' firing regime, suitable for the development of relatively stable internal representations. This advantage, however, was only maintained if all three mechanisms were present.

The synaptic modifications produced by STDP require homeostatic regulation (provided by SN), otherwise the network develops excessive spiking activity, with some neurons firing at every time step (if STDP is the only mechanism present) or with synchronous activity bursts (if IP is also present). The intrinsic plasticity rule was shown to be of particular importance, because of its stabilizing effects, achieved by spreading the network activity evenly throughout the neurons (by modifying individual neurons' threshold values). So, only in combination do these mechanisms allow the network to stabilize its activity, settling into a regime suitable for acquiring stable internal representations of the input.

By projecting the symbol sequence information to a higher dimensional state-space, larger networks are better able to separate the trajectories, as well as retain information about longer sequences. So, as excepted, performance and memory capacity are directly related to network size, in both static and dynamic reservoirs.

Interesting results were obtained by changing the neurons' transfer functions. The clipped linear function allowed the neuron's to develop a smoother input-output transition, increasing the overall network capacity to reflect the temporal properties of the input sequences in its state dynamics. The presence of plasticity was shown to be even more relevant for this case. In fact, without plasticity, reservoirs of the clipped linear type, displayed negligible performance improvements, whereas when plasticity was present these network's separation properties and memory capacity were far better than their 'binary' counterparts.

Dynamic reservoirs of clipped linear neurons were shown to separate the symbols based only on the timing of their appearance in the sequence, i.e., their position within each word, which is a significantly distinct result, given that the other reservoirs mostly separated the input symbols based on which of the two sequences they belonged to.

Overall, the more relevant result was the fact that all dynamic reservoirs analyzed were capable of generating distinct activity patterns in response to similar input symbols (the repetitions of $b$ and $d$), based only on temporal context. Without plasticity shaping its properties, static reservoirs displayed a smaller capacity to do so, with the majority of the symbol repetitions being lumped into the same activity patterns. So, the combination of several plasticity mechanisms is clearly beneficial in shaping neural network's response characteristics and allowing them to naturally process temporal structure.

# A
# Methods Description

## Agglomerative Hierarchical Clustering

Hierarchical clustering is a family of methods that is used to build a cluster hierarchy. There is a concept of ordering involved. The algorithms find successive clusters, using previously established ones. The order is driven by how many observations can be combined at any given point, and how the significant distance is determined.

These algorithms are usually either agglomerative or divisive. *Agglomerative* algorithms begin by considering each element as a separate cluster and merging them into successively large clusters by evaluating the pairwise distances between clusters, according to some similarity criteria, until all the data is agglomerated in one cluster in a "bottom-up" approach. *Divisive* algorithms work the other way around, considering all observations as a single cluster and splitting them recursively in a "top-down" manner, until only clusters of individual objects remain.

The most important step in cluster analysis is the selection of an appropriate metric, which will determine the clusters' elements similarity or dissimilarity and how they are going to be combined or split, thus influencing the shape of the clusters. Because no provision is made for relocating "misplaced" objects in these methods, a correct choice of metric is fundamental and the results should always be closely examined to ensure they make sense.

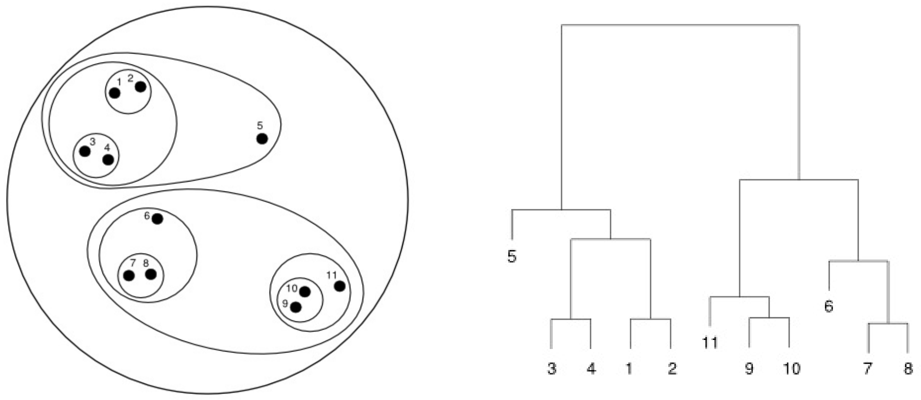


**Figure A.1:** Example of data clustering (left) and the most common way of representing it (dendrogram) (right).

The most commonly used **distance metrics** are:

**The Euclidean distance** is the most broadly used measure. Also called the two-norm distance, it has its origins on the Pythagorean Theorem. If $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ are two points in Euclidean n-space, the distance between them is

$$d(p,q) = ||p-q||_2\sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + ... + (q_n-p_n)^2}$$

**The Manhattan distance** also known as the rectilinear distance, one-norm distance or city-block distance. If $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ are two vectors in n-dimensional real vector space, the distance between them is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes

$$d(p,q) = ||p-q||_1 = \sum_{i=1}^{n}(p_i - q_i)$$

**The Mahalanobis distance** also called the quadratic distance. It measures the separation between two groups of objects and it is mainly used in classification problems, where there are several groups (multivariate analysis) and the investigation concerns the affinities between

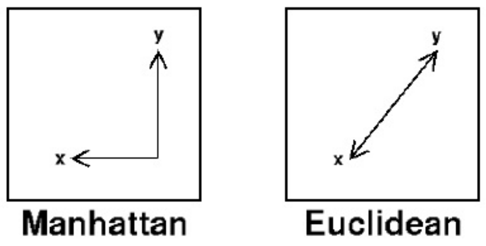


**Figure A.2:** Schematic representation of the Euclidean and Manhattan distances in two-dimensional space.

them. It is better adapted than the Euclidian distance to settings involving non spherically symmetric distributions. If $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ are two points in n-dimensional space and $C$, the covariance between them,

$$d^2_M(p, q) = (p - q)^T C^{-1}(p - q)$$

**The Chebychev distance** or maximum metric, determines the distance between two vectors, or points, as the maximum distance along any coordinate dimension. It is induced by the supremum norm or uniform norm. If $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ are two points in n-dimensional space

$$d_{Chebychev}(p, q) = max_i(|p_i - q_i|) = \lim_{k\to\infty}(\sum_{i=1}^{n} |p_i - q_i|^k)^{1/k}$$

**The Hamming distance** is mostly used for non-numeric data. It measures the minimum number of substitutions required to change one string into another, i. e., the number of positions at which the corresponding symbols are different. It was introduced in information theory as an error detection/correction code[42].

$$d(p, q) = (\sharp(q_i \neq p_i)), \forall i \in [1, n]$$

The main **distance criteria** embedded in most clustering algorithms are:

**Single linkage** also known as nearest neighbor approach, it considers the distance between any two clusters as the shortest distance from any point in one cluster to any point in the other. Two clusters are merged based on the single shortest or strongest link between them. If $\mathcal{A}$ and $\mathcal{B}$ are two clusters:

$$\min(d(x,y) : x \in \mathcal{A}, y \in \mathcal{B})$$

**Complete linkage** similar to single linkage, but based on a furthest neighbor approach, considering the maximum distance between clusters. The maximum distance between any two individual observations is represented as the smallest sphere that can enclose them:

$$\max(d(x,y) : x \in \mathcal{A}, y \in \mathcal{B})$$

**Average linkage** determines the mean distance between the elements in each cluster:

$$\frac{1}{|\mathcal{A}|.|\mathcal{B}|} \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{B}} d(x,y)$$

**Centroid linkage** this criterion is based on measuring the Euclidean distance between the centroids of two clusters:

$$d(x,y) = ||\bar{x}_i - \bar{y}_i||_2 \forall i \in [1,n],$$

where $||\,||_2$ is the Euclidean Distance and $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

**Ward's criterion** different from the previous ones, because it uses an analysis of variance to evaluate the distances between two clusters. It basically attempts to minimize the sum of squares of any two hypothetical clusters being formed:

$$d^2(x,y) = x_i y_i \frac{||\bar{x}_i - \bar{y}_i||_2^2}{(x_i + y_i)}$$

**NOTE** One can decide to stop the clustering process either when the clusters are too far apart to be merged (distance criterion) or when there is a sufficiently small number of clusters (number criterion).

## Principal Component Analysis

Principal component analysis (PCA) is a method of identifying patterns in large data sets and reshaping those data sets as to highlight the similarities and differences of their elements. It is a powerful technique for analyzing high-dimensional data, where direct graphical representation is not always feasible. In large data sets it is important to reduce the number of observed dimensions to a smaller number of artificial dimensions (the principal components) that account for most of the variance in the observed data. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. The biggest possible amount of variation will be crammed into the smallest possible amount of dimensions. The data set is thus compressed, while the redundant dimensions are ignored. The main objectives of PCA are to reduce the dimensionality of the data set without significant loss of information and to identify hidden and underlying patterns in the data set.

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \xrightarrow[\text{Reduce Dimensionality}]{} y = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} \; (K \ll N)$$

Suppose $x_1, x_2, \dots, x_M$ are $N \times 1$ vectors.

- The first step is to calculate the mean:

$$\bar{x} = \frac{1}{M} \sum_{i=1}^{M} x_i,$$

which is then subtracted from each individual observation $\Phi_i = x_i - \bar{x}$, leading to the creation of a $(N \times M)$ matrix $A = \begin{bmatrix} \Phi_1 & \Phi_2 & \dots & \Phi_M \end{bmatrix}$

- Based on A, the sample covariance is calculated:

$$C = \frac{1}{M} \sum_{n=1}^{M} \Phi_n \Phi_n^T = AA^T$$

- The eigenvalues of $C$ are calculated $\lambda_1 > \lambda_2 > \cdots > \lambda_N$, as well as and the eigenvectors $u_1, u_2, \dots, u_N$

- Since $C$ is symmetric, $u_1, u_2, \dots, u_N$ form a basis, meaning that any vector $(x - \bar{x})$ can be written as a linear combination of the eigenvectors:

$$x - \bar{x} = b_1 u_1 + b_2 u_2 + \cdots + b_N u_N = \sum_{i=1}^{N} b_i u_i$$

- The dimensionality reduction step consists of choosing the terms corresponding to the K largest eigenvalues:

$$\hat{x} - \bar{x} = \sum_{i=1}^{K} (K \ll N)$$

- The representation of $\hat{x} - \bar{x}$ into the basis $u_1, u_2, \dots, u_N$ is thus

$$\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix}$$

So, the overall linear transformation $\mathbb{R}^N \to \mathbb{R}^K$ performed by PCA is:

$$\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} = \begin{bmatrix} u_1^T \\ u_2^T \\ \dots \\ u_K^T \end{bmatrix} (x - \bar{x}) = U^T (x - \bar{x})$$

# B
# Matlab Code

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%      ######   #######  ########  ##    ##     %%%%%%%%%%%%%
%%%%%%%%%%%%%     ##    ## ##     ## ##     ## ###   ##     %%%%%%%%%%%%%
%%%%%%%%%%%%%     ##       ##     ## ##     ## ####  ##     %%%%%%%%%%%%%
%%%%%%%%%%%%%      ######  ##     ## ########  ## ## ##     %%%%%%%%%%%%%
%%%%%%%%%%%%%           ## ##     ## ##   ##   ##  ####     %%%%%%%%%%%%%
%%%%%%%%%%%%%     ##    ## ##     ## ##    ##  ##   ###     %%%%%%%%%%%%%
%%%%%%%%%%%%%      ######   #######  ##     ## ##    ##     %%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%  SELF-ORGANIZING RECURRENT NETWORK  %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%========================================================================
% author(s):    RD, KMP
% date:         15/08/2011
%-------------------------------------------------------------------------
clear all; clc;
%=========================================================================
% Simulation parameters
%-------------------------------------------------------------------------
nLSteps=50000;                        % number of learning steps;
nEstSteps=5000;
nPredSteps=5000;
nSteps=nLSteps+nEstSteps+nPredSteps;
%-------------------------------------------------------------------------
learning=1;                           % plasticity mechanisms on/off = 1/0;
wrdLgth=8;                            % word length n(+2); here n = 8;
strIn=1;                              % input structured or not = 1/0;
%------------------------ node type --------------------------------------
mcp=1;                                % McCulloch-Pitts nodes;
lif=0;                                % leaky integrate-and-fire nodes;
%-------------------- transfer functions ---------------------------------
hstep=0;                              % Heaviside step function;
```

```
clpln=0; gain=1;                      % clipped linear function;
sigm=1;                               % tanh
%-------------------- performance analysis --------------------------------
performance=1;                        % performance analysis on/off = 1/0;
perturbation=0;                       % perturbation analysis on/off = 1/0;
activity=1;
%--------------------------------------------------------------------------
%==========================================================================
% Network parameters
%--------------------------------------------------------------------------
nE=200; nI=.2*nE;             % number of units in the network;
nU=.05*nE;                    % number of symbol specific input neurons;

tEmax=0.5; tImax=1.4;          % maximum threshold values;
rSTDP=.001;                   % STDP adaptation parameter;
rIP=.001;                     % IP adaptation parameter and ...
hIP=2*nU/nE;                  % ... target rate;
lambda=10/(nE-1);             % the probability of a wEE-connection;
                              % defines the sparsity of wEE; in this case,
                              % the mean number of input (or output) wEE-
                              % connections = 10;
%--------------------------------------------------------------------------
%==========================================================================
% Network connectivity
%--------------------------------------------------------------------------
wEI=rand(nE,nI);                   % static weights I->E , uniform[0,1];
wEI=normalizeRows(wEI);            % normalize rows of wEI to 1;
%--------------------------------------------------------------------------
wIE=rand(nI,nE);                   % static weights E->I , uniform[0,1];
wIE=normalizeRows(wIE);            % normalize rows of wIE to 1;
%--------------------------------------------------------------------------
% Initialization of wEE
%--------------------------------------------------------------------------
wEE=zeros(nE,nE);                  % dynamic weights E->E;
ind=find(rand(nE,nE)≤lambda);     % apply wEE sparseness;
wEE(ind)=rand(length(ind),1);
wEE(eye(nE)>0)=0;                  % self-connections = 0;
wEE=normalizeRows(wEE);            % normalize rows of wEE to 1;
wEC=wEE>0;                         % to force wEE sparsity during learning;
%--------------------------------------------------------------------------
% Initial firing thresholds
%--------------------------------------------------------------------------
tE=tEmax*rand(nE,1);               % excitatory thresholds;
tI=tImax*rand(nI,1);               % inhibitory thresholds;
%--------------------------------------------------------------------------
%==========================================================================
% GENERATOR
%--------------------------------------------------------------------------
% inputMatrix generates input sequences
%--------------------------------------------------------------------------
```

```
[input,sY]=inputMatrix(nSteps,nU,nE,wrdLgth,strIn);
%------------------------------------------------------------------------
% pre-allocation
%------------------------------------------------------------------------
stE=zeros(nE,size(input,2)); stI=zeros(nI,size(input,2));
rDrive=zeros(nE,size(input,2));
stateE=zeros(nE,size(input,2)); stateI=zeros(nI,size(input,2));
pState=zeros(nE,size(input,2));
pertStateE=zeros(nE,size(input,2)); pertStateI=zeros(nI,size(input,2));
hammingD=zeros(1,nSteps);
%------------------------------------------------------------------------
% GENERATOR LOOP
%------------------------------------------------------------------------
for t=1:(size(input,2)-1)
  %---------------------- internal state update ---------------------------
  if mcp
    rDrive(:,t+1)=wEE*stateE(:,t)-wEI*stateI(:,t)-tE;
    stE(:,t+1)=rDrive(:,t+1)+input(:,t);
    stI(:,t+1)=wIE*stateE(:,t)-tI;
    % elseif lif
    % ...
  end
  %---------------------- output state update -----------------------------
  if hstep
    stateE(:,t+1)=stE(:,t+1)>0;
    stateI(:,t+1)=stI(:,t+1)>0;
    pState(:,t+1)=rDrive(:,t+1)>0;
  elseif clpln
    stateE(:,t+1)=clplin(nE,stE(:,t+1),gain);
    stateI(:,t+1)=clplin(nI,stI(:,t+1),gain);
    pState(:,t+1)=clplin(nE,rDrive(:,t+1),gain);
  elseif sigm
    stateE(:,t+1)=sigmoid(nE,stE(:,t+1),gain);
    stateI(:,t+1)=sigmoid(nI,stI(:,t+1),gain);
    pState(:,t+1)=sigmoid(nE,rDrive(:,t+1),gain);
  end
  %---------------------- plasticity mechanisms ---------------------------
  if learning
    if t<=nLSteps
      %--------- STDP ---------
      wEE=wEE+rSTDP*wEC.* ...
        (stateE(:,t+1)*stateE(:,t)'-stateE(:,t)*stateE(:,t+1)');
      wEE(wEE<0)=0;
      %--------- SN ----------
      wEE=normalizeRows(wEE);
      %--------- IP ----------
      tE=tE+rIP*(stateE(:,t)-hIP);
    end
  end
  %---------------------- perturbation analysis ---------------------------
  if perturbation
    pertStateE(:,t)=stateE(:,t);
    r=randperm(nE); r=r(1);
    pertStateE(r,t)=mod(stateE(r,t)+1,2);
    pertStateE(:,t+1)=(wEE*pertStateE(:,t)−wEI*pertStateI(:,t)−...
      tE+input(:,t))>0;
    pertStateI(:,t+1)=(wIE*pertStateE(:,t)−tI)>0;
    hammingD(t)=pdist2(stateE(:,t+1)',pertStateE(:,t+1)','hamming');
  end
end
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
% END OF GENERATOR LOOP
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
[Bhat] = estimatewOut(pState(:,(nLSteps+1):end),sY((nLSteps+1):end,:)...
    ,nEstSteps);

[perf,perfL2] = Performance (Bhat,sY((nLSteps+nEstSteps+1):end,:),...
    pState(:,(nLSteps+nEstSteps+1):end),nPredSteps);

fRate = mean(sum(pState(:,nLSteps+1:end))/nE);

%========================================================================
% SUB−FUNCTIONS
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
function [perf,perfL2] = Performance (Bhat,sY,pState,nSteps)
Y=sY(2:nSteps,:);
%−−−−−−−−−− time−series prediction −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
tsPred=pState(:,2:nSteps)'*Bhat;
%−−−−−−−−−− performance −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
[¬,sMax]=max(tsPred,[],2); [¬,ssY]=max(Y,[],2);
prdiff=(ssY==sMax); perf=mean(prdiff);
%−−−−−−−−−− L2−performance −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
perfL2=sqrt(mean(((Y−tsPred).^2))')';
%−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
end

function [Bhat] = estimatewOut(pState,sY,nSteps)
%−−−−−−−−−− define linear readout by least−squares estimate −−−−−−
X=pState(:,2:nSteps)';
sY=sY(2:nSteps,:);
Bhat=pinv(X)*sY;
end

function [input,sY]=inputMatrix(nSteps,nU,nE,wrdLgth,strIn)
% − Generates input sequences consisting of random alternations of
% 2 "words" with n+2 letters ('a n*b c' / 'e n*d f');
% − Generates sequence of letter identifiers (sSqn) (1−6,
% corresponding to a−e);
% − Generates time−series of correct readout patterns sY for supervised
% learning in the readout layer;
```

```
%------------------------------------------------------------------------
symbol=[[ones(nU,1);zeros(5*nU,1)], ...
    [zeros(nU,1);ones(nU,1);zeros(4*nU,1)], ...
    [zeros(2*nU,1);ones(nU,1);zeros(3*nU,1)], ...
    [zeros(3*nU,1);ones(nU,1);zeros(2*nU,1)], ...
    [zeros(4*nU,1);ones(nU,1);zeros(nU,1)], ...
    [zeros(5*nU,1);ones(nU,1)]];
if strIn
    word1=[symbol(:,1),repmat(symbol(:,2),1,wrdLgth),symbol(:,3)];
    word2=[symbol(:,4),repmat(symbol(:,5),1,wrdLgth),symbol(:,6)];

    Lgth=wrdLgth+2; m=ceil(nSteps/(2*Lgth));
    input=zeros(nE,2*m*Lgth); wSeq=[ones(1,m),zeros(1,m)];
    wSeq=wSeq(randperm(2*m));
    input(1:6*nU,:)=kron(wSeq,word1)+kron(1-wSeq,word2);

    yw1=eye(2*Lgth); yw2=yw1(:,Lgth+1:2*Lgth); yw1=yw1(:,1:Lgth);
    sY=(kron(wSeq,yw1)+kron(1-wSeq,yw2))';

else
    word=[symbol(:,1),symbol(:,2),symbol(:,3),...
        symbol(:,4),symbol(:,5),symbol(:,6)];
    w=1:6; m=ceil(nSteps/length(w));
    input=repmat(word,1,m);
    rperm=randperm(nSteps);
    input=[input(:,rperm);zeros(nE-6*nU,nSteps)];
    sY=repmat(eye(length(w)),m,1);
    sY=sY(rperm(1:nSteps),:);
end

end

function [sta]=clplin(n,preE,m)
for i=1:n
    if preE(i,1)<0
        sta(i,1)=0;
    elseif preE(i,1)≥0 && preE(i,1)≤1/m
        sta(i,1)=m*preE(i,1);
    elseif preE(i,1)>1/m
        sta(i,1)=1;
    end
end
end

function [nW]=normalizeRows(W)
c=size(W,2); nW=diag(1./(W*ones(c,1)))*W;
end
```

# C

# Batch Mode (v0.1)

The batch mode is an automated way of implementing the simulation studies described throughout the thesis and acquiring relevant data. It is a preliminary version (with a lot of room for improvements) comprising a basic user interface with the integration of all the relevant analysis methods described. The results of the simulations performed can either be saved to disk, or plotted (depending on the initial options the user is asked to specify).

It may still contain some bugs, so, if errors occur while running the batch version, the user is advised to use the "normal" version (Appendix B).

## Analysis Modes

The whole interface revolves around the type of analysis to be performed. The data resulting from the simulations as well as the plots will depend on the chosen analysis. By running `main.m`, the user is first asked to define this parameter, followed by the output options (save and/or plot data).

Four analysis methods are allowed (see examples):

**Single network** (0) Simulates one network (either dynamic or static). The results of these simulations will depend on the pre-defined analysis parameters (performance, perturbation, activity). This is the only type of analysis that allows the user to perform PCA and hierarchical clustering[1].

[1]The plotting routines in the script that performs the cluster analysis are not entirely

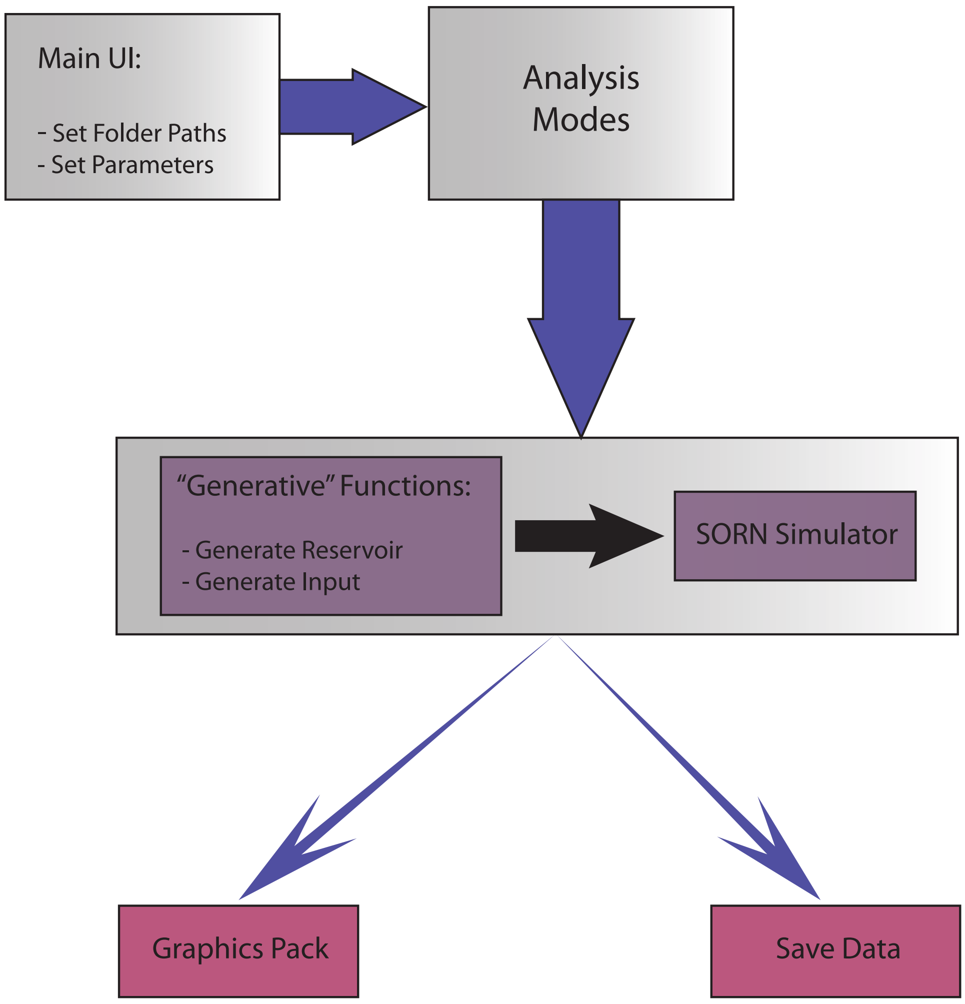


**Figure C.1:** Flow chart depicting the general components and structure of the code included in the batch version.

**Compare single networks** (1) Simulates a dynamic reservoir, then switches plasticity off, changes initial threshold values and simulates a static reservoir. The plotting routines that emerge from this analysis include a side-by-side comparison between the two reservoir types.

**Parameter sweep** (2) Analyzes the impact of varying a given variable value, over a given range in the pre-specified parameters (performance, crit-

developed.

icality, network activity). The user is asked to define the parameter to analyze (note that this is case sensitive, the variable name must be written exactly as it is defined in `set_parameters.m`), the interval to sweep and the number of simulations to average over.

**Compare parameter sweeps** (3) Performs the same analysis as the previous mode, but includes both types of reservoirs. It starts by analyzing dynamic reservoirs, after running the whole parameter range over all the simulations, learning is set to zero, threshold parameters modified and the analysis is repeated for the static reservoir. The output data and plots will include the comparison of both reservoir types.

Note that prior to running `main.m`, the network's parameters should be set (by editing `set_parameters.m` and saving the changes).

## Running the code

Start every analysis by editing `set_parameters.m`:

```
%==========================================================================
% Simulation parameters
%--------------------------------------------------------------------------
nLSteps=50000;                      % number of learning (plasticity)
                                    % steps;
nEstSteps=5000;                     % estimation time steps for
                                    % specifying the readout layer;
nPredSteps=5000;                    % number of prediction steps;
%------------------- Input characteristics --------------------------------
strIn=1;                            % input structured or not = 1/0;
wrdLgth=8;                          % word length n(+2);
%------------------- Network characteristics ------------------------------
node_type=0;                        % LIF-type node (1) or
                                    % McCulloch-Pitts-type (0)
trf_fcn=0;                          % transfer function: step(0), clipped
                                    % linear(1) or sigmoid(2)
gain=0;                             % steepness of the transfer function

learning=1;                         % plasticity mechanisms on/off = 1/0;

nE=100;                             % number of exc units in the network;
tEmax=.5; tImax=1.4;                % maximum threshold values;
```

```
%——————————————————— Analysis characteristics ———————————————————————————
performance=1;                    % performance analysis on/off = 1/0;
perturbation=0;                   % perturbation analysis on/off = 1/0;
activity=1;                       % measure firing rates
%————————————————————————————————————————————————————————————————————————
%========================================================================
% Aditional Network parameters
%————————————————————————————————————————————————————————————————————————
nI=.2*nE;                  % number of inhibitory units in the network;
nU=.05*nE;                 % number of symbol specific input neurons;
rSTDP=.001;                % STDP adaptation parameter;
rIP=.001;                  % IP adaptation parameter and ...
hIP=2*nU/nE;               % ... target rate;
lambda=10/(nE−1);          % the probability of a wEE−connection;
                           % — defines the sparsity of wEE; in this
                           % case, the mean number of input (or output)
                           % wEE−connections = 10;
```

The default values of those parameters are stored in the `default_pars.m`, located in the saved data folder. Note that when setting the analysis parameters (performance, perturbation, activity), at least one of them has to be set to 1.

After choosing the network and simulation parameters, run `main.m`. This script begins by setting the correct folder paths, then asks the user which type of simulation to run and whether or not to save and/or plot the data. After acquiring these values, the script calls other scripts or functions, depending on the analysis mode chosen.

## Single Network

```
Analysis Mode:  0
Save Data?  1
Plot Data?  1
```

After this, the script `run_SingleNet.m` is called, providing additional user options:

```
Perform PCA?  0
Cluster Data (takes a long time)?  0
Running...
```

The script calls the relevant functions (to understand what they do see the functions section below):

```
set_parameters
%==============================================================================
% Generate Reservoir
%------------------------------------------------------------------------------
[Parameters] = gen_reservoir (Parameters);
%=========================================================================
% Generate Input
%-------------------------------------------------------------------------
[Parameters] = gen_input(Parameters);
%=========================================================================
% Simulate
%-------------------------------------------------------------------------
[PlotData] = SORNsimulator(Parameters);
```

After the simulation is complete, the elapsed time is displayed and the relevant results plotted and saved (under the name `SingleNet.mat` in the saved data folder). The resulting plots will depend on the type of reservoir (dynamic or static) and on the chosen analysis parameters. For example, if the simulated network was a dynamic reservoir, the resulting plots will include the variation in $W^{EE}$ and $T^{E}$, not included in the static case.

If performance analysis was set to 1, the plots will include the readout error but not the performance per se (this result is only displayed in the command prompt).

## Compare single networks

```
Analysis Mode:  1
Save Data?  1
Plot Data?  1
```

In this case, the script called is `run_compareSNet.m`.

```
Running...
```

This script performs the same operations as the previous one, with the only difference that it simulates a dynamic and a static reservoir for a direct comparison. After simulating the dynamic reservoir, learning is set to 0, and the initial threshold values modified (lines $18-20$). The changes in threshold values are set according to the results presented in chapter 5. Alternatively, the user may wish to modify this by editing `run_compareSNet.m`.

```
1  set_parameters
2  Parameters.Network.Flags.Flagvalues(3)=1;
3  %==================================================================
4  % Generate Reservoir
5  %------------------------------------------------------------------
6  [Parameters] = gen_reservoir (Parameters);
7  %==================================================================
8  % Generate Input
9  %------------------------------------------------------------------
10 [Parameters] = gen_input(Parameters);
11 %==================================================================
12 % Simulate
13 %------------------------------------------------------------------
14 [Data] = SORNsimulator(Parameters);
15 
16 %==================================================================
17 %==================================================================
18 Parameters.Network.Flags.Flagvalues(3)=0;
19 Parameters.Network.Variables.Varvalues(4)=0.75;
20 Parameters.Network.Variables.Varvalues(5)=0.8;
21 %==================================================================
22 % Generate Reservoir
23 %------------------------------------------------------------------
24 [Parameters] = gen_reservoir (Parameters);
25 %==================================================================
26 % Generate Input
27 %------------------------------------------------------------------
28 [Parameters] = gen_input(Parameters);
29 %==================================================================
30 % Simulate
31 %------------------------------------------------------------------
32 [Data] = SORNsimulator(Parameters);
```

Throughout the simulation, the elapsed time for each reservoir is displayed:

```
Elapsed time (Dynamic Reservoir): 1.843789e+002
Elapsed time (Static Reservoir): 1.902951e+002
```

Upon completion, the results are plotted and/or saved (in the saved data folder, under the name `SingleNetCompare.mat`), similarly to what happens in the previous case the only difference being that plots for dynamic and static reservoirs are placed side-by-side, allowing the user to easily visualize the differences.

## Parameter Sweep

```
Analysis Mode:  2
Save Data?  1
Plot Data?  1
```

Running parameter sweeps has several peculiarities. By setting analysis mode to 2, the `main` script calls the function `run_parSweeps.m`. Within this function, the user is asked to specify the remaining parameters:

```
Parameter to evaluate: wrdLgth
Interval: 4:2:20
Number of simulations/condition: 10
```

When specifying the parameter to evaluate, caution must be taken to ensure its name is written exactly as it is defined, i.e., the variable name must be equal to the name that specifies it in `set_parameters`. This is because, during the simulation, the function will look for that name in Parameters.Network.Variables.Varnames (see data structures below) and replace the corresponding value in Parameters.Network.Variables.Varvalues.

The interval can be specified as in the displayed example (initial value: interval: final value), or accounting for each individual value ([4, 6, 8, ..., 20]).

Another important aspect in this analysis mode is the order of the for loops. If the parameter to analyze is relevant for setting the reservoir's initial states (weights, thresholds, number of neurons, etc.), a new reservoir must be generated for each parameter and the n simulations ran on that reservoir. If, for example, the parameter to evaluate was tEmax, the order of the loops would be:

```
    for p=1:size(range,2)
        Parameters.Network.Variables.Varvalues(ind) = range(p);
        %=================================================================
        % Generate Reservoir
        %-----------------------------------------------------------------
        [Parameters] = gen_reservoir (Parameters);

        for n=1:nSims
            %=============================================================
            % Simulation
            %-------------------------------------------------------------
            [Parameters] = gen_input(Parameters);
            [SimData] = SORNsimulator (Parameters);
```

Otherwise, one reservoir is generated and the parameter range swept through it. This is the case in the example above, where the parameter to assess is wrdLgth.

```
ind=find(ismember(Parameters.Network.Variables.Varnames,Par));

if ind≥7
    for n=1:nSims
        %=============================================================
        % Generate Reservoir
        %-------------------------------------------------------------
        [Parameters] = gen_reservoir (Parameters);

        for p=1:size(range,2)
            Parameters.Network.Variables.Varvalues(ind) = range(p);
            %=========================================================
            % Simulation
            %---------------------------------------------------------
            [Parameters] = gen_input(Parameters);
            [SimData] = SORNsimulator (Parameters);
```

This type of analysis tends to take some time, depending obviously on the specified parameter and interval. So, during the simulation, information about time is provided:

```
Elapsed time: 00:02:58    Time per condition: 00:02:58    Estimated time remaining: 00:08:54
Elapsed time: 00:06:13    Time per condition: 00:03:06    Estimated time remaining: 00:06:13
Elapsed time: 00:09:32    Time per condition: 00:03:10    Estimated time remaining: 00:03:10
Elapsed time: 00:12:56    Time per condition: 00:03:14    Estimated time remaining: 00:00:00
```

After the simulations are completed, results are saved (with the name `CompareParsonline.mat`) and/or plotted. The plots, in this case, are error bars, depicting the means and standard deviations over the defined number of simulations and the parameter range.

### Compare parameter sweeps

```
Analysis Mode:  3
Save Data?  1
Plot Data?  1
```

This is the final analysis mode. It is in every aspect equal to the normal parameter sweeps, the only difference being that it compares dynamics vs static reservoirs. The script `run_ComparePars.m`, runs a parameter sweep in the dynamic case, then switches plasticity off, resets the initial threshold values and runs the static reservoirs. The data structures are saved online (so, if the user decides to stop the process, the already obtained data is saved) and in the end of the simulation. The final plots will look like Figure 5.1.

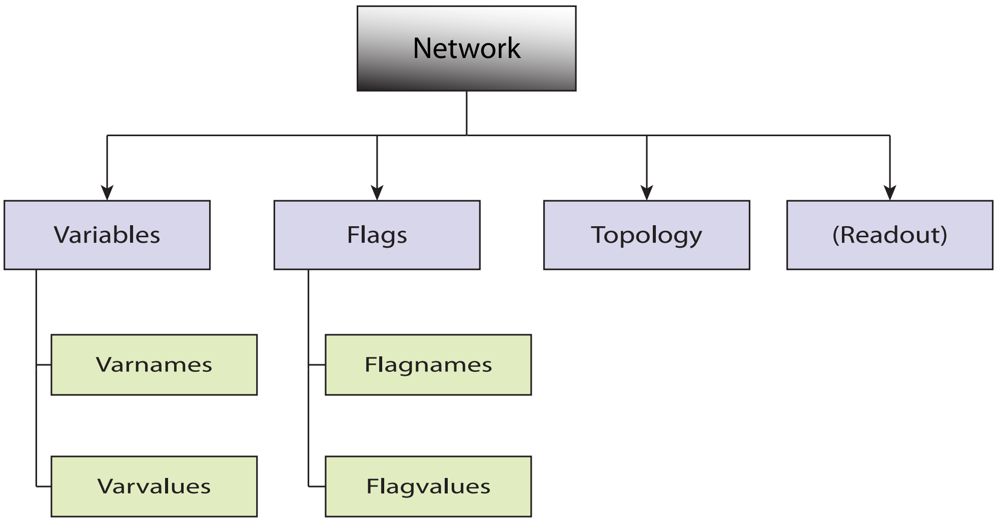
## Data Structures

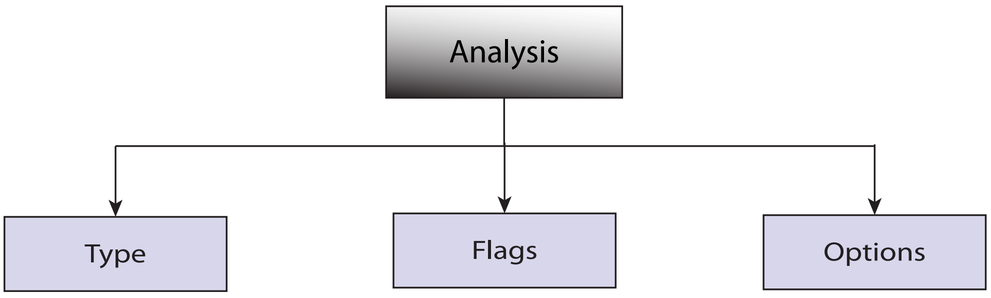
In order to save memory and keep the workspace clean, the majority of the data is stored in structure arrays. All functions and scripts extract only the relevant variables from these structures and assign them the correct names prior to running their computation. After the computation is performed all data is stored again in the correct structure location and the workspace cleared.

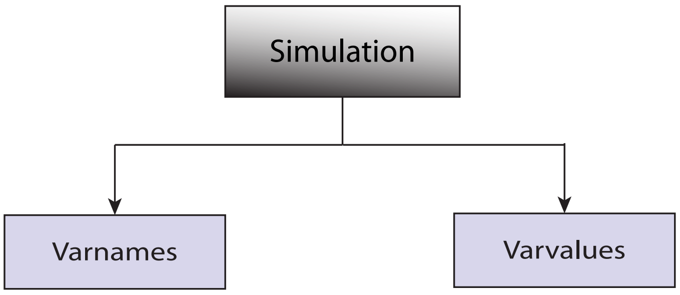
### Parameters

The Parameters structure initially stores the parameters defined in `set_parameters.m`, but will also store the weights matrices and unit's thresholds generated by gen_reservoir.m, the input matrices generated by gen_input.m and the estimated readout weights (if performance analysis is on).

```
Parameters =
Analysis: [1x1 struct]
Network: [1x1 struct]
Simulation: [1x1 struct]
Input: {[200x60000 double]  [10000x20 double]}
```







So, the Parameters structure contains the fields Network, Analysis, Simulation and Input, each with its own subfields. The field Input is the only that does not contain subfields, it is a cell array with the matrices input and sY.

The remaining fields in the structure contain the following data:

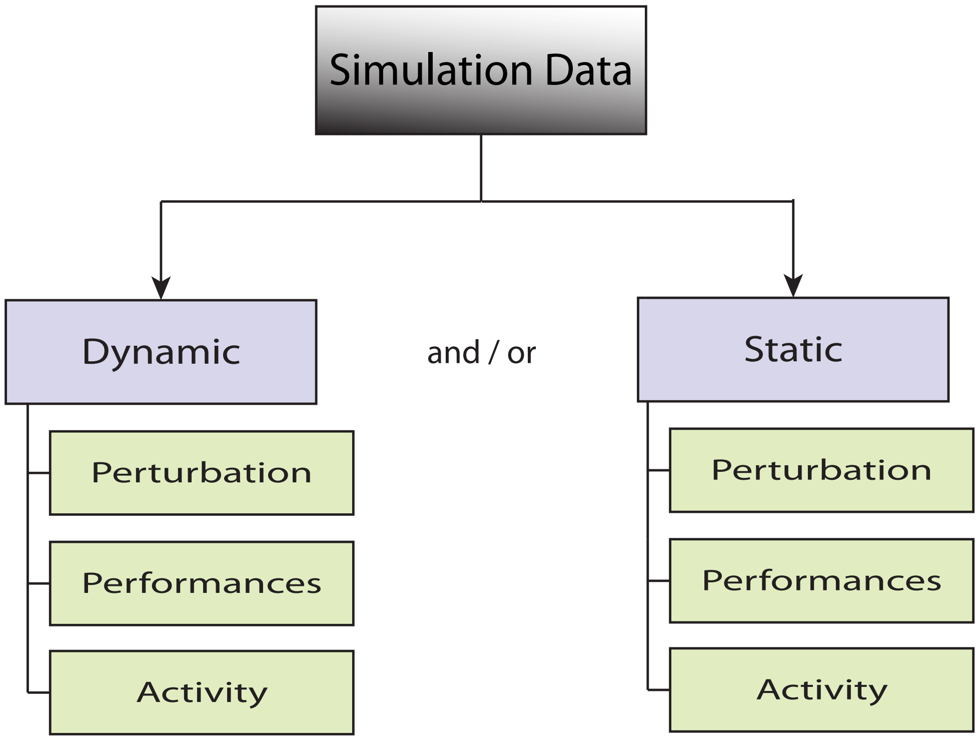
```
Parameters.Network.Variables.Varnames={'nE','nI','nU','tEmax','tImax',...
    'lambda','wrdLgth','rIP','hIP','rSTDP','gain'};
Parameters.Network.Variables.Varvalues=[nE,nI,nU,tEmax,tImax,lambda,...
    wrdLgth,rIP,hIP,rSTDP,gain];
%-----------------------------------------------------------------------------
Parameters.Network.Flags.Flagnames={'node_type','trf_fcn','learning',...
    'strIn'};
Parameters.Network.Flags.Flagvalues=[node_type,trf_fcn,learning,strIn];
%-----------------------------------------------------------------------------
Parameters.Analysis.Flags=[performance(1/0), perturbation(1/0), ...
    activity(1/0)];
Parameters.Analysis.Type=[Analysis mode(0, 1, 2 or 3)];
Parameters.Analysis.Options=[save data (1/0), plot data (1/0), ...
    perform PCA (1/0), perform clustering (1/0)];
%-----------------------------------------------------------------------------
Parameters.Simulation.Varnames={'nLSteps','nEstSteps','nPredSteps'};
```

To access the contents of any field, type `Parameters.(subfield1).(..)`. For example, to access the input matrix type:

```
Parameters.Input{1}
```

## Simulation Data

The data resulting from the simulations is stored in a structure, usually called PlotData, that will serve as input to the plotting functions. The contents of said structure will depend on the analysis mode.



For the analysis modes 0 and 1, the subfield Perturbation will contain the calculated hamming distances in the post-learning period (in the case of dynamic reservoirs); the subfield Performances is a cell array with the performance result and the squared error; the subfield Activity contains pState and the network topology. The topology structure saved here has the initial and final $W^{EE}$ on positions 7 and 3, respectively and the initial and final $T^{E}$, on positions 8 and 5.

For the analysis modes 2 and 3, the subfields are somewhat different. The subfield $Performance1$ contains an $n \times p$ matrix with the performance results for n simulations and p conditions, whereas $Performance2$ is an $n \times p$ cell array with the errors of each simulation. The subfield Activity has two

$n \times p$ matrices with the mean firing rates for each simulation and the mean number of inactive neurons. The subfield Criticality (note that the name is different), has the mean hamming distances in an $n \times p$ matrix. An additional subfield in included, called Parameters, containing the name of the analyzed parameter, the range and number of simulations.

This structure should not modified because it contains all the relevant information for the plotting routines, which may not work if the naming of the fields or their values are modified.

## Functions

The `run` functions described above, all call the same functions. We now provide a short description of what they do.

Note that included in the Functions and Scripts folder are two additional functions (`elapsedtimer.m` and `freezeColors.m`). They are freely distributed in the Matlab central[2], but any use has to retain the authors names, included in the preamble. `elapsedtimer` is used in analysis modes 2 and 3 to display the time information (it has been modified to suit the particular needs of this code) and `freezeColors` is used in some plotting routines, when setting different colormaps for different subplots in the same figure.

### Generate Reservoir

This function takes as input the Parameters structure, creates the initial values of weights and thresholds and saves them in the Topology subfield.

```
%=========================================================================
% Network connectivity
%-------------------------------------------------------------------------
wEI=rand(nE,nI);                    % static weights I->E , uniform[0,1];
wEI=normalizeRows(wEI);             % normalize rows of wEI to 1;
%-------------------------------------------------------------------------
wIE=rand(nI,nE);                    % static weights E->I , uniform[0,1];
```

[2] http://www.mathworks.com/matlabcentral/

```
wIE=normalizeRows(wIE);             % normalize rows of wIE to 1;
%-----------------------------------------------------------------------------
% Initialization of wEE
%-----------------------------------------------------------------------------
wEE=zeros(nE,nE);                   % dynamic weights E->E;
ind=find(rand(nE,nE)≤lambda);      % apply wEE sparseness;
wEE(ind)=rand(length(ind),1);
wEE(eye(nE)>0)=0;                   % self-connections = 0;
wEE=normalizeRows(wEE);             % normalize rows of wEE to 1;
wEC=wEE>0;                          % to force wEE sparsity during learning;
%-----------------------------------------------------------------------------
% Initial firing thresholds
%-----------------------------------------------------------------------------
tE=tEmax*rand(nE,1);                % excitatory thresholds;
tI=tImax*rand(nI,1);                % inhibitory thresholds;
%-----------------------------------------------------------------------------
% Save Data
%-----------------------------------------------------------------------------
Parameters.Network.Topology={wEI,wIE,wEE,wEC,tE,tI};
```

## Generate Input

This function generates the input matrix and training signal (sY).

```
if strIn
    word1=[symbol(:,1),repmat(symbol(:,2),1,wrdLgth),symbol(:,3)];
    word2=[symbol(:,4),repmat(symbol(:,5),1,wrdLgth),symbol(:,6)];
    Lgth=wrdLgth+2; m=ceil(nSteps/(2*Lgth));
    input=zeros(nE,2*m*Lgth); wSeq=[ones(1,m),zeros(1,m)];
    wSeq=wSeq(randperm(2*m));
    input(1:6*nU,:)=kron(wSeq,word1)+kron(1-wSeq,word2);

    yw1=eye(2*Lgth); yw2=yw1(:,Lgth+1:2*Lgth); yw1=yw1(:,1:Lgth);
    sY=(kron(wSeq,yw1)+kron(1-wSeq,yw2))';
    if learning
        sY=sY(nLSteps+1:end,:);
    end
else
    word=[symbol(:,1),symbol(:,2),symbol(:,3),...
        symbol(:,4),symbol(:,5),symbol(:,6)];
    w=1:6; m=ceil(nSteps/length(w));
    input=repmat(word,1,m);
    rperm=randperm(nSteps);
    input=[input(:,rperm);zeros(nE-6*nU,nSteps)];
    sY=repmat(eye(length(w)),m,1);
    sY=sY(rperm(1:nSteps),:);
    if learning
        sY=sY(nLSteps+1:end,:);
    end
end
%------------------------------------------------------------------------
% Save Data to structure
%------------------------------------------------------------------------
Parameters.Input{1}=input; Parameters.Input{2}=sY;
```

## SORN Simulator

This is the main simulator, it is in every aspect equal to the generator in Appendix B, except for the variable data manipulations it includes, that encompass the different analysis modes and parameters.

# References

[1] L. F. Abbott and S. B. Nelson. Synaptic plasticity: taming the beast. *Nature Neuroscience*, 3 Suppl(november):1178–1183, 2000.

[2] M. Abeles. *Corticonics: Neural Circuits of the Cerebral Cortex*. Number 4. Cambridge University Press, 1991.

[3] B. A. Arcas, A. L. Fairhall, and W. Bialek. What can a single neuron compute? *Advances in neural information processing systems*, 13:75–81, 2001.

[4] A Arieli, A Sterkin, A Grinvald, and A Aertsen. Dynamics of ongoing activity: explanation of the large variability in evoked cortical responses. *Science*, 273(5283):1868–1871, 1996.

[5] A F Atiya and A G Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709, 2000.

[6] Štefan Babinec and Jiří Pospíchal. Improving the prediction accuracy of echo state neural networks by anti-oja's learning. In *Proceedings of the 17th international conference on Artificial neural networks*, ICANN'07, pages 19–28, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] Edward L. Bartlett and Xiaoqin Wang. Long-lasting modulation by stimulus context in primate auditory cortex. *Journal of Neurophysiology*, 94:83–104, 2005.

[8] M F Bear and R C Malenka. Synaptic plasticity: Ltp and ltd. *Current Opinion in Neurobiology*, 4(3):389–399, 1994.

[9] C. C. Bell, V. Z. Han, Y. Sugawara, and K. Grant. Synaptic plasticity in a cerebellum-like structure depends on temporal order. *Nature*, 387(6630):278–281, 1997.

[10] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[11] Nils Bertschinger and Thomas Natschlger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413–1436, 2004.

[12] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of Neuroscience*, 18(24):10464–10472, 1998.

[13] A. R. Bizzarri. Convergence properties of a modified hopfield-tank model. *Biological Cybernetics*, 64(4):293–300, 1991.

[14] J. F. Brons and C. D. Woody. Long-term changes in excitability of cortical neurons after pavlovian conditioning and extinction. *Journal of Neurophysiology*, (44):605–615, 1980.

[15] Bede M. Broome, Vivek Jayaraman, and Gilles Laurent. Encoding and decoding of overlapping odor sequences. *Neuron*, 51:467–482, 2006.

[16] M. Brosch and C. E. Schreiner. Sequence sensitivity of neurons in cat primary auditory cortex. *Cerebral Cortex*, 10:1155–1167, 2000.

[17] Dean V Buonomano. Decoding temporal information: A model based on short-term synaptic plasticity. *Journal of Neuroscience*, 20(3):1129–41, 2000.

[18] Dean V Buonomano and Wolfgang Maass. State-dependent computations: spatiotemporal processing in cortical networks. *Nature Reviews Neuroscience*, 10(2):113–25, 2009.

[19] Matteo Carandini and Frank Sengpiel. Contrast invariance of functional maps in cat primary visual cortex. *Journal of Vision*, 4(3):130–143, 2004.

[20] H. Colman, J. Nabekura, and J. W. Lichtman. Alterations in synaptic strength preceding axon withdrawal. *Science*, 275(5298):356–361, 1997.

[21] Robert H Cudmore and Gina G Turrigiano. Long-term potentiation of intrinsic excitability in lv visual cortical neurons. *Journal of Neurophysiology*, 92(1):341–348, 2004.

[22] G C DeAngelis, I Ohzawa, and R D Freeman. Spatiotemporal organization of simple-cell receptive fields in the cat's striate cortex. *Neurophysiology*, 69(4):1091–1117, 1993.

[23] R C DeCharms. Optimizing sound features for cortical neurons. *Science*, 280(5368):1439–1444, 1998.

[24] N S Desai, L C Rutherford, and G G Turrigiano. Plasticity in the intrinsic excitability of cortical pyramidal neurons. *Nature Neuroscience*, 2(6):515–520, 1999.

[25] Morphological Development. Developmental neurogenesis. *Cortex*, pages 396–404, 2010.

[26] D. Durstewitz and G. Deco. Computational significance of transient dynamics in cortical networks. *European Journal of Neuroscience*, 27:217–227, 2008.

[27] X. Dutoit, B. Schrauwen, J. Van Campenhout, D. Stroobandt, H. Van Brussel, and M. Nuttin. Pruning and regularization in reservoir computing. *Neurocomput.*, 72(7-9):1534–1546, March 2009.

[28] Vaadia E., Aertsen A., and Nelken I. Dynamics of neuronal interactions cannot be explaned by neuronal transients. *Proc R Soc Lond B*, 261:407–410, 1995.

[29] J. Elman and D. Zipster. Learning the hidden structure of speech. *Journal of the Acoustic Society of America*, 83(4):1615–1626, 1988.

[30] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:172–211, 1990.

[31] Cagla Eroglu and Ben A. Barres. Regulation of synaptic connectivity by glia. *Nature*, 468(7321):223–231, 2010.

[32] European Symposium on Artifitial Neural Networks. *Perspectives on learning with recurrent neural networks*, 2002.

[33] D. E. Feldman. Timing-based ltp and ltd at vertical input to layer ii/iii pyramidal cells in rat barrel cortex. *Neuron*, 27:45–56, 2000.

[34] Chrisantha Fernando and Sampsa Sojakka. Pattern recognition in a bucket. *Advances in Artificial Life*, 2801:588–597, 2003.

[35] Karl J. Friston. The labile brain. i. neuronal transients and nonlinear coupling. *Philosophical Transactions of the Royal Society of London - Series B: Biological Sciences*, 355(1394):215–236, 2000.

[36] Karl J. Friston. The labile brain. ii. transients, complexity and selection. *Philosophical Transactions of the Royal Society of London - Series B: Biological Sciences*, 355(1394):237–252, 2000.

[37] Karl J. Friston. The labile brain. iii. transients and spatio-temporal receptive fields. *Philosophical Transactions of the Royal Society of London - Series B: Biological Sciences*, 355(1394):253–265, 2000.

[38] Steve Furber and Steve Temple. Neural systems engineering. *Journal of the Royal Society Interface the Royal Society*, 4(13):193–206, 2007.

[39] C. S. Goodman and C. Shatz. Developmental mechanisms that generate precise patterns of neural connectivity. *Cell Suppl*, 72:77–98, 1993.

[40] Carlos Maria Alaiz Gudin. Advanced methods for recurrent neural network design. Master's thesis, Universidad Autonoma de Madrid, Madrid, 2010.

[41] Kevin Gurney. *An introduction to neural networks.* CRC Press, 1997.

[42] Richard W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[43] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1998.

[44] Donald O. Hebb. *The Organization of Behavior*. Willey, 1949.

[45] G. E. Hinton. Boltzmann machine. *Scholarpedia*, 2(5):1668, 2007.

[46] C Holscher. Long-term potentiation: a good model for learning and memory? *Prog. Neuropsychopharmacol. Biol. Psychiatry*, 21(1):47–68, 1997.

[47] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[48] J. J. Hopfield. Hopfield network. *Scholarpedia*, 2(5):1977, 2007.

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

[50] K. Hornik, M. Stinchcombe, and H. White. Multilayered feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[51] Stefan Husler and Wolfgang Maass. A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models. *Cerebral Cortex*, 17(1):149–162, 2007.

[52] Kazuo Ishii, Tijn Van Der Zant, Vlatko Beucanovic, and Paul G Plger. Identification of motion with echo state network. *OCEANS04 MTTSIEEE TECHNOOCEAN04*, 3:1205–1230, 2004.

[53] Eugene M Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. MIT Press, 2007.

[54] H. Jaeger. The echo state approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, 2001.

[55] H Jaeger. Reservoir riddles: suggestions for echo state network research. *Proceedings 2005 IEEE International Joint Conference on Neural Networks 2005*, 3(3):1460–1462, 2005.

[56] H. Jaeger, Wolfgang Maass, and J. Principe. Special issue on echo state networks and liquid state machines. *Neural Networks*, 20(3):287–289, 2007.

[57] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[58] Herbert Jaeger, Mantas Lukosevicius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352, 2007.

[59] William James. *The Principles of Psychology*, volume I. Cosimo, New York, 1890.

[60] Sarah Jarvis, Stefan Rotter, and Ulrich Egert. Extending stability through hierarchical clusters in echo state networks. *Frontiers in neuroinformatics*, 4(July):11, 2010.

[61] Fei Jiang, Hugues Berry, and Marc Schoenauer. Supervised and evolutionary learning of echo state networks. *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature*, pages 215–224, 2008.

[62] Nir Kalisman, Gilad Silberberg, and Henry Markram. Deriving physical connectivity from neuronal morphology. *Biological Cybernetics*, 88(3):210–8, 2003.

[63] Gerd Kempermann. Adult neurogenesis. *Memory*, 5(7):722–728, 2006.

[64] Andreea Lazaar, Gordon Pipa, and Jochen Triesch. Sorn: a self-organizing recurrent neural network. *Frontiers in Computational Neuroscience*, 3(23):1–9, 2009.

[65] Andreea Lazar. *Self-organizing Recurrent Neural Networks*. PhD thesis, Department of Computer Science and Mathematics, Goethe-Universitt Frankfurt am Main, Frankfurt, 2009.

[66] Andreea Lazar, Gordon Pipa, and Jochen Triesch. Fading memory and time series prediction in recurrent networks with different forms of plasticity. *Neural Networks*, 20(3):312–322, 2007.

[67] Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. Theoretical analysis of learning with reward-modulated spike-timing-dependent plasticity. *Advances in Neural Information Processing Systems 20*, 20(1):1–8, 2007.

[68] John Lisman, Jeff W Lichtman, and Joshua R Sanes. Ltp: perils and progress. *Nature Reviews Neuroscience*, 4(11):926–929, 2003.

[69] Dmitri V. Lissin, Stephen N. Gomperts, Reed C. Carroll, Chadwick W. Christine, Daniel Kalman, Marina Kitamura, Stephen Hardy, Roger A. Nicoll, Robert C. Malenka, and Mark Von Zastrow. Activity differentially regulates the surface expression of synaptic ampa and nmda glutamate receptors. *Proceedings of the National Academy of Sciences of the United States of America*, 95(12):7097–7102, 1998.

[70] Y. J. Lo and M. M. Poo. Activity-dependent synaptic competition in vitro: heterosynaptic suppression of developing synapses. *Science*, 254(5034):1019–1022, 1991.

[71] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, August 2009.

[72] X F Ma and Z X Mei. Is ltp the neural basis of learning and memory? *Sheng li ke xue jin zhan Progress in physiology*, 29(2):137–140, 1998.

[73] Wolfgang Maas, Thomas Natschlger, and Henry Markram. Real-time computing withpout stable states: a new framework for neural computation based on perturbations. *Neural Computations*, 14:2531–2560, 2002.

[74] W. Maass, T. Natschlger, and H. Markram. Computational models for generic cortical microcircuits. *Computational Neuroscience A Comprehensive Approach*, pages 575–605, 2004.

[75] Wolfgang Maass, Prashant Joshi, and Eduardo D. Sontag. Principles of real-time computing with feedback applied to cortical microcircuit models. *Advances in Neural Information Processing Systems 18*, pages 835–842, 2006.

[76] Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of Computer and System Sciences*, 69(4):593–616, 2004.

[77] Larry Manevitz and Hananel Hazan. Stability and topology in reservoir computing. *Avances in Soft Computing*, 6438:245256, 2010.

[78] Eve Marder and Jean-Marc Goaillard. Variability, compensation and homeostasis in neuron and network function. *Nature Reviews Neuroscience*, 7(7):563–74, 2006.

[79] Henry Markram, Joachim Lbke, Michael Frotscher, and Bert Sackmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275:213–215, 1997.

[80] Henry Markram, Yun Wang, and Misha Tsodyks. Differential signaling via the same axon of neocortical pyramidal neurons. *Proceedings of the National Academy of Sciences of the United States of America*, 95(9):5323–5328, 1998.

[81] S. J. Martin, P. D. Grimwood, and R. G. Morris. Synaptic plasticity and memory: an evaluation of the hypothesis. *Annual Review of Neuroscience*, 23:649–711, 2000.

[82] N Mayer, J M Herrmann, and T Geisel. Retinotopy and spatial phase in topographic maps. *Neurocomputing*, 32-33:447–452, 2000.

[83] W. S. McCulloch and W. H. Pitts. A logical calculus of the ideas immanent in nervous system activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[84] Robert J. Mceliece, Edward C. Posner, Eugene R. Rodemich, and Santosh S. Venkatesh. The capacity of the hopfield associative memory. *IEEE Transactions on Information Theory*, 33(4):461–482, 1987.

[85] Peter McLeod, Kim Plunkett, and Edmund T. Rolls. *Introduction to Connectionist Modelling of Cognitive Processes.* Oxford University Press, 2nd edition, 1998.

[86] Samar B Mehta and David Kleinfeld. Frisking the whiskers: patterned sensory input in the rat vibrissa system. *Neuron*, 41(2):181–184, 2004.

[87] V. Mountcastle. *An organizing principle for cerebral function: the unit model and the distributed system*, pages 7–50. MIT Press, 1978.

[88] Thomas Natschlaeger, Nils Bertschinger, and Robert Legenstein. At the edge of chaos: Real-time computations and self-organized criticality in recurrent neural networks. *Advances in Neural Information Processing Systems 17*, 17:145–152, 2005.

[89] Sacha B. Nelson and Gina G. Turrigiano. Strength through diversity. *Neuron*, 60(3):477–482, 2008.

[90] Danko Nikoli, Stefan Haeusler, Wolf Singer, and Wolfgang Maass. Temporal dynamics of information content carried by neurons in the primary visual cortex. *Neural Information Processing Systems*, 2006.

[91] Esther A Nimchinsky, Bernardo L Sabatini, and Karel Svoboda. Structure and function of dendritic spines. *Annual Review of Physiology*, 64(1):313–353, 2002.

[92] D Norton and D Ventura. Preparing more effective liquid state machines using hebbian learning. *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 4243–4248, 2006.

[93] R J O'Brien, S Kamboj, M D Ehlers, K R Rosen, G D Fischbach, and R L Huganir. Activity-dependent modulation of synaptic ampa receptor accumulation. *Neuron*, 21(5):1067–1078, 1998.

[94] Simon A Overduin and Philip Servos. Symmetric sensorimotor somatotopy. *PLoS ONE*, 3(1):6, 2008.

[95] Mustafa C Ozturk, Dongming Xu, and Jos C Prncipe. Analysis and design of echo state networks. *Neural Computation*, 19(1):111–138, 2007.

[96] David Parker. Complexities and uncertainties of neuronal network function. *Philosophical Transactions of the Royal Society B Biological Sciences*, 361(1465):81–99, 2006.

[97] David Parker. Neuronal network analyses: premises, promises and uncertainties. *Philosophical Transactions of the Royal Society of London - Series B: Biological Sciences*, 365(1551):2315–2328, 2010.

[98] H. Paugam-Moisy and Sander Bohte. *Computing with Spiking Neuron Networks*, pages 1–47. PRePRINT, 2009.

[99] D Prokhorov. Echo state networks: appeal and challenges. *Proceedings 2005 IEEE International Joint Conference on Neural Networks 2005*, 3(3):1463–1466, 2005.

[100] M. Rabinovich, R. Huerta, and G. Laurent. Transient dynamics for neural processing. *Science*, 321:48–50, 2008.

[101] N. Rochester, J. H. Holland, L. H. Haibt, and W. L. Duda. Tests on a cell assembly theory of the action of the brain using a large digital computer. *Transaction of Information Theory*, IT-2:80–93, 1956.

[102] Ali Rodan and Peter Tino. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–144, 2011.

[103] Raul Rojas. 13 the hopfield model. *Neural Networks*, pages 337–371, 1996.

[104] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington D.C., 1962.

[105] D E Rumelhart, G E Hinton, and R J Williams. *Learning internal representations by error propagation*, volume 1, pages 318–362. MIT Press, 1986.

[106] Carl Sagan. Dragons of eden. page 269, 1977.

[107] Benjamin Schrauwen, Lars Bsing, and Robert Legenstein. On computational power and the order-chaos phase transition in reservoir computing. *Neural Information Processing Systems*, 2009:1–8, 2009.

[108] Harel Z. Shouval, Samuel S. H. Wang, and Gayle M. Wittenberg. Spike timing dependent plasticity: A consequence of more fundamental learning rules. *Frontiers in computational neuroscience*, 4(July):13, 2010.

[109] Gilad Silberberg, Anirudh Gupta, and Henry Markram. Stereotypy in neocortical microcircuits. *Trends in Neurosciences*, 25(5):227–30, 2002.

[110] Per Jesper Sjstrm and Michael Husser. A cooperative switch determines the sign of synaptic plasticity in distal dendrites of neocortical pyramidal neurons. *Neuron*, 51(2):227–238, 2006.

[111] Per Jesper Sjstrm, Gina G. Turrigiano, and Sacha B. Nelson. Multiple forms of long-term plasticity at unitary neocortical layer 5 synapses. *Neuropharmacology*, 52(1):176–184, 2007.

[112] D C Somers, S B Nelson, and M Sur. An emergent model of orientation selectivity in cat visual cortical simple cells. *Journal of Neuroscience*, 15(8):5448–5465, 1995.

[113] S Song and L F Abbott. Cortical development and remapping through spike timing-dependent plasticity. *Neuron*, 32(2):339–50, 2001.

[114] Sen Song and L. F. Abbott. Cortical development and remapping through spike timing-dependent plasticity. *Neuron*, 32(2):339–50, 2001.

[115] Sen Song, Per Jesper Sjstrm, Markus Reigl, Sacha Nelson, and Dmitri B. Chklovskii. Highly nonrandom features of synaptic connectivity in local cortical circuits. *PLoS Biology*, 3(3):e68, 2005.

[116] J J Steil. *Backpropagation-decorrelation: online recurrent learning with O(N) complexity*, volume 2, pages 843–848. IEEE, 2004.

[117] Jochen J Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning. *Neural Networks*, 20(3):353–364, 2007.

[118] Peter Stern. Glee for glia. *Science*, 330(November):2010–2010, 2010.

[119] Charles R. Tessier and Kendal Broadie. Activity-dependent modulation of neural circuit synaptic connectivity. *Frontiers in molecular neuroscience*, 2(July):13, 2009.

[120] T. Trappenberg. *Fundamentals of Computational Neuroscience.* Oxford University Press, 2nd edition, 2010.

[121] Jochen Triesch. A gradient rule for the plasticity of a neurons intrinsic excitability. *Artificial Neural Networks Formal Models and Their ApplicationsICANN 2005*, page 6570, 2005.

[122] Jochen Triesch. Synergies between intrinsic and synaptic plasticity mechanisms. *Neural Computation*, 19(4):885–909, 2007.

[123] A. M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 45(2):161–228, 1939.

[124] G. G. Turrigiano, R. Kenneth, S. Niraj, Rutherford Desai, C. Lana, and S. B. Nelson. Activity-dependent scaling of quantal amplitude in neocortical neurons. *Nature*, 391:892–895, 1998.

[125] Gina G. Turrigiano. The self-tuning neuron: synaptic scaling of excitatory synapses. *Cell*, 135(3):422–435, 2008.

[126] Gina G. Turrigiano, Kenneth R. Leslie, Niraj S Desai, Lana C. Rutherford, and Sacha B. Nelson. Activity-dependent scaling of quantal amplitude in neocortical neurons. *Nature*, 391(6670):892–6, 1998.

[127] Gina G Turrigiano and Sacha B Nelson. Homeostatic plasticity in the developing nervous system. *Nature Reviews Neuroscience*, 5(2):97–107, 2004.

[128] Ganesh K Venayagamoorthy and Bashyal Shishir. Effects of spectral radius and settling time in the performance of echo state networks. *Neural networks : the official journal of the International Neural Network Society*, 22(7):861–3, September 2009.

[129] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20:391–403, 2007.

[130] D. Verstraeten, B. Schrauwen, and D. Stroobandt. Reservoir-based techniques for speech recognition. *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1050–1053, 2006.

[131] D. Verstraeten, Benjamin Schrauwen, and D. Stroobandt. *Isolated word recognition using a liquid state machine*, page 435440. Citeseer, 2005.

[132] Pascal Wallisch, Michael Lusignan, Marc Benayoun, Tanya I. Baker, Adam S. Dickey, and Nicholas G. Hatsopoulos. *MATLAB for Neuroscientists - An Introduction to Scientific Computing in MATLAB*. Elsevier, 2009.

[133] Marion Wardermann and Jochen Steil. *Intrinsic plasticity for reservoir learning algorithms*, pages 513–518. d-side publi, 2007.

[134] A J Watt, M C W Von Rossum, K M MacLeod, S B Nelson, and G G Turrigiano. Activity co-regulates quantal ampa and nmda current at neocortical synapses. *Neuron*, 23:659–670, 2000.

[135] P J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[136] R J Williams and D Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.

[137] Ji Yu, Jie Xiao, Xiaojia Ren, Kaiqin Lao, and X Sunney Xie. Probing gene expression in live cells, one protein molecule at a time. *Science*, 311(5767):1600–3, 2006.

[138] W Zhang and D J Linden. The other side of the engram. *Nature Reviews Neuroscience*, 4:885–900, 2003.