Markovian bias of neural-based architectures with feedback connections

Peter Tiňo[1], Barbara Hammer[2], and Mikael Bodén[3]

[1] University of Birmingham, Birmingham, UK p.tino@cs.bham.ac.uk
[2] Clausthal University of Technology, Germany hammer@in.tu-clausthal.de
[3] University of Queensland, Brisbane, Australia mikael@itee.uq.edu.au

Summary. Dynamic neural network architectures can deal naturally with sequential data through recursive processing enabled by feedback connections. We show how such architectures are predisposed for suffix-based Markovian input sequence representations in both supervised and unsupervised learning scenarios. In particular, in the context of such architectural predispositions, we study computational and learning capabilities of typical dynamic neural network architectures. We also show how efficient finite memory models can be readily extracted from *untrained* networks and argue that such models should be used as baselines when comparing dynamic network performances in a supervised learning task. Finally, potential applications of the Markovian architectural predisposition of dynamic neural networks in bioinformatics are presented.

1 Introduction

There has been a considerable research activity in connectionist processing of sequential symbolic structures. For example, researchers have been interested in formulating models of human performance in processing linguistic patterns of various complexity (e.g. [1]).

Of special importance are dynamic neural network architectures capable of naturally dealing with sequential data through recursive processing. Such architectures are endowed with feedback delay connections that enable the models to operate with a "neural" memory" in the form of past activations of a selected subset of neurons, sometimes referred to as recurrent neurons. It is expected that activations of such recurrent neurons will code all the "important" information from the past that is needed to solve a given task. In other words, the recurrent activations can be thought of as representing, in some "relevant" way, the temporal context in which the current input item is observed. The "relevance" is given by the nature of the task, be it next symbol prediction, or mapping sequences in a topographically ordered manner.

It was a bit surprising then, when researchers reported that when training dynamic neural networks to process language structures, activations of re-

current neurons displayed a considerable amount of structural differentiation even *prior to learning* [2, 3, 1]. Following [1], we refer to this phenomenon as the *architectural bias* of dynamic neural network architectures.

It has been recently shown, both theoretically and empirically, that the structural differentiation of recurrent activations before the training has much deeper implications [4, 5]. When initialized with small weights (standard initialization procedure), typical dynamic neural network architectures will organize recurrent activations in a suffix-based Markovian manner and it is possible to extract from such *untrained* networks predictive models comparable to efficient finite memory source models called variable memory length Markov models [6]. In addition, in such networks, recurrent activations tend to organize along a self-similar fractal substrate the fractal and multifractal properties of which can be studied in a rigorous manner [7]. Also, it is possible to rigorously characterize learning capabilities of dynamic neural network architectures in the early stages of learning [5].

Analogously, the well-known Self-Organizing Map (SOM) [8] for topographic low-dimensional organization of high-dimensional vectorial data has been reformulated to enable processing of sequential data. Typically, standard SOM is equipped with additional feed-back connections [9, 10, 11, 12, 13, 14]. Again, various forms of inherent predispositions of such models to Markovian organization of processed data have been discovered [15, 13, 16].

The aim of this paper is to present major developments in the phenomenon of architectural bias of dynamic neural network architectures in a unified framework. The paper has the following organization: First, a general state-space model formulation used throughout the paper is introduced in section 2. Then, still on the general high description level, we study in section 3 the link between Markovian suffix-based state space organization and contractive state transition maps. Basic tools for measuring geometric complexity of fractal sets are introduced as well. Detailed studies of several aspects of the architectural bias phenomenon in the supervised and unsupervised learning scenarios are presented in sections 4 and 5, respectively. Section 6 brings a flavour of potential applications of the architectural bias in bioinformatics. Finally, key findings are summarized in section 7.

2 General model formulation

We consider models that recursively process inputs $\mathbf{x}(t)$ from a set $\mathcal{X} \subseteq \mathbb{R}^{N_I}$ by updating their state $\mathbf{r}(t)$ in a bounded set $\mathcal{R} \subseteq \mathbb{R}^N$, and producing outputs $\mathbf{y}(t) \in \mathcal{Y}$, $\mathcal{Y} \subseteq \mathbb{R}^{N_O}$. State space model representation takes the form:

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{r}(t)) \tag{1}$$

$$\mathbf{r}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{r}(t-1)). \tag{2}$$

Processing of an input time series $\mathbf{x}(t) \in \mathcal{X}$, $t = 1, 2, ...$, starts by initializing the model with some $\mathbf{r}(0) \in \mathcal{R}$ and then recursively updating the state $\mathbf{r}(t) \in$

$\mathcal{R}$ and output $\mathbf{y}(t) \in \mathcal{Y}$ via functions $\mathbf{f} : \mathcal{X} \times \mathcal{R} \to \mathcal{R}$ and $\mathbf{h} : \mathcal{R} \to \mathcal{Y}$, respectively. This is illustrated in figure 1.

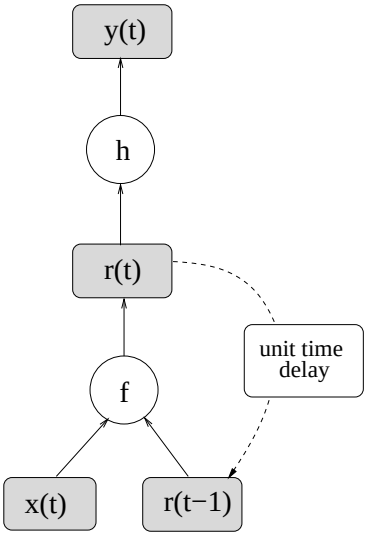

Fig. 1. The basic state space model used throughout the paper. Processing of an input time series $\mathbf{x}(t) \in \mathcal{X}$, $t = 1, 2, ...$, is done by recursively updating the state $\mathbf{r}(t) \in \mathcal{R}$ and output $\mathbf{y}(t) \in \mathcal{Y}$ via functions $\mathbf{f} : \mathcal{X} \times \mathcal{R} \to \mathcal{R}$ and $\mathbf{h} : \mathcal{R} \to \mathcal{Y}$, respectively.

We consider inputs coming from a finite alphabet $\mathcal{A}$ of A symbols. The set of all finite sequences over $\mathcal{A}$ is denoted by $\mathcal{A}^*$. The set $\mathcal{A}^*$ without the empty string ϵ is denoted by $\mathcal{A}^+$. The set of all sequences over $\mathcal{A}$ with exactly n symbols (n-blocks) is denoted by $\mathcal{A}^n$. Each symbol $s \in \mathcal{A}$ is mapped to its real-vector representation $\mathbf{c}(s)$ by an injective coding function $\mathbf{c} : \mathcal{A} \to \mathcal{X}$. The state transition process (2) can be viewed as a composition of fixed-input maps

$$\mathbf{f}_s(\mathbf{r}) = \mathbf{f}(\mathbf{c}(s), \mathbf{r}), \quad s \in \mathcal{A}. \tag{3}$$

In particular, for a sequence $s_{1:n} = s_1...s_{n-2}s_{n-1}s_n$ over $\mathcal{A}$ and $\mathbf{r} \in \mathcal{R}$, we write

$$\begin{aligned} \mathbf{f}_{s_{1:n}}(\mathbf{r}) &= \mathbf{f}_{s_n}(\mathbf{f}_{s_{n-1}}(...(\mathbf{f}_{s_2}(\mathbf{f}_{s_1}(\mathbf{r})))...)) \\ &= (\mathbf{f}_{s_n} \circ \mathbf{f}_{s_{n-1}} \circ ... \circ \mathbf{f}_{s_2} \circ \mathbf{f}_{s_1})(\mathbf{r}). \end{aligned} \tag{4}$$

3 Contractive state transitions

In this section we will show that by constraining the system to contractive state transitions we obtain state organizations with Markovian flavour. Moreover, one can quantify geometric complexity of such state space organizations using tools of fractal geometry.

3.1 Markovian state-space organization

Denote a Euclidean norm by $\|\cdot\|$. Recall that a mapping $\mathbf{F}: \mathcal{R} \to \mathcal{R}$ is said to be a contraction with contraction coefficient $\rho \in [0,1)$, if for any $\mathbf{r}, \mathbf{r}' \in \mathcal{R}$, it holds

$$\|\mathbf{F}(\mathbf{r}) - \mathbf{F}(\mathbf{r}')\| \leq \rho \cdot \|\mathbf{r} - \mathbf{r}'\|. \tag{5}$$

$\mathbf{F}$ is a contraction if there exists $\rho \in [0,1)$ so that $\mathbf{F}$ is a contraction with contraction coefficient ρ.

Assume now that each member of the family $\{\mathbf{f}_s\}_{s\in\mathcal{A}}$ is a contraction with contraction coefficient ρ_s and denote the weakest contraction rate in the family by

$$\rho_{max} = \max_{s\in\mathcal{A}} \rho_s.$$

Consider a sequence $s_{1:n} = s_1 ... s_{n-2} s_{n-1} s_n \in \mathcal{A}^n$, $n \geq 1$. Then for any two prefixes u and v and for any state $\mathbf{r} \in \mathcal{R}$, we have

$$\begin{aligned}
\|\mathbf{f}_{us_{1:n}}(\mathbf{r}) - \mathbf{f}_{vs_{1:n}}(\mathbf{r})\| &= \|\mathbf{f}_{s_{1:n-1}s_n}(\mathbf{f}_u(\mathbf{r})) - \mathbf{f}_{s_{1:n-1}s_n}(\mathbf{f}_v(\mathbf{r}))\| && (6)\\
&= \|\mathbf{f}_{s_n}(\mathbf{f}_{s_{1:n-1}}(\mathbf{f}_u(\mathbf{r}))) - \mathbf{f}_{s_n}(\mathbf{f}_{s_{1:n-1}}(\mathbf{f}_v(\mathbf{r})))\| && (7)\\
&\leq \rho_{max} \cdot \|\mathbf{f}_{s_{1:n-1}}(\mathbf{f}_u(\mathbf{r})) - \mathbf{f}_{s_{1:n-1}}(\mathbf{f}_v(\mathbf{r}))\|, && (8)
\end{aligned}$$

and consequently

$$\begin{aligned}
\|\mathbf{f}_{us_{1:n}}(\mathbf{r}) - \mathbf{f}_{vs_{1:n}}(\mathbf{r})\| &\leq \rho_{max}^n \cdot \|\mathbf{f}_u(\mathbf{r}) - \mathbf{f}_v(\mathbf{r})\| && (9)\\
&\leq \rho_{max}^n \cdot diam(\mathcal{R}), && (10)
\end{aligned}$$

where $diam(\mathcal{R})$ is the diameter of the set $\mathcal{R}$, i.e. $diam(\mathcal{R}) = \sup_{x,y\in\mathcal{R}} \|x - y\|$.

By similar arguments, for $L < n$,

$$\begin{aligned}
\|\mathbf{f}_{s_{1:n}}(\mathbf{r}) - \mathbf{f}_{s_{n-L+1:n}}(\mathbf{r})\| &\leq \rho^L \cdot \|\mathbf{f}_{s_{1:n-L}}(\mathbf{r}) - \mathbf{r}\| && (11)\\
&\leq \rho^L \cdot diam(\mathcal{R}). && (12)
\end{aligned}$$

There are two related lessons to be learnt from this exercise:

1. No matter what state $\mathbf{r} \in \mathcal{R}$ the model is in, the final states $\mathbf{f}_{us_{1:n}}(\mathbf{r})$ and $\mathbf{f}_{vs_{1:n}}(\mathbf{r})$ after processing two sequences with a long common suffix $s_{1:n}$ will lie close to each other. Moreover, the greater the length n of the common suffix, the closer lie the final states $\mathbf{f}_{us_{1:n}}(\mathbf{r})$ and $\mathbf{f}_{vs_{1:n}}(\mathbf{r})$.

2. If we could only operate with finite input memory of depth L, then all reachable states from an initial state $\mathbf{r}_0 \in \mathcal{R}$ could be collected in a finite set
$$\mathcal{C}_L(\mathbf{r}_0) = \{\mathbf{f}_w(\mathbf{r}_0) \mid w \in \mathcal{A}^L\}.$$
Consider now a long sequence $s_{1:n}$ over $\mathcal{A}$. Disregarding the initial transients for $1 \leq t \leq L-1$, the states $\mathbf{f}_{s_{1:t}}(\mathbf{r}_0)$ of the original model (2) can be approximated by the states $\mathbf{f}_{s_{t-L+1:t}}(\mathbf{r}_0) \in \mathcal{C}_L(\mathbf{r}_0)$ of the finite memory model to arbitrarily small precision, as long as the memory depth L is long enough.

What we have just described can be termed *Markovian organization of the model state space*. Information processing states that result from processing sequences sharing a common suffix naturally cluster together. In addition, the spatial extent of the cluster reflects the length of the common suffix - longer suffixes lead to tighter cluster structure.

If, in addition, the readout function $\mathbf{h}$ is Lipschitz with coefficient $\kappa > 0$, i.e. for any $\mathbf{r}, \mathbf{r}' \in \mathcal{R}$,
$$\|\mathbf{h}(\mathbf{r}) - \mathbf{h}(\mathbf{r}')\| \leq \kappa \cdot \|\mathbf{r} - \mathbf{r}'\|,$$
then
$$\|\mathbf{h}(\mathbf{f}_{us_{1:n}}(\mathbf{r})) - \mathbf{h}(\mathbf{f}_{vs_{1:n}}(\mathbf{r}))\| \leq \kappa \cdot \rho^n \cdot \|\mathbf{f}_u(\mathbf{r}) - \mathbf{f}_v(\mathbf{r})\|.$$
Hence, for arbitrary prefixes u, v, we have that for sufficiently long common suffix length n, the model outputs resulting from driving the system with $us_{1:n}$ and $vs_{1:n}$ can be made arbitrarily close. In particular, on the same long input sequence (after initial transients of length $L-1$), the model outputs (1) will be closely shadowed by those of the finite memory model operating only on the most recent L symbols (and hence on states from $\mathcal{C}_L(\mathbf{r})$), as long as L is large enough.

3.2 Measuring complexity of state space activations

Let us first introduce measures of size of geometrically complicated objects called fractals (see e.g. [17]). Let $K \subseteq \mathcal{R}$. For $\delta > 0$, a *δ-fine cover of K* is a collection of sets of diameter $\leq \delta$ that cover K. Denote by $N_\delta(K)$ the smallest possible cardinality of a δ-fine cover of K.

Definition 1. *The* upper *and* lower box-counting dimensions *of K are defined as*
$$\dim_B^+ K = \limsup_{\delta \to 0} \frac{\log N_\delta(K)}{-\log \delta} \quad \textit{and} \quad \dim_B^- K = \liminf_{\delta \to 0} \frac{\log N_\delta(K)}{-\log \delta}, \tag{13}$$
respectively.

Let $\gamma > 0$. For $\delta > 0$, define

$$\mathcal{H}_\delta^\gamma(K) = \inf_{\Gamma_\delta(K)} \sum_{B \in \Gamma_\delta(K)} (diam(B))^\gamma, \tag{14}$$

where the infimum is taken over the set $\Gamma_\delta(K)$ of all countable δ-fine covers of K. Define

$$\mathcal{H}^\gamma(K) = \lim_{\delta \to 0} \mathcal{H}_\delta^\gamma(K).$$

Definition 2. *The* Hausdorff dimension of the set K *is*

$$\dim_H K = \inf\{\gamma |\ \mathcal{H}^\gamma(K) = 0\}. \tag{15}$$

It is well known that

$$\dim_H K \leq \dim_B^- K \leq \dim_B^+ K. \tag{16}$$

The Hausdorff dimension is more "subtle" than the box-counting dimensions: the former can capture details not detectable by the latter. For a more detailed discussion see e.g. [17].

Consider now a Bernoulli source $\mathcal{S}$ over the alphabet $\mathcal{A}$ and (without loss of generality) assume that all symbol probabilities are nonzero.

The system (2) can be considered an *Iterative Function System* (IFS) [18] $\{\mathbf{f}_s\}_{s \in \mathcal{A}}$. If all the fixed input maps $\mathbf{f}_s$ are contractions (contractive IFS), there exists a unique set $K \subseteq \mathcal{R}$, called the IFS attractor, that is invariant under the action of the IFS:

$$K = \bigcap_{s \in \mathcal{A}} \mathbf{f}_s(K).$$

As the system (2) is driven by an input stream generated by $\mathcal{S}$, the states $\{\mathbf{r}(t)\}$ converge to K. In other words, after some initial transients, state trajectory of the system (2) "samples" the invariant set K (chaos game algorithm).

It is possible to upper-bound fractal dimensions of K [17]:

$$\dim_B^+ K \leq \gamma, \quad \text{where} \quad \sum_{s \in \mathcal{A}} \rho_s^\gamma = 1.$$

If the IFS obeys the open set condition, i.e. all $\mathbf{f}_s(K)$ are disjoint, we can get a lower bound on the dimension of K as well: assume there exist $0 < \kappa_s < 1$, $s \in \mathcal{A}$, such that

$$\|\mathbf{f}_s(\mathbf{r}) - \mathbf{f}_s(\mathbf{r}')\| \geq \kappa_s \cdot \|\mathbf{r} - \mathbf{r}'\|.$$

Then [17],

$$\dim_H K \geq \gamma, \quad \text{where} \quad \sum_{s \in \mathcal{A}} \kappa_s^\gamma = 1.$$

4 Supervised learning setting

In this section, we will consider the implications of Markovian state-space organization for recurrent neural networks (RNN) formulated and trained in a supervised setting. We present the results for a simple 3-layer topology of RNN. The results can be easily generalized to more complicated architectures.

4.1 Recurrent Neural Networks and Neural Prediction Machines

Let $\mathbf{W}^{r,x}$, $\mathbf{W}^{r,r}$ and $\mathbf{W}^{y,r}$ denote real $N \times N_I$, $N \times N$ and $N_O \times N$ weight matrices, respectively. For a neuron activation function g, we denote its element-wise application to a real vector $\mathbf{a} = (a_1, a_2, ..., a_k)^T$ by $\mathbf{g}(\mathbf{a})$, i.e. $\mathbf{g}(\mathbf{a}) = (g(a_1), g(a_2), ..., g(a_k))^T$. Then, the state transition function $\mathbf{f}$ in (2) takes the form

$$\mathbf{f}(\mathbf{x}, \mathbf{r}) = \mathbf{g}(\mathbf{W}^{r,x}\mathbf{x} + \mathbf{W}^{r,r}\mathbf{r} + \mathbf{t}_f), \tag{17}$$

where $\mathbf{t}_f \in \mathbb{R}^N$ is a bias term.

Often, the output function $\mathbf{h}$ reads:

$$\mathbf{h}(\mathbf{r}) = \mathbf{g}(\mathbf{W}^{y,r}\mathbf{r} + \mathbf{t}_h), \tag{18}$$

with the bias term $\mathbf{t}_h \in \mathbb{R}^{N_O}$. However, for the purposes of symbol stream processing, the output function $\mathbf{h}$ is sometimes realized as a piece-wise constant function consisting of a collection of multinomial next-symbol distributions, one for each compartment of the partitioned state space $\mathcal{R}$. RNN with such an output function were termed *Neural Prediction Machines* (NPM) in [4]. More precisely, assuming the symbol alphabet $\mathcal{A}$ contains A symbols, the output space $\mathcal{Y}$ is the simplex

$$\mathcal{Y} = \{\mathbf{y} = (y_1, y_2, ..., y_A)^T \in \mathbb{R}^A \mid \sum_{i=1}^{A} y_i = 1 \text{ and } y_i \geq 0\}.$$

The next-symbol probabilities $\mathbf{h}(\mathbf{r}) \in \mathcal{Y}$ are estimates of the relative frequencies of symbols, conditional on RNN state $\mathbf{r}$. Regions of constant probabilities are determined by vector quantizing the set of recurrent activations that result from driving the network with the training sequence:

1. Given a training sequence $s_{1:n} = s_1 s_2 ... s_n$ over the alphabet $\mathcal{A} = \{1, 2, ..., A\}$, first re-set the network to the initial state $\mathbf{r}(0)$, and then, while driving the network with the sequence $s_{1:n}$, collect the recurrent layer activations in the set $\Gamma = \{\mathbf{r}(t) \mid 1 \leq t \leq n\}$.
2. Run your favorite vector quantizer with M codebook vectors $\mathbf{b}_1, ..., \mathbf{b}_M \in \mathcal{R}$, on the set Γ. Vector quantization partitions the state space $\mathcal{R}$ into M Voronoi compartments $V_1, ..., V_M$, of the codebook vectors[4] $\mathbf{b}_1, ..., \mathbf{b}_M$:

[4] ties are broken according to index order

$$V_i = \{\mathbf{r} \mid \|\mathbf{r} - \mathbf{b}_i\| = \min_j \|\mathbf{r} - \mathbf{b}_j\|\}. \quad (19)$$

All points in V_i are allocated to the codebook vector $\mathbf{b}_i$ via a projection $\pi : \mathcal{R} \to \{1, 2, ..., M\}$,

$$\pi(\mathbf{r}) = i, \quad \text{provided} \quad \mathbf{r} \in V_i. \quad (20)$$

3. Re-set the network again with the initial state $\mathbf{r}(0)$.
4. Set the [Voronoi compartment,next-symbol] counters $N(i, a)$, $i = 1, ..., M$, $a \in \mathcal{A}$, to zero.
5. Drive the network again with the training sequence $s_{1:n}$.
   For $1 \leq t < n$, if $\pi(\mathbf{r}(t)) = i$, and the next symbol s_{t+1} is a, increment the counter $N(i, a)$ by one.
6. With each codebook vector associate the next symbol probabilities[5]

$$P(a \mid i) = \frac{N(i, a)}{\sum_{b \in \mathcal{A}} N(i, b)}, \quad a \in \mathcal{A}, \quad i = 1, 2, ..., M. \quad (21)$$

7. The output map is then defined as

$$\mathbf{h}(\mathbf{r}) = P(\cdot \mid \pi(\mathbf{r})), \quad (22)$$

where the a-th component of $\mathbf{h}(\mathbf{r})$, $h_a(\mathbf{r})$, is the probability of next symbol being $a \in \mathcal{A}$, provided the current RNN state is $\mathbf{r}$.

Once the NPM is constructed, it can make predictions on a test sequence (a continuation of the training sequence) as follows: Let $\mathbf{r}(n)$ be the vector of recurrent activations after observing the training sequence $s_{1:n}$. Given a prefix $u_1 u_2, ...u_\ell$ of the test sequence, the NPM makes a prediction about the next symbol $u_{\ell+1}$ by:

1. Re-setting the network with $\mathbf{r}(n)$.
2. Recursively updating states $\mathbf{r}(n + t)$, $1 \leq t \leq \ell$, while driving the network with $u_1 u_2 ... u_\ell$.

[5] For bigger codebooks, we may encounter data sparseness problems. In such cases, it is advisable to perform smoothing of the empirical symbol frequencies by applying Laplace correction with parameter $\gamma > 0$:

$$P(a \mid i) = \frac{\gamma + N(i, a)}{\gamma \cdot A + \sum_{b \in \mathcal{A}} N(i, b)}.$$

The parameter γ can be viewed in the Dirichlet prior interpretation for the multinomial distribution $P(a \mid i)$ as the effective number of times each symbol $a \in \mathcal{A}$ was observed in the context of state compartment i, *prior* to counting the conditional symbol occurrences $N(i, a)$ in the training sequence. Typically, $\gamma = 1$, or $\gamma = A^{-1}$.

3. The probability of $u_{\ell+1}$ occurring is

$$h_{u_{\ell+1}}(\mathbf{r}) = P(u_{\ell+1} \mid \pi(\mathbf{r}(n+\ell))). \tag{23}$$

A schematic illustration of NPM is presented in figure 2.

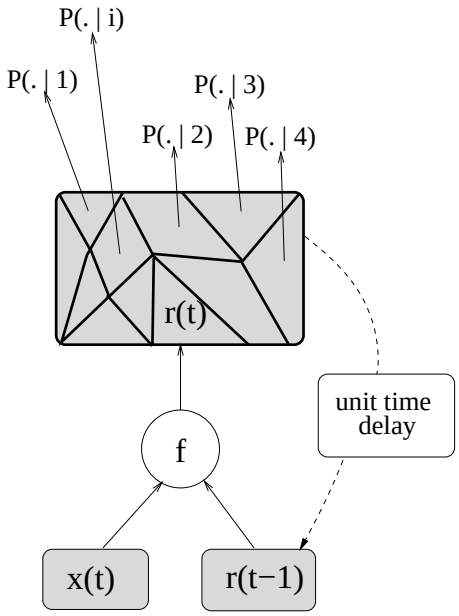

Fig. 2. Schematic illustration of Neural Prediction Machine. The output function is realized as a piece-wise constant function consisting of a collection of multinomial next-symbol distributions $P(\cdot|i)$, one for each compartment i of the state space $\mathcal{R}$.

4.2 Variable memory length Markov models and NPM built on untrained RNN

In this section we will present intuitive arguments for a strong link between Neural Prediction Machines (NPM) constructed on untrained RNN and a class of efficient implementations of Markov models called *Variable memory length Markov models* (VLMM) [6]. The arguments will be made more extensive and formal in section 4.5.

In *Markov models* (MMs) of (fixed) order L, the conditional next-symbol distribution over alphabet $\mathcal{A}$, given the history of symbols $s_{1:t} = s_1 s_2 ... s_t \in \mathcal{A}^t$ observed so far, is written in terms of the last L symbols ($L \leq t$),

$$P(s_{t+1} \mid s_{1:t}) = P(s_{t+1} \mid s_{t-L+1:t}). \tag{24}$$

For large memory lengths L, the estimation of prediction probabilities $P(s|w)$, $w \in \mathcal{A}^L$, can become infeasible. To overcome this problem, VLMM were introduced. The key feature of VLMMs is that they permit prediction contexts of variable length, depending on the particular history of observed symbols.

Suppose we are given a long training sequence on which we calculate empirical frequencies $\hat{P}_\ell$ of ℓ-blocks over $\mathcal{A}$. Let $w \in \mathcal{A}^n$ be a potential prediction context of length n used to predict the next symbol $s \in \mathcal{A}$ according to the empirical estimates

$$\hat{P}(s|w) = \frac{\hat{P}_{n+1}(ws)}{\hat{P}_n(w)}.$$

If for a symbol $a \in \mathcal{A}$, such that $aw \in \mathcal{A}^{n+1}$, the empirical probability of the next symbol s,

$$\hat{P}(s|aw) = \frac{\hat{P}_{n+2}(aws)}{\hat{P}_{n+1}(aw)},$$

with respect to the extended context aw differs "significantly" from $\hat{P}(s|w)$, then extending the prediction context w with $a \in \mathcal{A}$ helps in the next-symbol predictions. Several decision criteria have been suggested in the literature. For example, extend the prediction context w with $a \in \mathcal{A}$, if the Kullback-Leibler divergence between the next-symbol distributions for the candidate prediction contexts w and aw, weighted by the prior distribution of the extended context aw, exceeds a given threshold [19] [20]. For other variants of decision criteria see [21] [22].

A natural representation of the set of prediction contexts, together with the associated next-symbol probabilities, has the form of a prediction suffix tree (PST) [19] [23]. The edges of PST are labelled by symbols from $\mathcal{A}$. From every internal node there is at most one outgoing edge labelled by each symbol. The nodes of PST are labelled by pairs $(s, \hat{P}(s|w))$, $s \in \mathcal{A}$, $w \in \mathcal{A}^+$, where w is a string associated with the walk starting from that node and ending in the root of the tree. Given a history $s_{1:t} = s_1 s_2 ... s_t$ of observed symbols up to time t, the corresponding prediction context is the deepest node in the PST reached by taking a walk labelled by the reversed string, $s_t ... s_2 s_1$, starting in the root.

It is a common practise to initialize the RNN weights with small numbers. In such cases, *prior to RNN training*, the state transition function (17) is initialized as a contraction. There is a good reason for this practise: unless one has a strong prior knowledge about the network dynamics [24], the sequence of desired bifurcations during the training process may be hard to achieve when starting from arbitrarily complicated network dynamics [25]. But, as argued in section 3.1, in *contractive NPMs*, histories of seen symbols sharing long common suffixes are mapped close to each other. Under finite input memory assumptions, such histories are likely to produce similar continuations and in contractive NPMs they are likely to appear in the same quantization region.

Dense areas in the RNN state space correspond to symbol histories with long common suffixes and are given more attention by the vector quantizer. Hence, the prediction contexts are analogous to those of VLMM in that deep memory is used only when there are many different symbol histories sharing a long common suffix. In such situations it is natural to consider deeper past contexts when predicting the future. This is done automatically by the vector quantizer in NPM construction as it devotes more codebook vectors to the dense regions than to the sparsely inhabited areas of the RNN state space. More codebook vectors in dense regions imply tiny quantization compartments V_i that in turn group symbol histories with long shared suffixes.

Note, however, that while prediction contexts of VLMMs are built in a supervised manner, i.e. deep memory is considered only if it is dictated by the conditional distributions over the next symbols, in contractive NPMs prediction contexts of variable length are constructed in an unsupervised manner: prediction contexts with deep memory are accepted if there is a *suspicion* that shallow memory may not be enough, i.e. when there are many different symbol histories in the training sequence that share a long common suffix.

4.3 Fractal Prediction Machines

A special class of affine contractive NPMs operating with finite input memory has been introduced in [26] under the name *Fractal Prediction Machines* (FPM). Typically, the state space $\mathcal{R}$ is an N-dimensional unit hypercube[6] $\mathcal{R} = [0,1]^N$, $N = \lceil \log_2 A \rceil$. The coding function $\mathbf{c} : \mathcal{A} \to \mathcal{X} = \mathcal{R}$ maps each of the A symbols of the alphabet $\mathcal{A}$ to a unique vertex of the hypercube $\mathcal{R}$. The state transition function (17) is of particularly simple form, as the activation function g is identity and the connection weight matrices are set to $\mathbf{W}^{r,x} = \rho\mathbf{I}$ and $\mathbf{W}^{r,x} = (1-\rho)\mathbf{I}$, where $0 < \rho < 1$ is a contraction coefficient and $\mathbf{I}$ is the $N \times N$ identity matrix. Hence, the state space evolution (2) reads:

$$\begin{aligned} \mathbf{r}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{r}(t-1)) \\ &= \rho\mathbf{r}(t-1) + (1-\rho)\mathbf{x}(t). \end{aligned} \tag{25}$$

The reset state is usually set to the center of $\mathcal{R}$, $\mathbf{r}_0 = \{\frac{1}{2}\}^N$. After fixing the memory depth L, the FPM states are confined to the set

$$\mathcal{C}_L(\mathbf{r}_0) = \{\mathbf{f}_w(\mathbf{r}_0) \mid w \in \mathcal{A}^L\}.$$

FPM construction proceeds in the same manner as that of NPM, except for the states $\mathbf{f}_{s_{1:t}}(\mathbf{r}_0)$ coding history of inputs up to current time t are approximated by their memory-L counterparts $\mathbf{f}_{s_{t-L+1:t}}(\mathbf{r}_0)$ from $\mathcal{C}_L(\mathbf{r}_0)$, as discussed in section 3.1.

[6] for $x \in \Re$, $\lceil x \rceil$ is the smallest integer y, such that $y \geq x$

4.4 Echo and Liquid State Machines

A similar architecture has been proposed by Jaeger in [27, 28]. So-called *Echo State Machines* (ESM) combine an untrained recurrent reservoir with a trainable linear output layer. Thereby, the state space model has the following form

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{x}(t), \mathbf{r}(t), \mathbf{y}(t-1)) \tag{26}$$

$$\mathbf{r}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{r}(t-1), \mathbf{y}(t-1)). \tag{27}$$

For training, the function $\mathbf{f}$ is fixed and chosen in such a way that it has the echo state property, i.e. the network state $\mathbf{r}(t)$ is uniquely determined by left infinite sequences. In practise, the dependence of $\mathbf{r}$ on $\mathbf{y}$ is often dropped such that this property is equivalent to the fading property of recurrent systems and it can be tested by determining the Lipschitz constant of $\mathbf{f}$ (which must be smaller than 1). The readout $\mathbf{h}$ is trainable. It is often chosen as a linear function such that training can be done efficiently using linear regression. Unlike FPMs, ESMs usually work with very high dimensional state representations and a transition function which is obtained as the composition of a nonlinear activation function and a linear mapping with (sufficiently small) *random* weights. This combination should guarantee that the transition function has sufficiently rich dynamics to represent "important" properties of input series within its reservoir.

A similar idea is the base for so-called *Liquid State Machines* (LSM) as proposed by Maass [29]. These are based on a fixed recurrent mechanism in combination with a trainable readout. Unlike FPM and ESM, LSMs consider continuous time, and they are usually based on biologically plausible circuits of spiking neurons with sufficiently rich dynamics. It has been shown in [29] that LSMs are approximation complete for operators with fading memory under mild conditions on the models. Possibilities to estimate the capacity of such models and their generalization ability have recently been presented in [30]. However, the fading property is not required for LSMs.

4.5 On the computational power of recursive models

Here we will rigorously analyze the computational power of recursive models with contractive transition function. The arguments are partially taken from [5]. Given a sequence $s_{1:t} = s_1 \dots s_t$, the *L-truncation* is given by

$$T_L(s_{1:t}) = \begin{cases} s_{t-L+1:t} & \text{if } t \leq L \\ s & \text{otherwise} \end{cases}$$

Assume $\mathcal{A}$ is a finite input alphabet and $\mathcal{Y}$ is an output set. A *Definite Memory Machine* (DMM) computes a function $f : \mathcal{A}^* \to \mathcal{Y}$ such that $f(s) = f(T_L(s))$ for some fixed memory length L and every sequence s. A *Markov model* (MM) fulfils $P(s_{t+1}|s_{1:t}) = P(s_{t+1}|T_L(s))$ for some L. Obviously, the function which

maps a sequence to the next symbol probability given by a Markov model defines a definite memory machine. Variable length Markov models are subsumed by standard Markov models taking L as the maximum length. Therefore, we will only consider definite memory machines in the following.

State space models as defined in eqs. (1-2) induce a function $\mathbf{h} \circ \mathbf{f}_\bullet(\mathbf{r}) : \mathcal{A}^* \to \mathcal{Y}, s_{1:t} \mapsto (\mathbf{h} \circ \mathbf{f}_{s_t} \circ \ldots \circ \mathbf{f}_{s_1})(\mathbf{r})$ as defined in (4) given a fixed initial value $\mathbf{r}$. As beforehand, we assume that the set of internal states $\mathcal{R}$ is bounded. Consider the case that $\mathbf{f}$ is a contraction w.r.t. $\mathbf{x}$ with parameter ρ and $\mathbf{h}$ is Lipschitz continuous with parameter κ. According to (12), $|\mathbf{f}_s(\mathbf{r}) - \mathbf{f}_{T_L(s)}(\mathbf{r})| \leq \rho^L \cdot diam(\mathcal{R})$. Note that the function $\mathbf{h} \circ \mathbf{f}_{T_L(\bullet)}(\mathbf{r})$ can be given by a definite memory machine. Therefore, if we choose $L = \log_\rho(\epsilon/(\kappa \cdot diam(\mathcal{R})))$, we find $|\mathbf{h} \circ \mathbf{f}_s(\mathbf{r}) - \mathbf{h} \circ \mathbf{f}_{T_L(s)}(\mathbf{r})| \leq \epsilon$. Hence, we can conclude that every recurrent system with contractive transition function can be approximated arbitrarily well by a finite memory model. This argument proves the Markovian property of RNNs with small weights:

Theorem 1. *Assume* $\mathbf{f}$ *is a contraction w.r.t.* $\mathbf{x}$*, and* $\mathbf{h}$ *is Lipschitz continuous. Assume* $\mathbf{r}$ *is a fixed initial state. Assume* $\epsilon > 0$*. Then there exists a definite memory machine with function* $g : \mathcal{A}^* \to \mathcal{Y}$ *such that* $|g(s) - \mathbf{h} \circ \mathbf{f}_s(\mathbf{r})| \leq \epsilon$ *for every input sequence* s*.*

As shown in [5], this argument can be extended to the situation where the input alphabet is not finite, but given by an arbitrary subset of a real-vector space.

It can easily be seen that standard recurrent neural networks with small weights implement contractions, since standard activation functions such as tanh are Lipschitz continuous and linear maps with sufficiently small weights are contractive. RNNs are often initialized by small random weights. Therefore, RNNs have a Markovian bias in early stages of training, first pushing solutions towards simple dynamics before going towards more complex models such as finite state automata and beyond.

It has been shown in [27] that ESMs without output feedback possess the so-called shadowing property, i.e. for every ϵ a memory length L can be found such that input sequences the last L entries of which are similar or identical lead to images which are at most ϵ apart. As a consequence, such mechanisms can also be simulated by Markovian models. The same applies to FPMs (the transition is a contraction by construction) and NPMs built on untrained RNNs randomly initialized with small weights.

The converse has also been proved in [5]: every definite memory machine can be simulated by a RNN with small weights. Therefore, an equivalence of small weight RNNs (and similar mechanisms) and definite memory machines holds. One can go a step further and consider the question in which cases

randomly initialized RNNs with small weights can simulate a given definite memory machine provided a suitable readout is chosen.

The question boils down to the question whether the images of input sequences s of a fixed memory length L under the recursion $(\mathbf{f}_{s_L} \circ \ldots \circ \mathbf{f}_{s_1})(\mathbf{r})$ are of a form such that they can be mapped to desired output values by a trainable readout. We can assume that the codes for the symbols s_i are linearly independent. Further, we assume that the transfer function of the network is a nonlinear and non algebraic function such as tanh. Then the terms $(\mathbf{f}_{s_L} \circ \ldots \circ \mathbf{f}_{s_1})(\mathbf{r})$ constitute functions of the weights which are algebraically independent since they consist of the composition of linearly independent terms and the activation function. Thus, the images of sequences of length L are almost surely disjoint such that they can be mapped to arbitrary output values by a trainable readout provided the readout function $\mathbf{h}$ fulfils the universal approximation capability as stated e.g. in [31] (e.g. FNNs with one hidden layer). If the dimensionality of the output is large enough (dimensionality L^A), the outputs are almost surely linearly independent such that even a linear readout is sufficient. Thus, NPMs and ESMs with *random* initialization constitute universal approximators for finite memory models provided sufficient dimensionality of the reservoir.

4.6 On the generalization ability of recursive models

Another important issue of supervised adaptive models is their generalization ability, i.e. the possibility to infer from learnt data to new examples. It is well known that the situation is complex for recursive models due to the relatively complex inputs: sequences of priorly unrestricted length can, in principle, encode behavior of priorly unlimited complexity. This observation can be mathematically substantiated by the fact that generalization bounds for recurrent networks as derived e.g. in [32, 33] depend on the input distribution. The situation is different for finite memory models for which generalization bounds independent of the input distribution can be derived. A formal argument which establishes generalization bounds of RNNs with Markovian bias has been provided in [5], whereby the bounds have been derived also for the setting of continuous input values stemming from a real-vector space. Here, we only consider the case of a finite input alphabet $\mathcal{A}$.

Assume $\mathcal{F}$ is the class of functions which can be implemented by a recurrent model with Markovian bias. For simplicity, we restrict to the case of binary classification, i.e. outputs stem from $\{0, 1\}$, e.g. adding an appropriate discretization to the readout $\mathbf{h}$. Assume P is an unknown underlying input-output distribution which has to be learnt. Assume a set of m input-output pairs, $\mathcal{T}_m = \{(s^1, y^1), \ldots, (s^m, y^m)\}$, is given; the pairs are sampled from P in an i.i.d. manner[7]. Assume a learning algorithm outputs a function g when trained on the samples from $\mathcal{T}_m$. The *empirical error* of the algorithm

[7] independent identically distributed

is defined as

$$\hat{E}_{\mathcal{T}_m}(g) = \frac{1}{m}\sum_{i=1}^{m} |g(s^i) - y^i| \,.$$

This is usually minimized during training. The *real error* is given by the quantity

$$E_P(g) = \int |g(s) - y| \; d_P(s,y) \,.$$

A learning algorithm is said to generalize to unseen data if the real error is minimized by the learning scheme.

The capability of a learning algorithm of generalizing from the training set to new examples depends on the capacity of the function class used for training. One measure of the complexity is offered by the Vapnik Chervonenkis (VC) dimension of the function class $\mathcal{F}$. The VC dimension of $\mathcal{F}$ is the size of the largest set such that every binary mapping on this set can be realized by a function in $\mathcal{F}$. It has been shown by Vapnik [34] that the following bound holds for sample size m, $m \geq \mathcal{VC}(\mathcal{F})$ and $m > 2/\epsilon$:

$$\begin{aligned} &P^m\{\mathcal{T}_m \;\mid\; \exists g \in \mathcal{F} \text{ s.t. } \; |E_P(g) - \hat{E}_{\mathcal{T}_m}(g)| > \epsilon\} \\ &\leq 4 \cdot \left(\frac{2 \cdot \mathrm{e} \cdot m}{\mathcal{VC}(\mathcal{F})}\right) \cdot \exp(-\epsilon^2 \cdot m/8) \end{aligned}$$

Obviously, the number of different binary Markovian classifications with finite memory length L and alphabet $\mathcal{A}$ of A symbols is restricted by 2^{L^A}, thus, the VC dimension of the class of Markovian classifications is limited by L^A. Hence its generalization ability is guaranteed. Recurrent networks with Markovian bias can be simulated by finite memory models. Thereby, the necessary input memory length L can be estimated from the contraction coefficient ρ of the recursive function, the Lipschitz constant κ of the readout, and the extent of the state space $\mathcal{R}$, by the term $-\log_\rho(2 \cdot \kappa \cdot diam(\mathcal{R}))$. This holds since we can approximate every binary classification up to $\epsilon = 0.5$ by a finite memory machine using this length as shown in Theorem 1. This argument proves the generalization ability of ESMs, FPMs, or NPMs with fixed recursive transition function and finite input alphabet $\mathcal{A}$ for which a maximum memory length can be determined a priori. The argumentation can be generalized to the case of arbitrary (continuous) inputs as shown in [5].

If we train a model (e.g. a RNN with small weights) in such a way that a Markovian model with finite memory length arises, whereby the contraction constant of the recursive part (and hence the corresponding maximum memory length L) is not fixed, these bounds do not apply. In this case, one can use arguments from the theory of structural risk minimization as provided e.g. in [35]. Denote by $\mathcal{F}_i$ the set of recursive functions which can be simulated using memory length i (this corresponds to a contraction coefficient $(2 \cdot \kappa \cdot diam(\mathcal{R})^{-1/i}$). Obviously, the VC dimension of $\mathcal{F}_i$ is i^A. Assume a classifier is trained on a sample $\mathcal{T}_m$ of m i.i.d. input-output pairs (s^j, y^j), $j = 1, 2, ...m$,

generated from P. Assume the number of errors on $\mathcal{T}_m$ of the trained classifier g is k, and the recursive model g is contained in $\mathcal{F}_i$. Then, according to [35] (Theorem 2.3), the generalization error $E_P(g)$ can be upper bounded by the following term with probability $1 - \delta$:

$$\frac{1}{m}\left((2 + 4\ln 2)k + 4\ln 2i + 4\ln\frac{4}{\delta} + 4i^A \cdot \ln\left(\frac{2em}{i^A}\right)\right) .$$

(Here, we chose the prior probabilities used in [35] (Theorem 2.3) as $1/2^i$ for the probability that g is in $\mathcal{F}_i$ and $1/2^k$ that at most k errors occur.) Unlike the bounds derived e.g. in [15], this bound holds for any *posterior* finite memory length L obtained during training. In practise, the memory length can be estimated from the contraction and Lipschitz constants as shown above. Thus, unlike for general RNNs, generalization bounds which do not depend on the input distribution, but rather on the input memory length, can be derived for RNNs with Markovian bias.

4.7 Verifying the architectural bias

The fact that one can extract from untrained recurrent networks Neural Prediction Machines (NPM, section 4.1) that yield comparable performance to Variable Memory Length Markov Models (VLMM, section 4.2) has been demonstrated e.g. in [4]. This is of vital importance for model building and evaluation, since in the light of these findings, the base line models in neural-based symbolic sequence processing tasks should in fact be VLMMs. If a neural network model after the training phase cannot beat a NPM extracted from an untrained model, there is obviously no point in training the model first place. This may sound obvious, but there are studies e.g. in the field of cognitive aspects of natural language processing, where trained RNN are compared with fixed order Markov models [1]. Note that such Markov models are often inferior to VLMM.

In [4], five model classes were considered:

- RNN trained via Real Time Recurrent Learning [36] coupled with Extended Kalman Filtering [37, 38]. There were A input and output units, one for each symbol in the alphabet $\mathcal{A}$, i.e. $N_I = N_O = A$. The inputs and desired outputs were encoded through binary one-of-A encoding. At each time step, the output activations were normalized to obtain next-symbol probabilities.
- NPM extracted from trained RNN.
- NPM extracted from RNN *prior to training*.
- Fixed order Markov models.
- VLMM.

The models were tested on two data sets:

- A series of quantized activations of a laser in a chaotic regime. This sequence can be modelled quite successfully with finite memory predictors, although, because of the limited finite sequence length, predictive contexts of variable memory depth are necessary.
- An artificial language exhibiting deep recursive structures. Such structures cannot be grasped by finite memory models and fully trained RNN should be able to outperform models operating with finite input memory and NPM extracted from untrained networks.

Model performance was measured by calculating the average (per symbol) negative log-likelihood on the hold-out set (not used during the training). As expected, given fine enough partition of the state space, the performances of RNNs and NPMs extracted from trained RNNs were almost identical. Because of finite sample size effects, fixed-order Markov models were beaten by VLMMs. Interestingly enough, when considering models with comparable number of free parameters, the performance of NPMs extracted prior to training mimicked that of VLMMs. Markovian bias of untrained RNN was clearly visible.

On the laser sequence, negative log-likelihood on the hold out set improved slightly through an expensive RNN training, but overall, almost the same performance could be achieved by a cheap construction of NPMs from randomly initialized *untrained* networks.

On the deep-recursion set, trained RNNs could achieve much better performance. However, it is essential to quantify how much useful information has really been induced during the training. As argued in this study, this can only be achieved by consulting VLMMs and NPMs extracted *before* training.

A large-scale comparison in [26] of Fractal Prediction Machines (FPM, section 4.3) with both VLMM an fixed order Markov models revealed an almost equivalent performance (modulo model size) of FPM and VLMM. Again, fixed order Markov models were inferior to both FPM and VLMM.

4.8 Geometric complexity of RNN state activations

It has been well known that RNN state space activations $\mathbf{r}$ often live on a fractal support of a self-similar nature [39]. As outlined in section 3.2, in case of contractive RNN, one can actually estimate the size and geometric complexity of the RNN state space evolution.

Consider a Bernoulli source $\mathcal{S}$ over the alphabet $\mathcal{A}$. When we drive a RNN with symbolic sequences generated by $\mathcal{S}$, we get recurrent activations $\{\mathbf{r}(t)\}$ that tend to group in clusters sampling the invariant set K of the RNN iterative function system $\{\mathbf{f}_s\}_{s\in\mathcal{A}}$.

Note that for each input symbol $s \in \mathcal{A}$, we have an affine mapping acting on $\mathcal{R}$:

$$Q_s(\mathbf{r}) = \mathbf{W}^{r,r}\mathbf{r} + \mathbf{W}^{r,x}\mathbf{c}(s) + \mathbf{t}_f.$$

The range of possible net-in activations of recurrent units for input s is then

$$C_s = \bigcup_{1 \leq i \leq N} [Q_s(\mathcal{R})]_i, \tag{28}$$

where $[B]_i$ is the slice of the set B, i.e. $[B]_i = \{r_i |\ \mathbf{r} = (r_1, ..., r_N)^T \in B\}$, $i = 1, 2, ..., N$.

By finding upper bounds on the values of derivatives of activation function,

$$g'_s = \sup_{v \in C_s} |g'(v)|, \quad g'_{max} = \max_{s \in \mathcal{A}} g'_s,$$

and setting

$$\rho_s^{max} = \alpha_+(\mathbf{W}^{r,r}) \cdot g'_s, \tag{29}$$

where $\alpha_+(\mathbf{W}^{r,r})$ is the largest singular value of $\mathbf{W}^{r,r}$, we can bound the upper box-counting dimension of RNN state activations [40]: Suppose $\alpha_+(\mathbf{W}^{r,r}) \cdot g'_{max} < 1$ and let γ_{max} be the unique solution of $\sum_{s \in \mathcal{A}} (\rho_s^{max})^\gamma = 1$. Then, $\dim_B^+ K \leq \gamma_{max}$.

Analogously, Hausdorff dimension of RNN sates can be lower-bounded as follows: Define

$$q = \min_{s, s' \in \mathcal{A}, s \neq s'} \|\mathbf{W}^{r,x}(\mathbf{c}(s) - \mathbf{c}(s'))\|$$

and assume $\alpha_+(\mathbf{W}^{r,r}) < \min\left\{(g'_{max})^{-1}, q \cdot diam(\mathcal{R})\right\}$. Let γ_{min} be the unique solution of $\sum_{s \in \mathcal{A}} (\rho_s^{min})^\gamma = 1$, where $\rho_s^{min} = \alpha_-(\mathbf{W}^{r,r}) \cdot \inf_{v \in C_s} |g'(v)|$ and $\alpha_-(\mathbf{W}^{r,r})$ is the smallest singular value of $\mathbf{W}^{r,r}$. Then, $\dim_H K \geq \gamma_{min}$.

Closed-form (less tight) bounds are also possible [40]:

$$\dim_B^+ K \leq -\frac{\log A}{\log N + \log W_{max}^{r,r} + \log g'_{max}},$$

where $W_{max}^{r,r} = \max_{1 \leq i,j \leq N} |W_{ij}^{r,r}|$; and

$$\dim_H K \geq -\frac{\log A}{\log \rho^{min}},$$

where $\rho^{min} = \min_{s \in \mathcal{A}} \rho_s^{min}$.

A more involved multifractal analysis of RNN state activations under more general stochastic sources driving the RNN input can be found in [7].

5 Unsupervised learning setting

In this section, we will study model formulations for constructing topographic maps of sequential data. While most approaches to topographic map formation assume that the data points are members of a finite-dimensional vector space of a fixed dimension, there has been a considerable interest in extending topographic maps to more general data structures, such as sequences or

trees. Several modifications of standard Self-Organizing Map (SOM) [8] to sequences and/or tree structures have been proposed in the literature [41, 15]. Several modifications of SOM equip standard SOM with additional feed-back connections that allow for natural processing of recursive data types. Typical examples of such models are Temporal Kohonen Map [9], recurrent SOM [10], feedback SOM [11], recursive SOM [12], merge SOM [13] and SOM for structured data [14].

In analogy to well-known capacity results derived for supervised recursive models e.g. in [42, 43], some approaches to investigate the principled capacity of these models have been presented in the last few years. Thereby, depending on the model, the situation is diverse: whereas SOM for structured data and merge SOM (both with arbitrary weights) are equivalent to finite automata [44, 13], recursive SOM equipped with a somewhat simplified context model can simulate at least pushdown automata [45]. The Temporal Kohonen Map and recurrent SOM are restricted to finite memory models due to their restricted recurrence even with arbitrary weights [45, 44]. However, the exact capacity of most unsupervised recursive models as well as biases which arise during training have hardly been understood so far.

5.1 Recursive SOM

In this section we will analyse recursive SOM (RecSOM) [12]. There are two main reasons for concentrating on RecSOM. First, RecSOM transcends simple local recurrence of leaky integrators of earlier SOM-motivated models for processing sequential data and consequently can represent much richer dynamical behavior [15]. Second, the underlying continuous state-space dynamics of RecSOM makes this model particularly suitable for analysis along the lines of section 3.1.

The map is formed at the recurrent layer. Each unit $i \in \{1, 2, ..., N\}$ in the map has two weight vectors associated with it:

- $\mathbf{w}_i^{r,x} \in \mathcal{X}$ – linked with an N_I-dimensional input $\mathbf{x}(t)$ feeding the network at time t
- $\mathbf{w}_i^{r,r} \in \mathcal{R}$ – linked with the context

$$\mathbf{r}(t-1) = (r_1(t-1), r_2(t-1), ..., r_N(t-1))^T$$

containing map activations $r_i(t-1)$ from the previous time step.

Hence, the weight matrices $\mathbf{W}^{r,x}$ and $\mathbf{W}^{r,r}$ can be written as $\mathbf{W}^{r,x} = ((\mathbf{w}_1^{r,x})^T, (\mathbf{w}_2^{r,x})^T, ..., (\mathbf{w}_N^{r,x})^T)^T$ and $\mathbf{W}^{r,r} = ((\mathbf{w}_1^{r,r})^T, (\mathbf{w}_2^{r,r})^T, ..., (\mathbf{w}_N^{r,r})^T)^T$, respectively.

The activation of a unit i at time t is computed as

$$r_i(t) = \exp(-d_i(t)), \tag{30}$$

where

$$d_i(t) = \alpha \cdot \|\mathbf{x}(t) - \mathbf{w}_i^{r,x}\|^2 + \beta \cdot \|\mathbf{r}(t-1) - \mathbf{w}_i^{r,r}\|^2. \quad (31)$$

In eq. (31), $\alpha > 0$ and $\beta > 0$ are parameters that respectively quantify the influence of the input and the context on the map formation. The output of RecSOM is the index of the unit with maximum activation (best-matching unit),

$$y(t) = h(\mathbf{r}(t)) = \underset{i \in \{1,2,...,N\}}{\operatorname{argmax}} \; r_i(t). \quad (32)$$

Both weight vectors can be updated using a SOM-type learning [12]:

$$\Delta \mathbf{w}_i^{r,x} = \gamma \cdot \nu(i, y(t)) \cdot (\mathbf{x}(t) - \mathbf{w}_i^{r,x}), \quad (33)$$
$$\Delta \mathbf{w}_i^{r,r} = \gamma \cdot \nu(i, y(t)) \cdot (\mathbf{r}(t-1) - \mathbf{w}_i^{r,r}), \quad (34)$$

where $0 < \gamma < 1$ is the learning rate. Neighborhood function $\nu(i,k)$ is a Gaussian (of width σ) on the distance $d(i,k)$ of units i and k in the map:

$$\nu(i,k) = e^{-\frac{d(i,k)^2}{\sigma^2}}. \quad (35)$$

The "neighborhood width", σ, linearly decreases in time to allow for forming topographic representation of input sequences. A schematic illustration of RecSOM is presented in figure 3.

The time evolution (31) becomes

$$r_i(t) = \exp(-\alpha \|\mathbf{x}(t) - \mathbf{w}_i^{r,x}\|^2) \cdot \exp(-\beta \|\mathbf{r}(t-1) - \mathbf{w}_i^{r,r}\|^2). \quad (36)$$

We denote the Gaussian kernel of inverse variance $\eta > 0$, acting on $\mathbb{R}^N$, by $G_\eta(\cdot,\cdot)$, i.e. for any $\mathbf{u}, \mathbf{v} \in \mathbb{R}^N$,

$$G_\eta(\mathbf{u}, \mathbf{v}) = e^{-\eta \|\mathbf{u} - \mathbf{v}\|^2}. \quad (37)$$

Finally, the state space evolution (2) can written for RecSOM as follows:

$$\mathbf{r}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{r}(t-1)) = (f_1(\mathbf{x}(t), \mathbf{r}(t-1)), ..., f_N(\mathbf{x}(t), \mathbf{r}(t-1))), \quad (38)$$

where

$$f_i(\mathbf{x}, \mathbf{r}) = G_\alpha(\mathbf{x}, \mathbf{w}_i^{r,x}) \cdot G_\beta(\mathbf{r}, \mathbf{w}_i^{r,r}), \quad i = 1, 2, ..., N. \quad (39)$$

The output response to an input time series $\{\mathbf{x}(t)\}$ is the time series $\{y(t)\}$ of best-matching unit indexes given by (32).

5.2 Markovian organization of receptive fields in contractive RecSOM

As usual in topographic maps, each unit on the map can be naturally represented by its *receptive field* (RF). Receptive field of a unit i is the common suffix of all sequences for which that unit becomes the best-matching unit.

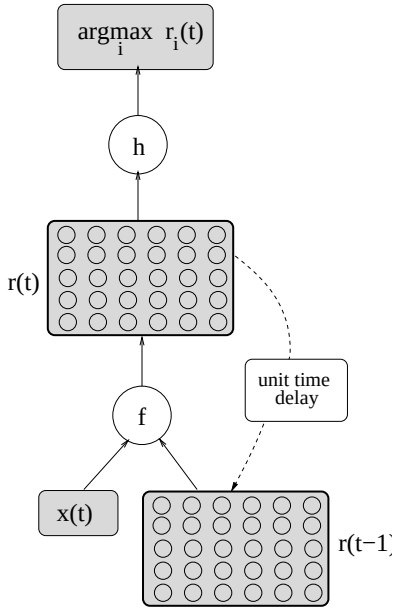

Fig. 3. Schematic illustration of RecSOM. Dimensions (neural units) of the state space $\mathcal{R}$ are topologically organized on a grid structure. The degree of "closeness" of units i and k on the grid is given by the neighborhood function $\nu(i,k)$. The output of RecSOM is the index of maximally activated unit (dimension of the state space).

It is common to visually represent trained topographic maps by printing out RFs on the map grid.

Let us discuss how *contractive* fixed-input maps $\mathbf{f}_s$ (3) shape the overall organization of RFs in the map. For each input symbol $s \in \mathcal{A}$, the autonomous dynamics

$$\mathbf{r}(t) = \mathbf{f}_s(\mathbf{r}(t-1)) \tag{40}$$

induces an output dynamics $y(t) = h(\mathbf{r}(t))$ of best-matching (winner) units on the map. Since for each input symbol $s \in \mathcal{A}$, the fixed-input dynamics (40) is a contraction, it will be dominated by a unique attractive fixed point $\mathbf{r}_s$. It follows that the output dynamics $\{y(t)\}$ on the map settles down in unit i_s, corresponding to the mode of $\mathbf{r}_s$. The unit i_s will be most responsive to input subsequences ending with long blocks of symbols s. Receptive fields of other units on the map will be organized with respect to the closeness of the units to the fixed-input winners i_s, $s \in \mathcal{A}$. When symbol s is seen at time t, the mode of the map activation profile $\mathbf{r}(t)$ starts drifting towards the unit i_s. The more consecutive symbols s we see, the more dominant the attractive fixed point of $\mathbf{f}_s$ becomes and the closer the winner position is to i_s.

It has indeed been reported that maps of sequential data obtained by RecSOM often seem to have a Markovian flavor: The units on the map become sensitive to recently observed symbols. Suffix-based RFs of the neurons are

topographically organized in connected regions according to last seen symbols. If two prefixes $s_{1:p}$ and $s_{1:q}$ of a long sequence $s_1...s_{p-2}s_{p-1}s_p...s_{q-2}s_{q-1}s_q...$ share a common suffix of length L, it follows from (10) that for any $\mathbf{r} \in \mathcal{R}$,

$$\|\mathbf{f}_{s_{1:p}}(\mathbf{r}) - \mathbf{f}_{s_{1:q}}(\mathbf{r})\| \leq \rho_{max}^L \cdot diam(\mathcal{R}).$$

For sufficiently large L, the two activations $\mathbf{r}^1 = \mathbf{f}_{s_{1:p}}(\mathbf{r})$ and $\mathbf{r}^2 = \mathbf{f}_{s_{1:q}}(\mathbf{r})$ will be close enough to have the same location of the mode,[8]

$$i_* = h(\mathbf{r}^1) = h(\mathbf{r}^2),$$

and the two subsequences $s_{1:p}$ and $s_{1:q}$ yield the same best matching unit i_* on the map, *irrespective of the position of the subsequences in the input stream*. All that matters is that the prefixes share a sufficiently long common suffix. We say that such an *organization of RFs on the map has a Markovian flavour*, because it is shaped solely by the suffix structure of the processed subsequences, and it does not depend on the temporal context in which they occur in the input stream[9].

RecSOM parameter β weighs the significance of importing information about possibly distant past into processing of sequential data. Intuitively, when β is sufficiently small, e.g. when information about the very recent inputs dominates processing in RecSOM, the resulting maps should have Markovian flavour. This intuition was given a more rigorous form in [16]. In particular, theoretical bounds on parameter β that guarantee contractiveness of the fixed input maps $\mathbf{f}_s$ were derived. As argued above, contractive fixed input mappings are likely to produce Markovian organizations of RFs on the RecSOM map.

Denote by $\mathbf{G}_\alpha(\mathbf{x})$ the collection of activations coming from the feed-forward part of RecSOM,

$$\mathbf{G}_\alpha(\mathbf{x}) = (G_\alpha(\mathbf{x}, \mathbf{w}_1^{r,x}), G_\alpha(\mathbf{x}, \mathbf{w}_2^{r,x}), ..., G_\alpha(\mathbf{x}, \mathbf{w}_N^{r,x})). \tag{41}$$

Then we have [16]:

Theorem 2. *Consider an input $\mathbf{x} \in \mathcal{X}$. If for some $\rho \in [0, 1)$,*

$$\beta \leq \rho^2 \ \frac{e}{2} \ \|\mathbf{G}_\alpha(\mathbf{x})\|^{-2}, \tag{42}$$

then the mapping $\mathbf{f}_{\mathbf{X}}(\mathbf{r}) = \mathbf{f}(\mathbf{x}, \mathbf{r})$ is a contraction with contraction coefficient ρ.

[8] or at least mode locations on neighboring grid points of the map

[9] Theoretically, there can be situations where **(1)** locations of the modes of $\mathbf{r}^1$ and $\mathbf{r}^2$ are distinct, but the distance between $\mathbf{r}^1$ and $\mathbf{r}^2$ is small; or where **(2)** the modes of $\mathbf{r}^1$ and $\mathbf{r}^2$ coincide, while their distance is quite large. This follows from discontinuity of the output map h. However, in our extensive experimental studies, we have registered only a negligible number of such cases.

Corollary 1. *Provided*

$$\beta < \frac{e}{2N}, \tag{43}$$

irrespective of the input symbol $s \in \mathcal{A}$, the fixed input map $\mathbf{f}_s$ of a RecSOM with N units will be a contraction.

The bound $e/(2N)$ may seem restrictive, but as argued in [15], the context influence has to be small to avoid instabilities in the model. Indeed, the RecSOM experiments of [15] used $N = 10 \times 10 = 100$ units and the map was trained with $\beta = 0.06$, which is only slightly higher than the bound $e/(2N) = 0.0136$. Obviously the bound $e/(2N)$ can be improved by considering other model parameters, as in Theorem 2.

These results complement Voegtlin's stability analysis of the parameter adaptation process during RecSOM training [12]: for $\beta < e/(2N)$, stability of weight updates with respect to small perturbations of the map activity $\mathbf{r}$ is ensured. Voegtlin also shows that if $\beta < e/(2N)$, small perturbations of the activities will decay (fixed input maps are locally contractive). The work in [16] extends this result to perturbations of arbitrary size. It follows that for each RecSOM model satisfying Voegtlin's stability bound on β, the fixed input dynamics for *any* input will be dominated by a unique attractive fixed point. This renders the map both Markovian quality and training stability.

5.3 Verifying Markovian organization of receptive fields in RecSOM

In [16] we posed the following questions:

- Is the architecture of RecSOM naturally biased towards Markovian representations of input streams? If so, under what conditions will Markovian organization of receptive fields (RF) occur? How natural are such conditions, i.e. can Markovian topographic maps be expected under widely-used architectures and (hyper)parameter settings in RecSOM?
- What can be gained by having a trainable recurrent part in RecSOM? In particular, how does RecSOM compare with a much simpler setting of *standard* SOM (no feedback connections) operating on a simple *non-trainable* FPM-based iterative function system (25)?

The experiments were performed on three data sets:

- A binary symbolic stream generated by a two-state first-order Markov chain used in the original RecSOM paper [12].
- The series of quantized activations of a laser in a chaotic regime from section 4.7.
- A corpus of written English - the novel "Brave New World" by Aldous Huxley. This is the second data set used to demonstrate RecSOM in [12].

Three methods of assessment of the topographic maps were used. The first two were suggested in [12].

- For each unit in the map its RF is calculated. The RFs are then visually presented on the map grid.
- As a measure of the amount of memory captured by the map, the overall quantizer depth of the map was calculated as

$$\mathrm{QD} = \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{p_i} \ell_{\mathrm{i}}, \tag{44}$$

  where p_i is the probability of the RF of neuron i and ℓ_i is its length.
- Quantifying topography preservation in recursive extensions of SOM is not as straightforward as in traditional SOM. To quantify the maps' topographic order we first calculated the length of the longest common suffix shared by RFs of that unit and its immediate topological neighbors. The topography preservation measure (TP) is the average of such shared RF suffix lengths over all units in the map.

For the Markov chain series, when we chose values of parameters α and β from the region of stable RecSOM behavior reported in [12], after the training we almost always got a trivial single-attractive-fixed-point behaviour of fixed input maps (40). As explained in section 5.2, this resulted in Markovian organization of RFs. Note that such a RF organization can be obtained "for free", i.e. without having a set of trainable feed-back connections, simply by constructing a standard SOM on a FPM-based iterative function system (25). We call such models IFS+SOM. This indeed turned out to be the case. Maps of RFs obtained by RecSOM and IFS+SOM of comparable sizes looked visually similar, with a clear Markovian organization. However, quantitative measures of quantizer depth (QD) and topography preservation (TP) revealed a significant advantage of RecSOM maps. The same applied to the chaotic laser data set. In most cases, the β parameter was above the upper bound of Theorem 2,

$$\Upsilon(s) = \frac{e}{2} \|\mathbf{G}_\alpha(\mathbf{c}(s))\|^{-2}, \tag{45}$$

guaranteeing contractive fixed input maps (40). Nevertheless, the organization of RFs was still Markovian. We performed experiments with β-values bellow $\epsilon/(2N)$ (see Corollary 1). In such cases, *irrespective of other training hyperparameters*, the fixed input maps (40) were always contractive, leading automatically to Markovian RF organizations.

When training RecSOM on the corpus of written English, for a wide variety of values of α and β that lead to stable map formation, non-Markovian organizations of RFs were observed. As a consequence, performance of IFS+SOM model, as measured by the QD and TP measures, was better, but it is important to realize that this does not necessarily imply that RecSOM maps were of inferior quality. The QD and TP are simply biased towards Markovian RF

organizations. But there may well be cases where non-Markovian topographic maps are more desirable, as discussed in the next section.

5.4 Beyond Markovian organization of receptive fields

Periodic (beyond period 1), or aperiodic attractive dynamics (40) yield potentially complicated non-Markovian map organizations with "broken topography" of RFs. Two sequences with the same suffix can be mapped into distinct positions on the map, separated by a region of different suffix structure. Unlike in contractive RecSOM or IFS+SOM models, such context-dependent RecSOM maps embody a potentially unbounded input memory structure: the current position of the winner neuron is determined by the whole series of processed inputs, and not only by a history of recently seen symbols. To fully appreciate the meaning of the RF structure, we must understand the driving mechanism behind such context-sensitive suffix representations.

In what follows we will try to suggest one possible mechanism of creating non-Markovian RF organizations. We noticed that often Markovian topography was broken for RFs ending with symbols of high frequency of occurrence. It seems natural for such RFs to separate on the map into distinct islands with a more refined structure. In other words, it seems natural to distinguish on the map between temporal contexts in which RFs with high frequency suffixes occur.

Now, the frequency of a symbol $s \in \mathcal{A}$, input-coded as $\mathbf{c}(s) \in \mathcal{X}$, in the input data stream will be reflected by the "feed-forward" part of the map consisting of weights $\mathbf{W}^{r,x}$. The more frequent the symbol is, the better coverage by the weights $\mathbf{W}^{r,x}$ it will get[10]. But good match with more feedforward weights $\mathbf{w}_i^{r,x}$ means higher kernel values of (see (37))

$$G_\alpha(\mathbf{c}(s), \mathbf{w}_i^{r,x}) = \exp\{-\alpha \|\mathbf{c}(s) - \mathbf{w}_i^{r,x}\|^2\},$$

and consequently higher L_2 norms of $\mathbf{G}_\alpha(\mathbf{c}(s))$ (41). It follows that the upper bound $\Upsilon(s)$ (45) on parameter β guaranteeing contractive fixed input map (40) for symbol s will be smaller. Hence, for more frequent symbols s, the interval $(0, \Upsilon(s))$ of β values for guaranteed Markovian organization of RFs ending with s shrinks and so a greater variety of stable maps can be formed with topography of RFs broken for suffixes containing s.

5.5 SOM for structured data

RecSOM constitutes a powerful model which has the capacity of at least pushdown automata as shown (for a simplified form) in [45]. It uses a very

[10] This is, strictly speaking, not necessarily true. It would be true in the standard SOM model. However, weight updates of $\mathbf{W}^{r,x}$ in RecSOM are influenced by the match in both the feedforward and recurrent parts of the model, represented by weights $\mathbf{W}^{r,x}$ and $\mathbf{W}^{r,x}$, respectively. Nevertheless, in all our experiments we found this correspondence between the frequency of an input symbol and its coverage by the feedforward part of RecSOM to hold.

complex context representation: the activation of all neurons of the map of the previous time step, usually a large number. This has the consequence that memory scales with N^2, N being the number of neurons, since the expected context is stored in each cell. SOM for structured data (SOMSD) and merge SOM (MSOM) constitute alternatives which try to compress the relevant information of the context such that the memory load is reduced.

SOMSD has been proposed in [14] for unsupervised processing of tree structured data. Here we restrict to sequences, i.e. we neglect branching. SOMSD compresses the temporal context by the *location of the winner* in the neural map. The map is given by units $i \in \{1, \ldots, N\}$ which are located on a lattice structure embedded in a real-vector space $\mathcal{R}^I \subset \mathbb{R}^d$, typically a two-dimensional rectangular lattice in $\mathbb{R}^2$. The location of neuron i in this lattice is referred to as I_i.

Each neuron has two vectors associated with it:

- $\mathbf{w}_i^{r,x} \in \mathcal{X}$ – linked with an N_I-dimensional input $\mathbf{x}(t)$ feeding the network at time t as in RecSOM
- $\mathbf{w}_i^{r,r} \in \mathcal{R}^I$ – linked with the context

$$\mathbf{r}^I(t-1) = I_{i_{t-1}} \text{ with } i_{t-1} = \underset{i}{\operatorname{argmin}}\{r_i(t-1) \mid i = 1, \ldots, N\}\,,$$

  i.e. it is linked to the winner location depending on map activations $r_i(t-1)$ from the previous time step.

The activation of a unit i at time t is computed as

$$r_i(t) = \alpha \cdot \|\mathbf{x}(t) - \mathbf{w}_i^{r,x}\| + \beta \cdot \|\mathbf{r}^I(t-1) - \mathbf{w}_i^{r,r}\|. \tag{46}$$

where, as beforehand,

$$\mathbf{r}^I(t-1) = I_{i_{t-1}} \text{ with } i_{t-1} = \underset{i}{\operatorname{argmin}}\{r_i(t-1) \mid i = 1, \ldots, N\}.$$

The readout extracts the winner

$$y(t) = h(\mathbf{r}(t)) = \underset{i \in \{1,\ldots,N\}}{\operatorname{argmin}} r_i(t)\,.$$

Note that this recursive dynamics is very similar to the dynamics of RecSOM as given in (31). The information stored in the context is compressed by choosing the winner location as internal representation.

Since $\mathcal{R}^I$ is a real-vector space, SOMSD can be trained in the standard Hebb style for both parts of the weights, moving $\mathbf{w}_i^{r,x}$ closer to $\mathbf{x}(t)$ and $\mathbf{w}_i^{r,r}$ closer to $\mathbf{r}^I(t-1)$ after the representation of a stimulus. The principled capacity of SOMSD can be characterized exactly. It has been shown in [15] that SOMSD can simulate finite automata in the following sense: assume $\mathcal{A}$ is a finite alphabet. Assume a finite automaton accepts the language $\mathcal{L} \subset \mathcal{A}^*$. Then there exists a constant delay τ, an encoding $C : \mathcal{A} \to \mathcal{X}^\tau$, and a SOMSD

network such that $s \in \mathcal{L}$ if and only if the sequence $C(s_1) \dots C(s_t)$ is mapped to neuron 1. Because of the finite context representation within the lattice in (46) (capacity at most the capacity of finite state automata), the equivalence of SOMSD for sequences and finite state automata results.

Obviously, unlike RecSOM, the dynamics of SOMSD is discontinuous since it involves the determination of the winner. Therefore, the arguments of section 5.2 which show a Markovian bias as provided for RecSOM do not apply to SOMSD. For SOMSD, an alternative argumentation is possible, assuming small β and two further properties which we refer to as *sufficient granularity* and as *distance preservation*. More precisely:

1. SOMSD has *sufficient granularity* given $\epsilon_1 > 0$ if for every $a \in \mathcal{A}$ and $\mathbf{r}^I \in \mathcal{R}^I$ a neuron i can be found such that $\|\mathbf{c}(a) - \mathbf{w}_i^{r,x}\| \leq \epsilon_1$ and $\|\mathbf{r}^I - \mathbf{w}_i^{r,r}\| \leq \epsilon_1$.
2. SOMSD is *distance preserving* given $\epsilon_2 > 0$ if for every two neurons i and j the following inequality holds $|\|I_i - I_j\| - \alpha \cdot \|\mathbf{w}_i^{r,x} - \mathbf{w}_j^{r,x}\| - \beta \cdot \|\mathbf{w}_i^{r,r} - \mathbf{w}_j^{r,r}\|| \leq \epsilon_2$. This inequality relates the distances of the locations of neurons on the lattice to the distances of their content.

Assume a SOMSD is given with initial context $\mathbf{r}_0$. We denote the output response to the empty sequence ϵ by $i_\epsilon = h(\mathbf{r}_0)$, the winner unit for an input sequence u by $i_u := h \circ \mathbf{f}_u(\mathbf{r}_0)$ and the corresponding winner location on the map grid by I_u. We can define an explicit Markovian metric on sequences in the following way: $d_{\mathcal{A}^*} : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}$, $d_{\mathcal{A}^*}(\epsilon, \epsilon) = 0$, $d_{\mathcal{A}^*}(u, \epsilon) = d_{\mathcal{A}^*}(\epsilon, u) = \|I_\epsilon - I_u\|$, and for two sequences $u = u_1 \dots u_{p-1} u_p$, $v = v_1 \dots v_{q-1} v_q$ over $\mathcal{A}$,

$$d_{\mathcal{A}^*}(u, v) = \alpha \cdot \|\mathbf{c}(u_p) - \mathbf{c}(v_q)\| + \beta \cdot d_{\mathcal{A}^*}(u_1 \dots u_{p-1}, v_1 \dots v_{q-1}). \qquad (47)$$

Assume $\beta < 1$, then, the output of SOMSD is Markovian in the sense that the metric as defined in (47) approximates the metric induced by the winner location of sequences. More precisely, the following inequality is valid:

$$| \; d_{\mathcal{A}^*}(u, v) - \|I_u - I_v\| \; | \leq (\epsilon_2 + 4 \cdot (\alpha + \beta)\epsilon_1)/(1 - \beta).$$

This equation is immediate if u or v are empty. For nonempty sequences $u = u_{1:p-1} u_p$ and $v = v_{1:q-1} v_q$, we find

$$\begin{aligned} | \; d_{\mathcal{A}^*}(u, v) - \|I_u - I_v\| \; | &= | \; \alpha \cdot \|\mathbf{c}(u_p) - \mathbf{c}(v_q)\| \\ &\quad + \beta \cdot d_{\mathcal{A}^*}(u_{1:p-1}, v_{1:q-1}) - \|I_u - I_v\| \; | \\ &\leq \epsilon_2 + \alpha \cdot | \; \|\mathbf{w}_{i_u}^{r,x} - \mathbf{w}_{i_v}^{r,x}\| - \|\mathbf{c}(u_p) - \mathbf{c}(v_q)\| \; | \\ &\quad + \beta \cdot | \; \|\mathbf{w}_{i_u}^{r,r} - \mathbf{w}_{i_v}^{r,r}\| - d_{\mathcal{A}^*}(u_{1:p-1}, v_{1:q-1})| \\ &= (*) \end{aligned}$$

Therefore, $\alpha \cdot \|\mathbf{w}_{i_u}^{r,x} - \mathbf{c}(u_p)\| + \beta \|\mathbf{w}_{i_u}^{r,r} - I_{u_{1:p-1}}\|$ is minimum. We have assumed sufficient granularity, i.e. there exists a neuron with weights at most ϵ_1 away from $\mathbf{c}(u_p)$ and $I_{u_{1:p-1}}$. Therefore,

$$\|\mathbf{w}_{i_u}^{r,x} - \mathbf{c}(u_p)\| \leq (\alpha + \beta)\epsilon_1/\alpha$$

and

$$\|\mathbf{w}_{i_u}^{r,r} - I_{u_{1:p-1}}\| \leq (\alpha + \beta)\epsilon_1/\beta.$$

Using the triangle inequality, we obtain the estimation

$$\begin{aligned}(*) &\leq \epsilon_2 + 4 \cdot (\alpha + \beta)\epsilon_1 + \beta \cdot \mid d_{\mathcal{A}^*}(u_{1:p-1}, v_{1:q-1}) - \|I_{u_{1:p-1}} - I_{v_{1:q-1}}\| \mid \\ &\leq \cdots \leq (\epsilon_2 + 4 \cdot (\alpha + \beta)\epsilon_1)(1 + \beta + \beta^2 + \ldots) \\ &\leq (\epsilon_2 + 4 \cdot (\alpha + \beta)\epsilon_1)/(1 - \beta),\end{aligned}$$

since $\beta < 1$.

5.6 Merge SOM

SOMSD uses a compact context representation. However, it has the drawback that it relies on a fixed lattice structure. Merge SOM (MSOM) uses a different compression scheme which does not rely on the lattice, rather, it uses the *content* of the winner. Here context representations are located in the same space as inputs $\mathcal{X}$. A map is given by neurons which are associated with two vectors

- $\mathbf{w}_i^{r,x} \in \mathcal{X}$ – linked with an N_I-dimensional input $\mathbf{x}(t)$ feeding the network at time t as in RecSOM
- $\mathbf{w}_i^{r,r} \in \mathcal{X}$ – linked with the context

  $$\mathbf{r}^M(t-1) = \gamma \cdot \mathbf{w}_{i_{t-1}}^{r,x} + (1-\gamma) \cdot \mathbf{w}_{i_{t-1}}^{r,r}$$

  where $i_{t-1} = \text{argmin}_i\{r_i(t-1) \mid i = 1, \ldots, N\}$ is the index of the best matching unit at time $t-1$. The feed-forward and recurrent weight vectors of the winner from the previous time step are merged using a fixed parameter $\gamma > 0$.

The activation of a unit i at time t is computed as

$$r_i(t) = \alpha \cdot \|\mathbf{x}(t) - \mathbf{w}_i^{r,x}\| + \beta \cdot \|\mathbf{r}^M(t-1) - \mathbf{w}_i^{r,r}\|, \tag{48}$$

where $\mathbf{r}^M(t-1) = \gamma \cdot \mathbf{w}_{i_{t-1}}^{r,x} + (1-\gamma) \cdot \mathbf{w}_{i_{t-1}}^{r,r}$ is the merged winner content of the previous time step. The readout is identical to SOMSD.

As for SOMSD, training in the standard Hebb style is possible, moving $\mathbf{w}_i^{r,x}$ towards $\mathbf{x}(t)$ and $\mathbf{w}_i^{r,r}$ towards the context $\mathbf{r}^M(t-1)$, given a history of stimuli up to time t. It has been shown in [13] that MSOM can simulate every finite automaton whereby we use the notation of simulation as defined for SOMSD. Since for every finite map only a finite number of different contexts can occur, the capacity of MSOM coincides with finite automata.

As for SOMSD, the transition function of MSOM is discontinuous since it incorporates the computation of the winner neuron. However, one can show that standard Hebbian learning biases MSOM towards Markovian models

in the sense that optimum context weights which have a strong Markovian flavour are obtained by Hebbian learning for MSOM as stable fixed points of the learning dynamics. Thereby, the limit of Hebbian learning for vanishing neighborhood size is considered which yields the update rules

$$\Delta \mathbf{w}_{i_t}^{r,x} = \eta \cdot (\mathbf{x}(t) - \mathbf{w}_{i_t}^{r,x}) \tag{49}$$

$$\Delta \mathbf{w}_{i_t}^{r,r} = \eta \cdot (\mathbf{r}^M(t-1) - \mathbf{w}_{i_t}^{r,r}) \tag{50}$$

after time step t, whereby $\mathbf{x}(t)$ constitutes the input at time step t, $\mathbf{r}^M(t-1) = \gamma \cdot \mathbf{w}_{i_{t-1}}^{r,x} + (1-\gamma) \cdot \mathbf{w}_{i_{t-1}}^{r,r}$ the context at time step t, and i_t the winner at time step t; $\eta > 0$ is the learning rate.

Assume an input series $\{\mathbf{x}(t)\}$ is presented to MSOM and assume the initial context $\mathbf{r}_0$ is the null vector. Then, if a sufficient number of neurons is available such that there exists a disjoint winner for every input $\mathbf{x}(t)$, the following weights constitute a stable fixed point of Hebbian learning as described in eqs. (49-50):

$$\mathbf{w}_{i_t}^{r,x} = \mathbf{x}(t) \tag{51}$$

$$\mathbf{w}_{i_t}^{r,r} = \sum_{j=1}^{t-1} \gamma \cdot (1-\gamma)^j \mathbf{x}(t-j). \tag{52}$$

The proof of this fact can be found in [15, 13]. Note that, depending on the size of γ, the context representation yields sequence codes which are very similar to the recursive states provided by fractal prediction machines (FPM) of section 4.3. For $\gamma < 0.5$ and unary inputs $\mathbf{x}(t)$ (one-of-A encoding), (52) yields a fractal whereby the global position in the state space is determined by the most recent entries of the input series. Hence, a Markovian bias is obtained by Hebbian learning.

This can clearly be observed in experiments, compare figure 4[11] which describes a result of an experiment conducted in [13]: a MSOM with 644 neurons is trained on DNA sequences. The data consists of windows around potential splice sites from C.elegans as described in [46]. The symbols TCGA are encoded as the edges of a tetrahedron in $\mathbb{R}^3$. The windows are concatenated and presented to a MSOM using merge parameter $\gamma = 0.5$. Thereby, the neighborhood function used for training is not a standard SOM lattice, but training is data driven by the ranks as described for neural gas networks [47]. Figure 4 depicts a projection of the learnt contexts. Clearly, a fractal structure which embeds sequences in the whole context space can be observed, substantiating the theoretical finding by experimental evidence.

Note that, unlike FPM which use a fixed fractal encoding, the encoding of MSOM emerges from training. Therefore, it takes the internal probability distribution of symbols into account, automatically assigning a larger space to those inputs which are presented more frequently to the map.

[11] We would like to thank Marc Strickert for providing the picture.

Fig. 4. Projection of the three dimensional fractal contexts which are learnt when training a MSOM on DNA-strings whereby the symbols ACTG are embedded in three dimensions.

6 Applications in bioinformatics

In this section we briefly describe applications of the architectural bias phenomenon in the supervised learning scenario. Natural Markovian organization of biological sequences within the RNN state space may be exploited when solving some sequence-related problems of bioinformatics.

6.1 Genomic data

Enormous amounts of genomic sequence data has been produced in recent years. Unfortunately, the emergence of experimental data for the *functional parts* of such large-scale genomic repositories is lagging behind. For example, several hundred thousands of proteins are known, but only about 35,000 have had their structure determined. One third of all proteins are believed to be membrane associated, but less than a thousand have been experimentally well-characterized to be so.

As commonly assumed, genomic sequence data specifies the structure and function of its products (proteins). Efforts are underway in the field of bioinformatics to link genomic data to experimental observations with the goal of *predicting* such characteristics for the yet unexplored data [48]. DNA, RNA and proteins are all chain molecules, made up of distinct monomers, namely *nucleotides* (DNA and RNA) and *amino acids* (protein). We choose to cast such chemical compounds as symbol sequences. There are four different nucleotides and twenty different amino acids, here referred to as alphabets $\mathcal{A}_4$ and $\mathcal{A}_{20}$, respectively.

A functional region of DNA typically includes thousands of nucleotides. The average number of amino acids in a protein is around the 300 mark but sequence lengths vary greatly. The combinatorial nature of genomic sequence data poses a challenge for machine learning approaches linking sequence data with observations. Even when thousands of samples are available, the sequence space is sparsely populated. The *architectural bias* of predictive models may thus influence the result greatly. This section demonstrates that a Markovian state-space organization–as imposed by the recursive model in this paper– discerns patterns with a biological flavor [49].

In bioinformatics *motifs* are perceived as recurring patterns in sequence data (DNA, RNA and proteins) with an observed or conjectured biological significance (the character varies widely). Problems like promoter and DNA-binding site discovery [50, 51], identification of alternatively spliced exons [52] and glycosylation sites are successfully approached using the concept of sequence motifs. Motifs are usually quite short and to be useful may require a more permissive description than a list of consecutive sequence symbols. Hence, we augment the motif alphabet $\mathcal{A}$ to include the so-called wildcard symbol $'*'$: $\tilde{\mathcal{A}} = \mathcal{A} \cup \{*\}$. Let $\mathcal{M} = m_1 m_2 ... m_{|\mathcal{M}|}$ be a motif where $m_i \in \tilde{\mathcal{A}}$. As outlined below the wildcard symbol has special meaning in the motif sequence matching process.

6.2 Recurrent architectures promote motif discovery

We are specifically interested if the Markovian state space organization promotes the discovery of a motif within a sequence. Following [49], *position-dependent* motifs are patterns of symbols relative to a fixed sequence-position. A motif *match* is defined as

$$\text{match}(s_{1:n}, \mathcal{M}) = \begin{cases} \text{T}, & \text{if } \forall i \in \{1, ..., |\mathcal{M}|\} \ [m_i = * \vee m_i = s_{n-|\mathcal{M}|+i}] \\ \text{F} & \text{otherwise} \end{cases}$$

where$n \geq |\mathcal{M}|$. Notice that the motif must occur at the end of the sequence.

To inspect how well sequences with and without a given motif $\mathcal{M}$ (T and F, respectively) are separated in the state-space, Bodén and Hawkins determined a motif match entropy for each of the Voronoi compartments $V_1, ..., V_M$, introduced in the context of NPM construction in section 4.1. The compartments were formed by vector quantizing the *final* activations produced by a neural-based architecture on a set of sequences (see Section 4). With each codebook vector $\mathbf{b}_i \in \mathcal{R}$, $i = 1, 2, ..., M$, we associate the probability $P(\text{T}|i)$ of observing a state $\mathbf{r} \in V_i$ corresponding to a sequence motif match. This probability is estimated by tracking the matching and non-matching sequences while counting the number of times V_i is visited.

The entropy for each codebook is defined as

$$H_i = -P(\text{T}|i) \log_2 P(\text{T}|i) - (1 - P(\text{T}|i)) \log_2 (1 - P(\text{T}|i)) \tag{53}$$

and describes the homogeneity within the Voronoi compartment by considering the proportion of motif *positives* and *negatives*. An entropy of 0 indicates perfect homogeneity (contained states are exclusively for matching or non-matching sequences). An entropy of 1 indicates random organization (a matching state is observed by chance). The average codebook entropy, $H = M^{-1} \sum_{i=1}^{M} H_i$, is used to characterize the overall state-space organization.

In our experiments, the weights $\mathbf{W}^{r,x}$ and $\mathbf{W}^{r,r}$ were randomly drawn from a uniform distribution $[-0.5, +0.5]$. This, together with activation function g, ensured contractiveness of the state transition map $\mathbf{f}$ and hence (as explained in section 3.1) Markovian organization of the RNN state space.

In an attempt to provide a simple baseline alternative to our recursive neural-based models, a feedforward neural network was devised accepting the whole sequence at its input without any recursion. Given the maximum input sequence length ℓ_{max}, the input layer has $N_I \cdot \ell_{max}$ units and the input code of a sequence $s_{1:n} = s_1 ... s_n$ takes the form

$$\tilde{\mathbf{c}}(s_{1:n}) = \mathbf{c}(s_1) \bullet ... \bullet \mathbf{c}(s_n) \bullet \{0\}^{N_I \cdot (\ell_{max} - n)},$$

where $\bullet$ denotes vector concatenation. Hence, $\tilde{\mathbf{c}}(s_{1:n})$ is a concatenation of input codes for symbols in $s_{1:n}$, followed by $N_I \cdot (\ell_{max} - n)$ zeros. The hidden layer response of the feedforward network to $s_{1:n}$ is then

$$\tilde{\mathbf{f}}_{s_{1:n}} = \tilde{\mathbf{f}}(\tilde{\mathbf{c}}(s_{1:n})) = \mathbf{g}(\tilde{\mathbf{W}}^{r,x} \tilde{\mathbf{c}}(s_{1:n}) + \mathbf{t}_f), \tag{54}$$

where the input-to-hidden layer weights (again randomly drawn from $[-0.5, +0.5]$) are collected in the $N \times (N_I \cdot \ell_{max})$ matrix $\tilde{\mathbf{W}}^{r,x}$.

In each trial, a random motif $\mathcal{M}$ was first selected and then 500 random sequences (250 with a match, 250 without) were generated. Ten differently initialized networks (both RNN and the feed-forward baseline) were then subjected to the sequences and the average entropy H was determined. By varying sequence and motif lengths, the proportion of wildcards in the motifs, state space dimensionality and number of codebook vectors extracted from sequence states, Bodén and Hawkins established that recurrent architectures were clearly superior in discerning random but pre-specified motifs in the state space (see Figure 5). However, there was no discernible difference between the two types of networks (RNN and feedforward) for shift-invariant motifs (an alternative form which allows matches to appear at any point in the sequence). To leverage the bias, a recurrent network should thus continually monitor its state space.

These synthetic trials (see [49]) provide evidence that a recurrent network naturally organizes its state space at each step to enable classification of extended and flexible sequential patterns that led up to the current time step. The results have been shown to hold also for higher-order recurrent architectures [53].

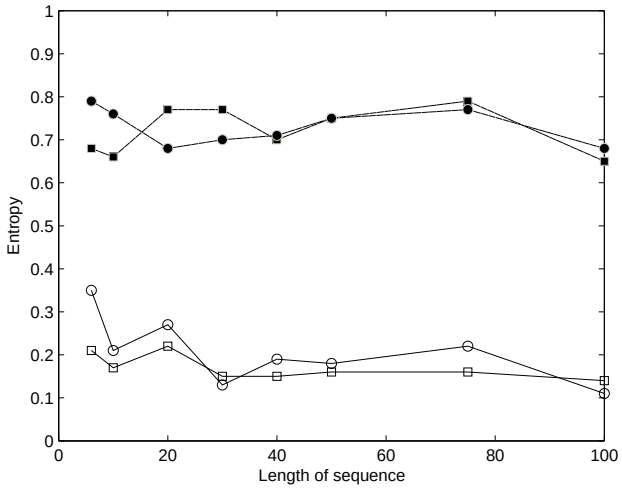

Fig. 5. The average entropy H (y-axis) for recurrent networks (unfilled markers) and baseline feedforward networks (filled markers) when $\mathcal{A}_4$ (squares) and $\mathcal{A}_{20}$ (circles) were used to generate sequences of varying length (x-axis) and position-dependent motifs (with length equal to half the sequence length and populated with 33% wild-cards). The number of codebooks, M, was 10.

6.3 Discovering discriminative patterns in signal peptides

A long-standing problem of great biological significance is the detection of so-called signal peptides. Signal peptides are reasonably short protein segments that fulfil a transportation role in the cell. Basically, if a protein has a signal peptide it translocates to the exoplasmic space. The signal peptide is cleaved off in the process.

Viewed as symbol strings, signal peptides are rather diverse. However, there are some universal characteristics. For example, they always appear at the N-terminus of the protein, they are 15-30 residues long, consist of a stretch of seven to 15 hydrophobic (lipid-favouring) amino acids and a few well-conserved, small and neutral monomers close to the cleavage site. In Figure 6 nearly 400 mammalian signal peptides are aligned at their respective cleavage site to illustrate a subtle pattern of conservation.

In an attempt to simplify the sequence classification problem, each position of the sequence can first be tagged as belonging (T) or not belonging (F) to a signal peptide [54]. SignalP is a feedforward network that is designed to look at all positions of a sequence (within a carefully sized context window) and trained to classify each into the two groups (T and F) [54].

The Markovian character of the recursive model's state space suggests that replacing the feedforward network with a recurrent network may provide a better starting point for a training algorithm. However, since biological

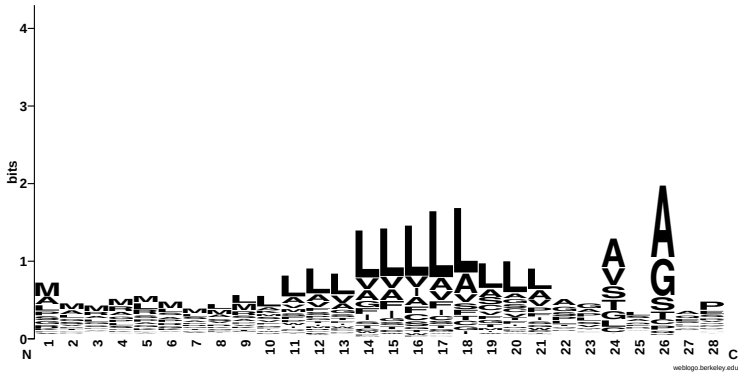

Fig. 6. Mammalian signal peptide sequences aligned at their cleavage site (between positions 26 and 27). Each amino acid is represented by its one-letter code. The relative frequency of each amino acid in each aligned position is indicated by the height of each letter (as measured in bits). The graph is produced using WebLogo.

sequences may exhibit dependencies in both directions (upstream as well as downstream) the simple recurrent network may be insufficient.

Hawkins and Bodén [53, 55] replaced the feedforward network with a *bi-directional* recurrent network [56]. The bi-directional variant is simply two conventional (uni-directional) recurrent networks coupled to merge two states each produced from traversing the sequence from one direction (either upstream or downstream relative the point of interest). In Hawkins and Bodén's study, the networks' ability was only examined after training.

After training, the average test error (as measured over known signal peptides and known negatives) dropped by approximately 25% compared to feedforward networks. The position-specific errors for both classes of architectures are presented in Figure 7. Several possible setups of the bi-directional recurrent network are possible, but the largest performance increase was achieved when the number of symbols presented at each step was 10 (for both the upstream and downstream networks).

Similar to signal peptides, mitochondrial targeting peptides and chloroplast transit peptides traffic proteins. In their case proteins are translocated to mitochondrion and chloroplast, respectively. Bi-directional recurrent networks applied to such sequence data show a similar pattern of improvement compared to feedforward architectures [53].

The combinatorial space of possible sequences and the comparatively small number of known samples make the application of machine learning to bioinformatics challenging. With significant data variance, it is essential to design models that guide the learning algorithm to meaningful portions of the model parameter space. By presenting exactly the same training data sets to two classes of models, we note that the architectural bias of recurrent networks seems to enhance the prospects of achieving a valid generalization. As noted

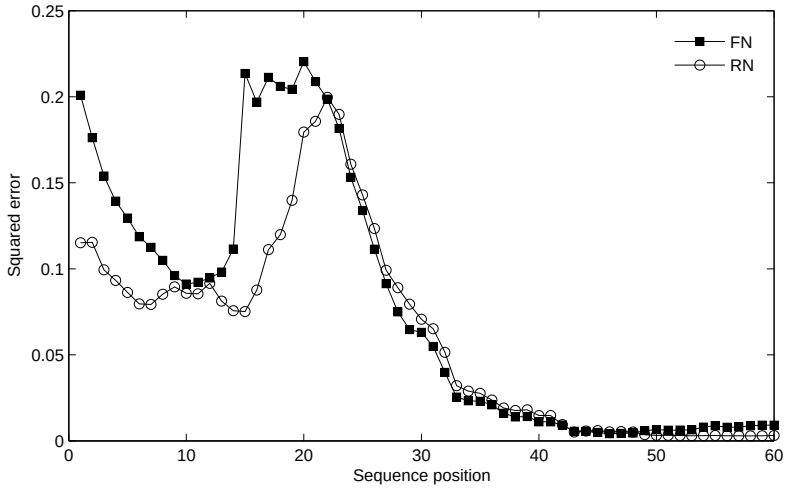

Fig. 7. The average error of recurrent and feedforward neural networks over the first 60 residues in a sequence test set discriminating between residues that belong to a signal peptide from those that do not. The signal peptide always appear at the N-terminus of a sequence (position 1 and onwards, average length is 23).

in [53] the locations of subtle sequential patterns evident in the data sets (cf. Figure 6) correspond well with those regions at which the recurrent networks excel. The observation lends empirical support to the idea that recursive models are well suited to this broad class of bioinformatics problems.

7 Conclusion

We have studied architectural bias of dynamic neural network architectures towards suffix-based Markovian input sequence representations. In the supervised learning scenario, the cluster structure that emerges in the recurrent layer *prior to training* is, due to the contractive nature of the state transition map, organized in a Markovian manner. In other words, the clusters emulate Markov prediction contexts in that histories of symbols are grouped according to the number of symbols they share in their suffix. Consequently, from such recurrent activation clusters it is possible to extract predictive models that correspond to variable memory length Markov models (VLMM). To appreciate how much information has been induced during the training, the dynamic neural network performance should always be compared with that of VLMM baseline models.

Apart from trained models (possibly with small weights, i.e. Markovian bias) various mechanisms which only rely on the dynamics of fixed or randomly initialized recurrence have recently been proposed, including fractal

prediction machines [26], echo state machines [27, 28], and liquid state machines [27, 28]. As discussed in the paper, these models almost surely allow an approximation of Markovian models even with random initialization provided a large dimensionality of the reservoir.

Interestingly, the restriction to Markovian models has benefits with respect to the generalization ability of the models. Whereas generalization bounds of general recurrent models necessarily rely on the input distribution due to the possibly complex input information, the generalization ability of Markovian models can be limited in terms of the relevant input length or, alternatively, the Lipschitz parameter of the models. This yields bounds which are independent of the input distribution. Smaller contraction coefficients imply better bounds. These bounds also hold for arbitrary real-valued inputs and for posterior bounds on the contraction coefficient achieved during training. The good generalization ability of RNNs with Markovian bias is particularly suited for typical problems in bioinformatics where dimensionality is high compared to the number of available training data. A couple of experiments which demonstrate this effect have been included in this article.

Unlike supervised RNNs, a variety of fundamentally different unsupervised recurrent networks has recently been proposed. Apart from comparably old biologically plausible leaky integrators such as the temporal Kohonen map or recurrent networks, which obviously rely on a Markovian representation of sequences, models which use a more elaborate temporal context such as the map activation, the winner location, or winner content have been proposed. These models have, in principle, the larger capacity of finite state automata or pushdown automata, respectively, depending on the context model. Interestingly, this capacity reduces to Markovian models under certain realistic conditions discussed in this paper. These include a small mixing parameter for recursive networks, a sufficient granularity and distance preservation for SOMSD, and Hebbian learning for MSOM. The field of appropriate training of complex unsupervised recursive models is, so far, widely unexplored. Therefore, the identification of biases of the architecture and training algorithm towards the Markovian property is a very interesting result which allows to gain more insight into the process of unsupervised sequence learning.

References

1. Christiansen, M., Chater, N.: Toward a connectionist model of recursion in human linguistic performance. Cognitive Science **23** (1999) 417–437
2. Kolen, J.: Recurrent networks: state machines or iterated function systems? In Mozer, M., Smolensky, P., Touretzky, D., Elman, J., Weigend, A., eds.: Proceedings of the 1993 Connectionist Models Summer School. Erlbaum Associates, Hillsdale, NJ (1994) 203–210
3. Kolen, J.: The origin of clusters in recurrent neural network state space. In: Proceedings from the Sixteenth Annual Conference of the Cognitive Science Society, Hillsdale, NJ: Lawrence Erlbaum Associates (1994) 508–513

4. Tiňo, P., Čerňanský, M., Beňušková, L.: Markovian architectural bias of recurrent neural networks. IEEE Transactions on Neural Networks **15** (2004) 6–15
5. Hammer, B., Tino, P.: Recurrent neural networks with small weights implement definite memory machines. Neural Computation **15** (2003) 1897–1929
6. Ron, D., Singer, Y., Tishby, N.: The power of amnesia. Machine Learning **25** (1996)
7. Tiňo, P., Hammer, B.: Architectural bias in recurrent neural networks: Fractal analysis. Neural Computation **15** (2004) 1931–1957
8. Kohonen, T.: The self-organizing map. Proceedings of the IEEE **78** (1990) 1464–1479
9. Chappell, G., Taylor, J.: The temporal kohonen map. Neural Networks **6** (1993) 441–445
10. Koskela, T., znd J. Heikkonen, M.V., Kaski, K.: Recurrent SOM with local linear models in time series prediction. In: 6th European Symposium on Artificial Neural Networks. (1998) 167–172
11. Horio, K., Yamakawa, T.: Feedback self-organizing map and its application to spatio-temporal pattern classification. International Journal of Computational Intelligence and Applications **1** (2001) 1–18
12. Voegtlin, T.: Recursive self-organizing maps. Neural Networks **15** (2002) 979–992
13. Strickert, M., Hammer, B.: Merge SOM for temporal data. Neurocomputing **64** (2005) 39–72
14. Hagenbuchner, M., Sperduti, A., Tsoi, A.: Self-organizing map for adaptive processing of structured data. IEEE Transactions on Neural Networks **14** (2003) 491–505
15. Hammer, B., Micheli, A., Strickert, M., Sperduti, A.: A general framework for unsupervised processing of structured data. Neurocomputing **57** (2004) 3–35
16. Tiňo, P., Farkaš, I., van Mourik, J.: Dynamics and topographic organization of recursive self-organizing maps. Neural Computation **18** (2006) 2529–2567
17. Falconer, K.: Fractal Geometry: Mathematical Foundations and Applications. John Wiley and Sons, New York (1990)
18. Barnsley, M.: Fractals everywhere. Academic Press, New York (1988)
19. Ron, D., Singer, Y., Tishby, N.: The power of amnesia. In: Advances in Neural Information Processing Systems 6, Morgan Kaufmann (1994) 176–183
20. Guyon, I., Pereira, F.: Design of a linguistic postprocessor using variable memory length markov models. In: International Conference on Document Analysis and Recognition, Monreal, Canada, IEEE Computer Society Press (1995) 454–457
21. Weinberger, M., Rissanen, J., Feder, M.: A universal finite memory source. IEEE Transactions on Information Theory **41** (1995) 643–652
22. Buhlmann, P., Wyner, A.: Variable length markov chains. Annals of Statistics **27** (1999) 480–513
23. Rissanen, J.: A universal data compression system. IEEE Trans. Inform. Theory **29** (1983) 656–664
24. Giles, C., Omlin, C.: Insertion and refinement of production rules in recurrent neural networks. Connection Science **5** (1993)
25. Doya, K.: Bifurcations in the learning of recurrent neural networks. In: Proc. of 1992 IEEE Int. Symposium on Circuits and Systems. (1992) 2777–2780
26. Tiňo, P., Dorffner, G.: Predicting the future of discrete sequences from fractal representations of the past. Machine Learning **45** (2001) 187–218

27. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology (2001)
28. Jaeger, H., Haas, H.: Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science **304** (2004) 78–80
29. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation **14** (2002) 2531–2560
30. Maass, W., Legenstein, R.A., Bertschinger, N.: Methods for estimating the computational power and generalization capability of neural microcircuits. In: Advances in Neural Information Processing Systems. Volume 17., MIT Press (2005) 865—872
31. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2** (1989) 359–366
32. Hammer, B.: Generalization ability of folding networks. IEEE Transactions on Knowledge and Data Engineering **13** (2001) 196–206
33. Koiran, P., Sontag, E.: Vapnik-chervonenkis dimension of recurrent neural networks. In: European Conference on Computational Learning Theory. (1997) 223–237
34. Vapnik, V.: Statistical Learning Theory. Wiley-Interscience (1998)
35. Shawe-Taylor, J., Bartlett, P.L.: Structural risk minimization over data-dependent hierarchies. IEEE Trans. on Information Theory **44** (1998) 1926–1940
36. Williams, R., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural Computation **1** (1989) 270–280
37. Williams, R.: Training recurrent networks using the extended kalman filter. In: Proc. 1992 Int. Joint Conf. Neural Networks. Volume 4. (1992) 241–246
38. Patel, G., Becker, S., Racine, R.: 2d image modelling as a time-series prediction problem. In Haykin, S., ed.: Kalman filtering applied to neural networks. Wiley (2001)
39. Manolios, P., Fanelli, R.: First order recurrent neural networks and deterministic finite state automata. Neural Computation **6** (1994) 1155–1173
40. Tiňo, P., Hammer, B.: Architectural bias in recurrent neural networks – fractal analysis. In Dorronsoro, J., ed.: Artificial Neural Networks - ICANN 2002. Lecture Notes in Computer Science, Springer-Verlag (2002) 1359–1364
41. de A. Barreto, G., Araújo, A., Kremer, S.: A taxanomy of spatiotemporal connectionist networks revisited: The unsupervised case. Neural Computation **15** (2003) 1255–1320
42. Kilian, J., Siegelmann, H.T.: On the power of sigmoid neural networks. Information and Computation **128** (1996) 48–56
43. Siegelmann, H.T., Sontag, E.D.: Analog computation via neural networks. Theoretical Computer Science **131** (1994) 331–360
44. Hammer, B., Micheli, A., Sperduti, A., Strickert, M.: Recursive self-organizing network models. Neural Networks **17** (2004) 1061–1086
45. Hammer, B., Neubauer, N.: On the capacity of unsupervised recursive neural networks for symbol processing. In d'Avila Garcez, A., Hitzler, P., Tamburrini, G., eds.: Workshop proceedings of NeSy'06. (2006)
46. Sonnenburg, S.: New methods for splice site recognition. Master's thesis, Diplom thesis, Institut für Informatik, Humboldt-Universität Berlin (2002)

47. Martinetz, T., Berkovich, S., Schulten, K.: 'neural-gas' networks for vector quantization and its application to time-series prediction. IEEE Transactions on Neural Networks **4** (1993) 558–569
48. Baldi, P., Brunak, S.: Bioinformatics: The machine learning approach. MIT Press, Cambridge, Mass (2001)
49. Bodén, M., Hawkins, J.: Improved access to sequential motifs: A note on the architectural bias of recurrent networks. IEEE Transactions on Neural Networks **16** (2005) 491–494
50. Bailey, T.L., Elkan, C.: Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In: Proceedings of ISMB, Stanford, CA (1994) 28–36
51. Stormo, G.D.: Dna binding sites: representation and discovery. Bioinformatics **16** (2000) 16–23
52. Sorek, R., Shemesh, R., Cohen, Y., Basechess, O., Ast, G., Shamir, R.: A non-EST-based method for exon-skipping prediction. Genome Research **14** (2004) 1617–1623
53. Hawkins, J., Bodén, M.: The applicability of recurrent neural networks for biological sequence analysis. IEEE/ACM Transactions on Computational Biology and Bioinformatics **2** (2005) 243–253
54. Dyrlöv Bendtsen, J., Nielsen, H., von Heijne, G., Brunak, S.: Improved prediction of signal peptides: SignalP 3.0. Journal of Molecular Biology **340** (2004) 783–795
55. Hawkins, J., Bodén, M.: Detecting and sorting targeting papetides with neural networks and support vector machines. Journal of Bioinformatics and Computational Biology **4** (2006) 1–18
56. Baldi, P., Brunak, S., Frasconi, P., Soda, G., Pollastri, G.: Exploiting the past and the future in protein secondary structure prediction. Bioinformatics **15** (1999) 937–946