Markovian Architectural Bias of Recurrent Neural Networks

Peter Tiňo Michal Čerňanský Ľubica Beňušková

Abstract

In this paper we elaborate upon the claim that clustering in the recurrent layer of recurrent neural networks (RNNs) reflects meaningful information processing states even *prior* to training [1] [2]. By concentrating on activation clusters in RNNs, while not throwing away the continuous state space network dynamics, we extract predictive models that we call neural prediction machines (NPMs). When RNNs with sigmoid activation functions are initialized with small weights (a common technique in the RNN community), the clusters of recurrent activations emerging *prior to training* are indeed meaningful and correspond to Markov prediction contexts. In this case the extracted NPMs correspond to a class of Markov models, called variable memory length Markov models (VLMMs). In order to appreciate how much information has really been induced during the training, the RNN performance should always be compared with that of VLMMs and NPMs extracted *before* training as the "null" base models. Our arguments are supported by experiments on a chaotic symbolic sequence and a context-free language with a deep recursive structure.

Keywords

Recurrent neural networks, complex symbolic sequences, information latching problem, Iterative Function Systems, Markov models.

I. Introduction

IT has been well appreciated for some time that, when trained via gradient-based techniques, recurrent neural networks (RNNs) have problems in catching long-term dependencies in the input stream. Most algorithms fail to learn long dependencies reliably even in the simplest toy problems [3]. This phenomenon is referred to as the problem of information latching [5]. As the temporal sequences

This work was supported by the VEGA grant 1/9046/02 (P.T., M.Č., Ľ.B.). Peter Tiňo is with the School of Computer Science, University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK. Michal Čerňanský is with the Faculty of Electrical Engineering and Information Technology, Slovak Technical University, Ilkovičova 3, 81219 Bratislava, Slovakia. Ľubica Beňušková is with the Faculty of Mathematics, Physics and Informatics, Comenius University, Mlynska dolina, 842 48 Bratislava 4, Slovakia. Corresponding author: Peter Tiňo, P.Tino@cs.bham.ac.uk.

increase in length, the influence of early components of the sequence have less impact on the network output. This causes the partial gradients, used to update the weights, to (exponentially) shrink to zero as the sequence length increases. Several approaches have been suggested to overcome this challenge, e.g. [6] [7]. However, this problem still remains one of the greatest challenges facing the field of neural sequence processing.

The problem of information latching is of special significance for the recurrent network community applying their models to grammar/automata inference tasks. A technique widely used in this community to stabilize the network performance is to extract *finite state* representations from trained networks [8] [9] [10] [11] [12] [13] [14] [15]. Usually, clusters in recurrent neurons' activation space are identified with states of the extracted machines [16]. This methodology was analyzed by Casey [14] and Tiňo et al. [17]. Tiňo and Köteleš [18] showed that extracting finite state representations from RNNs trained on *long* chaotic symbolic sequences reformulates the knowledge gained by the networks in a compact and easy-to-analyze form of finite state *stochastic* machines.

The intuition behind discretizing the state space of RNNs via clustering the recurrent activations is straightforward: Activation patterns across recurrent units can be thought of as spatial codes of the history of inputs seen so far. For example, when trained on symbolic sequences to perform the next-symbol prediction, RNNs tend to organize their state space so that "close" recurrent activation vectors correspond to histories of symbols yielding "similar" next-symbol distributions [18]. Yet, several researches issued a note of caution: When training RNNs to process language structures, activations of recurrent units display a considerable amount of structural differentiation even *prior to learning* [1] [2] [19]. Following [19], we refer to this phenomenon as the *architectural bias of RNNs*. In this paper we provide an explanation of this phenomenon and show that, even prior to any training, clusters in the recurrent layer of RNNs reflect Markov prediction states. In this case clustering recurrent activations leads to construction of predictive models similar in spirit to variable memory length Markov models (VLMMs) [20].

The paper has the following organization: After introducing the necessary notation in section II, we briefly describe Markov predictive models in section III. Section IV is devoted to the recurrent network model used in this study and prediction machines that can be extracted from RNN by quantizing

activations in the recurrent layer. The experiments are described in section V. The discussion in section VI puts our findings into the context of previous work. Section VII concludes the paper by summarizing the key messages of this study. Details related to training RNNs can be found in the Appendix.

II. Notation

Consider a finite alphabet $\mathcal{A} = \{1, 2, ..., A\}$. The set of all finite sequences over $\mathcal{A}$ is denoted by $\mathcal{A}^*$. The set $\mathcal{A}^*$ without the empty string e is denoted by $\mathcal{A}^+$. The set of all sequences over $\mathcal{A}$ with exactly n symbols (i.e. of length n) is denoted by $\mathcal{A}^n$.

The (empirical) probability of finding an n-block $w \in \mathcal{A}^n$ in a sequence S is denoted by $\hat{P}_n(w)$. A string $w \in \mathcal{A}^n$ is said to be an allowed n-block in the sequence S, if $\hat{P}_n(w) > 0$. The set of all allowed n-blocks in S is denoted by $[S]_n$.

Let $S = s_1 s_2 ... \in \mathcal{A}^+$ and $i \leq j$. By S_i^j we denote the string $s_i s_{i+1} ... s_j \in \mathcal{A}^{j-i+1}$, with $S_i^i = s_i \in \mathcal{A}$. The number of symbols in a string $S \in \mathcal{A}^+$ (the length of S) is denoted by $|S|$.

We write the distance between two points $\mathbf{x}$, $\mathbf{y}$ in a Euclidean metric space $\mathcal{X}$ as the norm of their difference $|\mathbf{x} - \mathbf{y}|_{\mathcal{X}}$. As it is always clear which space we refer to, we drop the explicit reference to $\mathcal{X}$ when writing $|\cdot|$.

III. Symbolic sequences and Markov models

In *Markov models* (MMs) of (fixed) order L, the conditional next-symbol distribution over $\mathcal{A}$, 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 ($L \leq t$),

$$P(s_{t+1} | \, S_1^t) = P(s_{t+1} | \, S_{t-L+1}^t). \tag{1}$$

For large memory lengths L, the estimation of prediction probabilities $P(s|w)$, $w \in \mathcal{A}^L$, can become infeasible. By increasing the model order L, the number of prediction contexts $w \in \mathcal{A}^L$, and hence the number of probability distributions to be estimated, rises by A^L, leaving the learner with the problem of the curse of dimensionality. To overcome this problem, several authors developed so-called *variable memory length Markov models* (VLMMs) [20]. 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 S over $\mathcal{A}$. Let $w \in [S]_n$ 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 $aw \in [S]_{n+1}$, the prediction 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 [20] [21]. For other variants of decision criteria see [22] [23].

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) [20] [24]. The edges of PST are labeled by symbols from $\mathcal{A}$. From every internal node there is at most one outgoing edge labeled by each symbol. The nodes of PST are labeled 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 labeled by the reversed string, $(S_1^t)^R = s_t ... s_2 s_1$, starting in the root.

IV. Recurrent neural network and neural prediction machines

Besides discrete state models, like Markov models, continuous state models, such as recurrent neural networks (RNNs), have been widely applied to sequence modeling tasks, e.g. [9][18][19][25]. In this study we work with Elman 1st-order recurrent networks schematically shown in figure 1. The network consists of an input layer, $\mathbf{I}^{(t)} = (I_1^{(t)}, ..., I_A^{(t)})$, and an output layer $\mathbf{O}^{(t)} = (O_1^{(t)}, ..., O_A^{(t)})$, each with A units corresponding to the symbols from the alphabet $\mathcal{A} = \{1, 2, ..., A\}$ appearing in the training and test sequences. The symbols $s \in \mathcal{A}$ are coded as binary vectors $\kappa(s)$ from $\{0, 1\}^A$ via a one-hot

unary encoding scheme: all elements are 0 except for the one corresponding to the coded symbol. In addition, the network contains a hidden recurrent layer, $\mathbf{R}^{(t)} = (R_1^{(t)}, ..., R_N^{(t)})$ and a context layer, $\mathbf{C}^{(t)} = (C_1^{(t)}, ..., C_N^{(t)}) = (R_1^{(t-1)}, ..., R_N^{(t-1)}) = \mathbf{R}^{(t-1)}$, connected through a unit time delay to the recurrent layer. A logistic sigmoid activation function $\sigma(x) = 1/(1+e^{-x})$, mapping $\Re$ into the interval $\Omega = (0, 1)$ of length $|\Omega| = 1$ was used. Note that the RNN state space is the N-dimensional interval Ω^N.

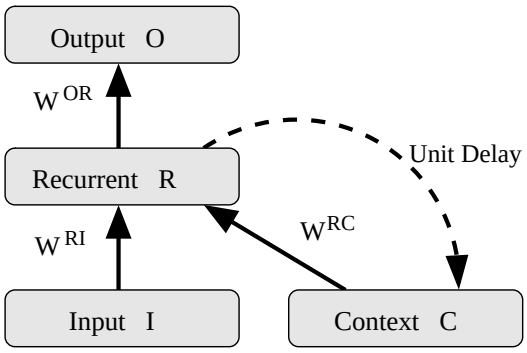

Fig. 1. Schematic representation of Elman RNN.

In terms of equations,

$$R_i^{(t)} = \sigma\left(\sum_j W_{ij}^{RI} \cdot I_j^{(t)} + \sum_j W_{ij}^{RC} \cdot C_j^{(t)} + T_i^R\right) \tag{2}$$

$$O_i^{(t)} = \sigma\left(\sum_j W_{ij}^{OR} \cdot R_j^{(t)} + T_i^O\right) \tag{3}$$

where $\mathbf{W}^{RI}$, $\mathbf{W}^{RC}$, $\mathbf{W}^{OR}$ and $\mathbf{T}^R$, $\mathbf{T}^O$ are real-valued connection weight matrices and threshold vectors, respectively.

We also write the state-transition and output functions, defined by eqs. (2) and (3) as

$$\mathbf{R}^{(t)} = f(\mathbf{I}^{(t)}, \mathbf{C}^{(t)}) = f(\mathbf{I}^{(t)}, \mathbf{R}^{(t-1)}) \tag{4}$$

and

$$\mathbf{O}^{(t)} = h(\mathbf{R}^{(t)}), \tag{5}$$

respectively.

The state-transition map f (eqs. (2), (4)) can be viewed as a family of maps parametrized by the input symbols from $\mathcal{A}$,

$$f_s : \Omega^N \to \Omega^N, \quad f_s(\mathbf{x}) = f(\kappa(s), \mathbf{x}), \quad s \in \mathcal{A}. \tag{6}$$

This notation can be extended to sequences over $\mathcal{A}$ by postulating

$$f_e = Id \tag{7}$$

$$f_{Us} = f_s \circ f_U, \quad s \in \mathcal{A},\ U \in \mathcal{A}^*, \tag{8}$$

where Id is the identity map.

Given an input sequence $S = s_1 s_2 ...$ over $\mathcal{A}$, the RNN state after t time steps is

$$\mathbf{R}^{(t)} = f_{s_1 ... s_t}(\mathbf{R}^{(0)}) = f_{S_1^t}(\mathbf{R}^{(0)}). \tag{9}$$

During training, the training sequence was presented at the rate of one symbol per clock tick. The networks were trained to predict the next symbol in a sequence given the previous context [9] [18] [19]. At the beginning of each training epoch the network is re-set with the same initial state $\mathbf{R}^{(0)} = \mathbf{C}^{(1)}$. The initial state is randomly chosen prior to the training session. Recurrent networks were trained with both the real time recurrent learning (RTRL) [26] [3] and extended Kalman filter (EKF) [27][28] methodologies.

A. Neural Prediction Machines

After training RNNs on symbolic sequences, the main effort is often concentrated on analyzing the resulting activation patterns in the recurrent layer $\mathbf{R}^{(t)}$. We make an explicit use of the induced state-space trajectories in RNNs by formulating prediction models, called *neural prediction machines* (NPM), that inherit state-space dynamics from RNNs. They differ from RNNs only in that they ignore the network output mapping h (5), and instead, based on the network dynamics f (4), use their own predictive (state-conditional) probabilities $P_{NPM}(a|\mathbf{R}^{(t)})$, $a \in \mathcal{A}$.

While the RNN output probabilities are determined using the map h as (e.g. [18] [19])

$$P_{RNN}(a|s_1 s_2 ... s_t) = \tilde{P}(a|\mathbf{R}^{(t)}) = \frac{O_a^{(t)}}{\sum_{b \in \mathcal{A}} O_b^{(t)}}, \tag{10}$$

calculation of the corresponding NPM predictive probabilities involves estimation of the relative frequencies of symbols, conditional on RNN state $\mathbf{R}^{(t)}$. It simply means that, instead of the map $\mathbf{R}^{(t)} \to \mathbf{O}^{(t)} \to P_{RNN}(\cdot|\mathbf{R}^{(t)})$, we have a piece-wise constant map, $\mathbf{R}^{(t)} \to P_{NPM}(\cdot|\mathbf{R}^{(t)})$, estimated on the training sequence. Regions of constant probabilities are determined by vector quantizing the

set of recurrent activations that result from driving the network (weights are fixed) with the training sequence. Bellow, we give a detailed description of the NPM construction.

1. Given a training sequence $S = s_1 s_2 ... s_n$ over the alphabet $\mathcal{A} = \{1, 2, ..., A\}$, first re-set the network to the initial state $\mathbf{C}^{(1)} = \mathbf{R}^{(0)}$, and then, while driving the network with the sequence S, collect the recurrent layer activation vectors in the set $\Gamma = \{\mathbf{R}^{(t)} |\ 1 \leq t \leq n\}$.

2. Run a vector quantizer with M codebook vectors $\mathbf{B}_1, ..., \mathbf{B}_M$, on the set Γ (we used K-means clustering [29]). Vector quantization partitions the space of recurrent activations Ω^N into M regions $V_1, ..., V_M$, that are Voronoi compartments of the codebook vectors $\mathbf{B}_1, ..., \mathbf{B}_M$,

$$V_i = \{\mathbf{R} |\ d(\mathbf{R}, \mathbf{B}_i) = \min_j d(\mathbf{R}, \mathbf{B}_j)\}, \tag{11}$$

where $d(\cdot, \cdot)$ is the Euclidean distance. All points in V_i are allocated[1] to the codebook vector $\mathbf{B}_i$. This is formalized by a projection $\pi : \Omega^N \rightarrow \{\mathbf{B}_1, ..., \mathbf{B}_M\}$,

$$\pi(\mathbf{x}) = \mathbf{B}_i, \quad \text{provided} \ \ \mathbf{x} \in V_i. \tag{12}$$

3. Re-set the network again with the initial state $\mathbf{C}^{(1)}$.

4. Set the [codebook-vector,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.

For $1 \leq t < n$, if $\pi(\mathbf{R}^{(t)}) = \mathbf{B}_i$, 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

$$P(a |\ \mathbf{B}_i) = \frac{\gamma + N(i, a)}{\gamma \cdot A + \sum_{b \in \mathcal{A}} N(i, b)}, \quad a \in \mathcal{A}, \ \ i = 1, 2, ..., M, \tag{13}$$

where, due to possible data sparseness caused by bigger codebooks, we perform smoothing of the empirical relative symbol frequencies $\frac{N(i,a)}{\sum_{b \in \mathcal{A}} N(i,b)}$ by applying Laplace correction with parameter[2] $\gamma > 0$. Typically, $\gamma = 1$, or $\gamma = A^{-1}$. Here we use the latter choice.

Once the NPM is constructed, it can make predictions on a test sequence (a continuation of the training sequence) as follows: Let $\tilde{\mathbf{C}}^{(1)}$ be the vector of recurrent activations after observing the training

[1] Ties as events of measure zero (points land on the border between the compartments) are broken according to index order

[2] The parameter γ can be viewed in the Dirichlet prior interpretation for the multinomial distribution $P(a |\ \mathbf{B}_i)$ as the effective number of times each symbol $a \in \mathcal{A}$ was observed in the context of state compartment V_i, *prior* to counting the conditional symbol occurrences $N(i, a)$ in the training sequence S.

sequence. Given a prefix $u_1u_2,...u_\ell$ of the test sequence, the NPM makes a prediction about the next symbol $u_{\ell+1}$ by:

1. Re-setting the network with the initial state $\tilde{\mathbf{C}}^{(1)}$.

2. Recursively updating recurrent activations $\mathbf{R}^{(t)}$ (4) while driving the network with $u_1u_2...u_\ell$.

3. Using (13), the next symbol distribution on the test sequence given by the NPM is

$$P_{NPM}(u_{\ell+1}|u_1u_2...u_\ell) = P(u_{\ell+1}|\ \pi(\mathbf{R}^{(\ell)})). \tag{14}$$

Note that the *dynamics* of NPM is the same as that of the associated RNN. In other words, NPM makes use of spatial representations of symbol histories encoded by the RNN. The history of symbols seen by the network up to time t is represented by the vector of recurrent activations $\mathbf{R}^{(t)}$. Quantization of RNN state space merely helps us to estimate, on a (finite) training sequence, the state-conditional predictive probabilities $P(\cdot|\mathbf{R}^{(t)})$ in a piece-wise constant manner by computing (quantization compartment conditional) relative frequencies of the next symbols. If there is a finite number of prediction contexts (e.g. in Markov models – L-blocks over $\mathcal{A}$, or in NPMs – codebook vectors $\mathbf{B}_1,...,\mathbf{B}_M$), then, given a sufficiently long training sequence S, the maximum likelihood estimates of the context-conditional next-symbol distributions over $\mathcal{A}$ are obtained by counting the context-conditional relative frequencies of the next symbols in S. For a related approach to building finite memory predictive models based on a hand-designed contractive iterative function system (IFS) [30], see [31].

B. Contractive NPMs

In this section we are interested in cases where all maps f_s, $s \in \mathcal{A}$, are contractions, i.e. there is a constant $0 \leq C < 1$, such that for all $\mathbf{x}, \mathbf{y} \in \Omega^N$,

$$|f_s(\mathbf{x}) - f_s(\mathbf{y})| \leq C \cdot |\mathbf{x} - \mathbf{y}|. \tag{15}$$

This is the case when RNNs with sigmoid activation functions are initialized with small weights [32] - a common technique used in the RNN community. We will show that in such situations NPMs essentially correspond to finite memory predictors.

Consider a string $S = s_1 s_2 ... s_r \in \mathcal{A}^r$, $r \geq 1$. For any two strings $U, V \in \mathcal{A}^*$,

$$
\begin{aligned}
|f_{US}(\mathbf{x}) - f_{VS}(\mathbf{x})| &= |f_{S_1^{r-1}s_r}(f_U(\mathbf{x})) - f_{S_1^{r-1}s_r}(f_V(\mathbf{x}))| && (16)\\
&= |f_{s_r}(f_{S_1^{r-1}}(f_U(\mathbf{x}))) - f_{s_r}(f_{S_1^{r-1}}(f_V(\mathbf{x})))| && (17)\\
&\leq C \cdot |f_{S_1^{r-1}}(f_U(\mathbf{x})) - f_{S_1^{r-1}}(f_V(\mathbf{x}))| && (18)
\end{aligned}
$$

Repeating this step $r = |S|$ times yields

$$
\begin{aligned}
|f_{US}(\mathbf{x}) - f_{VS}(\mathbf{x})| &\leq C^r \cdot |f_U(\mathbf{x}) - f_V(\mathbf{x})| && (19)\\
&\leq C^{|S|} \cdot diam(\Omega^N) && (20)\\
&= C^{|S|} \cdot \sqrt{N} \cdot |\Omega| && (21)
\end{aligned}
$$

Equations (19–21) imply that no matter what state $\mathbf{x} \in \Omega^N$ the RNN is initiated with, the neural codes $f_P(\mathbf{x})$, $f_Q(\mathbf{x})$ of two sequences $P, Q \in \mathcal{A}^+$ sharing a long common suffix S will lie close to each other. Furthermore, the longer is the common suffix shared by P and Q, the closer lie $f_P(\mathbf{x})$ and $f_Q(\mathbf{x})$.

In fixed-order Markov model of memory depth L, the prediction context on which we base our next-symbol prediction is simply the string of the last L observed symbols. In NPM, the prediction contexts are the quantization regions V_1,...,$V_M \subseteq \Omega^N$, or equivalently, the corresponding codebook vectors $\mathbf{B}_1, ..., \mathbf{B}_M \in \Omega^N$. The codebook vectors are obtained by vector quantizing the set $\Gamma = \{f_{S_1^t}(\mathbf{R}^{(0)}) |\ 1 \leq t \leq n\}$ of the neural codes of prefixes of the training sequence $S = s_1 s_2 ... s_n$. In contractive NPMs, histories of seen symbols sharing long common suffixes (i.e. histories that are likely to produce similar continuations) are mapped close to each other. This makes them likely to appear in the same quantization region. Dense areas in the RNN state space formed by the neural codes from Γ 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 S_1^t in S sharing a long common suffix. In such situations, when predicting the next symbol one has to be careful and consider more symbols into the past. Note that this is done automatically by the vector quantizer 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 group symbol histories S_1^t with long shared suffixes.

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.

We finish this section by noting that (19–21) also imply

$$\begin{aligned} |f_{S_1^t}(\mathbf{x}) - f_{S_{t-L+1}^t}(\mathbf{x})| &\leq C^L \cdot |f_{S_1^{t-L}}(\mathbf{x}) - \mathbf{x}| \quad (22) \\ &\leq C^L \cdot \sqrt{N} \cdot |\Omega|. \quad (23) \end{aligned}$$

This can be interpreted as follows: Imagine that while observing an input stream $S = s_1 s_2 ...$ we have at our disposal only a finite memory length L. In other words, all we can process are the most recent L input symbols[3] $s_{t-L+1}...s_t = S_{t-L+1}^t$. If we started processing each L-block from the same initial state $\mathbf{R}^{(0)}$, at time t we would end up in the state

$$\mathbf{R}_L^{(t)} = f_{S_{t-L+1}^t}(\mathbf{R}^{(0)}). \quad (24)$$

Equations (22–23) state that for *any* initial state $\mathbf{R}^{(0)} = \mathbf{x}$, *any* sequence $S \in \mathcal{A}^n$, and *any* memory length $L < n$, the activations $\mathbf{R}_L^{(n)}$ approximate $\mathbf{R}^{(n)}$ with precision $C^L \cdot \sqrt{N} \cdot |\Omega|$. It follows, that for sufficiently long strings S and sufficiently long fixed finite memory length L, $\mathbf{R}_L^{(n)}$ can approximate (in L_∞ norm) $\mathbf{R}^{(n)}$ to arbitrarily small precision. The activations $\mathbf{R}_L^{(t)}$ are states of a finite memory machine operating on a finite state set

$$\mathcal{R}_L = \{f_V(\mathbf{R}^{(0)}) |\ V \in \mathcal{A}^L\}. \quad (25)$$

V. Experiments

We tested the architectural bias hypothesis on a sequence of quantized activations of a laser in a chaotic regime and an artificial language exhibiting deep recursive structures. The laser sequence can

[3] we assume $L \leq t$

be modeled quite successfully with finite memory predictors, although, because of the limited finite sequence length, predictive contexts of variable memory depth are necessary. Recursive structures, however, cannot in principle be grasped by finite memory models and trained RNNs should be able to outperform their initial finite memory predictive capabilities.

A. RNN training

We trained RNNs using Real Time Recurrent Learning (RTRL) [26] and Extended Kalman Filtering (EKF) [27][28]. The initial state $\mathbf{R}^{(0)} = \mathbf{C}^{(1)}$ and weights of RNN were randomly initialized from the intervals $\Omega^N = (0.0, 1.0)^N$ and $(-0.5, 0.5)$, respectively. Unlike RTRL, EKF does not update all RNN weights with the same learning rate. The Kalman gain matrix adjusts the learning rate individually for each RNN weight parameter. Numerous simulations were performed with different learning (hyper)parameters. Although compared with RTRL, EKF is much more computationally expensive, it converges in a much smaller number of epochs, and is not that sensitive to the values of (hyper)parameters.

To emphasize the contrast between trained RNNs and prediction machines extracted from *untrained* networks, in each experiment we present the average results (Mean $\pm$ St.Dev.) over 10 simulation runs performed with a fixed set of (hyper)parameter values that were found to give the best overall RNN performance.

As expected, high momentum and learning rates in RTRL lead to a faster convergence at the cost of frequent significant error increases/fluctuations. On the other hand, smaller rates make training slow and susceptible to local minima. We tried to avoid these problems by resorting to annealed schedules for learning rate (e.g. [33]). Compared with the fixed learning rate regime, a piece-wise linear learning rate decrease: $0.05 \rightarrow 0.001$ over the first 600.000 input symbol presentations and $0.001 \rightarrow 0.0001$ over the next 600.000 – 6.000.000 time steps, was found to give consistently improved RNN performance on the data sets considered here. In general, RTRL achieves its minimum only after thousands of epochs, and this minimum is systematically higher, albeit slightly, than that obtained with EKF. Therefore, here we present only the results obtained with EKF.

State error covariance matrix $\mathbf{P}$ in the EKF training was initialized to $\mathbf{P} = p \cdot \mathbf{E}$, where $p = 1000$

and $\mathbf{E}$ is the identity matrix. The measurement noise covariance matrix $\mathbf{N}$ was set to $\mathbf{N} = n \cdot \mathbf{E}$, where $n = 100$ and the output noise covariance matrix $\mathbf{Q}$ was set to $\mathbf{Q} = q \cdot \mathbf{E}$, where $q = 0.0001$. More details about the EKF training are given in Appendix.

For each data set, we experimented with a range of RNNs having 4, 8, 16 and 32 recurrent units. Here we present simulation results for 16 recurrent units. We found that RNNs with 4 and 8 units could not be consistently trained to match the VLMM performance. On the other hand, RNNs with 32 units did not significantly outperformed their 16-unit counterparts. In any case, using more or less units does not alter the principal message we would like to stress in this paper - the architectural bias of RNNs towards finite memory models and the need to assess the amount of induced knowledge in RNNs by comparing them with Markovian models as the base "null" models.

B. Measuring the model performance

For each data set, after constructing predictive models on the training sequence, we calculated the normalized negative log-likelihood (NNL) given by the models on the test sequence $S_{test} = s_1 s_2 ... s_m$. For a predictive model $\mathcal{M}$, the likelihood is computed by multiplying all the next symbol probabilities given by $\mathcal{M}$ on S_{test}. The NNL measure is obtained by dividing the negative logarithm of this product by the number of tested symbols,

$$NNL_{\mathcal{M}}(S_{test}) = \frac{-\sum_{t=1}^{m-1} \log_A P_{\mathcal{M}}(s_{t+1}|\ s_1 ... s_t)}{m-1}, \tag{26}$$

where the base of the logarithm is the number of symbols A in the alphabet $\mathcal{A}$ and $P_{\mathcal{M}}(s_{t+1}|\ s_1 ... s_t)$ is given by (10) and (14) for $\mathcal{M} = RNN$ and $\mathcal{M} = NPM$, respectively. The higher are the predictive probabilities assigned to the actual next symbols, the smaller is NNL. Vanishing NNL corresponds to the perfect next-symbol prediction. The measure $NNL_{\mathcal{M}}(S_{test})$ can be viewed as the amount of "statistical surprise" experienced by the model $\mathcal{M}$ when seeing the sequence S_{test} [20].

While training the networks with both RTRL and EKF, we continually checked the RNN and NPM performance on the test sequence. The NPMs had a maximum codebook size $\mathcal{M}$ of 300. To assess the amount of learned information, we also measured the network and NPM performance *before training*, just after random initialization of RNN weights. In addition, to check the architectural bias hypothesis,

we trained and tested both the fixed-order and variable memory length Markov models (section III) on the sequences used for training and testing RNNs and NPMs. As with NPMs, MMs and VLMMs had up to 300 predictive contexts.

C. Laser in a chaotic regime

In the first experiment we trained the models on a sequence of quantized activity changes of a laser in a chaotic regime.

Deterministic chaotic dynamical systems usually organize their behavior around chaotic attractors containing regions of different levels of instability (sensitivity to small perturbations in initial conditions), measured e.g by the local Lyapunov exponents. Periods of relatively predictable behavior are followed by periods of unpredictable development (due to finite precision of our measuring devices and computing machines). By quantizing a chaotic trajectory into a symbolic stream (each symbol corresponds to a region of the state space where the system evolves), a technique well-known in symbolic dynamics, we obtain a rough picture about the basic topological, metric and memory structure of the trajectories (see e.g. [34]). Relatively predictable subsequences having various levels of memory structure are followed by highly unpredictable events usually requiring a deep memory. To model such sequences we need to vary the memory depth with respect to the context [31].

The data set was a long time series $\{D_t\}$ of 10 000 differences between the successive activations of a real laser in a chaotic regime. The series $\{D_t\}$ was quantized into a symbolic stream $S = \{s_t\}$ over four symbols ($\mathcal{A} = \{1, 2, 3, 4\}$) corresponding to low and high positive/negative laser activity change as shown in figure 2. The first 8000 symbols and the remaining 2000 symbols from the laser symbolic sequence S formed the training and test sequences, respectively.

The results are shown in figure 3. Fixed-order Markov models are beaten by VLMMs, since, as explained above, for this data set it is advantageous to use predictive contexts of variable memory length. The best NNL achieved by VLMM with 300 predictive contexts is equal to 0.2

The architectural bias is clearly visible. Note how the NPM performance before training (0th training epoch) follows the shape of the NNL curve for VLMMs. NNL can be improved by a small amount through an expensive RNN training. Interestingly enough, almost the same performance can

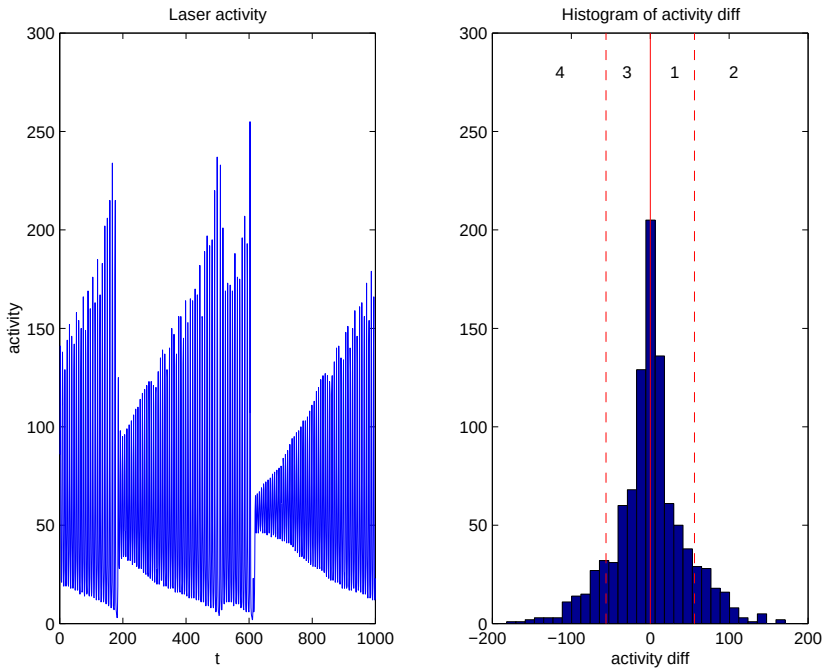

Fig. 2. Left – the first 1000 activations of a laser in a chaotic regime. Right – histogram of the differences between the successive activations. Dotted vertical lines show the cut values used to quantize the activation differences into four regions. Symbols 1, 2, 3 and 4, corresponding to the quantization regions appear on top of the figure.

be achieved by a cheap construction of NPMs from randomly initiated *untrained* networks. The initial average NNL of NPMs extracted from untrained RNNs is equal to 0.17. After training, the final average NNL of RNN outputs is equal to 0.14, the same value as the NNL of NPMs extracted from trained RNNs.

D. Context-Free Language with Deep Recursion

The second data set is composed of strings from a context-free language L_G with generating grammar $G = (R, \{R\}, \mathcal{A} = \{1, 2, 3, 4\}, P)$, where R is a (single) non-terminal symbol that is also the start symbol, and $1, 2, 3, 4 \in \mathcal{A}$ are terminal symbols. P is the set of three production rules

$$R \rightarrow 1R3 \tag{27}$$

$$R \rightarrow 2R4 \tag{28}$$

$$R \rightarrow e, \tag{29}$$

where e is the empty string. Finite memory models are not well-suited to model recursive structures and therefore we expect RNNs to be trained well beyond capabilities of MMs, VLMMs and NPMs

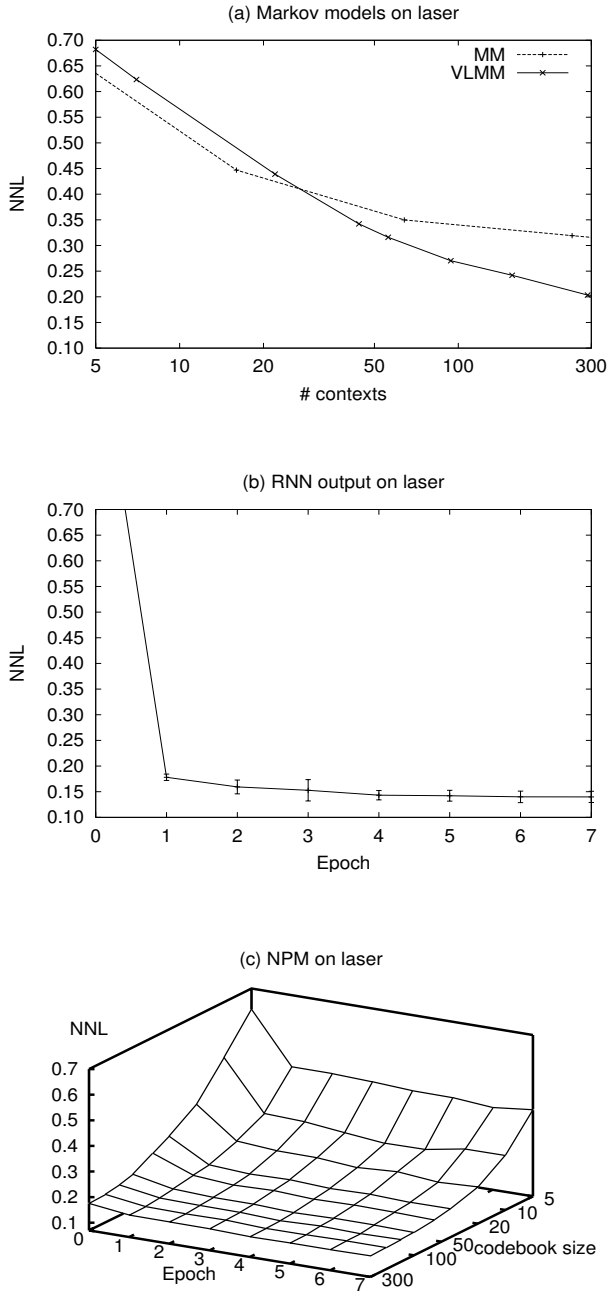

Fig. 3. NNL performance of MMs, VLMMs (a), RNNs (b), and NPMs extracted from RNNs trained with EKF (c) in the laser data experiment.

Length	Percent in the sequence	Examples
2	34.5 %	$13, 24$
4	20.6%	$1243, 1133$
6	13.6 %	$122443, 211344$
8	10.3%	22213444
10,12,14,...20	each length 3.5 %	121212111333434343

TABLE I

Distribution of string lengths in the training and test sequences in the deep recursion experiment.

extracted from untrained networks.

Both training and test sequences consisted of 1000 randomly generated concatenated strings. Shorter strings were more frequent than the longer ones, as specified in table I. The training and test sequences contained 6156 and 6192 symbols, respectively.

First half of any string from L_G contains only symbols 1 and 2, while the second half contains 3's and 4's. The first symbol is always 1 or 2, and so can be predicted with probability 50%. *Any* potential *sequential* predictor does not know where the first half of the string ends. All it can observe is that 3 cannot follow 2 and that 4 cannot follow 1. Hence, symbols from the first half of the string and the first symbol of the second half of the string can be predicted with 33% accuracy. Once it runs across the first 3 or 4 (start of the second half of the string), all the following symbols 3 and 4 can be predicted with 100% accuracy by looking at their counterparts 1 and 2 in the first half of the string. After all 3's and 4's were processed, the next string begins with symbol 1 or 2, which again can be predicted with probability 50%, etc. Based on these considerations, we calculated the "bottom line" NNL that can be achieved on the test sequence as 0.477.

The results are shown in figure 4. Note that the architectural bias of RNN is again clearly visible. The NNL performance of NPMs extracted from untrained networks (i.e., 0.68) roughly corresponds to the best performance of VLMMs (i.e., 0.62). As expected, RNNs could be trained to process recursive structures much better than finite memory models. The average final NNLs for RNNs and NPMs were

equal to 0.52 and 0.51 (for 20 to 140 quantization centers), respectively[4]. Even though NPMs do not perform worse than the RNNs they were extracted from, using a large codebook may not bring such a dramatic improvement in performance as seen in the case of untrained networks.

VI. Discussion

Kolen [1] [2] studied the clustering behavior in recurrent neural networks (RNNs) and explained the origin of clusters by viewing RNNs as non-linear iterative function systems. By showing that clusters emerge even prior to training[5], he argued against the interpretation of clusters as abstract information processing states discovered by RNN. In [1] he articulated two problems associated with extracting finite automata from trained networks. First, sensitivity to initial conditions produces nondeterministic machines, whose trajectories are specified by the initial state and the state-transformations. Second, trivial changes in observation strategies can lead to descriptions from a wide range of computational complexity classes. These issues were later addressed also by Blair and Pollack [4]. The substance of Kolen's criticism is that the (possibly delicate continuous state space) RNN dynamics is merely used as a pathway for designing finite state machines and the network is thrown away once the automaton is extracted. This is not the case in NPMs. Clusters of recurrent activations are interpreted as information processing states in that it is assumed that all states belonging to the same quantization region yield the same next-symbol distribution. On the other hand, state transitions are driven by the original RNN dynamics and so we throw away only the RNN output map (5). By the information processing state assumption, we can estimate the emission probabilities for each quantization region by simply counting the corresponding (quantization region conditional) next-symbol frequencies.

Obviously, by keeping the original RNN dynamics in NPMs we lose the possibility of compact finite-state reformulations of knowledge induced in RNN during the training. As shown in [1] [2], such reformulations may be questionable. On the other hand, since NPMs directly link dynamical features of RNN to the next-symbol prediction task, we could show that clusters appearing in the RNN state

[4]Note that despite the Laplace correction, VLMMs and NPMs with a large number of (deeper) prediction contexts tend to slightly overfit the training data.

[5]The tendency of RNNs to form clusters in the state space before training was first reported by Servan-Schreiber, Cleeremans and McClelland [37].

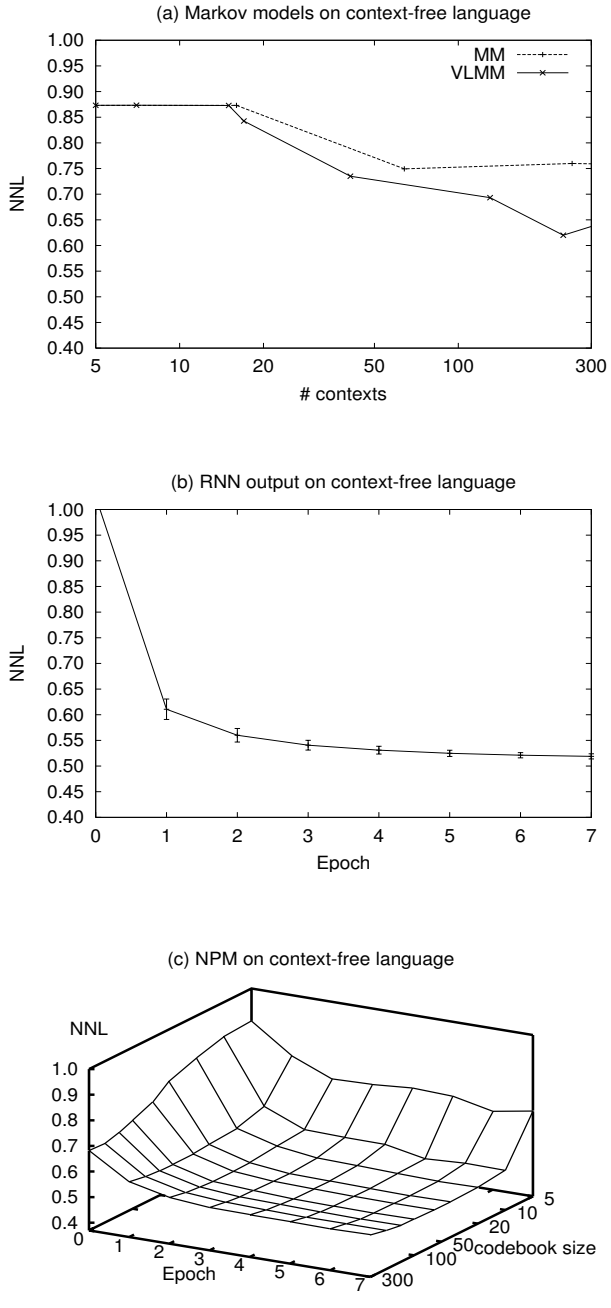

Fig. 4. NNL performance of MMs, VLMMs (a), RNNs (b), and NPMs extracted from RNNs trained with EKF (c) in the deep recursion, context-free language, experiment.

space *prior to training* can be interpreted as meaningful prediction contexts. When initializing RNN with small weights, each of the dynamical systems (6) is a contraction driven by a single attractive fixed point [17] [32][6]. Due to the contractive nature of the maps (6), the extracted NPMs roughly correspond to a class of Markov models, called variable memory length Markov models (VLMMs).

Christiansen and Chater trained Elman simple recurrent network [9] to perform next symbol prediction in a set of experiments with increasing recursion complexity [19]. They applied discriminant analysis to the RNN state space and concluded that some useful features of the state space organization appear prior to training. They referred to this phenomenon as the *architectural bias of recurrent networks*, but did not provide any possible explanation for it. They suggest to compare state space representations in RNNs before and after the training. Our present work offers an explanation for the architectural bias phenomenon.

As illustrated in our experiments, when the modeler trains a RNN on a complex sequence he/she may be tempted to think that the network extracts a lot of useful information. Often, due to the information latching problem [5] mentioned in the introduction, RNNs can beat finite memory models only at a price of rather lengthy and complicated training procedure (see e.g. [38]). But we have shown that finite memory predictive models can be readily extracted from RNNs without a need to ever train them. Moreover, the extracted models, that we call neural prediction machines (NPM), correspond to an efficient implementation of Markov models - VLMM - where the memory depth is made context-dependent [20]. Therefore, to see how much useful information has really been induced during the training, one has to consult VLMMs and NPMs extracted *before* training.

A. Related work

In [31] we studied a hand-designed affine iterative function system (IFS) for coding symbol histories on a fractal set. Prediction machines, similar to our NPMs, constructed on such a fractal set had competitive performance to both VLMMs and Markov models. Interestingly enough, we were able to rigorously connect the complexity measures on symbolic sequences (entropy spectra) with complexity

[6]Actually, this type of simple dynamics is the reason for starting to train recurrent networks from small weights. Unless one has a strong prior knowledge about the network dynamics [35], the sequence of bifurcations leading to a desired network behavior may be hard to achieve when starting from an arbitrarily complicated network dynamics [36].

measures of the corresponding fractal representations (multifractal generalized dimension spectra) [39] [40].

Our current work is also related to the recent research activity in fractal and multifractal analysis of strange sets arising in chaotic dynamical systems. The strange sets are modeled using Moran-like constructions [41] [42]. The Moran-like geometric constructions iteratively construct limit sets using a collection of basic sets that have symbolic addresses and are increasingly refined (with increasing address length) according to a given symbolic dynamical system. As the construction proceeds, the diameter of the basic sets diminishes to zero.

Tabor [43] studied metric relations between IFS representations of various forms of automata. He also investigated possibilities of coding complex symbolic structures, such as context-free grammars, on a fractal set using IFSs [44] [45].

VII. Conclusion

We have shown that clusters in the recurrent layer of recurrent networks can reflect meaningful information processing states, *even prior to training*. Christiansen and Chater refer to this phenomenon as the "architectural bias" of recurrent neural networks [19], but no explanation has been proposed so far.

We have demonstrated that when RNNs with sigmoid activation functions are initialized with small weights (a common technique in the RNN community),

1. the clusters of recurrent activations that emerge prior to training correspond to Markov prediction contexts – histories of symbols are grouped according to the number of symbols they share in their suffix, and

2. concentrating on the activation clusters, it is possible to extract from untrained RNNs predictive models that correspond to a class of Markov models, called variable memory length Markov models.

Recurrent networks have a potential to outperform finite memory models, but to appreciate how much information has really been induced during the training, the RNN performance should always be compared with that of VLMMs and NPMs extracted *before* training as the "null" base models.

Appendix

The Kalman filter (KF) is a set of equations describing a recursive solution of the linear discrete-data filtering problem [48]. Applying KF to the nonlinear system can be done in several ways. Probably the most straightforward way is to apply KF to the system linearized in every time step using Taylor expansion. This approach is called an extended Kalman filter (EKF). Training of Elman RNN and generally any other multilayer perceptron network (recurrent or not) can be regarded as an optimal filtering problem [46]. The set of EKF equations for the network training can be formulated as follows:

$$\mathbf{K}(t) = \mathbf{P}(t-1)\mathbf{H}^T(t)[\mathbf{H}(t)\mathbf{P}(t-1)\mathbf{H}^T(t) + \mathbf{N}(t)]^{-1}, \tag{30}$$

$$\mathbf{P}(t) = \mathbf{P}(t-1) - \mathbf{K}(t)\mathbf{H}(t)\mathbf{P}(t-1) + \mathbf{Q}(t), \tag{31}$$

$$\mathbf{W}(t) = \mathbf{W}(t-1) + \mathbf{K}(t)[\mathbf{D}(t) - \mathbf{O}(t)]. \tag{32}$$

Let n_w and n_o denote the number of all network weights and number of output units, respectively. In terms of RNN equations described in section IV, $\mathbf{W}$ is a vector of all weights (from matrices $\mathbf{W}^{RI}$, $\mathbf{W}^{RC}$, $\mathbf{W}^{OR}$ and vectors $\mathbf{T}^R$, $\mathbf{T}^O$) of length n_w. $\mathbf{H}$ is $n_o \times n_w$ Jacobian matrix calculated in every time step containing in rows derivatives of corresponding output activity with respect to all weights. These derivatives can be calculated by routines very similar to real time recurrent learning (RTRL) [26] or backpropagation through time (BPTT) [49]. $\mathbf{P}$ is $n_w \times n_w$ error covariance matrix, it holds error covariances corresponding to each pair of network weights. Ignoring error covariances between weights of different network units leads to decoupled extended Kalman filter (DEKF). Dividing matrix $\mathbf{P}$ into smaller matrices corresponding to network units (or groups of units) decreases computational requirements. On the other hand, full coupling - global EKF approach (eq. 30 – 32) - can yield better solutions [46]. $n_w \times n_o$ matrix $\mathbf{K}$, called Kalman gain, is used in updating weights $\mathbf{W}$ according to the difference between desired output vector $\mathbf{D}$ and actual network output $\mathbf{O}$. $n_o \times n_o$ matrix $\mathbf{N}$ stands for measurement noise covariance matrix and similarly to the learning rate in RTRL or BPTT can control training speed of EKF. Higher values represent higher amount of uncertainty attributed to the difference $\mathbf{D}(t) - \mathbf{O}(t)$ leading to slower training. $n_w \times n_w$ matrix $\mathbf{Q}$ stands for process noise covariance matrix. Nonzero process noise improves convergence of the filter.

In the EKF-based training described here, the EKF estimated state is a concatenation of network's weights. Alternatively, network training can be formulated as a dual estimation problem where both network weights and activities are to be estimated. Multiple approaches to dual estimation exist [47]. Other variations of EKF can be based on training on multiple input streams (multi-stream EKF [46]).

References

[1] J.F. Kolen, "Recurrent networks: state machines or iterated function systems?," in *Proc. 1993 Connectionist Models Summer School*, M.C. Mozer, P. Smolensky, D.S. Touretzky, J.L. Elman, and A.S. Weigend, Eds., Hillsdale, NJ: L. Erlbaum Associates, 1994, pp. 203–210.

[2] J.F. Kolen, "The origin of clusters in recurrent neural network state space," in *Proc. Sixteenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ: L. Erlbaum Associates, 1994, pp. 508–513.

[3] S.C. Kremer, "Spatio-temporal connectionist networks: A taxonomy and review," *Neural Computation*, vol. 13, no. 2, pp. 249–306, 2001.

[4] A.D. Blair and J.B. Pollack, "Analysis of Dynamical Recognizers", *Neural Computation*, vol. 9, pp. 1127–1142, 1997.

[5] Y. Bengio, P. Frasconi, and P Simard, "The problem of learning long-term dependencies in recurrent networks," in *Proc. 1993 IEEE International Conference on Neural Networks*, vol. 3, pp. 1183–1188, 1993.

[6] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[7] J. Hochreiter and J. Schmidhuber, "Long short term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] A. Cleeremans, D. Servan-Schreiber, and J.L. McClelland, "Finite state automata and simple recurrent networks," *Neural Computation*, vol. 1, no. 3, pp. 372–381, 1989.

[9] J.L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, 1990.

[10] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, and Y.C. Lee, "Learning and extracting finite state automata with second–order recurrent neural networks," *Neural Computation*, vol. 4, no. 3, pp. 393–405, 1992.

[11] P. Manolios and R. Fanelli, "First order recurrent neural networks and deterministic finite state automata," *Neural Computation*, vol. 6, no. 6, pp. 1155–1173, 1994.

[12] R.L. Watrous and G.M. Kuhn, "Induction of finite–state languages using second–order recurrent networks," *Neural Computation*, vol. 4, no. 3, pp. 406–414, 1992.

[13] P. Tiňo and J. Sajda, "Learning and extracting initial mealy machines with a modular neural network model," *Neural Computation*, vol. 7, no. 4, pp. 822–844, 1995.

[14] M.P. Casey, "The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction," *Neural Computation*, vol. 8, no. 6, pp. 1135–1178, 1996.

[15] P. Frasconi, M. Gori, M. Maggini, and G. Soda, "Insertion of finite state automata in recurrent radial basis function networks," *Machine Learning*, vol. 23, pp. 5–32, 1996.

[16] C.W. Omlin and C.L. Giles, "Extraction of rules from discrete-time recurrent neural networks," *Neural Networks*, vol. 9, no. 1, pp. 41–51, 1996.

[17] P. Tiňo, B.G. Horne, C.L. Giles, and P.C. Collingwood, "Finite state machines and recurrent neural networks – automata

and dynamical systems approaches," in *Neural Networks and Pattern Recognition*, J.E. Dayhoff and O. Omidvar, Eds., New York: Academic Press, 1998, pp. 171–220.

[18] P. Tiňo and M. Koteles, "Extracting finite state representations from recurrent neural networks trained on chaotic symbolic sequences," *IEEE Transactions on Neural Networks*, vol. 10, no. 2, pp. 284–302, 1999.

[19] M.H. Christiansen and N. Chater, "Toward a connectionist model of recursion in human linguistic performance," *Cognitive Science*, vol. 23, pp. 417–437, 1999.

[20] D. Ron, Y. Singer, and N. Tishby, "The power of amnesia," in *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann, 1994, pp. 176–183.

[21] I. Guyon and F. Pereira, "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, pp. 454–457

[22] M.J. Weinberger, J.J. Rissanen, and M. Feder, "A universal finite memory source," *IEEE Trans. Information Theory*, vol. 41, no. 3, pp. 643–652, 1995.

[23] P. Buhlmann and A.J. Wyner, "Variable length Markov chains," *Annals of Statistics*, vol. 27, pp. 480–513, 1999.

[24] J. Rissanen, "A universal data compression system," *IEEE Trans. Information Theory*, vol. 29, no. 5, pp. 656–664, 1983.

[25] P. Rodriguez, J. Wiles, and J.L. Elman, "A recurrent neural network that learns to count," *Connection Science*, vol. 11, pp. 5–40, 1999.

[26] R.J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 270–280, 1989.

[27] R.J. Williams, "Training recurrent networks using the extended kalman filter," in *Proc. 1992 Int. Joint Conf. Neural Networks*, vol. 4, 1992, pp. 241–246.

[28] G. Patel, S. Becker, and R. Racine, "2D image modelling as a time-series prediction problem," in *Kalman filtering applied to Neural Networks*, S. Haykin, Ed., Wiley, 2001.

[29] M.R. Anderberg, *Cluster Analysis for Applications*, New York: Academic Press, 1973.

[30] M.F. Barnsley, *Fractals everywhere*, New York: Academic Press, 1988.

[31] P. Tiňo and G. Dorffner, "Predicting the future of discrete sequences from fractal representations of the past," *Machine Learning*, vol. 45, no. 2, pp. 187–218, 2001.

[32] P. Tiňo, B.G. Horne, and C.L. Giles, "Attractive periodic sets in discrete time recurrent networks (with emphasis on fixed point stability and bifurcations in two–neuron networks)," *Neural Computation*, vol. 13, no. 6, pp. 1379–1414, 2001.

[33] S. Lawrence, C. L. Giles, and S. Fong, "Natural language grammatical inference with recurrent neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 1, pp. 126–140, 2000.

[34] A. Katok and B. Hasselblatt, *Introduction to the modern theory of dynamical systems*, Cambridge: Cambridge University Press, 1995.

[35] C.L. Giles and C.W. Omlin, "Insertion and refinement of production rules in recurrent neural networks," *Connection Science*, vol. 5, no. 3, pp. 307–318, 1993.

[36] K. Doya, "Bifurcations in the learning of recurrent neural networks," in *Proc. of 1992 IEEE Int. Symposium on Circuits and Systems*, 1992, pp. 2777–2780.

[37] D. Servan-Schreiber, A. Cleeremans, and J. McClelland, "Encoding sequential structure in simple recurrent networks," in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed., San Mateo, CA: Morgan Kaufmann, 1989.

[38] D.L.T. Rohde and D.C. Plaut, "Language acquisition in the absence of explicit negative evidence: How important is starting small?," *Cognition*, vol. 72, pp. 67–109, 1999.

[39] P. Tiňo, "Spatial representation of symbolic sequences through iterative function system," *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 29, no. 4, pp. 386–392, 1999.

[40] P. Tiňo, "Multifractal properties of Hao's geometric representations of DNA sequences," *Physica A: Statistical Mechanics and its Applications*, vol. 304, no. 3–4, pp. 480–494, 2002.

[41] Y. Pesin, *Dimension Theory in Dynamical Systems: Rigorous Results and Applications*, Chicago: University of Chicago Press, 1997.

[42] Y. Pesin and H. Weiss, "On the dimension of deterministic and Cantor-like sets, symbolic dynamics, and the Eckmann-Ruelle conjecture," *Comm. Math. Phys*, vol. 182, no. 1, pp. 105–153, 1996.

[43] W. Tabor, "Dynamical automata," Tech. Rep. TR98-1694, Cornell University, Computer Science Department, 1998.

[44] W. Tabor and M.K. Tanenhaus, "Dynamical models of sentence processing," *Cognitive Science*, vol. 23, no. 4, pp. 491–515, 2000.

[45] W. Tabor, "Fractal encoding of context free grammars in connectionist networks," *Expert Systems: The International Journal of Knowledge Engineering and Neural Networks*, vol. 17, no. 1, pp. 41–56, 2000.

[46] L. Feldkamp, D. Prokhorov, C. Eagen, and F. Yuan, "Enhanced multi-stream Kalman filter training for recurrent networks," in *Nonlinear Modeling: Advanced Black-Box Techniques*, J. Suykens and J. Vandewalle, Eds., Kluwer Academic Publishers, 1998, pp. 29-53.

[47] E. A. Wan and Alex T. Nelson, "Dual EKF methods," in *Kalman Filtering and Neural Networks*, S. Haykin, Ed., Wiley, 2000.

[48] G. Welch and G. Bishop, "An Introduction to the Kalman filter," TR NC27599-3175, University of North Carolina, 1997.

[49] P.J. Werbos, "Backpropagation through time; what it does and how to do it," *Proc. IEEE*, vol. 78, pp. 1550–1560, 1990.