Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 20 (2007) 236–244
Neural Networks
www.elsevier.com/locate/neunet

Organization of the state space of a simple recurrent network before and after training on recursive linguistic structures

Michal Čerňanský[a,*], Matej Makula[a], Ľubica Beňušková[b]

[a] Faculty of Informatics and Information Technologies, Slovak Technical University, Ilkovičova 3, 842 16 Bratislava 4, Slovakia
[b] Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University, Mlynská dolina, 842 48 Bratislava 4, Slovakia

Received 5 January 2005; accepted 30 January 2006

Abstract

Recurrent neural networks are often employed in the cognitive science community to process symbol sequences that represent various natural language structures. The aim is to study possible neural mechanisms of language processing and aid in development of artificial language processing systems. We used data sets containing recursive linguistic structures and trained the Elman simple recurrent network (SRN) for the next-symbol prediction task. Concentrating on neuron activation clusters in the recurrent layer of SRN we investigate the network state space organization before and after training. Given a SRN and a training stream, we construct predictive models, called *neural prediction machines*, that directly employ the state space dynamics of the network. We demonstrate two important properties of representations of recursive symbol series in the SRN. First, the clusters of recurrent activations emerging *before training* are meaningful and correspond to Markov prediction contexts. We show that prediction states that naturally arise in the SRN initialized with small random weights approximately correspond to states of Variable Memory Length Markov Models (VLMM) based on individual symbols (i.e. words). Second, we demonstrate that during training, the SRN reorganizes its state space according to word categories and their grammatical subcategories, and the next-symbol prediction is again based on the VLMM strategy. However, after training, the prediction is based on word categories and their grammatical subcategories rather than individual words. Our conclusion holds for small depths of recursions that are comparable to human performances. The methods of SRN training and analysis of its state space introduced in this paper are of a general nature and can be used for investigation of processing of any other symbol time series by means of SRN.

Keywords: Recurrent neural networks; Linguistic structures; Next-symbol prediction; State space analysis; Language processing; Markovian architectural bias; Neural prediction machines; Variable length Markov model

1. Introduction

A significant scientific effort is devoted to connectionist processing of complex language structures. One of the main driving forces behind such studies has been formulating the models of human performance in processing linguistic patterns of various complexity (Christiansen & Chater, 1999; Elman, 1990; Hanson & Negishi, 2002; Lawrence, Giles, & Fong, 2000). Also, the analysis of state space trajectories in recurrent neural networks (RNNs) has provided new insights into the types of processes which may account for the ability of learning devices to acquire and represent language, without appealing to traditional linguistic concepts (Parfitt, 1997; Servan-Schreiber, Cleeremans, & McClelland, 1989). To gain an insight into what RNNs have learned, trained networks are used as generators by transforming their outputs to "probabilities" of possible sentence continuations (by normalizing the sum of outputs to 1). One of these possible continuations is then chosen stochastically and can be fed back as the next input to the network to generate the next symbol (Christiansen & Chater, 1999; Elman, 1990).

Common algorithms usually used for the RNNs training are based on gradient minimization of error. One such algorithm, backpropagation through time (BPTT) (Werbos, 1990; Williams & Zipser, 1995), consists of unfolding a recurrent network in time and applying the well-known backpropagation algorithm directly. Another gradient based

* Corresponding author. Tel.: +421 2 602 91 219, +421 2 654 29 502; fax: +421 2 654 20 587.
E-mail addresses: cernansky@fiit.stuba.sk (M. Čerňanský), makula@fiit.stuba.sk (M. Makula), benus@ii.fmph.uniba.sk (L. Beňušková).

0893-6080/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2006.01.020

approach, where estimates of derivatives needed for evaluating error gradient are calculated in every time step, is called the real time recurrent learning algorithm (RTRL) (Williams & Zipser, 1989). In Čerňanský and Beňušková (2003) we compared two methods of training SRN, namely the RTRL and extended Kalman filter algorithms. We have shown that an approach of extended Kalman filter, although computationally more expensive, yields higher robustness with respect to the hyperparameter values, and the resulting next-symbol prediction is better than with RTRL. This result is consistent with the findings of Pérez-Ortiz, Gers, Eck, and Schmidhüber (2003) with training the LSTM (long short-term memory) network on artificial grammar sequences. Thus, in the present paper we use the extended Kalman filter algorithm for training the SRN.

Yet whichever the training approach, a note of caution should be issued: when training RNNs to process any kind of symbol time series (including language structures), activations of recurrent units display a considerable amount of structural differentiation even *prior to learning* (Hammer & Tiňo, 2003; Tiňo & Hammer, 2003; Kolen, 1994a, 1994b). According to Christiansen and Chater (1999) we refer to this phenomenon as the *architectural bias of RNNs*. In this paper we perform an investigation of this phenomenon using artificial languages constructed according to Christiansen and Chater (1999). These authors trained SRN on three artificial languages with recursive structures reflecting those found in human language. They showed close correspondence between the empirically observed limited ability of humans to process recursive structures and results obtained by experimenting with SRN.

We study the organization of the SRN internal representations before and after learning by discretizing its state space through clustering recurrent activations. The intuition behind clustering is this: activation patterns across recurrent units can be thought of as an n-dimensional spatial codes of history of inputs seen so far. For example, when trained on symbol 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 (Tiňo, 1999). We use this state space organization and the training set to extract neural predictive models that we call neural prediction machines. Based on comparison with the performance of Markov prediction models of finite and variable length memories (MM and VLMM, respectively), we show that neural prediction machines of *non-trained* RNNs initialized with small random weights are closely related to VLMMs (Ron, Singer, & Tishby, 1996). Previously, using the same training sets (Tiňo, Čerňanský, & Beňušková, 2002), we showed that neural prediction machines extracted from non-trained RNNs are closely related to the so-called fractal prediction machines (FPMs). FPMs introduced in Tiňo and Dorffner (2001) are constructed from RNNs treated as an affine iterative function system. Later (Tiňo, Čerňanský, & Beňušková, 2004), we used a chaotic symbolic time sequence and one example of context-free artificial language to demonstrate that the architectural bias of RNNs has the nature of VLMM.

We would like to answer the question how the RNN state space reorganizes during training on recursive linguistic structures and what is the nature of the resulting internal representation of these recursive structures. We introduce a simple visualization approach to investigate the organization of clusters of RNN recurrent activations. Based on comparison with the performance of VLMMs, we show that neural prediction machines of trained SRN are again closely related to VLMMs, but this time based on word categories and their grammatical subcategories rather than individual words.

2. Architecture of SRN

The architecture of the SRN is shown in Fig. 1 (Elman, 1990). Units of the input layer I and the recurrent layer R and the output layer O are fully connected through the first order weights $\mathbf{W}^{RI}$ and $\mathbf{W}^{OR}$, respectively, as in the feedforward multilayer perceptron (MLP). Time delay connections feed back current activities of recurrent units $\mathbf{R}^{(t)}$ to the context layer so that $\mathbf{C}^{(t)} = \mathbf{R}^{(t-1)}$. Hence, every recurrent unit is fed by activities of all recurrent units from previous time step through recurrent weights $\mathbf{W}^{RC}$. Recurrent units' activities from previous time step can be viewed as an extension of input to the recurrent layer. They represent the memory of the network, since they hold contextual information from previous time steps.

Given input pattern in time t, $\mathbf{I}^{(t)} = (I_1^{(t)}, \ldots, I_j^{(t)}, \ldots, I_{|I|}^{(t)})$, and recurrent activities $\mathbf{R}^{(t)} = (R_1^{(t)}, \ldots, R_j^{(t)}, \ldots, R_{|R|}^{(t)})$, the recurrent unit's net input $\tilde{R}_i^{(t)}$ and output activity $R_i^{(t)}$ are calculated as

$$\tilde{R}_i^{(t)} = \sum_j W_{ij}^{RI} I_j^{(t)} + \sum_j W_{ij}^{RC} R_j^{(t-1)}, \tag{1}$$

$$R_i^{(t)} = f(\tilde{R}_i^{(t)}). \tag{2}$$

Output unit k calculates its net input $\tilde{O}_i^{(t)}$ and output activity $O_i^{(t)}$ as:

$$\tilde{O}_i^{(t)} = \sum_j W_{ij}^{OR} R_j^{(t)}, \tag{3}$$

$$O_i^{(t)} = f(\tilde{O}_i^{(t)}), \tag{4}$$

where $|I|$, $|R|$ and $|O|$ are the number of input, hidden and output units, respectively, and f stands for the activation function. In this work we used the logistic sigmoid function $f(x) = \left(1 + \mathrm{e}^{-x}\right)^{-1}$.

In our experiments, symbols from input alphabet are encoded using one-hot encoding scheme: all input unit's or target activities are fixed to be inactive but for one unit corresponding to the input or target symbol. Hence, the number of input and output units is equal to the number of symbols in the alphabet of a given data set. The training sequence was presented at a rate of one symbol per clock tick. SRN was trained to predict the next item in a sequence given the previous context. At the beginning of each training epoch the network is re-set with the same initial state $\mathbf{C}^{(1)}$. The initial state is randomly chosen prior to the training session.

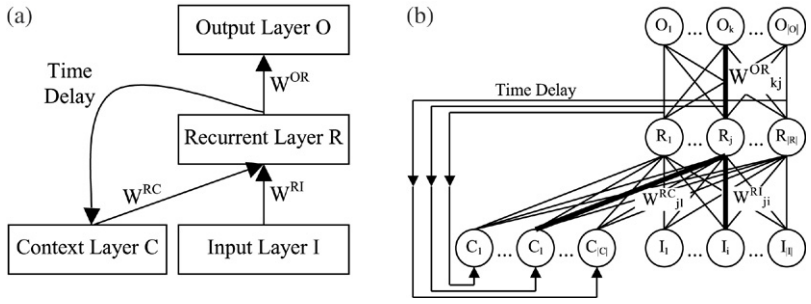

Fig. 1. Simplified (a) and more detailed (b) representation of Elman's SRN.

3. Extended Kalman filter algorithm for SRN

The Kalman filter (KF) is a set of equations describing a recursive solution of the linear discrete-data filtering problem (Welch & Bishop, 1995). 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 the Taylor expansion. This approach is called an extended Kalman filter (EKF). Training of Elman SRN, and generally any other multilayer perceptron network (recurrent or not), can be regarded as an optimal filtering (Feldkamp, Prokhorov, Eagen, & Yuan, 1998). The set of EKF equations for the network training can be formulated as follows:

$$\mathbf{K}(t) = \mathbf{P}(t-1)\mathbf{H}^T(t)[\mathbf{H}(t)\mathbf{P}(t-1)\mathbf{H}^T(t) + \mathbf{N}(t)]^{-1}, \quad (5)$$

$$\mathbf{P}(t) = \mathbf{P}(t-1) - \mathbf{K}(t)\mathbf{H}(t)\mathbf{P}(t-1) + \mathbf{Q}(t), \quad (6)$$

$$\mathbf{W}(t) = \mathbf{W}(t-1) + \mathbf{K}(t)[\mathbf{D}(t) - \mathbf{O}(t)]. \quad (7)$$

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 2, $\mathbf{W}$ is a vector of all weights (concatenated from matrices $\mathbf{W}^{RI}$, $\mathbf{W}^{RC}$, $\mathbf{W}^{OR}$) of the length n_w. $\mathbf{H}$ is the Jacobian matrix, $n_o \times n_w$, calculated in every time step and containing in rows the derivatives of corresponding output activity with respect to all weights. These derivatives can be calculated by routines used to calculate derivatives either for RTRL (Williams & Zipser, 1989) or for BPTT (Werbos, 1990). $\mathbf{P}$ is the $n_w \times n_w$ error covariance matrix, it holds error covariances corresponding to each pair of network weights. Ignoring error covariances of 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 (Eqs. (5)–(7)) – can lead towards better solutions (Feldkamp et al., 1998). The $n_w \times n_o$ matrix $\mathbf{K}$, called the Kalman gain, is used in updating the weights $\mathbf{W}$ according to the difference between the desired output vector $\mathbf{D}$ and actual network output $\mathbf{O}$. The $n_o \times n_o$ matrix $\mathbf{N}$ stands for the measurement noise covariance matrix. Similarly to the learning rate in RTRL or BPTT, it controls the training speed of EKF. Higher values represent a higher amount of uncertainty attributed to the difference $\mathbf{D}(t) - \mathbf{O}(t)$ thus leading to slower training. The $n_w \times n_w$ matrix $\mathbf{Q}$ stands for the process noise covariance matrix. Nonzero process noise improves convergence of the filter.

To initialize the EKF training, diagonal elements p_1 of the state error covariance matrix $\mathbf{P}$ were set to values of $p_1 = 10^3$. Off-diagonal elements p_2 were set to values of $p_2 = 10^2$. Diagonal elements of the measurement noise covariance matrix $\mathbf{N}$ were set to $n = 10^2$ and diagonal elements of the output noise covariance matrix $\mathbf{Q}$ were set to $q = 10^{-4}$ according to Feldkamp et al. (1998).

In the described version of an EKF-based training, the EKF estimated state is a concatenation of network's weights. Network training can be formulated as a dual estimation problem where both network weights and activities are to be estimated. There are several approaches to dual estimation (see for instance Wan & Nelson, 2000). Other variations of EKF can be based on training on multiple input streams (multi-stream EKF, Feldkamp et al., 1998).

4. Neural prediction machines

The main aim of this work is to analyze the state space of the RNN trained on linguistic data sets inspired by Christiansen and Chater (1999). Techniques frequently used in RNN state space analysis are clusterization of the recurrent unit's activities and attributing meaningful information to the extracted clusters.

When training the RNN on a sequence of symbols representing the words generated by simple regular grammars, clusters in the state space correspond to the states of finite state machine accepting words from given language. Thus, trained RNN state reflects the information of possible continuation of symbol sequence.

But also the state space of an untrained RNN initialized by small weights can reflect meaningful organization, as shown and explained in Hammer and Tiňo (2003), Tiňo and Hammer (2003), Tiňo et al. (2004). Recurrent units' activities are grouped in clusters that correspond to the symbols presented to the network. The state of the network (recurrent unit activities) is mostly determined by the most recent symbol presented to the network, then by the second last presented symbol, etc. Clusters in the state space correspond to the frequent sequences of symbols from the symbol dataset presented to the network and can be used to create prediction machines.

Neural prediction machines (NPM) are similar to MMs in a way they use predictive contexts. In fixed order MMs of order n, all n^A strings of length n created from alphabet $\mathcal{A} = \{1, 2, \ldots, A\}$ are predictive contexts. VLMMs use more sophisticated methods for creating contexts: only relevant

contexts (substrings of variable length) are kept by the model (see the section Markov Models). Contexts of NPM are clusters extracted from the RNN state space. The symbol just presented to the network corresponds to some cluster if and only if the RNN state belongs to this cluster. The next-symbol probabilities for all A symbols from alphabet $\mathcal{A}$ can be calculated and attributed for every cluster in the same way as in MMs.

The next-symbol probability of a symbol $a \in \mathcal{A}$ for a cluster c is calculated by relating the number of times (counter N_c^a) when symbol a follows a symbol driving the RNN to a given cluster c with the number of times the RNN state belongs to the cluster c ($\sum_{b\in\mathcal{A}} N_c^b$). The next-symbol probability distributions are smoothed by applying Laplace correction parameter γ, hence the probability of predicting symbol a when in cluster c can be calculated as:

$$P_{\mathrm{NPM}}(a|c) = \frac{P(a,c)}{P(c)} \doteq \frac{\gamma + N_c^a}{\gamma A + \sum_{b\in\mathcal{A}} N_c^b}. \tag{8}$$

We set the Laplace correction parameter γ to the value of A^{-1}. This can be seen as if we initialized counters to the value γ prior to counting any symbol occurrences thus attributing some probabilities also to symbols not present in the training sequence. The more important the context, the less smoothing is performed. On the other hand, the probability distribution of rare (statistically less convincing) context is smoothed more heavily.

5. Markov models

In MM of fixed order L, the next-symbol distribution in sequence of t symbols $a_1 a_2..a_t \in \mathcal{A}^t$ from alphabet $\mathcal{A} = \{1, 2, \ldots, A\}$ is written in the terms of the last L symbols

$$P(a_{t+1}|a_1..a_t) = P(a_{t+1}|a_{t-L}..a_t). \tag{9}$$

Building a Markov next-symbol prediction model over symbol sequence is straightforwardly based on empirical estimation of prediction probability such that

$$P_{\mathrm{MM}}(a|w) = \frac{P(wa)}{P(w)} \doteq \frac{\gamma + N_w^a}{\gamma A + \sum_{b\in\mathcal{A}} N_w^b}, \tag{10}$$

where $a \in \mathcal{A}$ is the next symbol and $w \in \mathcal{A}^L$ is called a prediction context. Probability distribution $P_{\mathrm{MM}}(a|w)$ is estimated by counting occurrences of sequence w followed by symbol a in the training sequence, i.e. N_w^a. Similar to NPMs, we perform smoothing by Laplace correction with $\gamma = 1/A$. However, for large memory lengths L, the estimation can be infeasible because the number of different prediction contexts $w \in \mathcal{A}^L$ increases exponentially by factor A. Using of large order MMs thus results in highly variable estimates.

On the other hand, VLMM use prediction contexts of variable length, and thus make use of deeper temporal information represented by longer prediction context only when it is really necessary. Creating VLMM is an iterative process, in which prediction contexts are systematically extended in the following manner. Let us suppose a candidate prediction context $w \in \mathcal{A}^m$ of length m with empirical next-symbol probability $P(a|w)$. If for a symbol $u \in \mathcal{A}$ the probability $P(a|uw)$ defined as

$$P(a|uw) = \frac{P(uwa)}{P(uw)}, \tag{11}$$

differs significantly from $P(a|w)$, then extending the prediction context w by symbol $u \in \mathcal{A}$ improves the next-symbol prediction. A typical decision criterion used for extending the candidate prediction context is the Kullback–Leibler divergence between the next-symbol distributions $P(a|w)$ and $P(a|uw)$ expressed through parameter ϵ_1. Once appropriate prediction contexts are chosen according to the foregoing formalism, the next-symbol probabilities are estimated similarly to MM, i.e.:

$$P_{\mathrm{VLMM}}(a|w) = \frac{P(wa)}{P(w)} \doteq \frac{\gamma + N_w^a}{\gamma A + \sum_{b\in\mathcal{A}} N_w^b}, \tag{12}$$

where Laplace correction parameter γ is equal to $1/A$. For practical reasons the size of VLMM is usually limited by maximal length of prediction context $|w| < L$ or the minimal probability $P(w) > \epsilon_2$.

6. Description of data sets

We performed experiments on three artificial languages constructed according to Christiansen and Chater (1999). Each language involves one of the three complex recursions taken from Chomsky (1957), interleaved with right-branching recursions (RBR). Language symbols belong to four categories: n, v, N, V, where n stands for a singular noun, N for a plural noun, v for a singular verb and V for a plural verb. Let e be the empty string. The four complex recursions are:

(1) Center-embedding (mirror) recursion (CER), where the strings exhibit mirror symmetry around the midpoint. Example: "the boy girls like runs". CER can be defined by the following production rule: $X \rightarrow NXV|nXv|e$. Starting with X we can generate the strings like $e, \ldots, nNVv, NnnNVvvV$, etc. The simplest way to process CER sentences is to develop a last-in-first-out memory or stack to store agreement information. CER strings form context-free language that can be processed by push-down automata.

(2) Identity (cross-dependency) recursion (IR), where the strings consist of the concatenation of the sequence of nouns and the sequence of corresponding verbs. Example: "the boy girls runs like". IR can be defined by the following production rule: $X \rightarrow NnX|VvX|e$. Starting with X we can generate the strings like $e, \ldots, NnVv, nNvV, \ldots$, etc. The most straightforward way to process IR sentences is to develop a first-in-first-out memory or queue. This language belongs to the class of context-sensitive languages; linearly-bounded automata can process such languages.

(3) Counting recursion (CR), where in order to parse such strings from left to right it is necessary to count the number of N's and note whether it equals the number of V's, while ignoring singulars and plurals. IR can be defined by the following production rule: $X \rightarrow NXV|NXV|e$. Starting with X we can generate the strings

Table 1
Distribution of string lengths with examples from the CERandRBR data set

Length	Percent (%)	RBR examples	CER examples
2	15.0	nv, NV	nv, NV
4	27.5	$NVnv$	$nNVv$
6	7.0	$NVNVnv$	$nNNVVv$
8	0.5	$nvNVNVnv$	$nnNNVVvv$

Examples are given in categories. In actual data sets, every category was replaced by one of four "true" words.

like $e, NV, NNVV, NNNVVV, \ldots$, etc. Correct performance of the processing device can be achieved simply by counting the number of occurring nouns, and then predicting the same number of verbs, that is to develop a counter. Similarly to the CER, this is a context-free language.

(4) Right-branching recursions (RBR), where the strings can be generated by a simple generative process described by the production rule $X \rightarrow NVX|nvX|e$, to obtain constructions like $NVnv$. Example: "girls like the boy that runs". Even processing unbounded RBR structures does not involve unbounded memory because each noun is immediately followed by a verb which agrees with it. Simple finite-state automaton can be constructed to process this language belonging to the regular languages class.

Thus, our three benchmark recursive languages were: CERandRBR, CRandRBR, IRandRBR. Each of the three languages involves two kinds of recursive structures: one is a complex recursion that cannot be generated by a finite state machine, and the other one is RBR that can be produced by a simple iterative process carried out by a finite state machine. In such a way, each language contains a combination of context-sensitive or context-free and regular language features, like the natural languages do.

Each language had a sixteen word vocabulary with four singular nouns, four plural nouns, four singular verbs and four plural verbs. Words representing a given category were chosen with equal probability. They were encoded by "one-hot" encoding. Thus, the SRN had 17 input and output units, with one unit devoted to the end of a string marker. The training sequence consisted of 5000 strings of variable length, and the test set of 500 novel strings that were not contained in the training set. Both sets contained 50% of RBR strings interleaved with 50% of complex recursion strings in a random way. The mean sentence length was approximately 4.7 words. The distribution of string lengths of both types of recursions, i.e. complex and RBR, was the same for all three languages. It is shown in Table 1 together with examples from CERandRBR.

The aim of the methods described in the following section is to predict the next symbol having seen all previous symbols. As already mentioned, we used a 17 symbols alphabet (16 symbols represent words and one symbol is the end-of-string symbol). Since all data sets were generated stochastically with known probability distribution (see Table 1), at every point of the generative process the distribution of the next-symbol probabilities is known and the entropies of symbol sequences can be calculated. The entropy of the data sets was estimated by Monte Carlo method.

Although languages have also context-sensitive features, the length of word strings is constrained to 8 (see Table 1). Hence, languages were generated by grammars with only regular and some context-free production rules enhanced with probabilities of applying the corresponding rule. While generating a string from language, all sentence forms having the same prefix of terminals are maintained together with the probabilities of generating them. Probabilities of all symbols that can be generated after an already generated prefix can be evaluated easily and are used to make entropy estimation more accurate. One symbol is then chosen with calculated probability, all maintained sentence forms capable of generating the given symbol are enhanced with this symbol and their generating probabilities are updated. Sentence forms that are unable to generate given symbols are thrown away. By this iterative Monte Carlo method normalized entropies were estimated as 0.595 for the CERandRBR data set, 0.647 for CRandRBR and 0.595 for IRandRBR.

7. Training and evaluation

An EKF was used to train Elman's SRN as described in Section 3. We trained a SRN to predict the next symbol in a sequence given previous context. That is, the SRN gets a word as an input at time t and has to predict the next word at time $t + 1$. During the test phase, test symbol series were presented to the network and the predictive performance evaluated while the weights remained frozen. The SRN weights and initial state (starting activation of the context layer) were randomly initialized from intervals $(-0.5, 0.5)$ and $(0.0, 1.0)$, respectively, for each simulation run. The SRN state (i.e. context) units were reset before training or testing epoch to the initial state and 30 dummy steps were performed (no error calculation or weight updates were done during dummy steps). Recurrent units' count varied from 4 to 32 but the performance was not getting better for more than 16 hidden units. The number of input and output units corresponded to the symbol count in an alphabet of a given data set. Unipolar 0–1 sigmoid activation function was used. Diagonal elements of the state error covariance matrix $\mathbf{P}$ in the EKF training were initialized to the value of p_1 and off-diagonal elements were set to p_2 with $p_1 = 10^3$ and $p_2 = 10^2$. The diagonal elements of the measurement noise covariance matrix $\mathbf{N}$ were set to n, where $n = 10^2$ and diagonal elements of the output noise covariance matrix $\mathbf{Q}$ were set to q, where $q = 10^{-4}$ (Feldkamp et al., 1998).

Before and after training, NPMs were constructed based on recurrent state activations. K-means clusterization was used with center counts varying from 5 to 300.

Multiple VLMMs with L varying from 1 to 7 and with different ϵ parameters were built and the best results for a given number of prediction contexts were chosen.

Predictive performance of all methods (RNNs output, NPMs and VLMMs) was evaluated by means of a *normalized negative log-likelihood* (NNL) calculated over the test symbol sequence

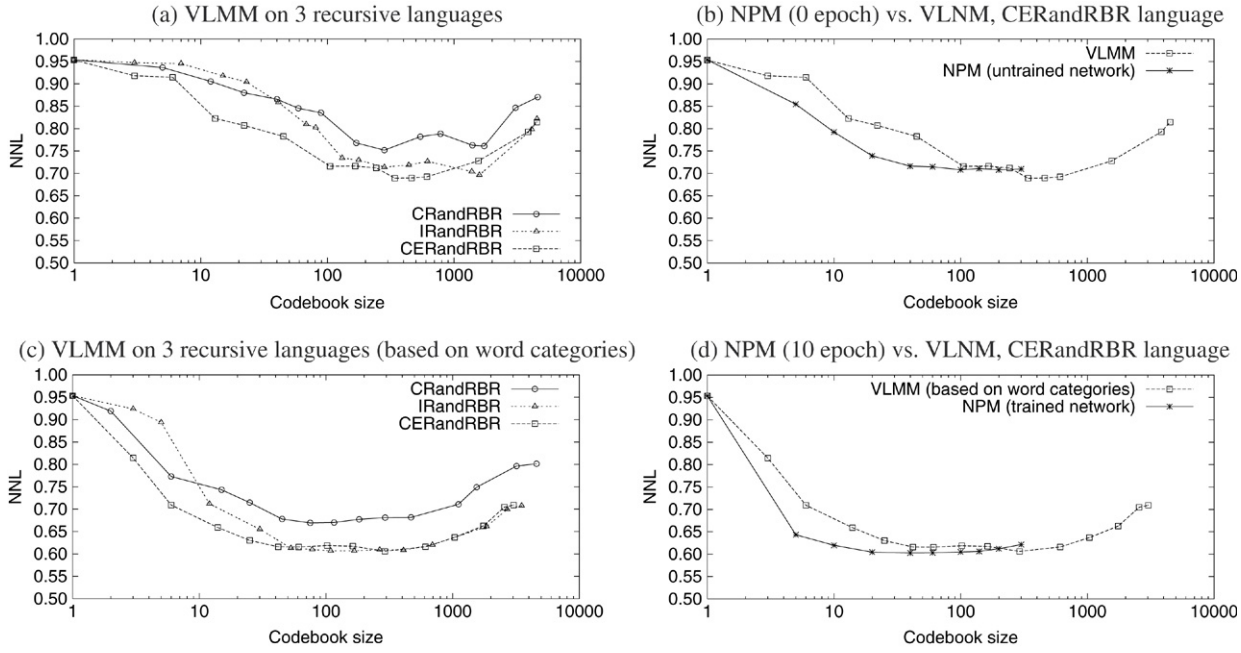

Fig. 2. NNL achieved on Christiansen and Chater recursion data sets by (a) VLMM based on individual words (i.e. symbols), (b) NPM of the naive SRN and VLMM based on individual words for the CERandRBR language, (c) VLMMs based on word categories and their grammatical subcategories, (d) NPM of the trained SRN and VLMM based on word categories and their grammatical subcategories for the CERandRBR language.

from time step $t = 1$ to T as

$$\mathrm{NNL} = -\frac{1}{T}\sum_{t=1}^{T} \log_A p_t(a_t), \tag{13}$$

where the base of the logarithm A is the number of symbols in the alphabet $\mathcal{A} = \{1, 2, \ldots, A\}$, and the $p_t(a_t)$ is the probability of predicting symbol a_t in the time step t. Value $p_t(a_t)$ is equal either to P_{RNN} (Eq. (14)), P_{NPM} (Eq. (8)) or P_{VLMM} (Eq. (12)), depending on whether we evaluate the predictive performance of the RNN output, NPM or VLMM, respectively. P_{RNN} was calculated by normalizing output neurons activations as:

$$P_{\mathrm{RNN}}(a|R^{(t)}) = \frac{O_a^{(t)}}{\sum_{b\in\mathcal{A}} O_b^{(t)}}, \quad a \in \mathcal{A}. \tag{14}$$

8. Results

In Fig. 2 we show NNL achieved by VLMMs based on single words (a) and grammatical subcategories of word categories (c). These results are for the CERandRBR language compared with NPM results (b), (d). NNL results based on the RNN output and corresponding standard deviations are shown in Fig. 3(a) together with NPMs (b)–(d). Standard deviations in the 3-D plots are not shown but generally they are smaller than 5% of the mean value.

Compare the NPM performances for 0 training epochs in Fig. 3(b)–(d) with the VLMM performance in Fig. 2(a). These results for CERandRBR language are plotted in Fig. 2(b). NPMs extracted from *untrained* networks achieved NNL comparable with NNL given by VLMM. This phenomenon, called Markovian architectural bias of RNNs, has been thoroughly investigated and explained elsewhere (Hammer & Tiňo, 2003; Tiňo & Hammer, 2003; Tiňo et al., 2004).

However, comparing the results of trained SRN with the results obtained with VLMM reveals that SRN can acquire some amount of information during training. Not only the predictive performance of RNN outputs improves (see Fig. 3(a)), but also the predictive results of NPMs extracted from the recurrent layer improve consistently (Fig. 3(b)–(d)). For instance, the final NNL performance of NPM trained on CERandRBR (i.e. 0.60) decreases almost down to an ideal NNL of 0.59. Likewise with the other languages (NNL of 0.59 for CRandRBR and 0.69 for IRandRBR).

It means that the training has led to a better reorganization of the SRN state space. The dynamics of RNNs initialized with small weights is trivial. With a fixed input, it is completely dominated by a single attractive fixed point in the recurrent activation space (the RNN state space). Different input codes of symbols $1, 2, \ldots, A$, from $\mathcal{A}$ correspond to A different attractive fixed points in the RNN state space. Distances between these point attractors are random (see Fig. 4(a)). During training, the point attractors for symbols (words) have clustered together according to the word categories and their grammatical subcategories (i.e. singular nouns, singular verbs, plural nouns and plural verbs) as can be seen in Fig. 4(b). The same reorganization happens in the state space of SRNs trained on IRandRBR and CRandRBR. Thus, in the trained dynamics, we still have fixed point attractors, each for each input symbol, but their positions in the state space are no longer random. Instead these positions depend on the word category and its grammatical subcategory, i.e. one place (cluster) is occupied

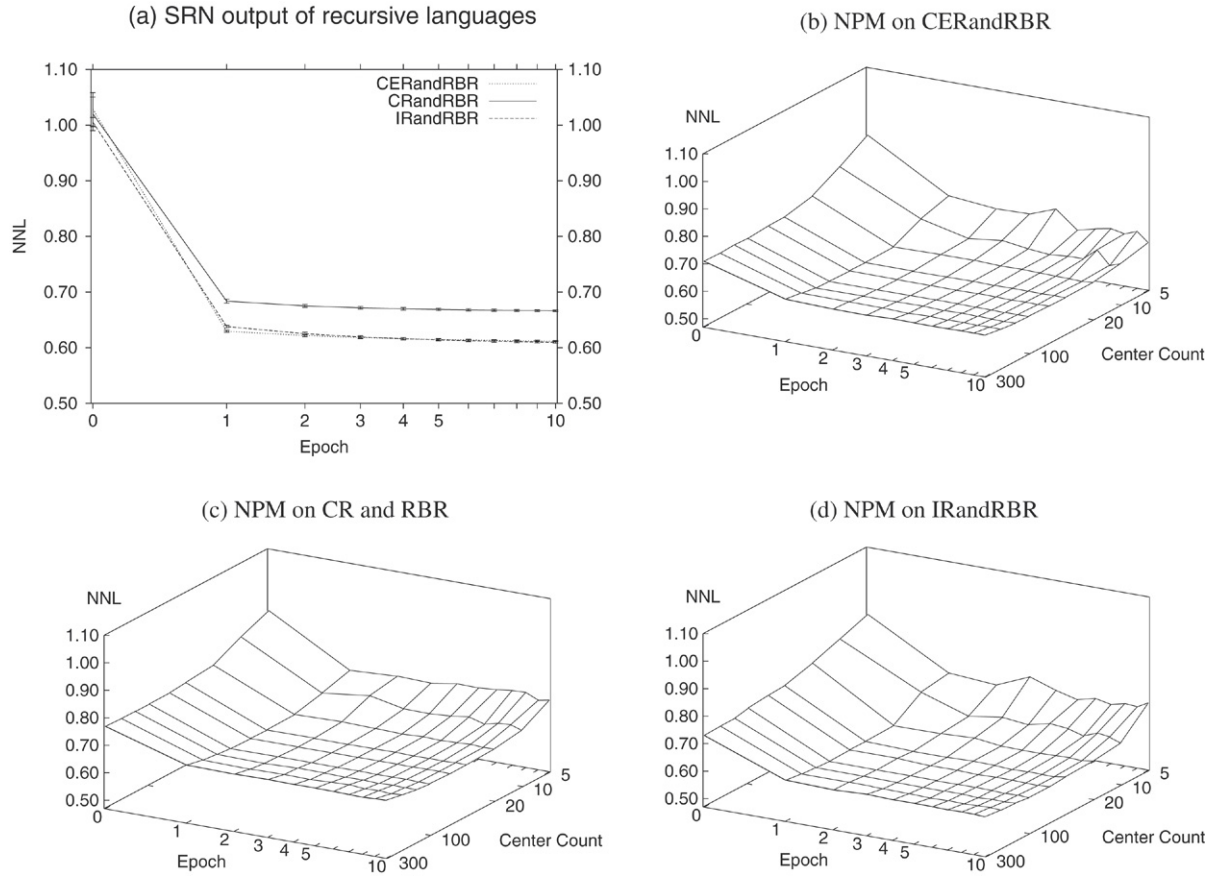

Fig. 3. NNL achieved on Christiansen and Chater recursion data sets before and during training by (a) SRN output, (b) NPM on CERandRBR, (c) NPM on CRandRBR, and (d) NPM on IRandRBR.

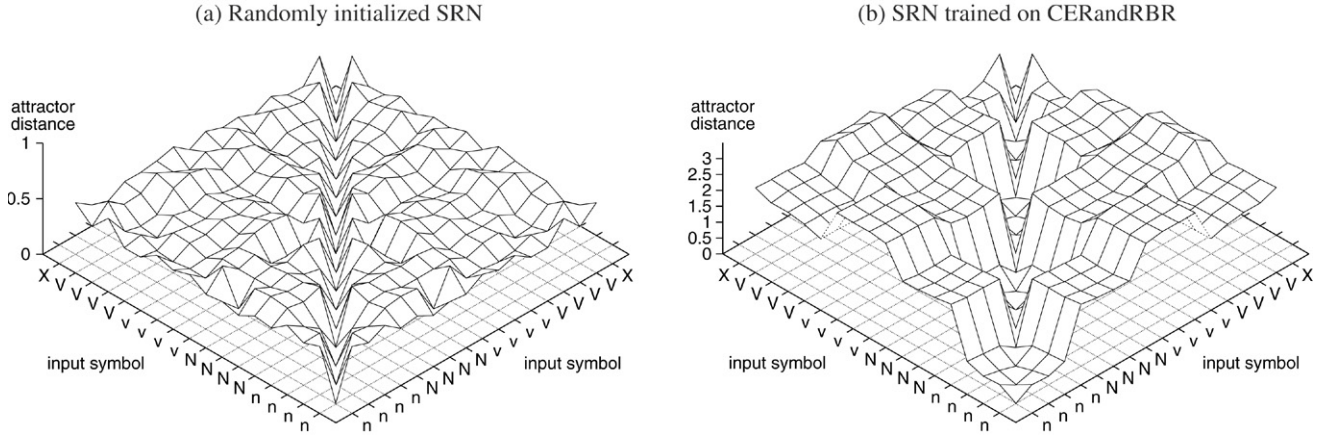

Fig. 4. Euclidean distances among fixed point attractors belonging to 17 input symbols in the SRN state space (a) before training and (b) after training on CERandRBR. Note that during training, the symbols (words) have clustered according to the word categories and their grammatical subcategories (i.e. singular nouns, singular verbs, etc). The same reorganization emerges after training on IRandRBR and CRandRBR.

by attractors for all four singular nouns from the language vocabulary, another cluster is occupied by all four attractors for singular verbs, etc.

Such a reorganization according to word categories and their grammatical subcategories leads us to the proposal that a SRN after training still builds its predictions for language recursions upon Markovian principles, but this time based on word categories and their grammatical subcategories. Thus, we calculated NNLs for VLMMs based on word categories in singular and plural (four categories) for prediction of the next symbol, for all three languages, see figure Fig. 2(c). We compared these NNLs with final best NNLs achieved by NPMs after training. Results for the CERandRBR language are shown in Fig. 2(d) and they are the same for all three languages. By taking into account the fixed point attractor state space dynamics, we can conclude that for achieving ideal possible

predictions on these recursive languages with a small depth of recursion, it is sufficient to work with a Markovian approach with variable length of memory based on word categories and their grammatical subcategories.

9. Discussion

RNNs initialized with small weights are biased towards the class of Markov models known as variable memory length Markov models (Tiňo et al., 2004; Hammer & Tiňo, 2003; Tiňo & Hammer, 2003). We constructed predictive models, called neural prediction machines, that share the state space dynamics of RNNs, but are not dependent on the RNN output map. Generally, an NPM seeks state clusters in a network state space and associates them with conditional probabilities for the next-symbol prediction. Each quantization center is identified with the prediction context for the next-symbol prediction. Using NPMs we were able to test the usefulness of state space representations in RNNs for building probabilistic models of recursive linguistic data. Experiments on recursion data of Christiansen and Chater confirmed our Markovian architectural bias hypothesis.

When RNNs with sigmoid activation functions are initialized with small weights, the clusters of recurrent activations emerging *before training* are 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. The basic property of classical fixed order Markov models of the order m is that distribution of probabilities for each next symbol in the sequence depends on the immediate predecessors of that symbol. Thus, only sufficient number of symbols (i.e. m) is relevant for the next-symbol prediction. VLMM approach does not use fixed memory depth, but instead it uses deeper memory for suffixes only when it is really needed. 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.

However, comparing the results of trained SRN with the results obtained with VLMM reveals that SRN and NPMs can acquire some amount of information during training. Dynamics of randomly initialized RNN with small weights is quite simple. It contains a set of fixed point attractors, each for one input symbol. By visualization of distances between fixed point attractors before and after learning we have revealed that in the trained dynamics, we still have fixed point attractors, each for each input symbol, but their positions in the state space are no longer random. Instead these positions depend on the word category and its grammatical subcategory, i.e. one place (cluster) is occupied by attractors for all four singular nouns from the language vocabulary, another cluster is occupied by all four attractors for singular verbs, etc.

Such a reorganization according to the word categories and their grammatical subcategories has led us to the proposal that a SRN after training still builds its predictions for language recursions upon Markovian principles, but this time based on the word categories in singular and plural. Thus, we calculated NNLs for VLMMs based on word categories in singular and plural, for all three languages. By comparing these NNLs with final best NNLs achieved by NPMs, and taking into account the fixed point attractor state space dynamics, we can conclude that for achieving ideal possible predictions on these recursive languages with a small depth of recursion, it is sufficient to work with the Markovian approach with variable length of memory based on the word categories and their grammatical subcategories, like plural and singular.

If we want to understand the strategy SRN and RNNs in general use for predictive tasks, our approach demonstrates the usefulness of building NPMs using clustering the state space and comparing the performance with suitable statistical models. This can aid the choice of an appropriate artificial neural network architecture for a given task and show how much have RNNs actually learned during the training.

Acknowledgments

This work was supported by the APVT grant 20-030204 and the VEGA grant 1/2045/05. We would like to thank Dr. Peter Tiňo for useful discussions.

References

Čerňanský, M., & Beňušková, L. (2003). Simple recurrent network trained by RTRL and extended Kalman filter algorithms. *Neural Network World*, *13*(3), 223–234.

Chomsky, N. (1957). *Syntactic structures*. Mouton: The Hague.

Christiansen, M. H., & Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, *23*, 417–437.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Feldkamp, L., Prokhorov, D., Eagen, C., & Yuan, F. (1998). Enhanced multi-stream Kalman filter training for recurrent networks. In J. Suykens, & J. Vandewalle (Eds.), *Nonlinear modeling: Advanced black-box techniques* (pp. 29–53). Kluwer Academic Publishers.

Hammer, B., & Tiňo, P. (2003). Neural networks with small weights implement definite memory machines. *Neural Computation*, *15*(8), 1897–1929.

Hanson, S. J., & Negishi, M. (2002). On the emergence of rules in neural networks. *Neural Computation*, *14*(9), 2245–2268.

Kolen, J. F. (1994a). The origin of clusters in recurrent neural network state space. In *Proceedings of the 16th annual conference of the cognitive science society* (pp. 508–513). Hillsdale, NJ: Lawrence Erlbaum Associates.

Kolen, J. F. (1994b). Recurrent networks: state machines or iterated function systems? In M. C. Mozer, et al., (Eds.), *Proceedings of the 1993 connectionist models summer school* (pp. 203–210). Hillsdale, NJ: Lawrence Erlbaum Associates.

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

Parfitt, S. (1997). Aspects of anaphora resolution in artificial neural networks: Implications for nativism. Ph.D. thesis. London: Imperial College.

Pérez-Ortiz, J. A., Gers, F. A., Eck, D., & Schmidhüber, J. (2003). Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. *Neural Networks*, *16*(2), 241–250.

Ron, D., Singer, Y., & Tishby, N. (1996). The power of amnesia: learning probabilistic automata with variable memory length. *Machine Learning*, *25*, 117–149.

Servan-Schreiber, D., Cleeremans, A., & McClelland, J. L. (1989). Graded state machines: the representation of temporal contingencies in Simple Recurrent Networks. *Machine Learning*, *7*, 161–193.

Tiňo, P., & Dorffner, G. (2001). Predicting the future of discrete sequences from fractal representations of the past. *Machine Learning*, *45*(2), 187–217.

Tiňo, P., & Hammer, B. (2003). Architectural bias in recurrent neural networks — fractal analysis. *Neural Computation*, *15*(8), 1931–1957.

Tiňo, P., Čerňanský, M., & Beňušková, L. (2002). Markovian architectural bias of recurrent neural networks. In P. Sinčák, et al., (Eds.), *Intelligent technologies, theory and applications* (pp. 17–23). Amsterdam: IOS Press.

Tiňo, P., Čerňanský, M., & Beňušková, L. (2004). Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, *15*(1), 6–15.

Tiňo, P. (1999). Spatial representation of symbolic sequences through iterative function systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, *10*, 284–302.

Wan, E. A., & Nelson, A. T. (2000). Dual EKF methods. In S. Haykin (Ed.), *Kalman filtering and neural networks* (pp. 123–173). New York: Wiley.

Welch, G., & Bishop, G. (1995). An introduction to the Kalman filter, TR95-041. Department of Computer Science, University of North Carolina.

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

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*, 270–280.

Williams, R. J., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin, & D. E. Rumelhart (Eds.), *Back-propagation: Theory, architectures and applications* (pp. 433–486). Hillsdale, NJ: Lawrence Erlbaum Publishers.