Markovian Architectural Bias of Recurrent Neural Networks

Peter Tiňo[1,2], Michal Čerňanský[2], and Lubica Beňušková[3]

[1] Neural Computing Research Group, Aston University, Birmingham, B4 7ET, UK
p.tino@aston.ac.uk, http://www.ncrg.aston.ac.uk/~tinop
[2] Department of Computer Science and Engineering FEI, Slovak Technical University, Ilkovičova 3, 812 19 Bratislava 1, Slovakia
cernans@dcs.elf.stuba.sk, http://www.dcs.elf.stuba.sk/~cernans
[3] Institute of Informatics FMFI, Comenius University, Mlynská dolina, 842 48 Bratislava 4, Slovakia
benus@ii.fmph.uniba.sk, http://www.ii.fmph.uniba.sk/~benus

Abstract. Several studies in the cognitive science community (see [1–3]) reported that when training recurrent neural networks (RNNs) to process language structures, activations of recurrent units display a considerable amount of structural differentiation even *prior to learning*. Following [3], we refer to this phenomenon as the *architectural bias of RNNs*. In this paper, we show, that when initialized with small weights, RNNs produce recurrent activations that cluster according to a Markovian strategy: i.e. subsequences sharing a long common suffix are represented as points in a dense cluster. Prediction states that naturally arise in such networks loosely correspond to states of variable memory length Markov models (VLMM). After training RNNs, the main effort in related studies is concentrated on analyzing activation patterns in the recurrent layer. Given a (trained, or non-trained) RNN and a training stream, we test usefulness of recurrent representations by constructing predictive models, called *neural prediction machines* (NPM), that directly employ state-space dynamics of the network. Based on comparison with the so-called *fractal prediction machines* (FPMs), we demonstrate that NPMs of non-trained RNNs are closely related to VLMMs. Reports on learning and processing linguistic structures using RNNs that concentrate on state-space representations in trained networks should, prior to making any conclusion about the importance of detected recurrent representations, compare RNN results with those of NPMs extracted from untrained nets and/or Markovian models.

1 Introduction

There is a considerable amount of literature devoted to connectionist processing of complex language structures. One of the main driving forces behind such studies has been formulating models of human performance in processing linguistic patterns of various complexity (e.g. [3]). 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 [4, 5]. To gain an additional insight into what the networks have learned, trained networks were sometimes used as generators by transforming their outputs to "probabilities"[4] of possible sentence continuations. One of these possible continuations is then chosen stochastically and fed back as the next input to the network [3, 6].

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*. We refer to this phenomenon as the *architectural bias of RNNs* [3]. In this paper we provide an explanation of this phenomenon and issue an additional note of caution that the link between the quality of trained RNNs as language generators and the complexity of dynamical state-space scenarios inside the networks is less straightforward than might have been previously thought.

2 Neural Prediction Machines

We work with Elman-type 1st-order randomly-initialized recurrent networks trained using RTRL. The networks consist of 1 input layer, $\mathbf{I}^{(t)}$, and 1 output layer $\mathbf{O}^{(t)}$, each with A units corresponding to the symbols from $\mathcal{A} = \{1, 2, ..., A\}$ appearing in the training and test sequences. There are 2 hidden layers, $\mathbf{R}^{(t)}$ and $\mathbf{H}^{(t)}$, and 1 context layer, $\mathbf{C}^{(t)} = \mathbf{R}^{(t-1)}$, connected through a unit time delay to the first (recurrent) hidden layer (**Fig. 1**(f)). The second (non-recurrent) hidden layer, $\mathbf{H}^{(t)}$, was used to increase the flexibility of the maps between the hidden representations in the recurrent portion, $\mathbf{R}^{(t)}$, and the activations at the output layer, $\mathbf{O}^{(t)}$. A logistic sigmoid activation function was used, the learning rate and momentum were set to 0.05, and the training sequence was presented at the rate of one symbol per clock tick. The networks were trained to predict the next item in a sequence given the previous context [3, 7]. 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.

After training RNNs on linguistic data, the main effort is often concentrated on analyzing the resulting activation patterns in the recurrent layer $\mathbf{R}^{(t)}$. We make 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 $\mathbf{R}^{(t)} \rightarrow \mathbf{H}^{(t)} \rightarrow \mathbf{O}^{(t)}$, and instead, based on the network dynamics $[\mathbf{I}^{(t)}, \mathbf{C}^{(t)}] \rightarrow \mathbf{R}^{(t)}$, use their own predictive probabilities $P_{NPM}(a|\mathbf{R}^{(t)})$, $a \in \mathcal{A}$.

[4] By normalizing the sum of outputs to 1.

While the RNN output probabilities are determined, using the map $\mathbf{R}^{(t)} \rightarrow \mathbf{H}^{(t)} \rightarrow \mathbf{O}^{(t)}$, at the network output

$$P_{RNN}(a|\mathbf{R}^{(t)}) = \frac{O_a^{(t)}}{\sum_{b=1}^{A} O_b^{(t)}}, \tag{1}$$

calculation of the corresponding NPM predictive probabilities involves estimation of the relative frequencies of symbols, conditional on RNN state $\mathbf{R}^{(t)}$. Instead of the map $\mathbf{R}^{(t)} \rightarrow \mathbf{H}^{(t)} \rightarrow \mathbf{O}^{(t)} \rightarrow P_{RNN}(\cdot|\mathbf{R}^{(t)})$, we have a piece-wise constant map, $\mathbf{R}^{(t)} \rightarrow P_{NPM}(\cdot|\mathbf{R}^{(t)})$, estimated on the training sequence. Regions of constant probabilities are determined by vector quantizing recurrent activations that result from driving the network (weights are fixed) with the training sequence. In the following, we give a more rigorous description of the NPM construction.

1. Given a training sequence $S = s_1 s_2 ... s_N$ over an alphabet $\mathcal{A} = \{1, 2, ..., A\}$, first re-set the network to the initial state $\mathbf{C}^{(1)}$, and then, while driving the network with the sequence S, collect the recurrent layer activation vectors in the set $\Omega = \{\mathbf{R}^{(t)}|\ 1 \leq t \leq N\}$.
2. Run a vector quantizer with M codebook vectors $\mathbf{B}_1, ..., \mathbf{B}_M$, on the set Ω (we used K-means clustering). Vector quantization partitions the space of recurrent activations into M regions $V_1, ..., V_M$, that are Voronoi compartments of the codebook vectors $\mathbf{B}_1, ..., \mathbf{B}_M$,
$$V_i = \{\mathbf{R}|\ d(\mathbf{R}, \mathbf{B}_i) = \min_j d(\mathbf{R}, \mathbf{B}_j)\},$$
where $d(\cdot, \cdot)$ is the Euclidean distance. All points in V_i are allocated[5] to the codebook vector $\mathbf{B}_i$.
3. Re-set the network again with the initial state $\mathbf{C}^{(1)}$.
4. Set the [codebook-vector, next-symbol] counters $N(i, a)$, $i = 1, ..., M$, $a = 1, ..., A$, to zero.
5. Drive the network once more with the training sequence S.
For $1 \leq t \leq N$, if $\mathbf{R}^{(t)} \in V_i$, and the next symbol s_{t+1} is a, increment the counter $N(i, a)$ by one.
6. With each codebook vector (or, equivalently, quantization compartment) $V_1, ..., V_M$, associate the next symbol probabilities
$$P(a|\ V_i) = \frac{N(i, a)}{\sum_{b \in \mathcal{A}} N(i, b)}, \quad a \in \mathcal{A}. \tag{2}$$

Given a sequence $u_1, u_2, ..., u_L$ over the alphabet $\mathcal{A}$, the NPM makes a prediction about the next symbol u_{L+1} as follows:

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

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

2. Drive the network with the sequence $u_1, u_2, ..., u_L$, i.e. for $1 \leq t \leq L$, recursively compute the vectors $\mathbf{R}^{(t)}$.
3. The next symbol distribution given by NPM is
$P_{NPM}(u_{L+1}|\mathbf{R}^{(L)}) = P(u_{L+1}|\ V_i)$, provided $\mathbf{R}^{(L)} \in V_i$.

Note, that NPM is *not* a finite state predictor. The dynamics of NPM is completely the same as that of the associated RNN. 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 $P(\cdot|\mathbf{R}^{(t)})$ by a piece-wise constant map.

2.1 Fractal Prediction Machines

In [8] we introduced a special type of NPM, called *fractal prediction machine* (FPM). FPMs are constructed from RNNs with dynamics of an affine iterative function system(IFS). Assuming the network input at time t is $a \in \mathcal{A} = \{1, 2, ..., A\}$, the dynamics is defined by

$$\mathbf{R}^{(t)} = \frac{1}{2}(\mathbf{C}^{(t)} + \mathbf{T}_a), \tag{3}$$

where $\mathbf{T}_1, ..., \mathbf{T}_A$, $\mathbf{T}_a \neq \mathbf{T}_b$, for $a \neq b$, are symbol-related vertices of the hypercube $[0,1]^D$, and the dimension D is no less than $\log_2 A$. Note that such non-autonomous dynamics[6] is driven by A attractive fixed points $\mathbf{T}_1, ..., \mathbf{T}_A$, corresponding to symbols from the alphabet $\mathcal{A}$.

We showed that FPMs are closely related to variable memory length Markov models (VLMM) [9], which are Markov models of flexible memory depth, depending on the prediction context. Roughly speaking, VLMMs can utilize deep memory only when it is really needed, thus avoiding explosion in the number of free parameters of the fixed-order Markov models. The IFS dynamics, Eq. (3), driven by attractive fixed points associated with symbols from $\mathcal{A}$, transforms subsequences appearing in the training sequence into a set of points in Euclidean space, such that points corresponding to blocks sharing a long common suffix are mapped close to each other. Vector quantization on such a set partitions histories of seen symbols into several classes dominated by common suffixes. Quantization centers of FPMs play the role of predictive contexts in VLMMs. FPMs and VLMMs achieved comparable performances across a collection of symbolic sequences from a wide range of complexity and memory structures (see [8]).

[6] Strictly speaking, FPMs were in [8] constructed as finite-memory sources, where the IFS dynamics was driven only by the last L recently seen symbols. Typically, $L = 20$. However, the distance between recurrent activations resulting from 2 sequences that share a common suffix of length L is less than $2^{-L}\sqrt{D}$ (see [11]). So, for long memory depths L, the dynamics of the FPM introduced above is virtually the same as that of the FPM described in [8].

3 Processing recursive structures

In this section we train RNNs and construct NPMs on three data sets of the type used by Christiansen and Chater to asses connectionist modeling of recursion in human linguistic performance [3]. They trained simple recurrent network [7] on the next-symbol prediction and also assessed the trained networks as sequence generators.

Each language used in [3] involves one of three complex recursions taken from Chomsky [10], interleaved with right-branching recursions (RBR). The latter is generated by a simple iterative process to obtain constructions like: $P_N P_V S_N S_V$, where P stands for plural, S for singular, N for noun and V for verb category. Example: "girls like the boy that runs". The three complex recursions are:

(1) Counting recursion (CR): $\{\}, NV, NNVV, NNNVVV, ...$, while ignoring singulars and plurals.

(2) Center-embedding recursion (CER): $\{\}, ..., S_N P_N P_V S_V, P_N S_N S_V P_V, ...$ Example: "the boy girls like runs".

(3) Identity (cross-dependency) recursion (IR): $\{\}, ..., S_N P_N S_V P_V, P_N S_N P_V S_V, ...$ Example: "the boy girls runs like".

Thus, our three benchmark recursive languages were: CRandRBR, CERandRBR, IRandRBR. Each language had 16 word vocabulary with 4 words from each category, i.e. 4 singular nouns, 4 singular verbs, 4 plural nouns and 4 plural verbs. The RNN had 17 input and output units, where each unit represented one word or the end of sentence mark. There were 10 hidden and 10 recurrent neurons. The networks were trained in 10 training runs starting from different random weight initializations (from $[-0.5, 0.5]$). The training set of each language consisted of 5000 sentences and the test set of 500 novel sentences. One half of each set was comprised of RBR constructions and another half of appropriate complex recursions. Depths of embedding ranged from 0 to 3, with the following distributions: depth 0 – 15 %, depth 1 – 27.5 %, depth 2 – 7 %, depth 3 – 0.5 % (together 50 %). The mean sentence length was approximately 4.7 words.

In **Fig. 1**(a)–(e) we show the mean (across 10 training runs) normalized (per symbol) negative log likelihoods (NNL) achieved on the test set by FPMs, RNNs and the corresponding NPMs. In the 2-D plots, we also show the corresponding standard deviations. Standard deviations in the 3-D plots are not shown, but generally are less than 5 % of the mean value.

The effect of architectural bias is clearly visible. This is confirmed by comparing NPM performances (for 0 training epochs) in **Fig. 1**(c)–(e) with the FPM performance in **Fig. 1**(a), effectively implementing VLMMs [8]. About 40 codebook vectors in NPMs are sufficient to make full use of the associated RNNs dynamics. Moreover, NPMs corresponding to non-trained networks achieved NNL comparable with NNL given by RNNs after 10 epochs of training (**Fig. 1**(b)). What is the explanation for this, rather surprising, behavior? The dynamics of RNNs initialized with small weights is trivial. With a fixed input, it is completely dominated by a single attractive fixed point close to the center of the recurrent activation space (RNN state space) [12]. Different input codes of symbols 1,2,...,A, form $\mathcal{A}$ correspond to A different attractive fixed points in the RNN state space. Hence, when driving the network with an input sequence over $\mathcal{A}$, the network codes histories of seen symbols in its recurrent layer in the same Markovian manner as FPMs do (subsection **2.1**). NPMs of non-trained RNNs, initialized with small weights, are much like FPMs. Before the training, RNN outputs are driven by randomly initialized weights in the output mapping $R^{(t)} \to H^{(t)} \to O^{(t)}$ and it takes some time for the mapping to adjust to the statistics of the training sequence. Also, since initially the A attractive fixed points lie in a close neighborhood of the center of the state space, it may be difficult for the RNN to adjust the smooth output map to the dynamics that takes place in a rather tiny portion of the state space. But, the "useful" dynamics is already there, even prior to the training (!), and NPMs are able to make full use of it, since NPM predictive probabilities are computed as piece-wise constant maps based on (scale-free) vector quantization of RNN state space.

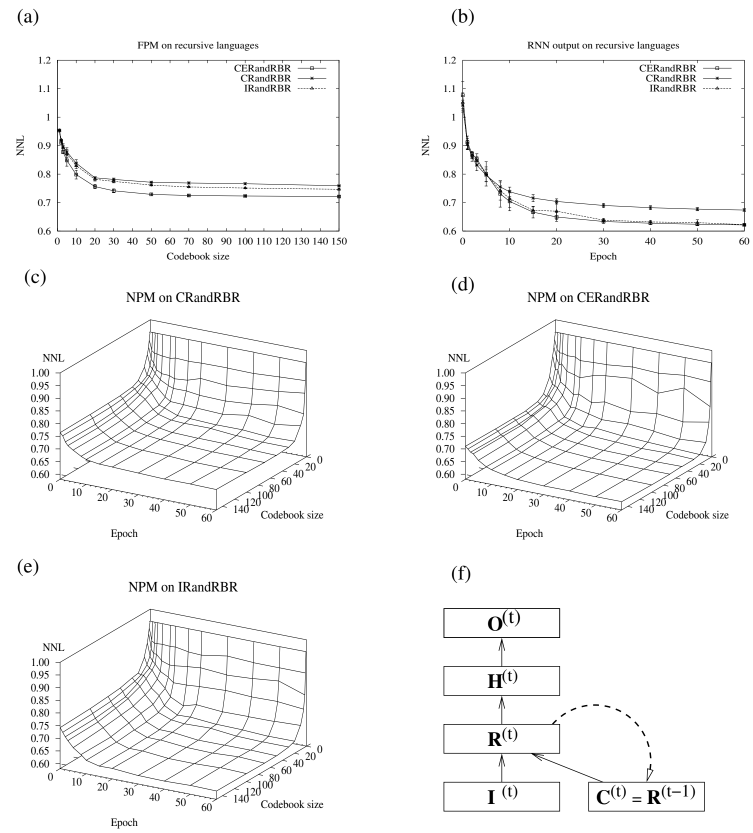

Fig. 1. Normalized negative log likelihoods (NNL) achieved on Christiansen and Chater recursion data sets by (a) FPMs, (b) output of RNNs and (c)–(e) NPMs. (f) RNN.

4 Conclusion

We have offered an explanation of the phenomenon known to the cognitive science community as the architectural bias in recurrent neural networks (RNNs). RNNs initialized with small weights are biased towards the class of Markov models known as variable memory length Markov models. We constructed predictive models, called neural prediction machines (NPM), that share the state-space dynamics of RNNs, but are not dependent on the RNN output map. Using NPMs we were able to test the usefulness of state space representations in RNNs for building probabilistic models of linguistic data. Experiments on recursion data of Christiansen and Chater confirmed our Markovian architectural bias hypothesis. Although not reported here, we obtained very similar results on a minimal-pair linguistic data set based on the University of Pennsylvania 'Brown' corpus.

Acknowledgements: Supported by VEGA 1/9046/02.

References

1. Kolen, J.F.: The origin of clusters in recurrent neural network state space. *Proc. 16th Annual Conf. Cognitive Sci. Soc.*, Lawrence Erlbaum Assoc., Hillsdale, NJ (1994) 508–513
2. Chater, N., Conkey P.: Finding linguistic structure with recurrent neural networks. *Proc. 14th Annual Meet. Cognitive Sci. Soc.*, Lawrence Erlbaum Assoc., Hillsdale (1992) 402–407
3. Christiansen, M.H., Chater, N.: Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, **23** (1999) 157–205
4. Parfitt, S.: Aspects of anaphora resolution in artificial neural networks: Implications for nativism. *PhD thesis*, Imperial College, London (1997)
5. Servan-Schreiber, D. *et al.*: Graded state machines: The representation of temporal contingencies in Simple Recurrent Networks. In *Advances in Neural Information Processing Systems.* Morgan-Kaufmann (1989) 643–652
6. Mozer, M., Soukup T.: Connectionist music composition based on melodic and stylistic constraints. In *Advances in Neural Information Processing Systems 3.* Morgan-Kaufmann (1991) 789–796
7. Elman, J.L.: Finding structure in time. *Cognitive Science* **14** (1990) 179–211
8. Tiňo, P., Dorffner, G.: Predicting the future of discrete sequences from fractal representations of the past. *Machine Learning* **45** (2001) 187–218
9. Ron, D., Singer, Y., Tishby, N.: The power of amnesia. *Machine Learning* **25** (1996) 138–153
10. Chomsky, N.: *Syntactic Structures.* Mouton, The Hague (1957)
11. Tiňo, P.: Spatial representation of symbolic sequences through iterative function systems. *IEEE Tran. Syst., Man, and Cyber. Part A: Systems and Humans* **10** (1999) 284–302
12. Tiňo, P., Horne, B.G., Giles, C.L.: Attractive periodic sets in discrete time recurrent networks (with emphasis on fixed point stability and bifurcations in two–neuron networks). *Neural Computation* **13** (2001) 1379–1414