ANALYSIS OF STATE SPACE OF RNNs TRAINED ON A CHAOTIC SYMBOLIC SEQUENCE

Matej Makula [*] Ľubica Beňušková [†]

Abstract

We investigate solutions provided by the finite-context predictive model called neural prediction machine (NPM) built on the recurrent layer of the two types of recurrent neural networks (RNNs). The first type is the first-order Elman's simple recurrent network (SRN) trained for the next symbol prediction by the technique of extended Kalman filter (EKF). The second type of RNN is an interesting unsupervised counterpart to the "classical" SRN, that is a recurrent version of the Bienenstock, Cooper, Munro (BCM) network that performs a kind of time-conditional projection pursuit. As an experimental data we chose a complex symbolic sequence with both long and short memory structures. We compared solutions achieved by both types of RNNs with Markov models to find out whether training can improve initial solutions reached by random network dynamics that can be interpreted as an iterated function system (IFS). Results of our simulations indicate that SRN trained by EKF achieves better next symbol prediction than its unsupervised counterpart. Recurrent BCM network can provide only Markovian solution that is not able to cover long memory structures in sequence and thus beat SRN.

1 Introduction

Recurrent neural networks (RNNs) represent a powerful computational model for time series processing. Training of such networks is performed by well-known gradient-based algorithms such as back propagation through time (BPTT) or real time recurrent learning (RTRL) [13]. In each training time step the correct output is presented to the network in order to calculate error signal and perform weight adjustments. Another approach is the technique of extended Kalman filter (EKF) that can be applied to training of RNNs [6, 12]. However, there is no direct correspondence of this kind of error backpropagation with learning in biological neural networks.

On the other hand unsupervised learning, where no error signals are backpropagated and training is based only on input statistics, represents biologically plausible type of weight adjustment. In the category of RNNs, one of the

[*]Department of Computer Science and Engineering, Slovak Technical University, Ilkovičova 3, 812 19 Bratislava, Slovakia, E–mail: *makula@dcs.elf.stuba.sk*

[†]Institute of Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University, Mlynská dolina, 842 48 Bratislava, Slovakia, E–mail: *benus@ii.fmph.uniba.sk*

unsupervised alternatives can be a recurrent Bienenstock, Cooper and Munro (BCM) network [1]. It was constructed from the feedforward BCM network by extending its input with a copy of neurons' activations from the previous time step. Training rule for the feedforward BCM network was introduced to explain self-organization in the developing visual cortex [3]. It was shown that the feedforward BCM neural network can perform projection pursuit on input data. Because a recurrent version of the BCM (RBCM) network has an extended input with neurons' activations from previous time step, it can perform a sort of time conditional projection pursuit.

In previous works, it has been shown that the 1st- and 2nd-order RBCM networks give comparable predictive performance on chaotic sequences as their counterpart RNNs trained by RTRL [8, 9]. In this paper we use an EKF based training for SRN. It was demonstrated that EKF is more effective in RNN training than RTRL (see the paper of M. Čerňanský and Ľ. Beňušková in this issue), thus it might improve solution performed by SRN. We focus on simple original analysis of solutions provided by SRN and RBCM and compare them with variable length Markov model (VLMM) [7] that was shown to be equivalent to solutions performed by randomly initialized RNNs [4, 10].

2 The First-Order Recurrent BCM Network

Recurrent BCM network has an extended input that contains a copy of recurrent neurons' activations from previous time step (Fig. 1a). In our experiments we use the first-order recurrent BCM network where the ith recurrent neuron activation for the next time step is calculated as

$$c_i(t+1) = \sigma\Big(\sum_j w_{ij}(t) \cdot d_j(t) + \sum_k m_{ik}(t) \cdot c_k(t) + \vartheta_i^C(t)\Big) \tag{1}$$

where $\sigma(x) = 1/(1+\exp(-\lambda \cdot x)$ is the sigmoid activation function with slope λ, ϑ_i^C is the bias, and w_{ij} and m_{ik} are feedforward and recurrent weights, respectively. The binary input vector $d(t) = \{d_j(t)\}_{j=1}^J$ at time t codes one input symbol. Adaption of weights is derived by means of the gradient descent minimization of the loss function $L(t) = \sum L_i(t)$, where

$$L_i(t) = -\left\{\frac{1}{3}c_i^3(t) - \frac{1}{4}E[c_i^2(t)]c_i^2(t)\right\} \tag{2}$$

while $E\left[c_i^2(t)\right] = \theta_M^i(t)$ is the moving synaptic modification threshold of the ith neuron, in our case calculated as

$$\theta_M^i(t) = \frac{1}{\tau}\int\limits_{-\infty}^{t} c_i^2(t') \cdot e^{-\frac{t-t'}{\tau}} dt' \tag{3}$$

where τ is the averaging period. Definition of the synaptic modification threshold as an average square value of the neuron's activity guarantees the correct behavior of the BCM network. The weight changes are determined by

$$\frac{dw_{ij}(t+1)}{dt} = -\eta\frac{\partial L(t+1)}{\partial w_{ij}(t)} = \eta\left[\sum_\alpha \phi_\alpha(t+1)\frac{\partial c_\alpha(t+1)}{\partial w_{ij}(t)}\right] \tag{4}$$

where η is the learning rate and $\phi_\alpha(t)$ is defined as $\phi_\alpha(t) = c_\alpha(t)\left[c_\alpha(t) - \theta_M^\alpha(t)\right]$ and represents the synaptic modification function for αth neuron. When the neuronal activity $0 < c_\alpha(t) < \theta_M^i(t)$, all neuron's active synapses weaken. However, when $\theta_M^i(t) < c_\alpha(t)$, all neuron's active synapses potentiate. Thus, each neuron is trying to divide its activities by its modification threshold to be greater for some inputs and smaller for the others. For the partial derivates of the neuronal activity according to the weights we get

$$\frac{dc_\alpha(t+1)}{dw_{ij}(t)} = \sigma'(t+1)\left[\delta_{i\alpha}^{Kron.} d_j(t) + \sum_\beta m_{\alpha\beta}(t)\frac{\partial c_\beta(t)}{\partial w_{ij}(t-1)}\right] \tag{5}$$

where $\delta_{i\alpha}^{Kron.}$ is the Kronecker's delta equal to 1 when $i = \alpha$, and otherwise equal to 0. Equations for m_{ik} analogical to 4 and 5 can be written simply by replacing w_{ij} with m_{ik}.

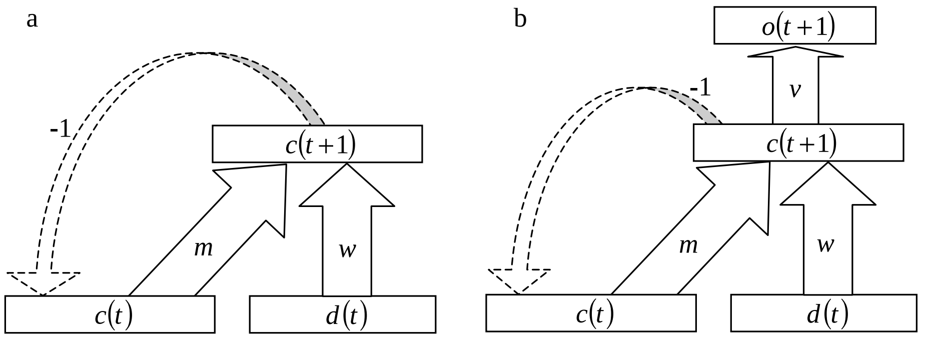

Figure 1: (a) Architecture of the first-order recurrent BCM network. (b) "Classical" first-order SRN. Wide arrows represent feed-forward connecti (i.e. full neuron interconnection between layers). Dashed arrows represent feedback connections (i.e. one to one time delayed copy of neuron activations).

3 "Classical" First-Order SRN

Layers of SRN are interconnected by connections represented by the weight matrices v, w, m (Fig. 1b). Output of the mth neuron is

$$o_m(t+1) = \sigma\left(\sum_i v_{mi}(t+1)\,c_i(t+1) + \vartheta_m^O(t)\right) \tag{6}$$

where ϑ_m^O is the bias. Recurrent neurons activities for the next time step are calculated as in eqn. 1, that is

$$c_i(t+1) = \sigma\left(\sum_j w_{ij}(t)\,d_j(t) + \sum_k m_{ik}(t)\,c_k(t) + \vartheta_i^C(t)\right) \tag{7}$$

where $d(t) = \{\, d_j(t)\,\}_{j=1}^{J}$ is the binary input vector and ϑ_i^C is the bias. In our experiments, we use the SRN output layer only in the training process. In evaluation process, we aim at the state representation and thus we analyze activations of the recurrent layer (i.e. the state of network). Training of SRN is performed through minimization of the error function $E(t) = \frac{1}{2}\sum_{m}(d_m(t) - o_m(t))^2$ by mens of EKF. (For details of the EKF training see the paper of M. Čerňanský and Ľ. Beňušková in this issue.)

4 Chaotic Symbolic Sequence

As a training sequence we chose a long symbolic sequence of quantized activity changes of laser in chaotic regime. Data set includes various levels of memory structures, i.e. relatively predictable subsequences followed by global, harder to predict events, that require a deeper memory. More precisely, it contains simple oscillations with increasing amplitude, followed by rapid decays of amplitude of oscillations with activity collapses (Fig 2).

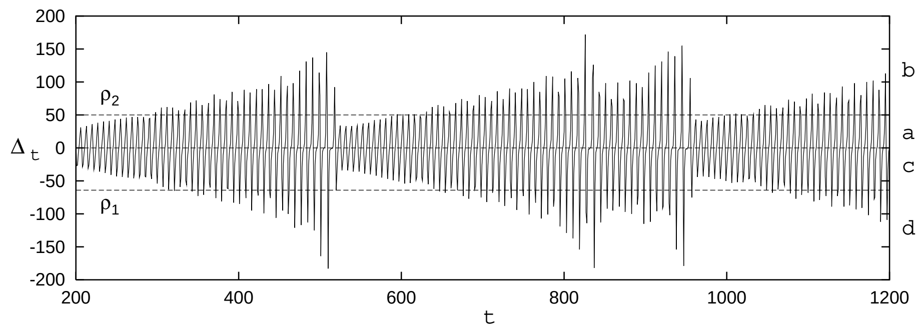

Figure 2: Example of 1000 differences Δ_t of the laser activations. Dotted horizontal lines coresponds to 10% and 90% sample quantiles. Letters on the right vertical axis indicate quantization into four symbols.

Whole sequence[1], i.e. 10 000 differences Δ_t between two subsequent activations, was quantized into a symbolic stream $S = \{ s_t\}$ of four symbols from the alphabet $A = \{a, b, c, d\}$, corresponding to low / high and positive / negative activity changes

$$s_t \equiv \begin{cases} a & (low\ positive) & if\ 0 \le \Delta_t < \rho_2 \\ b & (high\ positive) & if\ \rho_2 \le \Delta_t \\ c & (low\ negative) & if\ \rho_1 \le \Delta_t < 0 \\ d & (high\ negative) & if\ \Delta_t < \rho_1 \end{cases} \tag{8}$$

where ρ_1 and ρ_2 are 10% and 90% sample quantiles, respectively. Entire sequence S was divided into the train sequence S_{train} (the first 8000 symbols) and the test sequence S_{test} (the last 2000 symbols).

5 Neural Prediction Machine

In order to compare the state representations of SRN with recurrent BCM network, we use an abstract model, called neural prediction machine (NPM).

[1]Taken from http://www-psych.stanford.edu/˜andreas/Time-Series/SantaFe.html

Generally, NPM seeks state clusters in a network state space and associates them with conditional probabilities for the next symbol prediction. Before building of NPM we remove the output layer of SRN and thus unify output behavior of both architectures.

The procedure of building NPM starts after training phase of RNNs with fixing and storing all weights. For each symbol in the train sequence S_{train} we calculate activations $c(t) = \{c_i(t)\}$ of recurrent neurons, what creates an activation sequence S^{act}_{train} for the network. In the next step we perform vector quantization on S^{act}_{train} by using the K-means clustering algorithm for desired number of quantization centers. Each center is identified with the prediction context for the next symbol prediction. In order to obtain context-conditional probabilities, we associate a set of counters with each context, one counter for each symbol from input alphabet A. Moving through the train sequence S^{act}_{train}, nearest center C is found for each activation vector $c(t)$. Then according to the next symbol s_{next} in the sequence S_{train}, the counter for relevant center C and symbol s_{next} is increased by one. After checking the whole training sequence, conditional probabilities $P(s|C)$ are calculated by normalization of counters for each quantization center C.

In evaluation of NPM we run network over the test sequence $S_{test} = s_1, s_2, ...,$ s_m, while predicing each next symbol with probability calculated as follows: each symbol $s_t \in S_{test}$ drives network to recurrent activation $c(t) = \{c_i(t)\}$, with the nearest quantization center C_t. Probability for the next symbol s_{t+1} in sequence is the conditional probability $P(s_{t+1}|C_t)$ calculated in the NPM building phase.

Performance of NPM was evaluated by means of the normalized negative log-likelihood (NNL) on the test sequence calculated as

$$NNL(S_{test}) = \frac{-\sum\limits_{t=1}^{m-1} \log_{|A|} P(s_{t+1}|C_t)}{m-1} \tag{9}$$

where the base of logarithm is the number of symbols in alphabet A. The value of $NNL(S_{test})$ can be interpreted as the total amount of "statistical surprise" induced by the NPM [7]. Note, that NPM is not a finite state predictor. It has finite number of prediction contexts, but its dynamics is completely equal to the dynamics of an original recurrent network.

6 Simulations and Results

First, we trained recurrent BCM network with various numbers of recurrent units to obtain the best performance (i.e. the lowest NNL values). We also tried to use lateral inhibition, however it had no significant influence on network performance. Finally, we trained both SRN and recurrent BCM network with similar parameters and performed analysis of the solution provided by optimal settings.

The values of parameters for both networks were set to: the number of recurrent neurons 16, number of input and output neurons 4, averaging period $\tau = 100$ (for BCM), learning rate $\eta = 0.01$, and a unipolar 0-1 sigmoid with the slope $\lambda = 1$. Both weights and initial network state were initialized from uniform distribution over interval $(-0.5, 0.5)$. The number of training epochs was 10 for

EKF and 40 for BCM training. (For the values of other EKF parameters see the paper of M. Čerňanský and Ľ. Beňušková in this issue.) Input sequence was encoded by one-hot coding (i.e. $d_a = (1,0,0,0)$, $d_b = (0,1,0,0)$, etc.). All presented values are the averages with standard deviations from 10 runs of network training and NPM building for both SRN and recurrent BCM network (Fig. 3).

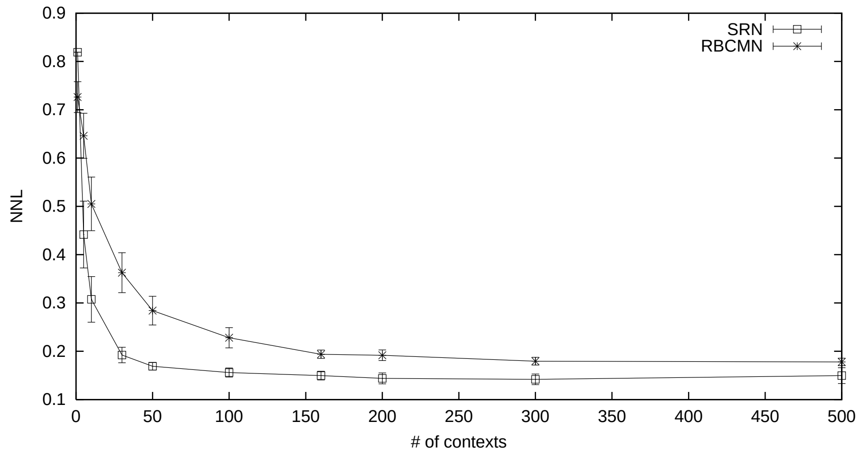

Figure 3: Normalized negative log-likelihood (NNL) for neural prediction machines (NPMs) built from SRN and recurrent BCM network after training. NNL was calculated for NPMs with different numbers of prediction contexts (i.e. vector quantization centers, # of contexts). We can see that SRN outperforms recurrent BCM network in its prediction. Ideal NNL would be equal to 0.

7 Analysis of the State Space

We want to understand what makes SRN to achieve better performance than a recurrent BCM network. As we mentioned earlier, the dynamics of the NPM and RNN are equal, thus by analyzing NPM with finite number of prediction contexts, we can analyze the dynamics of RNN.

It was shown that dynamics of randomly initialized RNN is closely related to the variable memory length Markov models (VLMM) [10]. The basic property of classical fixed order Markov models (MM) 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. However with increasing m, the number of prediction contexts and complexity of prediction increase exponentially. VLMM approach does not use fixed memory depth, but instead it uses deeper memory for suffixes only when it is really needed [7].

Dynamics of randomly initialized RNN with small weights is quite simple. It contains a set of attractors, each for one input symbol, located near the center of the state space. It was shown that the state representation produced by this attractor dynamics results in fractal structure of the RNN state space [5]. In this fractal structure, history of seen symbols is encoded in the same way as in the VLMM, i.e. sequences sharing the same suffix lead the network state to exactly the same cluster in the state space [11]. This property of RNNs to implement VLMM even without learning is called the architectural bias [4, 10]. In our experiment we choose the VLMM model as a reference to compare quality of solutions provided by both RNN learning rules.

The basic property of VLMM is the dependency of prediction context on the suffix in the sequence in the sense that the same suffix leads the network to the same state cluster. Thus, if identical suffixes in the test sequence lead to the same prediction context (i.e. quantization cluster) in NPM, the behavior of original RNN is equal to VLMM. On the other hand, if there are lots of identical subsequences that lead RNN to different state clusters (i.e. different prediction contexts in NPM), the network must come up with more complex, that is non-Markovian dynamics. This simple criterion allows us to determine whether the internal dynamics of RNN is Markovian (in the sense of VLMM) or it can extract another, obviously more complex dependencies from the input sequence.

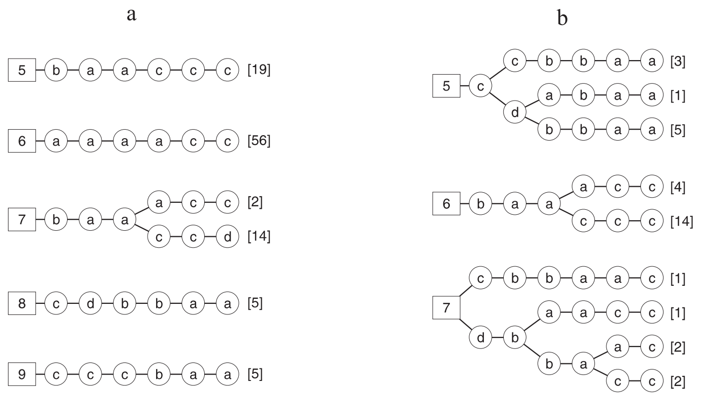

Figure 4: Examples of suffix trees for different prediction contexts of NPMs built from trained (a) recurrent BCM network and (b) SRN. Numbers in squares at the roots of suffix trees merely denote different quantization centers in the state space, i.e. different prediction contexts. Interconnected rings show sequences of input symbols leading NPM to the relevant context (number of sequence occurrences is in brackets). On average, SRN has much more patulous suffix trees than recurrent BCM network, thus the SRN state clusters do not encode only one single sequence of symbols.

For ilustration we choose NPM with 160 prediction contexts, which were sufficient for optimal clustering of the state space, because with higher number of prediction contexts, the number of unused clusters rapidly increases, and NNL does not decrease significantly (Fig. 3). The test sequence includes 2000 symbols. The first 20 symbols were used for network warming-up and were excluded from both NNL calculation and our analysis. First, we visualized and analyzed sequences of symbols that lead NPM to the same prediction context in the form of suffix trees. Characteristic examples are shown in (Fig. 4).

We also built NPM for randomly initialized networks and analyzed occurrences of identical suffixes in different prediction contexts. We ran each test ten times and calculated the mean value and standard deviation. Results for trained and untrained RNNs are shown in Table 1.

NPM built from	Number of identical suffixes for different centers	Average distance between centers	symmetric Kullback-Leibler distance
Randomly initialized network	112.7 (12.21)	0.000068 (0.000026)	0.783 (0.127)
Trained recurrent BCM network	228.9 (51.12)	0.0635 (0.02416)	0.83 (0.14)
Trained SRN	812.6 (111.8)	0.781037 (0.109299)	1.155 (0.120)

Table 1: Distribution of identical suffixes of length 6 and corresponding cluster center distances obtained from analysis of NPM built from different RNNs. Shown values are averages obtained from 10 runs with standard deviations in parenthesis.

In suffix trees of NPMs built from randomly initialized RNN and from trained recurrent BCM network, identical subsequences that lead NPM to different prediction contexts occurred on average from ~100 to ~200 times. However for trained SRN, it was more than 800 occurrences, what indicates that dynamics performed by SRN is more complex than dynamics of both, trained recurrent BCM network and randomly initialized untrained RNN. To find out, why identical suffixes belonging to the same centers occurred in untrained RNN, we calculated an average Euclidean distance between centers of state clusters for corresponding prediction contexts (Table 1, column 2). Because the average distance between centers in the case of randomly initialized RNN and trained recurrent BCM network is relatively small, duplicating of identical suffixes over clusters may be caused by the low degree of state space clustering. For the trained SRN, however, the average distance between centers is much greater, what can be interpreted in such a way that there can be a lot of identical subsequences that can lead RNN to different states, thus SRN in its state clusters encodes more information than could be obtained just from short suffixes. To confirm differences of information encoded in clusters, we calculated symmetric version of Kullback-Leibler distance of chosen context-conditional probabilities. Kullback-Leibler distance (also known as relative entropy) is not symmetric, thus we use its modified symmetric version defined as follows

$$D_{SYM}\left(C_i, C_j\right) = \frac{1}{2} \cdot \left[D_{KL}\left(C_i, C_j\right) + D_{KL}\left(C_j, C_i\right)\right] \tag{10}$$

where $D_{KL}(C_i, C_j)$ is a "classical" Kullback-Leibler distance calculated as

$$D_{KL}\left(C_i, C_j\right) = \sum_{s \in A} P\left(s|C_i\right) \cdot \log\left(\frac{P\left(s|C_i\right)}{P\left(s|C_j\right)}\right) \tag{11}$$

where C_i and C_j are chosen contexts (centers). In the last column of Table 1, there is an average value obtained from all different contexts C_i and C_j that appeared after the same sequence of symbols. Thus, SRN trained by EKF has such a state space organization that leads network state to different clusters with different probability distributions after seeing the identical short sequence. Clusters of trained SRN store the highest amount of information about longer time dependencies compared to untrained naive RNN, and trained recurrent BCM network.

8 Conclusion

In order to compare the state representations of SRN with recurrent BCM network, we use an abstract model, called neural prediction machine (NPM). Generally, 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. In this work, we have introduced a novel method for analysis of the dynamics of RNNs including the gain of information obtained by learning. Thus, organization of the RNN state space is analyzed using calculations of the number of identical suffixes leading to different prediction contexts (depending on the deeper temporal dependencies), evaluation of average distances between prediction contexts in the state space, and calculation of the information gain by prediction contexts due to learning.

For chosen symbolic sequence of a very high complexity, SRN trained by EKF can gain information not only from short suffixes but also from longer time dependencies. On the other hand, the unsupervised recurrent BCM network provides solution of Markovian nature, thus it is not able to overcome the well-known information latching problem in RNNs [2] nor the initial Markovian bias [10].

Acknowledgments

This work was supported by the VEGA grant 1/9046/02. We thank Michal Čerňanský for the computer code of EKF for SRN.

References

[1] Bachman C.M., Musman S.A., Luong D. and Shultz A. Unsupervised BCM projection pursuit algorithms for classification simulated radar presentations. *Neural Networks*, 7:709–728, 1994.

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

[3] Bienenstock E.L., Cooper L.N and Munro P.W. Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2(1):32–48, 1982.

[4] Hammer B. and Tiňo P. Neural networks with small weights implement finite memory machines. To appear in *Neural Computation.*

[5] Kolen J.F. The origin of clusters in recurrent neural network state space. In *Proc. 16th Annual Conf. of the Cognitive Sci. Soc.*, pp. 508–513. Hillsdale, NJ: Lawrence Erlbaum Associates, 1994.

[6] Matthews M.B. and Moschytz G.S. Neural network non-linear adaptive filtering using the extended kalman filter algorithm. In: *Proceedings of the INNC'90 Paris*, vol. 1, pp. 115–119, 1990.

[7] Ron D., Singer Y. and Tishby N. The power of amnesia. *Machine Learning*, 25, 1996.

[8] Tiňo P., Stančík M. and Beňušková L. Building predictive models on complex symbolic sequences via a first-order recurrent BCM network with lateral inhibition. In P. Sinčák and J. Vašcák (eds) *Quo Vadis Computational Intelligence? New Trends and Approaches in Computational Intelligence*, pp. 42–50. Physica-Verlag, Heidelberg, 2000.

[9] Tiňo P., Stančík M. and Beňušková L. Building predictive models on complex symbolic sequences with a second-order recurrent BCM network with lateral inhibition. In: *Proc. IEEE Intl. Joint Conf. Neural Networks*, pp. 265–270, 2000.

[10] Tiňo P., Čerňanský M. and Beňušková L. Markovian architectural bias of recurrent neural networks. To appear in *IEEE Trans. Neural Net.*

[11] Tiňo P. and Hammer B. Architectural bias in recurrent neural networks - fractal analysis. To appear in *Neural Computation.*

[12] Williams R.J. Training recurrent networks using the extended Kalman filter. In *Proc. Intl. Joint Conf. Neural Networks, Baltimore*, vol. 4, pp. 241–246, June 1992.

[13] Williams R.J. and Zipser D. Gradient-based learning algorithms for recurrent networks and the computational complexity. In: Y. Chauvin and D. E. Rumelhart (eds) *Back-propagation Theory, Architectures and Applications*, pp. 433–486. Lawrence Erlbaum Publ., Hillsdale, N.J., 1995.