Neural Network World, 13 (3), pp. 223-234. http://www.nnw.cz/obsahy03.html#3-2003

SIMPLE RECURRENT NETWORK TRAINED BY RTRL AND EXTENDED KALMAN FILTER ALGORITHMS

Michal Čerňanský * Ľubica Beňušková †

Abstract

Recurrent neural networks (RNNs) have much larger potential than classical feed-forward neural networks. Their output responses depend also on the time position of a given input and they can be successfully used in spatio-temporal task processing. RNNs are often used in the cognitive science community to process symbol sequences that represent various natural language structures. Usually they are trained by common gradient-based algorithms such as real time recurrent learning (RTRL) or backpropagation through time (BPTT). This work compares the RTRL algorithm that represents gradient based approaches with extended Kalman filter (EKF) methodology adopted for training the Elman's simple recurrent network (SRN). We used data sets containing recursive structures inspired by studies of cognitive science community and trained SRN for the next symbol prediction task. EKF approach, although computationally more expensive, shows higher robustness and the resulting next symbol prediction performance is higher.

1 Introduction

Feedforward neural networks are unable to process data with time dependent information. A network has to be provided with some kind of memory. One possibility of how to accomplish this task is to incorporate feedback connections between units. Network's hidden or output units can be provided with an extended input that is composed of current input activities together with activities from previous time steps. Because of these recurrent feedback connections we call this type of dynamical neural networks, recurrent neural networks (RNNs).

Common algorithms usually used for the RNNs training are based on gradient minimization of error. One of such algorithms, backpropagation through time (BPTT) [15, 19], consists of unfolding a recurrent network in time and applying the well-known backpropagation algorithm directly. Another gradient based 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) [18].

*Department of Computer Science and Engineering, Slovak Technical University, Ilkovičova 3, 812 19 Bratislava, Slovakia, E–mail: *cernansky@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*

In the cognitive science community researchers often try to use recurrent neural networks to establish links between human ability to process linguistic structures and the potential of RNNs [2, 3, 6]. One example of connectionist processing of artificial languages reflecting recursive real-language structures can be found in the work of Christiansen and Chater [2]. Authors trained SRN on three simple artificial languages with recursive structures similar to those found in human speech. They showed correspondence between empirically observed limited ability of humans to process recursive structures and results obtained by experimenting with SRN.

Data sets used in our work are inspired by these artificial languages. We compare two methods of training recurrent neural networks. RTRL represents common gradient-based approaches to weight modification. Other approach is the adaptation of the extended Kalman filter into the RNNs training framework.

2 Recurrent Neural Network Architecture

In this work we used the first order simple recurrent network (SRN) [3]. It is an example of multilayer perceptron with feedback connections (Fig. 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 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.

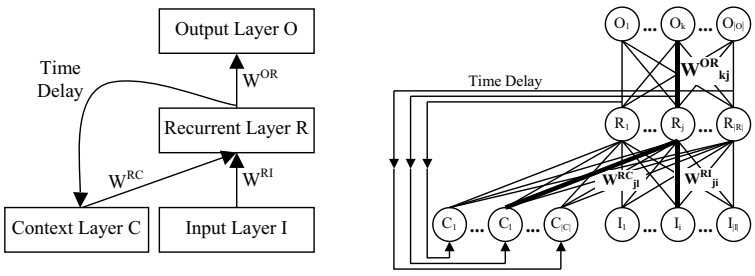

Figure 1: Simplified and more detailed representation of Elman's SRN.

Units of the input layer I and the recurrent layer R and the output layer O are fully connected through weights W^{RI} and W^{OR}, respectively, as in the feedforward multilayer perceptron (MLP). Time delay connections feed back current activities of recurrent units $R^{(t)}$ to the context layer so that $C^{(t)} = R^{(t-1)}$. Hence, every recurrent unit is fed by activities of all recurrent units from previous time step through recurrent weights 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, $I^{(t)} = (I_1^{(t)}, ..., I_j^{(t)}, ..., I_{|I|}^{(t)})$, and recurrent activities $R^{(t)} = (R_1^{(t)}, ..., R_j^{(t)}, ..., R_{|R|}^{(t)})$, 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 $f(x) = (1 + e^{-x})^{-1}$.

3 RTRL Algorithm for SRN

RTRL is based on approximate on-line gradient computation and was described in details in [18], however for an RNN architecture different from SRN. Here, we describe in detail the RTRL equations for SRN. Network weights are updated in every time step in order to minimize the current output error with respect to the calculated approximate gradient. In a given time t, modifications of weights connecting output and recurrent units are calculated as:

$$\Delta W_{ij}^{OR} = \alpha (D_i^{(t)} - O_i^{(t)}) f'(\tilde{O}_i^{(t)}) R_j^{(t)}, \tag{5}$$

where $D^{(t)} = (D_1^{(t)}, ..., D_j^{(t)}, ..., D_{|O|}^{(t)})$ is the desired output pattern and α is the learning speed. Modifications of weights connecting recurrent and input units are calculated as:

$$\Delta W_{ji}^{RI} = \alpha \sum_k^{|O|} \left[(D_k^{(t)} - O_k^{(t)}) f'(\tilde{O}_k^{(t)}) \sum_{h=1}^{|R|} W_{kh}^{RC} \frac{\partial R_h^{(t)}}{\partial W_{ji}^{RI}} \right], \tag{6}$$

where

$$\frac{\partial R_h^{(t)}}{\partial W_{ji}^{RI}} = f'(\tilde{R}_i^{(t)}) \left[I_i^{(t)} \delta_{hj}^{kron} + \sum_{l=1}^{|R|} W_{hl}^{RC} \frac{\partial R_l^{(t-1)}}{\partial W_{ji}^{RI}} \right]. \tag{7}$$

In a similar way one can calculate the modifications of weights connecting context and recurrent units:

$$\Delta W_{ji}^{RC} = \alpha \sum_k^{|O|} \left[(D_k^{(t)} - O_k^{(t)}) f'(\tilde{O}_k^{(t)}) \sum_{h=1}^{|R|} W_{kh}^{RC} \frac{\partial R_h^{(t)}}{\partial W_{ji}^{RC}} \right], \tag{8}$$

where

$$\frac{\partial R_h^{(t)}}{\partial W_{ji}^{RC}} = f'(\tilde{R}_i^{(t)}) \left[R_i^{(t-1)} \delta_{hj}^{kron} + \sum_{l=1}^{|R|} W_{hl}^{RC} \frac{\partial R_l^{(t-1)}}{\partial W_{ji}^{RC}} \right]. \tag{9}$$

δ_{hj}^{kron} is the Kronecker's delta and $\delta_{hj}^{kron} = 1$ if $h = i$, otherwise $\delta_{hj}^{kron} = 0$.

4 Extended Kalman Filter Algorithm for SRN

The Kalman filter is a set of equations describing a recursive solution of the discrete-data linear filtering problem. It is an effective solution to the least-square minimization problem. Good tutorials covering this topic are [8, 14] and the first chapter of [7].

Assume a system governed by a linear stochastic difference equation called the state or process equation

$$x_{k+1} = F_k x_k + w_k. \tag{10}$$

Unobservable state of the system x_k in the step k is calculated by applying the known state transition matrix from time k to $k+1$, F_k, to the previous state x_k, and the white Gaussian noise w_k is added. The measurement equation is given by

$$z_k = H_k x_k + v_k, \tag{11}$$

where z_k stands for an observed measurement, H_k for the known measurement matrix and v_k is the white Gaussian noise. Covariance matrices Q_k and R_k of the process and measurement noise

$$Q_k = E\left[w_k w_k^T\right], \tag{12}$$

$$R_k = E\left[v_k v_k^T\right] \tag{13}$$

have zero off-diagonal elements. Note, that the state transition matrix F_k and the measurement matrix H_k can change over time.

The aim of the Kalman filter is to obtain the best state estimate $\hat{x}$ from an observed noisy measurements z. We can define the state estimate error e_k and an estimate error covariance P_k as

$$e_k = x_k - \hat{x}_k, \tag{14}$$

$$P_k = E\left[e_k e_k^T\right]. \tag{15}$$

The Kalman filter works in the two step cycle. The first step, the time update step, is a simple prediction of the so-called a priori state estimate $\hat{x}_k^-$ and an estimate error covariance P_k^- based on previous $\hat{x}_k$ and P_k. Superscript "$-$" indicates the *a priori estimates*. Thus,

$$\hat{x}_k^- = F_{k-1}\hat{x}_{k-1}, \tag{16}$$

$$P_k^- = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1}. \tag{17}$$

The second, measurement update step, performs corrections of the state estimate and estimate error covariance based on an actual measurement. The Kalman gain K is calculated as

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}, \tag{18}$$

and *a posteriori* state estimate $\hat{x}_k$ and estimate error covariance P_k are calculated as

$$P_k = P_k^- - K_k H_k P_k^-, \tag{19}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H_k\hat{x}_k^-). \tag{20}$$

A standard Kalman filter can be applied to linear system affected by white Gaussian zero mean noise. If we loosen this assumption and consider nonlinar system such as SRN with sigmoidal units, the Kalman filter looses its optimality properties and the so-called extended Kalman filter (EKF) can be applied. Consider the time update and measurement update equations expressed by

$$x_{k+1} = f_k(x_k) + w_k, \tag{21}$$

$$z_k = h_k(x_k) + v_k, \tag{22}$$

where f_k and h_k are nonlinear vector functions of the state. By linearisation of these equations, using the Taylor series, we can derive the time update equations

$$\hat{x}_k^- = f_{k-1}(\hat{x}_{k-1}), \tag{23}$$

$$P_k^- = F_{k-1} P_{k-1} F_{k-1}^T + Q_{k-1}, \tag{24}$$

and the measurement update equations

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}, \tag{25}$$

$$P_k = P_k^- - K_k H_k P_k^-, \tag{26}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h_k(\hat{x}_k^-)). \tag{27}$$

where F_k and H_k are Jacobian matrices

$$F_k = \frac{\partial f_k(\hat{x}_k)}{\partial x}, \tag{28}$$

$$H_k = \frac{\partial h_k(\hat{x}_k)}{\partial x}. \tag{29}$$

Elman's SRN and generally any other feedforward or recurrent multilayer perceptron network can be described as a nonlinear system by

$$x_{k+1} = x_k, \tag{30}$$

$$z_k = h_k(x_k, u_k) + v_k, \tag{31}$$

instead of eqn. 21 and 22, where the state x_k is a vector of network weights. The weights of the trained network do not change and thus the state transition matrix $F = I$ where I is an identity matrix. The measurement z_k stands for the desired output values. The measurement function h_k is a nonlinear function of network weights x_k and input u_k. Jacobian matrix H_k is calculated in every time step k as

$$H_k = \frac{\partial h_k(\hat{x}_k, u_k)}{\partial x}, \tag{32}$$

using equations 7 and 9. Extended Kalman functions are simplified to

$$K_k = P_{k-1}(H_k P_{k-1} H_k^T + R_k)^{-1}, \tag{33}$$

$$P_k = P_{k-1} - K_k H_k P_{k-1} + Q_k, \tag{34}$$

$$\hat{x}_k = \hat{x}_{k-1} + K_k(z_k - h_k(\hat{x}_{k-1}, u_k)). \tag{35}$$

Note, that the small state transition noise with covariance matrix Q_k is still considered. In the described version of an EKF-based training, the state x is a concatenation of network's weights. The state x can also include activities of the context units as described in [16, 17]. This approach leads to the simpler Jacobian matrix H_k calculation but the state transition equation cannot be omitted, and the matrix H_k has a higher dimension.

5 Description of Data Sets

We performed experiments on two artificial languages that exhibit recursive structures found in natural languages. The first language is taken from the work of Christiansen and Chater [2]. Language symbols belong to four categories: N_P, V_P, N_S, V_S, where N stands for a noun, V for a verb, P for a plural and S for a singular. Language involves two kinds of recursive structures: one is a complex center-embedding recursion (CER) that cannot be generated by a finite state machine, and the other one is a right-branching recursion (RBR) that can be produced by a simple iterative process carried out by a finite state machine. Thus, this language contains the combination of context-free and regular language features, similarly like the natural languages do. RBR can be generated by a simple generative process described by the production rule $X \rightarrow N_P V_P X | N_S V_S X | e$, where e is the empty string, to obtain constructions like $N_P V_P N_S V_S$. The center embedding recursion (CER) can be defined by the following production rule: $X \rightarrow N_P X V_P | N_S X V_S | e$. Starting with X we can generate the strings like $N_P N_S V_S V_P$, $N_P N_S N_S N_P V_P V_S V_S V_P$, etc. The strings of this first language contained both CER and RBR structures. At the beginning of a string it is impossible to know whether the string will involve CER or RBR. Once the second word is encountered, a verb indicates RBR whereas another noun indicates CER. A CER structure may end after the first noun/verb pair, or continue with one or more embeddings. With another noun, it is not possible to predict how many nouns will follow. However, after encountering the first verb, it is possible to predict the number and type of verbs, provided the system can learn the number and the type of nouns.

We use a sixteen word vocabulary with four singular nouns, for plural nouns, four singular verbs and four plural verbs. Words representing 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. Sets contained 50 % of RBR strings interleaved with 50 % of CER strings in a random way. The distribution of string lengths of both types of recursion was the same and is described in the Table 1.

Length	**Percent**	**RBR Examples**	**CER Examples**
2	15.0 %	$N_S V_S, N_P V_P$	$N_S V_S, N_P V_P$
4	27.5 %	$N_P V_P N_S V_S$	$N_S N_P V_P V_S$
6	7.0 %	$N_P V_P N_P V_P N_S V_S$	$N_S N_P N_P V_P V_P V_S$
8	0.5 %	$N_S V_S N_P V_P N_P V_P N_S V_S$	$N_S N_S N_P N_P V_P V_P V_S V_S$

Table 1: Distribution of string lengths in the CERandRBR data set. Examples are given in categories. In actual data sets, every category was replaced by one of four "true" words.

The second, deep recursion data set, with embeddings reaching the value of 10, is composed of strings of the context-free language L_G. Its generating grammar is $G = (R, \{R\}, \{a, b, A, B\}, P)$, where R is the single non-terminal symbol which is also a starting symbol, and a, b, A, B are terminal symbols. The set of production rules P is composed of a single rule: $R \rightarrow aRb | R \rightarrow$

$ARB|R \rightarrow e$ where e is an empty string. This language is in [10] called the palindrome language.

The training and testing data sets consist of 1000 randomly generated concatenated strings. No end-of-string symbol was used. Shorter strings were more frequent in the training set then the longer ones as shown in Table 2. The total length of the training set was 6156 symbols and the length of the testing set was 6190 symbols.

Length	Percent	Example
2	35.0 %	ab, AB
4	20.0 %	$aABb, aabb$
6	14.0 %	$aAABBb, AaabBB$
8	10.0 %	$AAAabBBB$
10	3.5 %	$aAAAABBBBb$
12	3.5 %	$AAaAAabBBbBB$
14	3.5 %	$AaaaAaabbBbbbB$
16	3.5 %	$aaAaAAaABbBBbBbb$
18	3.5 %	$aAAAaaaaabbbbbBBBb$
20	3.5 %	$aaAaAaAaaabbbBbBbBbb$

Table 2: Distribution of string lengths in the context-free language.

6 Methods

RTRL and EKF were used to train Elman's SRN. Numerous simulations were performed with different parameters. 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. 10 training epochs (one epoch – one presentation of the training set) for EKF and up to 1000 epochs for RTRL were done when needed. SRN state (i.e. context) units were reset before training or testing epoch to initial state and 30 dummy steps were performed (no error calculation or weight updates were done during dummy steps). Recurrent unit count varied from 4 to 32 but the performance was not getting better for more than 16 hidden units. Number of input and output units corresponded to the symbol count in alphabet of a given data set. Unipolar 0-1 sigmoid activation function was used.

Predictive performance was evaluated by means of a normalized negative log-likelihood (NNL) calculated over symbolic sequence from time step $t = 1$ to T as

$$NNL = -\frac{1}{T}\sum_{t=1}^{T} \log_{|A|} p^{(t)}\left(s^{(t)}\right), \tag{36}$$

where the base of the logarithm $|A|$ is the number of symbols in the alphabet A and the $p^{(t)}\left(s^{(t)}\right)$ is the probability of predicting symbol $s^{(t)}$ in the time step t. Value $p^{(t)}\left(s^{(t)}\right)$ is obtained by normalizing activities of output units and choosing normalized output activity corresponding to the symbol $s^{(t)}$.

Our EKF training was not so sensitive to parameter values as the RTRL was. Diagonal elements of the state error covariance matrix P were initialized

to value 1000, all other elements were set to 100 [8]. The measurement noise covariance matrix R was set to $R = 1/Pr * I$ where Pr $= 0.01$, and the process noise covariance matrix Q was set to $Q = Pq * I$, where $Pq = 0.0001$. These values were used throughout all experiments. 10 training epochs were sufficient for reaching the steady state, no significant NNL improvement has occured after 10 epochs in any experiment. Partial derivatives needed for Jacobian matrix H were calculated by the RTRL algorithm. Although standalone RTRL does not follow the true gradient and should be used with small learning rate [18], the EKF with RTRL Jacobian matrix calculation worked very well.

Different RTRL parameters were used for each data set to obtain the best results. Using high momentum and learning rate causes faster convergence but significant error increases occur frequently, and the NNL fluctuations are high. On the other hand, using smaller values makes training slow and susceptible to local minima. We tried to avoid these problems by using some of the methods used by Lawrence et al. [6]. After detecting a significant error increase, the network weights were restored from the previous epoch and were randomly perturbated to prevent updating to the same point. Finding parameters for this method seemed to be difficult. Small weight perturbations were not sufficient to prevent a significant error increase. On the other hand, perturbing weights too much caused loosing information aquired up till then and error increased immediately. An easier way to improve training phase was to use the scheduled learning rate. We used linearly decreasing learning rate in predefined intervals. But no improvements made the RTRL training as stable and fast as the EKF training (taking into account the number of epochs). The momentum rate for the context-free language with deep recursion was disabled (set to 0) and the learning rate was set to initial value of 0.05. It decreased linearly during presentation of first 600000 symbols until it reached value of 0.001. For the CERandRBR data set, the momentum rate was set to 0.90 and learning rate to 0.05.

7 Results

We present mean and standard deviations of NNL results of 10 simulations with fixed parameters and different initializations. Although few simulations have been suppressed because of high NNL fluctuation (some RTRL and also EKF simulations), final results represent general behavior for the parameters described in the previous section. Fig. 2 shows results obtained on the center-embedding and right-branching recursion data (CERandRBR) and context-free languages, respectively. Generally, NNL performances of RNNs trained by EKF are better, although not dramatically. We were able to train few RNNs by RTRL to have similar performance as the networks trained by EKF, but it usually required much more overhead (i.e. choosing only few from many networks, more than one thousand of training epochs, extensive experimenting with learning and momentum rates).

RNNs behave as iterated function system [5] and have properties of Markov models even before training [1, 4, 12, 13]. Thus, it is important to compare results obtained by RNNs with results given by fixed order Markov models (MM) and variable length Markov models (VLMM) [11]. NNL results of Markov models as a function of number of context are shown in Fig. 3.

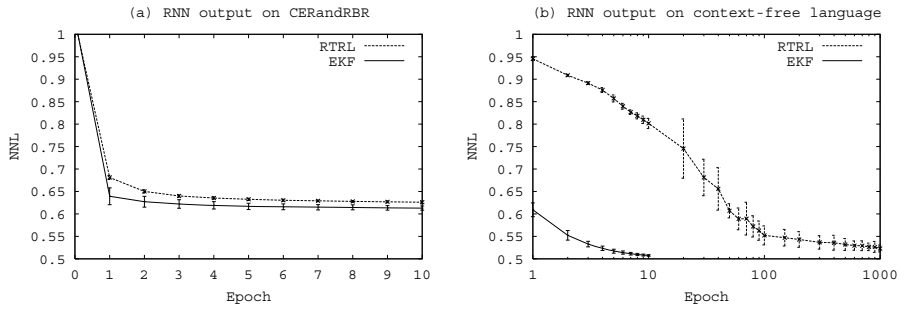

Figure 2: NNL performance of the output layer of SRN trained either by RTRL or EKF on the CERandRBR and context-free languages.

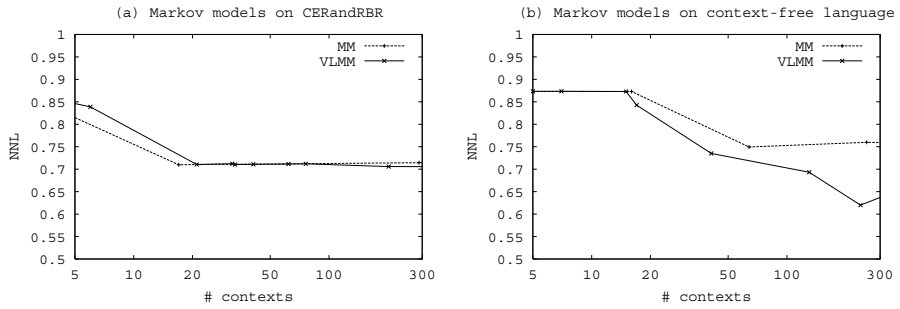

Figure 3: NNL performance of Markov models with fixed (MM) and variable length memory (VLMM) on the CERandRBR and context-free languages.

For the CERandRBR data set, both RTRL and EKF reached similar results. This data set is inspired by Christiansen and Chater's simulations in the cognitive field. If we take into account the size of the input alphabet (17 symbols), then the length of the training data set (23366 symbols) is small. NNL performance of trained SRN is better then NNL obtained by Markov models, and decreases almost down to an ideal NNL of 0.59 (estimated by means of Monte Carlo simulations).

Training SRN on context-free deep recursion data set also improves NNL performance compared to MMs. Recursions require sort of counting mechanisms that are not present in MMs. Estimated ideal NNL of our context-free language obtained by means of Monte Carlo simulations is equal to 0.41. Comparing the results of trained SRN with the results obtained with Markov models reveals that SRN can acquire some amount of information during training, although for very deep recursions it is difficult to reach an ideal NNL. Thus, in this case the performance of RNN is similar to the human performance, which are known not to be able to process deep recursions [2].

8 Conclusion

In this paper we describe in detail the equations of RTRL and EKF training for Elman SRN. We experimented with two kinds of complex symbolic time series to find out which of the two training methods can bring better results. Center-embedding and right-branching recursion language (CERandRBR) combines context-free and regular features of natural languages [2]. The second artificial language, was entirely context-free with deep recursions of up to 10 embeddings and was designed by us. EKF can show a significantly faster convergence in terms of number of epochs. Resulting NNLs are reliably better than for RTRL, although not dramatically. Standard deviations of results obtained by RTRL algorithm can get quite high revealing RTRL's sensitivity to the weight initialization. For each data set, different RTRL learning parameters had to be found – what is not the case of EKF. Although computationally more demanding, EKF approach to training RNNs on symbolic sequences shows higher robustness and better resulting performance (see also the paper of M. Makula and Ľ. Beňušková in this issue). It is easy to implement and should be considered when choosing appropriate algorithm for RNN training. Similar results were obtained by using the decoupled variant of EKF (DEKF) to train long short-term memory networks (LSTM networks) on artificial grammars [9].

Acknowledgments

This work was supported by the VEGA grant 1/9046/02. We would also like to thank Peter Tiňo for useful discussions.

References

[1] Čerňanský M. and Beňušková L. Finite-state Reber automaton and the recurrent neural networks trained in supervised and unsupervised manner. In: H. Bischof, G. Dorffner and K. Hornik (eds) *LNCS 2130. Artificial Neural Networks - ICANN'2001*, pp. 737–742, Springer-Verlag, 2001.

[2] Christiansen M.H. and Chater N. Toward a connectionist model of recursion in human linguistic performance. *Cognitive Sci.* 23: 417–437, 1999.

[3] Elman J.L. Finding structure in time. *Cognitive Sci.*, 14: 179–211, 1990.

[4] Hammer B. and Tiňo P. Recurrent 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] Lawrence S., Giles C.L. and Fong S. Natural language grammatical inference with recurrent neural networks. *IEEE Trans. Knowledge and Data Engineering*, 12(1):126–140, 2000.

[7] Maybeck P.S. *Stochastic models, estimation, and control, vol. 1.* New York Academic Press, 1979.

[8] Patel G.S.. Modeling nonlinear dynamics with extended Kalman filter trained recurrent multilayer perceptrons. *Thesis*, McMaster Univ., Canada, 2000.

[9] Pérez-Ortiz J.A., Gers F.A., Eck D. and Schmidhuber J. Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. *Neural Networks*, 16(2): 1–23, 2003.

[10] Rodriguez P. Simple recurrent networks learn contex-free and contex-sensitive languages by counting. *Neural Computation*, 13: 2093–2118, 2001.

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

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

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

[14] Welch G. and Bishop G.. An introduction to the Kalman filter. TR95-041, Dept. Computer Science, Univ. North Carolina, 1995.

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

[16] Williams R.J. Some observations on the use of the extended Kalman filter as a recurrent network learning algorithm. TR NU-CCS-92-1, Boston, 1992.

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

[18] Williams R.J. and Zipser D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

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