Recurrent Neural Networks trained by RTRL and Kalman Filter Algorithms

Michal Čerňanský * Ľubica Beňušková †

Abstract

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

1 Introduction

Feedforward neural networks are unable to process data with time dependant information. Network has to be provided with some kind of memory. One possibility how to accomplish this task is to incorporate feedback connection between units. Network's units can be provided with 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 RNNs training are based on gradient minimization of the error. One such algorithm, backpropagation through time (BPTT) [14, 18], consist of unfolding recurrent network in time and to directly apply well-known backpropagation algorithm. Extensions of this method can be used to train networks on infinite input sequences. Partial derivatives needed for evaluating error gradient are calculated over given interval of time. Another gradient based approach where estimates of derivatives and weight changes are calculated in every time step is called real time recurrent learning algorithm (RTRL) [17].

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

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

In cognitive science community researchers often get use of recurrent neural networks and try to establish links between human ability to process linguistic structures and the potential of RNNs [1, 2, 3, 6]. General believe behind this approach of studying human behavior is usually not to directly compare artificial units with neurons in human brain nor to completely simulate areas in brain by means of RNN's. Highly simplified models of biological neural networks can hardly ever mimic complex behavior of even very simple animals. Nevertheless researches believes that studying these simple artificial connectionist models can help to understand some aspects of the functioning of the human brain that power also comes from massive parallel computation of relatively very simple biological neurons.

One example of connectionist processing of artificial languages reflecting recursive real-language structures can be found in the work of Christiansen and Charter [1]. Authors trained SRN on three simple artificial languages with recursive structures similar to those found in human speech. They showed correspondence between empirically observed people's limited ability to process recursive structures and results obtained by experimenting with SRN.

Datasets 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 extended Kalman filter into the neural network's training framework.

2 Recurrent Neural Network Architecture

In this work we used first order simple recurrent network SRN [2]. It is an example of multilayer preceptron with feedback connections. Network is composed of four layers. 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 input or target symbol, that is highly active. Hence, the number of the input and output units is equal to the number of the symbols in the alphabet of a given data set. Network hidden layer is fed through recurrent connections by activities of context layer. Context units hold activities of hidden layer from previous time step. In this way SRN and generally all recurrent neural networks are provided by contextual information and cen process input data with time structure.

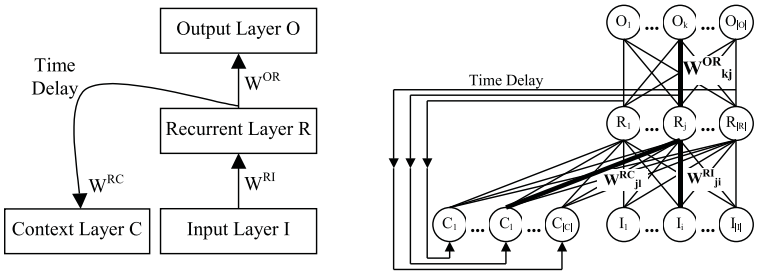

Figure 1: Simplified and more detailed representation of Elman's SRN.

SRN was proposed by Elman [2]. Units between input layer I and hidden layer R and between hidden layer and output layer O are fully connected through weights W^{OR} as in the feedforward MLP. Feedback time delay connections link current hidden units $R^{(t)}$ with previous time step hidden units $R^{(t-1)} = C^{(t)}$. Hence every hidden unit is fed through recurrent weights W^{RC} by activities of all hidden units from previous time step. Hidden layer is also called recurrent layer. Hidden units' activities from previous time step can be viewed as extended inputs to the recurrent layer. They represent the memory of the network, they hold contextual information from previous time steps and can be represented by context layer C. Simplified and also more detailed representations of SRN are shown in figure 1.

Given input patter in time t $I^{(t)} = (I_1^{(t)}, ..., I_j^{(t)}, ..., I_{|I|}^{(t)})$ and hidden activities from previous time steps $t-1$ $R^{(t-1)} = (R_1^{(t)}, ..., R_j^{(t)}, ..., R_{|R|}^{(t)})$ hidden 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 $|O|$ is the number of input, hidden and output units respectively. W_{ij}^{OR} represent weight connecting output unit i with hidden unit j, W_{ij}^{RI} and W_{ij}^{RC} are weights connecting hidden unit i with jth input or context unit. f stand for activation function. In this work we used logistic sigmoid function $f(x) = (1 + e^{-x})^{-1}$.

3 Real Time Recurrent Learning (RTRL) algorithm

Real Time Recurrent Learning is on-line algorithm for training recurrent networks. It is based on approximate on-line gradient computation and was described in details in [17]. Network weights are updated in every time step in order to minimize current output error with respect to the calculated approximate gradient. In given t time, modifications of weights connecting output and recurrent units are defined as:

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

where $D^{(t)} = (D_1^{(t)}, ..., D_j^{(t)}, ..., D_{|I|}^{(t)})$ is the desired output pattern and α is the learning speed. Modifications of weights connecting recurrent and input units are defined 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 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 Kroneckers delta and $\delta_{hj}^{kron} = 1$ if $h = 1$, otherwise $\delta_{hj}^{kron} = 0$.

4 Extended Kalman Filter Algorithm

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 estimation problem. Good tutorials covering this topic are [13] and the first chapter of [8].

Suppose system governed by linear stochastic difference equation called state equation

$$x_k = F_k x_k + w_{k-1}. \tag{10}$$

Unobservable state of the system in step x_k is calculated by applying known state transition matrix F_k to the previous state x_{k-1} and white gaussian noise w_{k-1} is added. Measurement equation is given by

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

where z_k stands for observed measurement, H_k for known measurement matrix and v_k is white gaussian noise. Covariance matrices R and Q of process and measurement noise have zero off-diagonal elements.

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

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

Note, that state matrix F_k and measurement matrix H_k can change over time.

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

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

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

Kalman filter works in two step cycle. The first step, time updates is simple prediction of so called a priori state estimate $\hat{x}_k^-$ and estimate error covariance

P_k^- based on previous $\hat{x}_{k-1}$ and P_{k-1}. Superscript $-$ indicates the a priori estimates.

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

$$P_k^- = FP_{k-1}F^T + Q. \tag{17}$$

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

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

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

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

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

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, Kalman filter looses its optimality properties and so call extended Kalman filter can be applied as a sub-optimal filter. Consider time update and measurement update equations described by

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

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

where f and h are nonlinear functions. By linearisation of these equations we can derive time update eqations

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

$$P_k^- = FP_{k-1}F^T + Q, \tag{24}$$

and measurement update equations

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

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

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

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 nonlinear system by

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

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

where state x_k is vector of network weights. Weights of trained network do not change and so state transition matrix $F = I$ where I is identity matrix. Measurement z_k stands for desired values and measurement function h_k is nonlinear function of network weights x_k and input u_k. Jacobian matrix H_k is calculated in every time step k as

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

and extended Kalman functions are simplified into

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

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

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

Note, that small state transition noise with covariance matrix Q is still considered. In described version of EKF-based training algorithm state x is concatenation of network's weighs. State x can also include activities on context units as described in [15, 16].

5 Description of Dataset

5.1 Center-Embedding and Right-Branching Recursion Data Set

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 Charter [1]. 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 languages features, similarly like the natural languages do. An RBR can be generated by a simple generative process described by the production rule $X \to N_P V_P X | N_S V_S X | e$ 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: $X \to N_P X V_P | N_S X V_S | e$. 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 either CER or 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 a RBR whereas another noun indicates a 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 cathegory were choosen with eqal probability. They were encoded by "one-hot" encoding.

Thus, SRN had 17 input and output units, with one unit devoted to the end of 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 embedding of both types of recursion was the same and is described in the table.

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	1.5 %	$N_S V_S N_P V_P N_P V_P N_S V_S$	$N_S N_S N_P N_P V_P V_P V_S V_S$

Table 1: Distribution of string in CERandRBR dataset. Examples are given in categories. In final datasets every category was replaced by one of four words that represent given gathegory.

Since in this, though quite complex, language the maximal depth of embedding within a string was 3, we have decided to employ a purely context-free artificial language with larger maximal embedding equal to 10.

5.2 Deep Recursion Data Set

Deep recursion data set is composed of strings of 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 that is also the 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 the empty string. This language is in [9] called 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 the table. 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 in DeepRec dataset.

6 Method

Real Time Recurrent Learning (RTRL) and extended Kalman Filtering (EKF) have been used to train Elman's simple recurrent network (SRN). Numerous simulations were performed with different parameters.

SRN weights and initial state (starting activation of state units) were randomly initialized from interval $(-0.5, 0.5)$ and $(0.0, 1.0)$ respectively for each simulation. 10 training epochs (one epoch - one presentation of the training set) for EKF and 100 epochs for RTRL were done for each simulation. SRN state 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).

Hidden unit count varied from 4 to 32 but we present simulation results only for 8 and 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.

The predictive performance was evaluated by means of 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 he logarithm $|A|$ is the number of symbol in the alphabet A and the $p^{(t)}\left(s^{(t)}\right)$ is the probability of predicting symbol $s^{(t)}$ in 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 symbol $s^{(t)}$.

EKF training wasn't as sensitive to initial parameters as RTRL was. Diagonal elements of state error covariance matrix P were initialized to value 1000, all other elements were set to 100. measurement noise covariance matrix R was set to $R = 1/Pr * I$ where $\mathrm{Pr} = 0.01$ and output 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 acquiring steady state, no significant NNL improvement occurs after 10 epochs in any experiment. Partial derivatives needed for Jacobian matrix H were calculated by the RTRL algorithm. Although standalone RTRL doesn't follow the true gradient and should be used with small learning rate [17], EKF with RTRL Jacobian matrix calculation worked fine.

Different training algorithm 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 NNL fluctuation is 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 methods used by Lawrence, Giles and Fong [6], but the improvements weren't significant and didn't made RTRL training as stable and fast as EKF training (taking into account number of epochs). Momentum rate for Context-Free Language with Deep Recursion Deep Recursion was disabled (set to 0) and 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 CERandRBR Data Set momentum rate was set to 0.90 and learning rate to 0.05.

7 Results

We present results of 10 simulations performed with best parameters. In this work we present mean and st. dev. of NNL results of 10 simulations with fixed parameters and different initial values. Although few simulations have been suppressed because of high NNL fluctuation (some RTRL and also EKF simulations), final results represents general behavior for the parameters described in the previous section. Figure 3 and 2 shows results obtained on CERandRBR and Deep Recursion data set respectively. Generally, NNL performances of networks trained by EKF are better. We were able to train few networks 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 meny networks, more then thousand of training epochs, extensive experimenting with learning and momentum rate for individual simulation).

Recurrent neural networks (RNNs) behave as iterated function system [4, 5] and have properties of Markov model (MM) even before training [7, 11, 12]. It is important to compare results obtained by RNNs with results given by fixed order Markov models or variable length Markov models [10]. Markov models NNL results as a function of numer of context are shown in figure 4.

For CERandRBR dataset, both RTRL and EKF training were insensitive to training parameters and acquired similar results. This data set is inspired by Christiansen and Cahrter's simulations in cognitive field. If we take into account the size of the input alphabet (17 symbols), length of the training data set (23366 symbols) is small. Although NNL performance of trained SRN is better then NNL obtained by Markov models, the improvement is small and the amount of information obtained during training is small. ≥ 20.

Training SRN on Deep Recursion data set significantly improves NNL performance. Deep recursions require sort of counting mechanisms that are not present in MMs. Comparing results of trained SRN with results obtained with Markov models reveals that SRN acquired significant amount of information during training. Actually resulting NNLs are similar to analytically calculated ones.

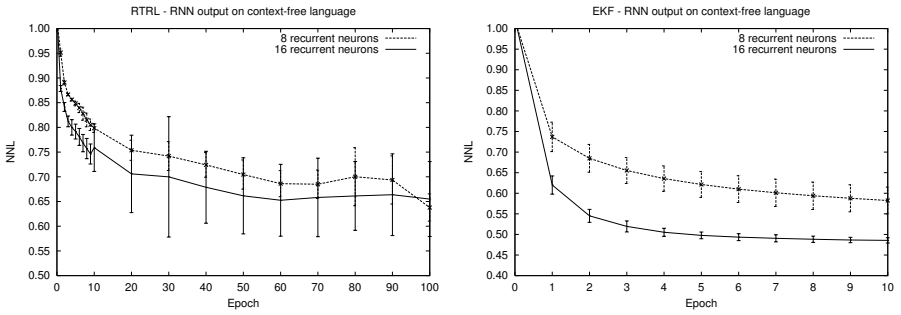

Figure 2: NNL performance of SRN output layer on Deep Recursion data set.

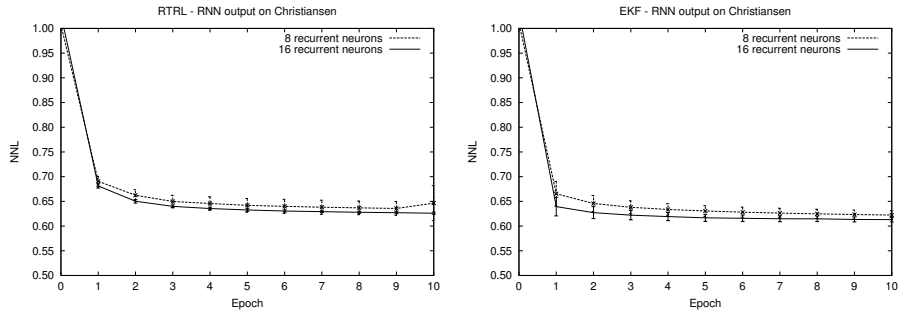

Figure 3: NNL performance of SRN output layer on Deep Cristiansen and Charter's data set.

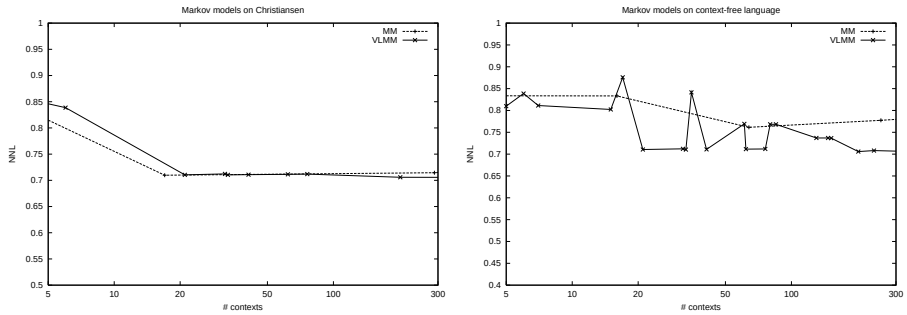

Figure 4: NNL performance of Markov models with fixed and variable length memory.

8 Conclusion

Extended Kalman filter shows significantly faster convergence in terms of number of epochs and resulting NNLs are better. Standard devitaion of results obtained by RTRL algorithm are high revealing RTRL's sensitivity to initial weight setting. Although computationally more difficult, extended Kalman filter approach to trainig recurrent networks on symbolic sequencies shows higher robustness and better resulting performance. It is easy to implement and should be considered when choosing approprieate algorithm for recurrent neural network training.

References

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

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

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

[4] J.F. Kolen. The origin of clusters in recurrent neural network state space. In *Proceedings from the Sixteenth Annual Conference of the Cognitive Science Society*, pages 508–513. Hillsdale, NJ: Lawrence Erlbaum Associates, 1994.

[5] J.F. Kolen. Recurrent networks: state machines or iterated function systems? In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A.S. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 203–210. Erlbaum Associates, Hillsdale, NJ, 1994.

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

[7] M. Čerňanský and L. Beňušková. Finite-state reber automaton and the recurrent neural networks trained in supervised and unsupervised manner. In H. Bischof Lecture Notes in Computer Science 2130. G. Dorffner and K. Hornik (Eds), editors, *Artificial Neural Networks - ICANN'2001*, pages 737–742. Springer-Verlag, 1998.

[8] P. S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. 1979.

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

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

[11] P. Tino, M. Cernansky, and L. Benuskova. Markovian architectural bias of recurrent neural networks, 2002.

[12] P. Tino and B. Hammer. Architectural bias in recurrent neural networks - fractal analysis. Technical Report NCRG/2002/010, NCRG, Aston University, UK, 2002.

[13] G. Welch and G. Bishop. An introduction to the kalman filter. Technical Report TR95-041, Department of Computer Science, University of North Carolina, 1995.

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

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

[16] R. J. Williams. Training recurrent networks using the extended Kalman filter. In *Proceedings of the International Joint Conference on Neural Networks, Baltimore*, June 1992.

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

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