Extension of Backpropagation through Time for Segmented-memory Recurrent Neural Networks

Stefan Glüge, Ronald Böck and Andreas Wendemuth
Faculty of Electrical Engineering and Information Technology, Chair of Cognitive Systems,
Otto von Guericke University Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany

Keywords: Recurrent Networks, Segmented-memory Recurrent Neural Networks, Backpropagation through Time, Real-time Recurrent Learning, Information Latching Problem, Vanishing Gradient Problem.

Abstract: We introduce an extended Backpropagation Through Time (eBPTT) learning algorithm for Segmented-Memory Recurrent Neural Networks. The algorithm was compared to an extension of the Real-Time Recurrent Learning algorithm (eRTRL) for these kind of networks. Using the information latching problem as benchmark task, the algorithms' ability to cope with the learning of long-term dependencies was tested. eRTRL was generally better able to cope with the latching of information over longer periods of time. On the other hand, eBPTT guaranteed a better generalisation when training was successful. Further, due to its computational complexity, eRTRL becomes impractical with increasing network size, making eBPTT the only viable choice in these cases.

1 INTRODUCTION

Conventional Recurrent Neural Networks suffer from the vanishing gradient problem in learning long-term dependencies (Bengio et al., 1994). To overcome this problem, the Segmented-Memory Recurrent Neural Network (SMRNN) architecture fractionises long sequences into segments. In the end, the single segments are connected in series and form the final sequence. The same procedure can be observed in human memorization, for instance, when people break up long numbers like telephone or bank account numbers in digits, such that 4051716 becomes 40 - 51 - 716.

Yet, SMRNNs are trained with an extended Real-Time Recurrent Learning (eRTRL) algorithm introduced by Chen and Chaudhari (2009). The underlying Real-Time Recurrent Learning algorithm (Williams and Zipser, 1989) has an average time complexity in order of magnitude $O(n^4)$, with n denoting the number of network units in a fully connected network (Williams and Zipser, 1995). Because of this complexity, the algorithm is often inefficient in practical applications where considerably big networks are used. Further, the time consuming training makes it difficult to perform a parameter search for the optimal number of hidden units, learning rate and so forth, for a specific application, cf. (Glüge et al., 2011).

In this paper we introduce an extension for the Backpropagation Through Time (Werbos, 1990) algorithm for SMRNNs, which we call extended Backpropagation Through Time (eBPTT). Compared to Real-Time Recurrent Learning, the Backpropagation Through Time algorithm has a much smaller time complexity of $O(n^2)$ (Williams and Zipser, 1995).

We compared both algorithms on a benchmark problem designed to test the ability of the networks to store information for a certain period of time. In comparison to eRTRL we found eBPTT less capable to learn the latching of information for long time periods. On the other hand, those networks that nonetheless were trained successful with eBPTT guaranteed better generalisation, that is, higher accuracy on the test set.

2 METHODS

The SMRNN architecture consists of two Simple Recurrent Networks (SRNs) (Elman, 1990) arranged in a hierarchical fashion as illustrated in Fig. 1. The first SRN processes the symbol level and the second the segment level of the input sequence.

Glüge S., Böck R. and Wendemuth A..
Extension of Backpropagation through Time for Segmented-memory Recurrent Neural Networks.
DOI: 10.5220/0004103804510456
In *Proceedings of the 4th International Joint Conference on Computational Intelligence* (NCTA-2012), pages 451-456
ISBN: 978-989-8565-33-4

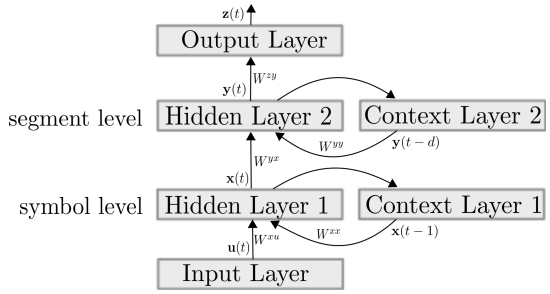

Figure 1: SMRNN topology.

2.1 Forward Processing in SMRNNs

We use the receiver-sender-notation to describe the processing in the network. The upper index of the weight matrices refer to the corresponding layer and the lower index to the single units. For example, W_{ki}^{xu} denotes the connection between the kth unit in hidden layer 1 (x) and the ith unit in the input layer (u) (cf. Fig. 1). Moreover, f_{net} is the transfer function of the network and n_u, n_x, n_y, n_z are the number of units in the input, hidden 1, hidden 2, and output layer.

The introduction of the parameter d on segment level makes the main difference between a cascade of SRNs and an SMRNN. It denotes the length of a segment, which can be fixed or variable. The processing of an input sequence starts with the initial symbol level state $\mathbf{x}(0)$ and segment level state $\mathbf{y}(0)$. At the beginning of a segment (segment head SH) $\mathbf{x}(t)$ is updated with $\mathbf{x}(0)$ and input $\mathbf{u}(t)$. On other positions $\mathbf{x}(t)$ is obtained from its previous state $\mathbf{x}(t-1)$ and input $\mathbf{u}(t)$. It is calculated by

$$x_k(t) = \begin{cases} f_{\text{net}}\left(\sum_j^{n_x} W_{kj}^{xx} x_j(0) + \sum_i^{n_u} W_{ki}^{xu} u_i(t)\right), \text{ if SH} \\ f_{\text{net}}\left(\sum_j^{n_x} W_{kj}^{xx} x_j(t-1) + \sum_i^{n_u} W_{ki}^{xu} u_i(t)\right), \\ \qquad\qquad \text{otherwise} \end{cases} \tag{1}$$

where $k = 1, \ldots, n_x$. The segment level state $\mathbf{y}(0)$ is updated at the end of each segment (segment tail ST) as

$$y_k(t) = \begin{cases} f_{\text{net}}\left(\sum_j^{n_y} W_{kj}^{yy} y_j(t-1) + \sum_i^{n_x} W_{ki}^{yx} x_i(t)\right), \\ \qquad\qquad \text{if ST} \\ y_k(t-1), \text{ otherwise} \end{cases} \tag{2}$$

where $k = 1, \ldots, n_y$. The network output results in forwarding the segment level state

$$z_k(t) = f_{\text{net}}\left(\sum_j^{n_y} W_{kj}^{zy} y_j(t)\right) \quad \text{with} \quad k = 1, \ldots, n_z\,. \tag{3}$$

While the symbol level is updated on a symbol by symbol basis, the segment level changes only after d symbols. At the end of the input sequence the segment level state is forwarded to the output layer to generate the final output. The dynamics of an SMRNN processing a sequence is shown in Fig. 2.

2.2 Extension of BPTT for SMRNNs

In the following, we describe how to adapt online Backpropagation Through Time to SMRNNs. That is, the error at the output at the end of a sequence is used instantaneously for weight adaptation of the network. Learning is based on minimizing the sum of squared errors at the end of a sequence of N segments,

$$E(t) = \begin{cases} \sum_{k=1}^{n_z} \frac{1}{2}(z_k(t) - d_k(t))^2, \text{ if } t = Nd \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

where $d_k(t)$ is the desired output and $z_k(t)$ is the actual output of the kth unit in the output layer.

The error is propagated back through the network and also back through time to adapt the weights. Further, it is not reasonable to keep the initial states $\mathbf{y}(0) = f_{\text{net}}(\mathbf{a}^{yy}(0))$ and $\mathbf{x}(0) = f_{\text{net}}(\mathbf{a}^{xx}(0))$ fixed, thus, the initial activations $\mathbf{a}^{yy}(0)$ and $\mathbf{a}^{xx}(0)$ are also learned. Here, the upper index of the activations refer to the corresponding layer and a lower index to the single units. For example, a_k^{yx} is the activation at the kth unit in the hidden layer 2 that results from connections from the hidden layer 1, which is simply $a_k^{yx}(t) = \sum_i^{n_x} W_{ki}^{yx} x_i(t)$.

The gradient of $E(t)$ can be computed from the injecting error

$$e_k(t) = z_k(t) - d_k(t). \tag{5}$$

Using the back propagation procedure we compute the delta error. The $\delta_k(t)$ is a short hand for $\partial E(t)/\partial a_k$ representing the sensitivity of $E(t)$ to small changes of the kth unit activation. The deltas for the output units δ^{zy}, hidden layer 2 units δ^{yy}, and hidden layer 1 units δ^{yx} at the end of a sequence ($t = Nd$) are

$$\delta_k^{zy}(t) = f'_{\text{net}}(a_k^{zy}(t)) e_k(t), \tag{6}$$

$$\delta_k^{yy}(t) = f'_{\text{net}}(a_k^{yy}(t)) \sum_{i=1}^{n_z} W_{ik}^{zy} \delta_i^{zy}(t) \tag{7}$$

$$\delta_k^{yx}(t) = f'_{\text{net}}(a_k^{yx}(t)) \sum_{i=1}^{n_z} W_{ik}^{zy} \delta_i^{zy}(t). \tag{8}$$

At that point, we unroll the SMRNN on segment level to propagate the error back in time. The state of the hidden layer 2 changes *only* at the end of a segment $t = nd$ and $n = 0, \ldots, N-1$. Therefore, the delta error for the hidden layer 2, and hidden layer 1 units results

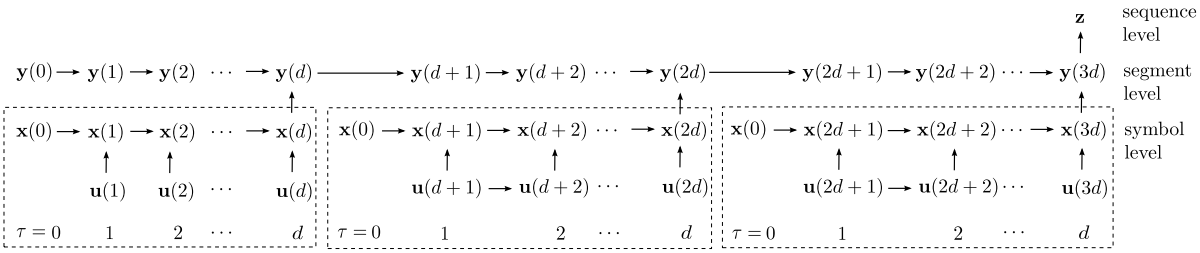

Figure 2: SMRNN dynamics for a sequence of three segments with fixed interval d. Processing on symbol level is described by Eq. 1 and illustrated in the dashed squares. The segment level above, is processes as described by Eq. 2. Finally, Eq. 3 describes how the network output z is obtained for the sequence.

in

$$\delta_k^{yy}(nd) = f'_{\text{net}}(a_k^{yy}(nd)) \sum_{i=1}^{n_y} W_{ik}^{yy} \delta_i^{yy}((n+1)d), \quad (9)$$

$$\delta_k^{yx}(nd) = f'_{\text{net}}(a_k^{yx}(nd)) \sum_{i=1}^{n_y} W_{ik}^{yy} \delta_i^{yy}((n+1)d). \quad (10)$$

Once the computation was performed down to the beginning of the sequence ($t = 0$), the gradient of the weights and initial activation on segment level is computed by

$$\Delta W_{ij}^{zy} = \delta_i^{zy}(Nd) y_j(Nd), \quad (11)$$

$$\Delta W_{ij}^{yy} = \sum_{n=1}^{N} \delta_i^{yy}(nd) y_j((n-1)d), \quad (12)$$

$$\Delta W_{ij}^{yx} = \sum_{n=2}^{N} \delta_i^{yx}(nd) x_j((n-1)d), \quad (13)$$

$$\Delta a_i^{yy} = \delta_i^{yy}(0). \quad (14)$$

For the adaptation of the weights on symbol level we apply the Backpropagation Through Time procedure repetitively for every time step $\tau = 0, \ldots, d$ for every segment of the sequence. That is, for the end of a segment ($\tau = d$)

$$\delta_k^{xx}(d) = f'_{\text{net}}(a_k^{xx}(d)) \sum_{i=1}^{n_y} W_{ik}^{yx} \delta_i^{yx}(d), \quad (15)$$

$$\delta_k^{xu}(d) = f'_{\text{net}}(a_k^{xu}(d)) \sum_{i=1}^{n_y} W_{ik}^{yx} \delta_i^{yx}(d), \quad (16)$$

and for $\tau < d$ we get

$$\delta_k^{xx}(\tau) = f'_{\text{net}}(a_k^{xx}(\tau)) \sum_{i=1}^{n_x} W_{ik}^{xx} \delta_i^{xx}(\tau+1), \quad (17)$$

$$\delta_k^{xu}(\tau) = f'_{\text{net}}(a_k^{xu}(\tau)) \sum_{i=1}^{n_x} W_{ik}^{xx} \delta_i^{xx}(\tau+1). \quad (18)$$

When the computation was performed to the beginning of a segment ($\tau = 0$), the gradient of the weights and initial activation on symbol level is computed by

$$\Delta W_{ij}^{xx} = \sum_{\tau=1}^{d} \delta_i^{xx}(\tau) x_j(\tau-1), \quad (19)$$

$$\Delta W_{ij}^{xu} = \sum_{\tau=2}^{d} \delta_i^{xu}(\tau) u_j(\tau-1), \quad (20)$$

$$\Delta a_i^{xx} = \delta_i^{xx}(0). \quad (21)$$

Note that the sums in Eq. 13 and 20 start at $n = 2$ and $\tau = 2$, respectively. This is due to the fact, that at time $t = 0$ the hidden layer 2 has no input from hidden layer 1 and further, hidden layer 1 has no input from the input layer (cf. Fig. 2).

The computed gradients can be used right away to change the networks weights and initial activations to

$$\tilde{W}_{ij} = W_{ij} - \alpha \Delta W_{ij} + \eta \Delta' W_{ij} \quad (22)$$

with a learning rate α and the momentum term η. The value $\Delta' W_{ij}$ represents the change of W_{ij} in the previous iteration. Figure 3 illustrates the error flow in the SMRNN for one sequence of length Nd.

2.3 Information Latching Problem

Typically dynamic systems change the output on current or immediate past inputs. Nevertheless, it is often desired that even inputs that occurred much earlier affect the system's output. The information latching problem was designed to test a system's ability to model dependencies of the output on earlier inputs (Bengio et al., 1994). In this context, "information latching" refers to the storage of information in the internal states of the system over some period of time.

The task is to distinguish two classes of sequences where the class C of the sequence $i_1, i_2, \ldots, i_T$ depends on the first L items

$$C(i_1, i_2, \ldots, i_T) = C(i_1, i_2, \ldots, i_L) \in \{0, 1\} \text{ with } L < T \quad (23)$$

For our experiments the class-defining start of a sequence had a fixed length of $L = 50$. To test the

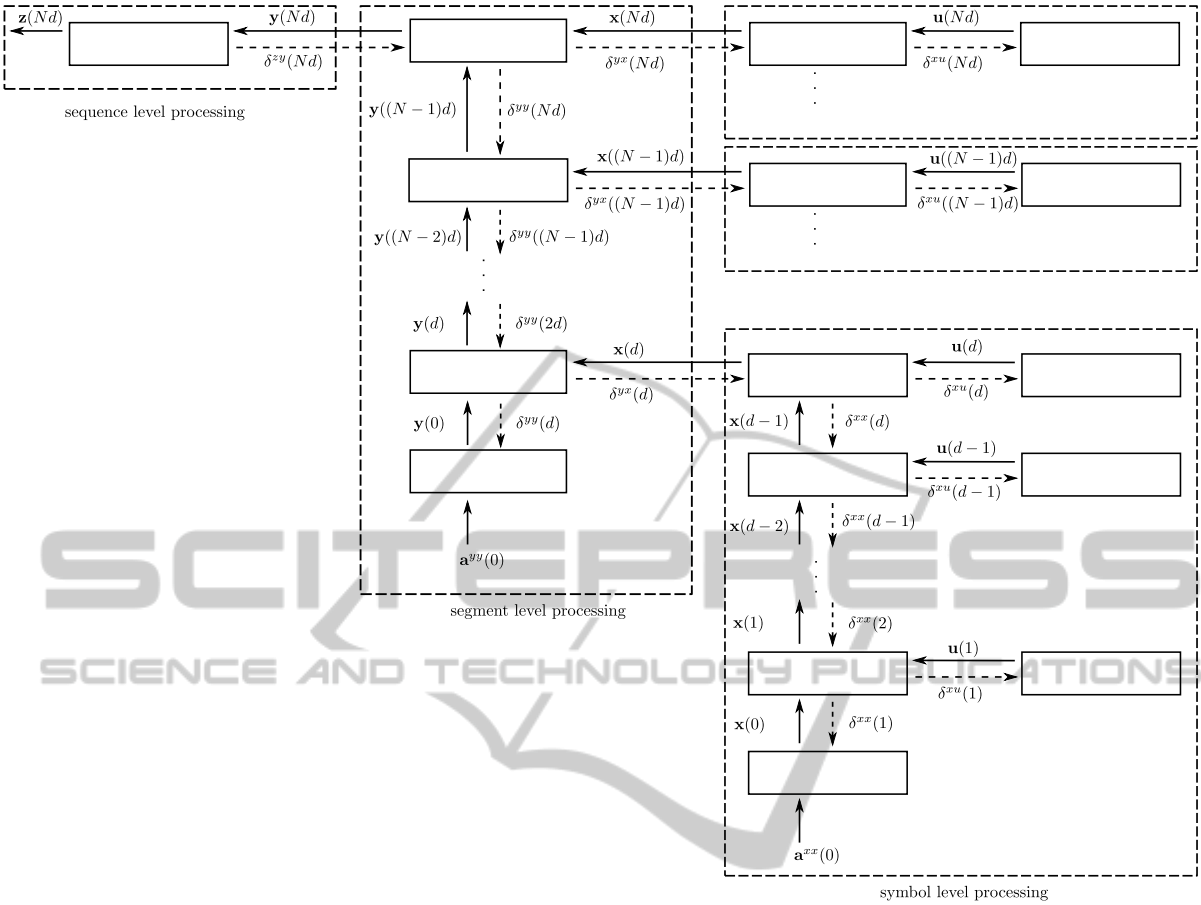

Figure 3: Errorflow of the eBPTT algorithm in an SMRNN for a sequence of length Nd. The solid arrows indicate the development of the states of the layers in the network. The dashed arrows show the propagation of the error back through the network and back through time.

networks ability to store the initial inputs over an arbitrary period of time, we gradually increased the length of the sequence T. The sequences were generated from an alphabet of 26 letters (a-z), such that the number of input neurons was 26 (1-of-N coding). A sequence was considered to be class $C = 1$ if the items $i_1, i_2, \ldots, i_L$ match a predefined string $s_1, s_2, \ldots, s_L$, otherwise it was class $C = 0$. All items i of a sequence that were not predefined were chosen randomly from the alphabet.

For each sequence length T we created two sets of sequences for training and testing. With increasing length of the sequences T the set of training and test samples was enlarged to ensure generalisation.

3 RESULTS

For every sequence length T we trained 100 networks with eRTRL (Chen and Chaudhari, 2009) and eBPTT, respectively. This was done to determine the algorithms' ability to learn the task in general. In every epoch the sequences of the training set were shown in a random order.

The networks' configuration and the size of the training/test sets are adopted from (Chen and Chaudhari, 2009) where SMRNNs and SRNs are compared on the information latching problem. Accordingly, the SMRNNs are comprised of $n_u = 26$ input units, $n_x = n_y = 10$ units in the hidden layers, and one output unit ($n_z = 1$). Further, the length of a segment was set to $d = 15$ and the sigmoidal transfer function $f_{\text{net}}(x) = 1/(1+\exp(-x))$ was used for the hidden and output units. The input units simply forwarded the input data which were $\in \{-1, 1\}$. Initial weights were set to uniformly distributed random values in the range of $(-1, 1)$.

Learning rate and momentum for each algorithm were chosen after testing 100 networks on all combinations $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$ and $\eta \in \{0.1, 0.2, \ldots, 0.9\}$ on the shortest sequence $T = 60$.

The combinations that yielded the highest mean accuracy on the test set were chosen for the experiment, that is $\alpha = 0.1$, $\eta = 0.4$ for eRTRL and $\alpha = 0.6$, $\eta = 0.5$ for eBPTT.

Training was stopped when the mean squared error of an epoch fell below 0.01 and thus, the network was considered to have successfully learned the task. For other cases training was cancelled after 1000 epochs. Table 1 shows the results for eRTRL and eBPTT for sequences of length T from 60 to 130.

The column for the number of successfully trained networks (#suc) in Tab. 1 clearly shows a decrease for eBPTT with the length of the sequences T. On the other hand, nearly all networks were trained successfully with eRTRL. Therefore, we can state that eRTRL is generally able to cope better with longer ranges of output dependencies than eBPTT.

The pure mean number of epochs (#eps) that were needed for training is somewhat misleading. Over the whole experiment eBPTT needs an average of 243.3 epochs for successful training while eRTRL needs only 67.1 epochs. It is important to note that this does not indicate that eRTRL training takes less time than eBPTT. The high computational complexity of Real-Time Recurrent Learning ($O(n^4)$), and therefore also of eRTRL, results in a much longer computation time for a single epoch compared to eBPTT. This becomes more and more evident with increasing network size. Figure 4 shows the time that is needed to train an SM-RNN for 100 epochs ($T = 60$, set size 50) depending on the number of neurons in the hidden layers[1]. For a considerable big network with $n_x = n_y = 100$ the training took about 3 minutes with eBPTT and 21.65 hours with eRTRL.

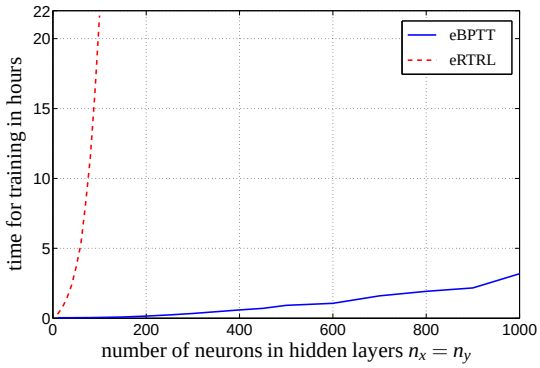

Figure 4: Computation time for training depending on the number of neurons in the hidden layers of the network. Training lasted 100 epochs of 50 sequences of length $T = 60$.

[1]Both algorithms were implemented in Matlab. Training was done on a AMD Opteron 8222 (3GHz), 8GB RAM, CentOS, Matlab R2011b (7.13.0.564) 64-bit.

The third column in Tab. 1 shows the performance of successfully trained networks on the test set (Acc.). For eBPTT we could observe higher accuracies than for eRTRL. It is also reflected by the overall accuracy of 96.8% for eBPTT compared to 89.2% for eRTRL. This implies, that successful learning with eBPTT guaranteed better generalisation.

4 DISCUSSION

Even though eRTRL was generally better able to cope with the latching of information over longer periods of time, the networks that finally learned the task with eBPTT showed higher accuracies on the test set.

Altogether, the question which learning algorithm to use for a specific task strongly depends on the character of the problem at hand. For small networks, as used for the experiment in Tab. 1, the choice depends on the timespan that has to be bridged. If we expect the output to be dependent on inputs that are comparatively shortly ago ($T = 60, \ldots, 100$) eBPTT provides the better choice. There is a high chance for a successful training of the network with a good generalisation. When the outputs depend on inputs that appeared long ago ($T > 130$), the eRTRL algorithm provides the better solution. It guarantees a successful network training where eBPTT could hardly train the network.

In real world problems, as speech recognition, handwriting recognition or protein secondary structure prediction the data to be classified has not such a compact representation as the strings in the information latching task. To be able to learn from such data the network size, that is, number of processing units, has to be increased. As shown in Fig. 4, eRTRL becomes simply impractical for large networks (training time: 3 minutes with eBPTT in contrast to 21.65 hours with eRTRL / $n_x = n_y = 100$). In these cases, eBPTT becomes the only viable choice of a training algorithm.

In future, the combination of both learning algorithms might be a possibility to overcome the drawbacks of both methods. It could reduce the computational complexity of eRTRL and increase eBPTT's ability to learn long-term dependencies.

ACKNOWLEDGEMENTS

The authors acknowledge the support provided by the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive

Table 1: Information latching problem with increasing sequence length T and fixed predefined string ($L = 50$). The size of sets for training/testing was increased too. 100 SMRNNs with parameters $n_x = n_y = 10$, $d = 15$ were trained on each sequence length. The number of networks that learned the task (#suc of 100) and the mean value of number of epochs (#eps) is shown together with the mean accuracy (Acc.) of successful networks on the test set and its standard deviation (Std. Dev.).

T	set size	eBPTT				eRTRL			
		#suc	#eps	Acc.	Std. Dev.	#suc	#eps	Acc.	Std. Dev.
60	50	79	230.6	0.978	0.025	100	44.3	0.978	0.025
70	80	58	285.7	0.951	0.047	100	63.9	0.861	0.052
80	100	61	215.2	0.974	0.024	100	66.2	0.862	0.088
90	150	48	240.4	0.951	0.123	100	52.4	0.940	0.044
100	150	43	241.4	0.968	0.018	100	82.1	0.778	0.065
110	300	36	250.0	0.977	0.049	100	69.6	0.868	0.052
120	400	17	305.4	0.967	0.050	100	56.7	0.950	0.040
130	500	14	177.6	0.978	0.017	96	101.4	0.896	0.078
mean			243.3	0.968	0.044		67.1	0.892	0.056

Technical Systems" funded by the German Research Foundation (DFG).

REFERENCES

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

Chen, J. and Chaudhari, N. S. (2009). Segmented-memory recurrent neural networks. *IEEE Transactions Neural Networks*, 20(8):1267–1280.

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

Glüge, S., Böck, R., and Wendemuth, A. (2011). Segmented-memory recurrent neural networks versus hidden markov models in emotion recognition from speech. In *International Conference on Neural Computation Theory and Applications (NCTA 2011)*, pages 308–315, Paris.

Werbos, P. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550 –1560.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Williams, R. J. and Zipser, D. (1995). *Gradient-based learning algorithms for recurrent networks and their computational complexity*, pages 433–486. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.