ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

# Auto-Encoder Pre-Training of Segmented-Memory Recurrent Neural Networks

Stefan Glüge, Ronald Böck and Andreas Wendemuth

Faculty of Electrical Engineering and Information Technology
Cognitive Systems Group, Otto von Guericke University Magdeburg
and Center for Behavioral Brain Science
Universitätsplatz 2, 39106 Magdeburg, Germany

**Abstract**. The extended Backpropagation Through Time (eBPTT) learning algorithm for Segmented-Memory Recurrent Neural Networks (SMRNNs) yet lacks the ability to reliably learn long-term dependencies. The alternative learning algorithm, extended Real-Time Recurrent Learning (eRTRL), does not suffer this problem but is computational very intensive, such that it is impractical for the training of large networks. The positive results reported with the pre-training of deep neural networks give rise to the hope that SMRNNs could also benefit of a pre-training procedure. In this paper we introduce a layer-local pre-training procedure for SMRNNs. Using the information latching problem as benchmark task, the comparison of random initialised and pre-trained networks shows the beneficial effect of the unsupervised pre-training. It significantly improves the learning of long-term dependencies in the supervised eBPTT training.

## 1 Introduction

Conventional recurrent neural networks suffer from the vanishing gradient problem in learning long-term dependencies [1]. The Segmented-Memory Recurrent Neural Network (SMRNN) architecture approaches the problem based on the observation on human memorization. Yet, these networks may be trained either with extended Real-Time Recurrent Learning (eRTRL) [2] or extended Backpropagation Through Time (eBPTT) [3]. Because of the time complexity in order of magnitude $\mathcal{O}(n^4)^*$ of the original Real-Time Recurrent Learning algorithm [4] the extended version for SMRNNs is unsuitable in practical applications where considerably big networks are used. For comparison, the original Backpropagation Through Time has a time complexity of $\mathcal{O}(n^2)$ [4].

A comparison of both algorithms on the information latching problem showed that eBPTT is generally less capable to learn long-term dependencies than eRTRL. Nevertheless, a successfull training with eBPTT led to a better generalisation compared to eRTRL, i.e. higher accuracy on the test set [3].

In this paper we show that an unsupervised layer-local pre-training improves eBPTT's ability to learn long-term dependencies significantly preserving the good generalisation performance.

*$n$ denoting the number of network units of a fully connected network

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

## 2 Methods

### 2.1 Forward processing in SMRNNs

The SMRNN architecture consists of two Simple Recurrent Networks (SRNs) 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. In the

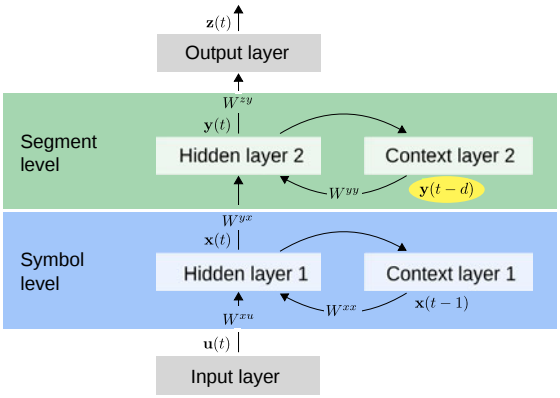


Fig. 1: SMRNN topology: The parameter $d$ on segment level makes the difference between a cascade of SRNs and an SMRNN. Only after a segment of length $d$ the segment level context is updated.

following, we use the receiver-sender-notation. The upper indices of the weight matrices refer to the corresponding layer and the lower indices to the single units. For example, $W^{xu}_{ki}$ denotes the connection between the $k^{\text{th}}$ unit in hidden layer 1 ($x$) and the $i^{\text{th}}$ unit in the input layer ($u$) (cf. Fig. 1). Moreover, $f_{\text{net}}$ 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

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

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

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

segment (segment tail ST) as

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

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.

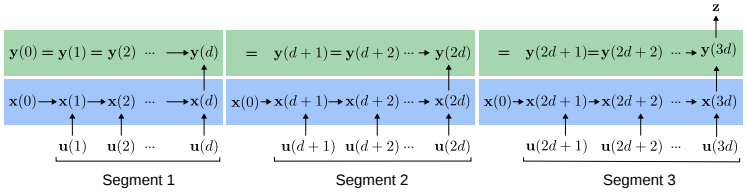


Fig. 2: SMRNN dynamics for a sequence of three segments with fixed interval $d$

### 2.2 Pre-Training of Segmented-Memory Recurrent Neural Networks

The positive results reported with the pre-training of deep neural networks [5] give rise to the hope that SMRNNs could also benefit of a pre-training procedure. Generally, it should lead to initial weights that lie in a region of the parameter space where it is more likely to find a solutions.

If we look on an SMRNN as a stack of two SRNs (cf. Fig. 1) the idea of a layer-local pre-training seems pretty natural. Even though, the architecture itself may not be regarded as *deep* in the conventional way, the recurrent character of a hidden – context layer pair allows the composition of a complex non-linear operation. Therefore, such layer-pair can be viewed as being *deep* in itself. Anyway, unfolded in time a recurrent network can be seen as a very deep multi-layer feedforward neural network.

Following the idea of layer-local pre-training, the single SRNs on symbol and segment level are separately trained as auto-encoders. In that way, the procedure does not differ from the pre-training of multi-layer feedforward neural networks. However, as the segment level processes the input of the symbol level only at the end of a segment, only these symbol level outputs are used for the segment level pre-training. So, for segment length $d$ every $d^{\text{th}}$ output of the symbol level auto-encoder SRN is used for the segment level pre-training. Hereafter, the

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

initialised weights are used as starting point for the supervised training. Figure 3 illustrates the pre-training procedure.

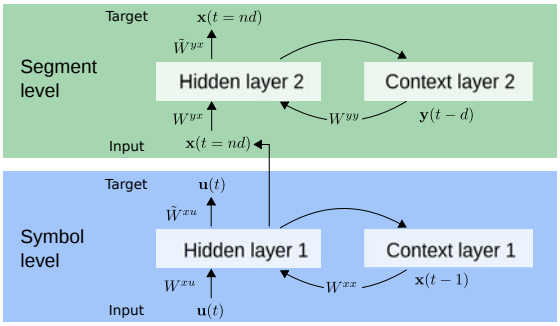


Fig. 3: Layer-local pre-training of an SMRNN. Each SRN's weights are initialised separately by training as an auto-encoder ($W^{xu}, W^{xx}, W^{yx}, W^{yy}$). The output of the symbol level SRN serves as input for the segment level SRN ($\mathbf{x}(t = nd)$ with $n = 1, \ldots, N$), which is trained in the same way.

### 2.3 Information Latching Problem

The information latching problem was designed by [1] to test a system's ability to model dependencies of the output on earlier inputs. In this context, "information latching" refers to the storage of information in the system's internal states for some time. Basically, it is a sequence classification problem. The idea 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

$$C(i_1, i_2, \ldots, i_T) = C(i_1, i_2, \ldots, i_L) \in \{0, 1\} \text{ with } L < T \tag{4}$$

The sequences were generated from an alphabet of 26 letters (a - z), such that the number of input units was $n_u = 26$ (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. As the class label was provided at the end of each sequence, the network needs to bridge at least $T - L$ time steps to relate the label to the class-defining string. So, if $L$ is kept fixed the problem gets harder with increasing sequence length $T$.

## 3 Results

For the experiment a fixed string $L = 50$ was used and the length of the sequence $T$ was increased gradually. For each sequence length $T$ two sets for training and testing were created. The sets were enlarged with increasing $T$ to ensure generalisation. Further, 100 networks were pre-trained and random initialised

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

(uniformly distributed random values in the range of $(-1, 1)$), respectively. This was done for every sequence length $T$. The sequences of the training set were shown in a random order in every epoch of the training.

During pre-training each SRN was trained to reproduce its input with the scaled conjugate gradient backpropagation algorithm [6] (cf. Fig. 3). Pre-training was stopped after 1000 epochs or when validation performance has increased more than six times since the last time it decreased. After pre-training, the weights $(W^{xu}, W^{xx}, W^{yx}, W^{yy})$ were used for the supervised eBPTT training on the information latching problem.

The networks' configuration and the size of the training/test sets were adopted from [2] where SMRNNs and SRNs are compared on the information latching problem. Accordingly, the SMRNNs comprised of $n_u = 26$ input units, $n_x = n_y = 10$ hidden layer units, and one output unit $n_z = 1$. Further, the length of a segment was set to $d = 15$ and the hyperbolic tangent $f_{\text{net}}(x) = \tanh(x)$ was applied in the hidden layers and the sigmoidal transfer function $f_{\text{net}}(x) = 1/\left(1 + \exp(-x)\right)$ was used for the output unit. The input units simply forwarded the input data $\mathbf{u}(t) \in \{-1, 1\}$. Learning rate $\alpha = 0.2$ and momentum $\eta = 0.1$ were used. This combination yielded the highest mean accuracy on the test set 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.

Supervised 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 sequences of length $T$ between 60 and 130.

Table 1: 100 SMRNNs were trained on each sequence length $T$. 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 of successful networks on the test set (ACC) and its standard deviation (STD).

| $T$ | set size | random initialised | | | | pre-trained | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | #suc | #eps | ACC | STD | #suc | #eps | ACC | STD |
| 60 | 50 | 80 | 122.6 | 0.966 | 0.061 | 91 | 69.8 | 0.968 | 0.034 |
| 70 | 80 | 83 | 80.3 | 0.962 | 0.040 | 96 | 41.9 | 0.971 | 0.047 |
| 80 | 100 | 65 | 123.3 | 0.968 | 0.038 | 95 | 31.5 | 0.979 | 0.039 |
| 90 | 150 | 41 | 180.3 | 0.978 | 0.022 | 77 | 29.4 | 0.977 | 0.053 |
| 100 | 150 | 37 | 147.1 | 0.971 | 0.023 | 82 | 40.3 | 0.979 | 0.051 |
| 110 | 300 | 26 | 204.2 | 0.980 | 0.010 | 75 | 55.6 | 0.981 | 0.057 |
| 120 | 400 | 16 | 239.6 | 0.954 | 0.123 | 49 | 32.4 | 0.987 | 0.028 |
| 130 | 500 | 6 | 194.8 | 0.987 | 0.011 | 52 | 39.2 | 0.977 | 0.069 |
| mean | | 44.5 | 161.5 | 0.972 | 0.041 | 77.1 | 42.5 | 0.977 | 0.047 |

One can observe the decrease of successfully trained networks with the length of the sequences $T$. This general trend holds for random initialised as well as

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

for pre-trained SMRNNs. However, the pre-trained networks do not suffer from that behaviour as much as the random initialised. For the longest sequence $T = 130$, 52 of 100 networks were trained successfully when pre-trained, while only 6 of 100 were obtained from random initialisation. The accuracy on the test set (ACC) is not influenced by the pre-training, as it does not differ significantly for both cases. Further, pre-trained networks needed less epochs (#eps) for the supervised training, but this saving was spent on pre-training.

## 4 Discussion

The main result of the experiment is that pre-training improves eBPTT's ability to learn long-term dependencies significantly. It reduces the chance to get stuck in local minima or plateaus and therefore increases the number of successfully trained networks (#suc in Tab. 1). This makes the computational intensive eRTRL dispensable, even for task where the outputs depend on inputs that appeared long ago. Compared to eRTRL, eBPTT guarantees better generalisation with less time consuming training (cf. [3]).

The pre-training procedure mainly effected the direct forward connections $W^{xu}$ and $W^{xy}$ of the SMRNNs, while the context weights $W^{xx}$ and $W^{yy}$ tended to zero. This supports the conclusion that during pre-training only representations of the actual input vectors were learned in the hidden layers and no temporal dependency was found between them.

Pre-training showed no effect on the generalisation error. This may be a consequence of the very low error, i.e. high accuracy, that is already achieved with random initialised weights. There is the possibility of a positive effect of pre-training on the generalisation error in a different scenario, for instance, a more complex sequence classification task.

Altogether, pre-training extends the area of application of eBPTT to long(er)-term dependencies in sequence classification tasks.

## References

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

[2] J. Chen and N. S. Chaudhari. Segmented-memory recurrent neural networks. *Neural Networks, IEEE Transactions*, 20(8):1267–80, August 2009.

[3] S. Glüge, R. Böck, and A. Wendemuth. Extension of backpropagation through time for segmented-memory recurrent neural networks. In *Proceedings of the 4th International Joint Conference on Computational Intelligence*, pages 451–456, 2012.

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

[5] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[6] M. F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, 1993.