This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier’s archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/authorsrights
Learning long-term dependencies in segmented-memory recurrent neural networks with backpropagation of error

Stefan Glüge, Ronald Böck, Günther Palm, 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
School of Life Science and Facility Management, Institute of Applied Simulation, Zurich University of Applied Sciences, Einsiedlerstrasse 31a, 8820 Wädenswil, Switzerland
Faculty of Computer Science, Institute of Neural Information Processing, Ulm University, 89069 Ulm, Germany

Abstract

In general, recurrent neural networks have difficulties in learning long-term dependencies. The segmented-memory recurrent neural network (SMRNN) architecture together with the extended real-time recurrent learning (eRTRL) algorithm was proposed to circumvent this problem. Due to its computational complexity eRTRL becomes impractical with increasing network size. Therefore, we introduce the less complex extended backpropagation through time (eBPTT) for SMRNN together with a layer-local unsupervised pre-training procedure. A comparison on the information latching problem showed that eRTRL is better able to handle the latching of information over longer periods of time, even though eBPTT guaranteed a better generalisation when training was successful. Further, pre-training significantly improved the ability to learn long-term dependencies with eBPTT. Therefore, the proposed eBPTT algorithm is suited for tasks that require big networks where eRTRL is impractical. The pre-training procedure itself is independent of the supervised learning algorithm and can improve learning in SMRNN in general.

1. Introduction

Conventional recurrent neural networks (RNNs) have difficulties in modelling the so-called long-term dependencies, i.e., learning a relationship between inputs that may be separated over several time steps. Since the mid-1990s a lot of research effort was put on the investigation of this problem, e.g., [1–6]. Usually recurrent networks are trained with a gradient based learning algorithm like back-propagation through time (BPTT) [7] and real-time recurrent learning (RTRL) [8]. Bengio et al. [1] and Hochreiter [9] found that the error gradient vanishes when it is propagated back through time and back through the network, respectively. There are basically two ways to circumvent this vanishing gradient problem. One possibility is to use learning algorithms that simply do not use gradient information, e.g., simulated annealing [1], cellular genetic algorithms [10] and the expectation-maximisation algorithm [11]. Alternatively, a variety of network architectures were suggested to tackle the vanishing gradient problem, e.g., second-order recurrent neural network [12], non-linear autoregressive model with exogenous inputs recurrent neural network (NARX) [3,13], hierarchical recurrent neural network [2], long short-term memory (LSTM) network [14], anticipation model [15], echo state network [16,17], latched recurrent neural network [5], recurrent multiscale network [18,19], modified distributed adaptive control (DAC) architecture [20], and segmented-memory recurrent neural network (SMRNN) [6,21].

Encouraging results with SMRNN have been reported on the problem of emotion recognition from speech [22] and protein secondary structure (PSS) prediction [6,21]. In [6] it was shown that SMRNN performs competitive to LSTM on an artificial benchmark problem (two-sequence problem). Further, bidirectional SMRNN outperforms bidirectional LSTM networks on PSS prediction. The SMRNN training is essentially gradient descent. Therefore, it does not get rid of the vanishing gradients, but attenuates the problem. A comprehensive discussion on the effect of the segmented memory is given in [6]. This paper addresses the gradient based training of SMRNNs. Basically, the architecture fractionates long sequences into segments. Then, these segments form the final sequence if connected in series. Such procedure can be observed in human memorisation of long sequences, e.g., for phone numbers.

So far, SMRNNs are trained with an extended real-time recurrent learning (eRTRL) algorithm [6]. The underlying RTRL algorithm has

* Corresponding author.
E-mail address: stefan.gluege@zhaw.ch (S. Glüge).

http://dx.doi.org/10.1016/j.neucom.2013.11.043 0925-2312/c 2014 Elsevier B.V. All rights reserved.
an average time complexity in the order of magnitude \(O(n^4)\), with \(n\) denoting the number of network units in a fully connected network [23]. Because of this complexity, the algorithm is often inefficient in practical applications where considerably big networks are used, as the time consuming training prohibits a complete parameter search for the optimal number of hidden units, learning rate, and so forth.

In this paper we adapt BPTT for SMRNNs, calling it extended backpropagation through time (eBPTT). Compared to RTRL, the underlying BPTT algorithm has a much smaller time complexity of \(O(n^2)\) [23]. We compared both algorithms on a benchmark problem designed to test the network’s ability to store information for a certain period of time. In comparison to eRTRL we found eBPTT being less capable to learn the latching of information for longer periods of time. However, those networks that were trained successfully with eBPTT showed a better generalisation as the time consuming training prohibits a complete parameter search for the optimal number of hidden units, learning rate, and so forth.

The remainder of the paper is organised as follows. Section 2 introduces the SMRNN architecture together with the eBPTT training algorithm. Further, the layer-local pre-training procedure is described and the information latching benchmark problem is introduced. Following this, Section 3 provides experimental results on the benchmark problem for eRTRL and eBPTT training. Further, randomly initialised and pre-trained networks with subsequent supervised eBPTT training are tested. Additionally, we investigate the effect of the pre-training and alternative weight initialisation procedures. Finally, the results are discussed in Section 4 and some concluding remarks on future work are given in Section 5.

2. Methods

2.1. Segmented-memory recurrent neural network

The basic limitation of gradient descent learning for the weight optimisation in recurrent networks led to the development of alternative network architectures. One particular approach is the segmented-memory recurrent neural network (SMRNN) architecture proposed in [21]. From a cognitive science perspective, the idea has the pleasant property that it is inspired by the memorisation process of long sequences, as it is observed in humans. Usually people fractionate sequences into segments to ease memorisation. Afterwards, the single segments are combined to form the final sequence. For instance, telephone numbers are broken into segments of two or three digits, such that 7214789 becomes 72 - 14 - 789. This behaviour is not just plausible from everyday life, but evident in studies in the field of experimental psychology [24–28].

The SMRNN architecture mimics this behaviour. It consists of two simple recurrent networks (SRNs) [29] arranged in a hierarchical fashion as illustrated in Fig. 1. A sequence of inputs is presented to the network symbol by symbol, i.e., input vector by input vector. Separate internal states store the symbol level context (short-term information) as well as the segment level context (long-term information). The symbol level state \(x(t)\) is updated for each input \(u(t)\), while the segment level state \(y(t)\) is updated at the end of a segment.

In the following the receiver–sender-notation is used to describe the processing in the network. The upper indices of the weight matrices refer to the corresponding layer and the lower indices to the single units. For example, \(W_{uy}^{k}\) denotes the connection between the \(k\)th unit in hidden layer 1 \((x)\) and the \(k\)th unit in the input layer \((u)\) (cf. Fig. 1). Moreover, \(f\) is the transfer function of the network’s units, and \(n_u, n_x, n_y, n_t\) are the number of units in the input, hidden 1, hidden 2, and output layers respectively.

The introduction of the parameter \(d\) on segment level distinguishes a cascade of SRNs from 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 \(x(0)\) and segment level state \(y(0)\). At the beginning of a segment (segment head = SH) \(x(t)\) is updated with \(x(0)\) and input \(u(t)\). On other positions \(x(t)\) is obtained from its previous state \(x(t-1)\) and input \(u(t)\). It is calculated by

\[
x_k(t) = \begin{cases} 
  f\left(\sum_{j} W_{uj}^k x_j(0) + \sum_{i} W_{ui}^k u_i(t)\right) & \text{if } t = \text{SH} \\
  f\left(\sum_{j} W_{uj}^k x_j(t-1) + \sum_{i} W_{ui}^k u_i(t)\right) & \text{else,}
\end{cases}
\]

where \(k = 1, \ldots, n_x\). The segment level state \(y(t)\) keeps its value during the processing of a segment and is updated at the end of each segment (segment tail = ST):

\[
y_k(t) = \begin{cases} 
  f\left(\sum_{j} W_{yj}^k y_j(t-1) + \sum_{i} W_{ui}^k x_i(t)\right) & \text{if } t = \text{ST} \\
  y_k(t-1) & \text{else,}
\end{cases}
\]

where \(k = 1, \ldots, n_y\). The network output is obtained by forwarding the segment level state:

\[
z_k(t) = f\left(\sum_{j} W_{yz}^k y_j(t)\right) \quad \text{with } k = 1, \ldots, n_z.
\]

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.

Concerning the segment length \(d\) for a sequence of length \(T\) the SMRNN turns into a recurrent network with multiple hidden layers if \(d > T\). For \(d = 1\) one gets a recurrent network with multiple hidden layers and multiple feedback connections. The advantage of a segmented memory and the slower vanishing gradient occurs only if \(1 < d < T\). In other words, the length of the interval \(d\) affects the performance of an SMRNN. If it is too small or too large it fails to bridge long time lags. Obviously, the optimal value for \(d\) is task-dependent, so the choice depends on a priori knowledge of the typical time lag size [6].
2.2. Extension of backpropagation through time for SMRNNs

The high computational complexity of RTRL [23], and therefore eRTRL [6], makes it impractical for applications where considerably big networks are used. In the following, we introduce an extension for the BPTT algorithm [7] which has a much smaller time complexity. For this, we adapt real-time BP to SMRNN, i.e., the output’s error at the end of a sequence is used instantaneously for weight adaptation of the network.

First of all, to describe the algorithm, we define the activation of a network unit as the sum of its weighted inputs, e.g., $a^y_k(t) = \sum_b W^y_{bk} x_k(t)$. Here, the upper indices refer to the corresponding layer and the lower index to the single unit, i.e., $a^y_k$ is the activation at the $k$th unit in hidden layer 2 ($y$) that results from connections from hidden layer 1 ($x$).

Generally, to train the network, the error signal is propagated back through the network and back through time to adapt the network weights $W^y$, $W^x$, $W^{yx}$, and $W^{zy}$. Further, it is not reasonable to keep the initial states $y(0) = f(a^y(0))$ and $x(0) = f(a^x(0))$ fixed, thus, the initial activations $a^y(0)$ and $a^x(0)$ are also learned.

Training is based on minimizing the sum of squared errors at the end of a sequence of $N$ segments:

$$ E(t) = \sum_{k=1}^{N} (z_k(t) - y_k(t))^2 \quad \text{if } t = N \text{d} $$
$$ E(t) = 0 \quad \text{else,} $$

where $y_k(t)$ is the target value and $z_k(t)$ is the actual output of the $k$th unit in the output layer. The gradient of the error $E(t)$ can be computed using the injecting error $e_k(t) = z_k(t) - y_k(t)$.

Applying backpropagation we compute the delta error. Here $\delta_k(t)$ is a short hand for $\partial E(t)/\partial a_k(t)$, representing the sensitivity of $E(t)$ to small changes of the $k$th unit activation. The deltas for the output units $\delta^y$, hidden layer 2 units $\delta^x$, and hidden layer 1 units $\delta^x$ at the end of a sequence (t=Nd) are

$$ \delta_k^y(t) = f'(a^y_k(t)) e_k(t) $$

$$ \delta_k^x(t) = f'(a^x_k(t)) \sum_i W^{zy}_{ik} \delta_i^y(t) $$

$$ \delta_k^y(t) = f'(a^y_k(t)) \sum_i W^{yx}_{ik} \delta_i^x(t) $$

Note that Eqs. (7) and (8) are basically the same, besides the fact that the error is weighted with the different activations resulting from the context layer 2 $a^y$ or hidden layer 1 $a^x$.

At that point we enroll 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 $t=nd$, where $n=0, \ldots, N-1$. Therefore, the delta errors for the hidden layer 2 and hidden layer 1 units result in

$$ \delta_k^y(nd) = f'(a^y_k(nd)) \sum_{i=1}^{N_d} W^{zy}_{ik} \delta_i^y((n+1)d) $$

$$ \delta_k^x(nd) = f'(a^x_k(nd)) \sum_{i=1}^{N_d} W^{yx}_{ik} \delta_i^x((n+1)d). $$

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

$$ \Delta W^y_{yi} = \delta_k^y(nd) y_i(0) $$

$$ \Delta W^x_{yi} = \sum_{n=1}^{N} \delta_k^y(nd) y_i((n-1)d) $$

$$ \Delta W^{yx}_{yi} = \sum_{n=2}^{N} \delta_k^y(nd) x_i((n-1)d) $$

$$ \Delta a^y_i = \delta_k^y(0). $$

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

$$ \delta_k^y(d) = f'(a^y_k(d)) \sum_{i=1}^{N_d} W^{zy}_{ik} \delta_i^y(d) $$

$$ \delta_k^x(d) = f'(a^x_k(d)) \sum_{i=1}^{N_d} W^{yx}_{ik} \delta_i^x(d). $$

Further, for $\tau < d$ we get

$$ \delta_k^y(\tau) = f'(a^y_k(\tau)) \sum_{i=1}^{N_d} W^{zy}_{ik} \delta_i^y(\tau + 1) $$

$$ \delta_k^x(\tau) = f'(a^x_k(\tau)) \sum_{i=1}^{N_d} W^{yx}_{ik} \delta_i^x(\tau + 1). $$

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

$$ \Delta W^y_{yi} = \sum_{\tau=1}^{d} \delta_k^y(\tau) y_i(\tau-1) $$

$$ \Delta W^{yx}_{yi} = \sum_{\tau=2}^{d} \delta_k^x(\tau) x_i(\tau-1) $$

$$ \Delta a^y_i = \delta_k^y(0). $$

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

---

**Fig. 2.** SMRNN dynamics for a sequence of three segments with fixed segment length $d$. 

$$ y(0) = y(1) = y(2) \rightarrow y(d) $$

$$ x(0) = x(1) = x(2) \rightarrow x(d) $$

$$ z $$

**Segment 1**

**Segment 2**

**Segment 3**
and initial activations:
\[
\tilde{W}_{ij} = W_{ij} - \alpha \Delta W_{ij} + \eta \Delta \tilde{W}_{ij},
\]
where \(\alpha\) denotes the learning rate and \(\eta\) the momentum term. The value \(\Delta W_{ij}\) represents the change of \(W_{ij}\) in the previous iteration. The gradients may also be applied epoch-wise, i.e., for an epoch of \(s = 1, \ldots, M\) sequences
\[
\tilde{W}_{ij} = W_{ij} - \alpha \left( \sum_{s=1}^{M} \Delta W_{ij}(s) \right) + \eta \Delta \tilde{W}_{ij}.
\]

Fig. 3 illustrates the error flow in the SMRNN for one sequence of length \(Nd\). Unfolded in time the context layers of the network are represented by multiple hidden layers, as the context is simply the previous state of a hidden layer.

2.3. Auto-encoder pre-training of SMRNN

Generally, the idea of an unsupervised layer-local pre-training arose in the context of deep neural networks, i.e., feedforward networks with many hidden layers [30]. Experimental results [31,32] show that the training of deep architectures is considerably difficult compared to shallow architectures. Using the standard random weight initialisation deep architectures generally produce poor training and generalisation results [31]. This suggests that the supervised gradient-based training of deep multi-layer networks gets stuck in local minima or plateaus. The effect gets even worse when the architecture is deeper.

The success of layer-local unsupervised pre-training for deep architectures reported in the literature [30] may be due to several effects. Generally, it may help to guide the parameters of the layers towards better regions in parameter space. Of course, the question “What are ‘better regions’?” arises in this context. To be more specific, unsupervised pre-training has a regularising effect that leads to regions in the parameter space where solutions are allowed. That means, the solution is “near” those of the unsupervised training, that is near a solution that captures statistical structure of the input [32]. Other experiments in [32] show that the effect of unsupervised pre-training is an advantage mostly for the lower layers of a deep architecture, as these layers are poorly trained with random parameter initialisation.

With a better initialisation of the lower layers by a pre-training, the training, as well as the generalisation error of the supervised training can be lowered significantly. Bengio [30] hypothesises that in a “well trained” deep neural network the hidden layers form a good representation of the input data, which helps in obtaining good predictions. Further, the pre-training can be seen as a way to decompose the problem into sub-problems with different levels of abstraction. One layer of unsupervised learning could extract features which are regarded as low-level features, due to the limited capacity of one layer. Hence, learning a second layer using the previous layer’s output/features should result in hierarchically higher-level features. Therefore, one could imagine that higher-level abstractions of the input emerge in a deep architecture [30].

If we look on an SMRNN as a stack of two SRNs (cf. Fig. 1) the idea to apply a layer-local pre-training seems 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. Unfolded in time a recurrent network can be seen as a very deep multi-layer neural network. Hence, it is reasonable to assume that the SMRNN architecture could also benefit of a pre-training procedure.

Following the idea of layer-local pre-training [30] the single SRNs on symbol and segment level are separately trained as auto-encoders. This means that each SRN is trained to reproduce its input at the output. In that way, the procedure does not differ from pre-training of multi-layer feedforward networks. However, as the segment level processes the symbol level state 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 \(d\)th output of the symbol level auto-encoder SRN is used for the segment level pre-training. Note that the symbol level state \(x\), not the desired target \(u\) of the symbol level, is used as segment level input. After

![Fig. 3. Errorflow of the bPTT 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.](image-url)
pre-training the initialised weights are used as starting points for the supervised training. Fig. 4 illustrates this pre-training procedure. It is independent of the learning algorithm that is used in the supervised training and vice versa.

2.4. Information latching problem

To evaluate the capabilities and limitations of the proposed eBPTT algorithm (cf. Section 2.2) we used the information latching problem. This benchmark problem was designed to test a system’s ability to model dependencies of the output on earlier inputs [1]. 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:

\[
C(i_1, i_2, \ldots, i_T) = C(0, 0, \ldots, 0) \quad \text{with} \quad L < T. \tag{24}
\]

Chen and Chaudhari [6] used the task to compare SRN and SMRNN regarding their capability to robustly store information over longer periods of time. In the experiment, SMRNNs were able to capture much longer dependencies than SRNs, while both were trained with eRTRL and RTRL. We used the same experimental setup for a comparison of SMRNN training with eRTRL and eBPTT. Further, we used the task to evaluate the unsupervised pre-training procedure described in Section 2.3.

The sequences were generated from an alphabet of 26 letters (a-z), such that the number of input units was \( n_i = 26 \) (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 assigned to class \( C = 0 \). All items \( i \) of a sequence that were not predefined were chosen randomly from the alphabet. Table 1 illustrates the problem for a sequence of length \( T = 22 \) with a class-defining string of length \( L = 10 \).

![Fig. 4. Layer-local pre-training of an SMRNN. Each SRN’s weights are initialised separately by training as an auto-encoder (\( W^{xx}, W^{yy}, W^{yx}, W^{xy} \)). The output of the symbol level SRN serves as input for the segment level SRN (\( x(t = nd) \)) with \( n = 1, \ldots, 9 \)), which is trained in the same way.](image)

### Table 1

Information Latching problem for a sequence of length \( T = 22 \) with a class-defining string of length \( L = 10 \).

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Class C</th>
</tr>
</thead>
<tbody>
<tr>
<td>predefined random string</td>
<td>1</td>
</tr>
<tr>
<td>random string</td>
<td>0</td>
</tr>
<tr>
<td>h d g h r t z u s z j i t n o e r v y q d f</td>
<td>0</td>
</tr>
<tr>
<td>predefined gu k w a r n g t o h d</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3. Results

3.1. eBPTT Versus eRTRL on the information latching problem

From the literature we know that BPTT is extremely advantageous compared to RTRL concerning the computational costs for network training, i.e., \( O(n^2) \) compared to \( O(n^4) \), with \( n \) denoting the number of network units in a fully connected network [22]. As these algorithms underlie their extended equivalents for the SMRNN training, we have the educated guess that eBPTT requires far less computation than eRTRL. In the following, we compare their ability to deal with the learning of long-term dependencies on the basis of the information latching problem introduced in Section 2.4.

The class label is only provided at the end of each sequence. Therefore, the network needs to bridge at least \( T - L \) time steps to relate the label to the class-defining string. Hence, if \( L \) is kept fixed the problem gets harder with increasing sequence length \( T \). For the evaluation a fixed string of length \( L = 50 \) was used. Further, the length of the sequence \( T \) was increased gradually, testing the networks’ ability to store the initial inputs for periods of time that become longer. For each selected sequence length \( T \) two sets for training and testing were created. The sets were enlarged with increasing \( T \) to ensure generalisation. To determine the algorithms’ ability to learn the task in general, 100 networks were trained with eRTRL and eBPTT. This was done for each sequence length \( T \) on the created training sets. Further, the sequences of the training and test sets were shown in a random order in every epoch of the training and during testing.

The networks’ configuration and the size of the training/test sets were adopted from [6] where SMRNNs and SRNs are compared on the information latching problem. Accordingly, the SMRNNs comprised of \( n_i = 26 \) input units, \( n_o = 10 \) hidden layer units, and one output unit \( n_{o} = 1 \). The length of a segment was set to \( d = 15 \) and the sigmoidal transfer function \( f(x) = 1/(1 + \exp(-x)) \) was used for the hidden and output units. The input units simply forwarded the input data which were \( e = (-1, 1) \). Initial weights were set to uniformly distributed random values in the range of \((-1, 1)\). The network output was assigned to one class by the boundary at 0.5, i.e.,

\[
C = 1 \text{ if } z(t) \geq 0.5 \quad \text{and} \quad C = 0 \text{ if } z(t) < 0.5. \tag{25}
\]

Learning rate and momentum for each algorithm were chosen after training of 100 networks for each combination of \( \alpha \in \{0.1, 0.2, \ldots, 0.9\} \) and \( \eta \in \{0.1, 0.2, \ldots, 0.9\} \) on the set of the shortest sequence \( T = 60 \). We used the shortest sequences for practical reasons, as these require the smallest amount of samples to ensure generalisation. By that, we could try all \( \alpha-\eta \)-combinations within a reasonable amount of time. Those combinations that yielded the highest mean accuracy over 100 networks on the test set were chosen for the comparison on longer sequences, i.e., \( \alpha = 0.1, \eta = 0.4 \) for eRTRL and \( \alpha = 0.6, \eta = 0.5 \) for eBPTT. We are aware that the learning rate and the momentum found by this procedure must not be optimal for longer sequences with \( T > 60 \). However, as we tend to compare both algorithms it must not be optimal, but comparable and be found in a reproducible manner.

The training was stopped when the mean squared error of an epoch fell below 0.01 and thus, the network was considered to have successfully learnt the task. For other cases, training was cancelled after 1000 epochs and the network was considered to have not learnt the task. Table 2 shows the results for eRTRL and eBPTT on the information latching problem with fixed predefined string of length \( L = 50 \) and increasing sequence length \( T \). 100 SMRNNs were trained and tested on each sequence length \( T \) from 60 to 130.
was increased gradually. For each sequence length to reveal the effect of a pre-training on SMRNN. As in Section 3.1, a latching problem

3.2. Random initialised and pre-trained SMRNN on the information

weeks, such that the experiment was stopped at For larger networks the training with eRTRL would have taken

emphasis the order of magnitude that lies between the required

length $L$.

Table 2
eBPTT versus eRTRL: Information latching problem with fixed predefined string of length $L=50$ and increasing sequence length $T$. The size of training/test set was increased also to ensure generalisation. 100 SMRNNs were trained for each sequence length $T$. The number of networks that successfully learned the task ($\#\text{suc}$ of 100) and their mean value of number of epochs for training ($\#\text{eps}$) is shown together with their mean accuracy on the test set (ACC) and its standard deviation (STD).

<table>
<thead>
<tr>
<th>T</th>
<th>Set size</th>
<th>eBPTT</th>
<th>eRTRL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#suc</td>
<td>#eps</td>
<td>ACC</td>
</tr>
<tr>
<td>60</td>
<td>50</td>
<td>120.0</td>
<td>0.978</td>
</tr>
<tr>
<td>70</td>
<td>50</td>
<td>130.5</td>
<td>0.951</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
<td>121.2</td>
<td>0.951</td>
</tr>
<tr>
<td>90</td>
<td>150</td>
<td>240.4</td>
<td>0.951</td>
</tr>
<tr>
<td>100</td>
<td>150</td>
<td>241.4</td>
<td>0.968</td>
</tr>
<tr>
<td>110</td>
<td>150</td>
<td>250.0</td>
<td>0.977</td>
</tr>
<tr>
<td>120</td>
<td>400</td>
<td>305.4</td>
<td>0.967</td>
</tr>
<tr>
<td>130</td>
<td>500</td>
<td>48</td>
<td>177.6</td>
</tr>
<tr>
<td>Mean</td>
<td>44.5</td>
<td>243.3</td>
<td>0.968</td>
</tr>
</tbody>
</table>

The column entitled “#suc” in Table 2 clearly shows a decrease of successfully trained networks for eBPTT with the length of the sequences $T$. This means that the harder the task, i.e., the longer the time span that has to be bridged by the network, the less networks could be trained successfully with eBPTT. In contrast, nearly all networks were trained successfully with eRTRL. Therefore, we can state that eRTRL is generally better able to cope with longer ranges of input–output dependencies than eBPTT.

The third column in Table 2 shows the performance of successfully trained networks on the test set (ACC). For each single training/test set we observed higher accuracies for eBPTT than for eRTRL. This is further reflected by the overall accuracy of 96.8% for eBPTT compared to 89.2% for eRTRL. Hence, we may state that successful learning with eBPTT guaranteed better generalisation.

Regarding the training time, the mean number of epochs ($\#\text{eps}$) that were needed for training can be somewhat misleading. Over the whole experiment eBPTT needs an average of 243.3 epochs for a 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 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. To illustrate that, Fig. 5 shows the time that is needed to train an SMRNN for 100 epochs ($T=60$, set size 50) depending on the number of units in the hidden layers.1 Note that the time-axis is scaled logarithmically to emphasise the order of magnitude that lies between the required training time of both algorithms. For a network with $n_x = n_y = 100$ the training took about 3 min with eBPTT and 21.65 h with eRTRL. For larger networks the training with eRTRL would have taken weeks, such that the experiment was stopped at $n_x = n_y = 100$ for practical reasons.

3.2. Random initialised and pre-trained SMRNN on the information latching problem

We used the information latching problem described in Section 2.4 to reveal the effect of a pre-training on SMRNN. As in Section 3.1, a fixed string of length $L=50$ 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 and enlarged with increasing $T$. The enlargement is necessary to ensure generalisation. For each training/test set 100 networks were pre-trained and randomly initialised.

Pre-training of the symbol and segment level SRN was performed as described in Section 2.3 and depicted in Fig. 4. Each network was trained to reproduce its input at the output with the scaled conjugate gradient backpropagation algorithm [33]. For the pre-training, as for the supervised training afterwards, the same data was used. From this training material, 80% were shown for pre-training and 10% for validation and validation of the pre-training. Even though auto-encoders learn a direct input–output mapping, the sequences were presented one after another, such that the sequential structure of the material is preserved. This is desirable as SRNs are able to learn the temporal structure of the data implicitly [34]. Pre-training was stopped after 1000 epochs or when the validation performance has increased more than six times since the last time it decreased. Such early stopping is widely used, because it is comparably easy to implement and was reported to be superior to regularisation methods like Weight Decay [35] and Weight Elimination [36], for instance in [37].

After pre-training the weights ($W^{00}$, $W^{01}$, $W^{11}$, $W^{10}$) were used for the supervised eBPTT training on the information latching problem.

For the random initialised networks the weights were set to uniformly distributed values in the range of $\pm 1$. Afterwards supervised training was done with eBPTT. The sequences were shown in a random order in every epoch of the training and during testing.

The same network configuration as in Section 3.1 was used for the experiment. That is $n_x = 26$ input units, $n_y = 10$ hidden layer units, and one output unit $n_z = 1$. The input units, again, simply forwarded the input data $u(t) \in (-1, 1)$. Further, the length of a segment was set to $d=15$ and the output layer used the sigmoidal transfer function $f(x) = 1/(1 + \exp(-x))$. It is important to note that unlike in Section 3.1, the hyperbolic tangent $f(x) = \tanh(x)$ was applied in the hidden layers. This is advantageous as a symmetric activation function around 0 allows error gradients to flow backwards more easily [38]. We forbore to use the hyperbolic tangent in the comparison of eBPTT and eRTRL in Section 3.1, as we intended to use exactly the same network structure that was used in the original article [6] which introduced eRTRL.

With the change of the transfer function in the hidden layers, the optimal combination of learning rate and momentum for the eBPTT training had to be determined again. Therefore, 100

\[\text{Table 2}\]

<table>
<thead>
<tr>
<th>$T$</th>
<th>Set size</th>
<th>eBPTT</th>
<th>eRTRL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$#\text{suc}$</td>
<td>$#\text{eps}$</td>
<td>ACC</td>
</tr>
<tr>
<td>60</td>
<td>50</td>
<td>120.0</td>
<td>0.978</td>
</tr>
<tr>
<td>70</td>
<td>50</td>
<td>130.5</td>
<td>0.951</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
<td>121.2</td>
<td>0.951</td>
</tr>
<tr>
<td>90</td>
<td>150</td>
<td>240.4</td>
<td>0.951</td>
</tr>
<tr>
<td>100</td>
<td>150</td>
<td>241.4</td>
<td>0.968</td>
</tr>
<tr>
<td>110</td>
<td>150</td>
<td>250.0</td>
<td>0.977</td>
</tr>
<tr>
<td>120</td>
<td>400</td>
<td>305.4</td>
<td>0.967</td>
</tr>
<tr>
<td>130</td>
<td>500</td>
<td>48</td>
<td>177.6</td>
</tr>
<tr>
<td>Mean</td>
<td>44.5</td>
<td>243.3</td>
<td>0.968</td>
</tr>
</tbody>
</table>

Fig. 5. Computation time for training depending on the number of units in the hidden layers $n_x$, $n_y$.
networks were initialised randomly and trained with each combination $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$ and $\eta \in \{0.1, 0.2, \ldots, 0.9\}$ on the training set of the shortest sequence $T = 60$. The combination $\alpha = 0.2$ and $\eta = 0.1$ achieved the highest mean accuracy on the test set and thus, was used in the further experiment.

The supervised 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 3 shows the results for pre-trained and randomly initialised SMRNNs on the information latching problem with a fixed predefined string of length $L = 50$ and increasing sequence length $T$. 100 SMRNNs were trained and tested on each sequence length $T$ from 60 to 130.

As before in Table 2 for eBPTT, we observed a decrease of successfully trained networks with an increase of the length of the sequences $T$. This general trend holds for randomly initialised as well as for pre-trained SMRNNs. However, the pre-trained networks do not suffer from this behaviour as much as the randomly initialised networks and thus, pre-training improves eBPTT’s ability to learn long-term dependencies. For the longest sequences $T = 130$, 59 of 100 networks were trained successfully when pre-trained, whereas only 6 of 100 were obtained from random initialisation. The mean accuracy on the test set (ACC) and its standard deviation (STD) does not differ significantly for both cases and confirm the results for eBPTT in Table 2.

Table 4 shows the results of this experiment. The reset to zero of the context weights after pre-training apparently had no effect on the result of the supervised training. Other than some individual variations, there is no significant difference in the number of successfully trained networks (#suc), nor a significant difference in accuracy (ACC) and number of epochs (#eps) (cf. Table 4). This result supports the impression that we gained from Fig. 6b and d, that no temporal dependency in the input was learned during pre-training. In the next section we further discuss that point.

### 3.4. Alternative context weight initialisation

It seems that the auto-encoder pre-training procedure mainly affects the direct forward connections of the SMRNN, while the context weights tend to zero (cf. Fig. 6). Even a complete reset of the context weights had no significant effect on the subsequent supervised training (cf. Table 4). This supports the conclusion that during auto-encoder pre-training, only representations of the actual input vectors are learnt in the hidden layer weights ($W^\text{lu}$ and $W^\text{ly}$). At the same time, no temporal dependencies between those representations are learnt.

This is not a general behaviour of SRNs, but rather a consequence of the pre-training as auto-encoder. Symbol level as well as segment level SRNs do not need to encode temporal relations between their inputs to reproduce the actual input at the output. Therefore, the networks learn a direct representation of the input in the hidden layer. Further, due to the complexity of the temporal dependencies between the single inputs, this context is not learnt implicitly, which results in context weights around zero [34].

### 3.4.1. Forcing error propagation through context weights

For the task of information latching, the learning of representations for the single characters in the hidden layers during pre-training as auto-encoder turned out to be beneficial (cf. Table 3). However, the zeroed context weights, resulting from such pre-training, are rather counter-intuitive. The class of a string is limited interest as it is known a priori. It is plotted first and foremost for the comparison against the weight distribution after pre-training. Even though Fig. 6 shows the weight distributions for the SMRNNs trained on the information latching problem with sequence length $T = 120$, the same behaviour was observed for all other lengths $T = 60, \ldots, 130$.

For the direct forward connections of the SMRNNs $W^\text{lu}$ (Fig. 6a) and $W^\text{ly}$ (Fig. 6c), pre-training shifted the weights towards a distribution which appears Gaussian-shaped with a mean near zero. Compared to the initial random distribution, most of the weights attained smaller absolute values around zero, while a small number of weights attained bigger absolute values greater than 1. This leads to the hypothesis that the representation of the input in the hidden layers after pre-training is based on a rather small number of weights with large absolute values.

Interestingly, weights in the recurrent connections on symbol level and segment level, i.e., $W^\text{ux}$ (Fig. 6b) and $W^\text{yx}$ (Fig. 6d), show a different trend. They are still uniformly distributed after pre-training but in a smaller range. This means, at average, pre-training uniformly lowered the absolute value of all weights in the context layer. In other words, the weights tend to zero. According to [34] vanishing context weights indicate that there is no temporal dependency in the input. Therefore, we can assume that, for the task at hand, learning during pre-training mainly took place in the direct forward connections of the network $W^\text{lu}$ and $W^\text{ly}$.

We tested this hypothesis by setting the context weights $W^\text{ux}$ and $W^\text{yx}$ to zero after pre-training. Then the supervised training was done as before in Section 3.2.
The important information, determining the class, is the keyword at the beginning of the string. Therefore, it seems deceptive to use zeroed context weights, preventing a flow of information from one state to its previous state, or in other words, from the end of the string to its beginning.

A straightforward solution for this dilemma is the replacement of the context weights after the pre-training as auto-encoder. By setting the recurrent weights $W_{xx}$ and $W_{yy}$ to the identity matrix, one can force the network to provide the previous network state, when processing the actual input. This should facilitate the information flow from the end of the sequence to its beginning during supervised learning. Metaphorically spoken, the supervised training is started with a network which has the basic assumption, that its previous internal state is as important as the actual input.

To evaluate this approach, the experiment from Section 3.3 was repeated. This time, after pre-training as auto-encoder, the context weights of the networks $W_{xx}$ and $W_{yy}$ were replaced by the corresponding identity matrices. The remaining parameters, as network configuration, training algorithms, learning rate, training/test material, etc., were kept unchanged. Table 5 shows the results obtained from plain pre-training as auto-encoder along with the results obtained if the context weights were replaced. The replacement of the context weights by the identity matrix increased the number of successfully trained networks (#suc) at average by 13.5% (79.8 versus 90.6). Further, there is no significant difference in the number of training epochs (#eps), nor a significant difference in accuracy (ACC) and its standard deviation (STD). In conclusion, the replacement of the recurrent weights of

---

**Table 4**

<table>
<thead>
<tr>
<th>$T$</th>
<th>Set size</th>
<th>Pre-trained as auto-encoder</th>
<th>Pre-trained as auto-encoder and context reset to zero</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#suc</td>
<td>#eps</td>
<td>ACC</td>
</tr>
<tr>
<td>60</td>
<td>50</td>
<td>89</td>
<td>53.7</td>
</tr>
<tr>
<td>70</td>
<td>80</td>
<td>98</td>
<td>43.8</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
<td>90</td>
<td>37.3</td>
</tr>
<tr>
<td>90</td>
<td>150</td>
<td>85</td>
<td>26.8</td>
</tr>
<tr>
<td>100</td>
<td>150</td>
<td>88</td>
<td>28.6</td>
</tr>
<tr>
<td>110</td>
<td>300</td>
<td>73</td>
<td>24.5</td>
</tr>
<tr>
<td>120</td>
<td>400</td>
<td>56</td>
<td>34.1</td>
</tr>
<tr>
<td>130</td>
<td>500</td>
<td>59</td>
<td>32.5</td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td>79.8</td>
<td>35.2</td>
</tr>
</tbody>
</table>
the SMRNNs by a simple identity matrix enforced the backpropagation of the error signal, and thereby supported learning of long-term dependencies.

It should be noted that the modification of the weights followed the discussion about the information latching problem. This implies prior knowledge about the nature of the task to be solved, which in most cases is not available. However, generally it might be a good idea to support the error propagation through time at the beginning of the supervised training by the manual setup of the weights for the recurrent connections.

3.4.2. Pre-training as predictor

One way to enforce sequence learning during pre-training could be an SRN pre-training as predictor. That is, the SRNs are pre-trained to predict the next input of the sequence. Concerning the learning task, this is essentially different from the pre-training as auto-encoder, where SRNs are pre-trained to predict the actual input of the sequence. In theory, the prediction task forces a network to learn the temporal relations between inputs.

Technically, the pre-training procedure itself does not differ very much from the auto-encoder pre-training introduced in Section 2.3 and illustrated in Fig. 4. The single SRNs on the symbol and segment level are separately trained as predictors. Again, the segment level processes the input of the symbol level only at the end of a segment. Hence, only these symbol level outputs are used for the segment level pre-training. Afterwards, the initialised weights are used as starting points for the supervised training. Fig. 7 illustrates the pre-training as predictor.

The procedure of a pre-training as predictor was evaluated on the information latching problem according to the evaluation of the pre-training as auto-encoder described in Section 3.2. The only difference was the weight initialisation by a pre-training as predictor (cf. Fig. 7) instead of pre-training as auto-encoder (cf. Fig. 4). All parameters such as network configuration, training algorithms, learning rate, training/test material, etc. remained unchanged. Table 6 shows the results obtained from random initialisation and pre-training as predictor side by side.

Apparently the pre-training as predictor provides only a slight advantage over randomly initialised weights. There is a small improvement of successfully trained networks (#suc), especially for longer sequences. However, compared to the pre-training as auto-encoder (cf. Table 3) it performs rather poorly.

<table>
<thead>
<tr>
<th>T</th>
<th>Set size</th>
<th>Pre-trained as auto-encoder</th>
<th>Pre-trained as auto-encoder and context replaced by identity matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#suc</td>
<td>#eps</td>
<td>ACC</td>
</tr>
<tr>
<td>60</td>
<td>50</td>
<td>89</td>
<td>53.7</td>
</tr>
<tr>
<td>70</td>
<td>80</td>
<td>98</td>
<td>43.8</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
<td>90</td>
<td>37.3</td>
</tr>
<tr>
<td>90</td>
<td>150</td>
<td>85</td>
<td>26.8</td>
</tr>
<tr>
<td>100</td>
<td>150</td>
<td>88</td>
<td>28.6</td>
</tr>
<tr>
<td>110</td>
<td>300</td>
<td>72</td>
<td>24.3</td>
</tr>
<tr>
<td>120</td>
<td>450</td>
<td>56</td>
<td>34.1</td>
</tr>
<tr>
<td>130</td>
<td>500</td>
<td>59</td>
<td>32.5</td>
</tr>
<tr>
<td>Mean</td>
<td>79.8</td>
<td>35.2</td>
<td>0.981</td>
</tr>
</tbody>
</table>

Table 5: Pre-trained SMRNNs with and without replacement of the context weights: information latching problem with fixed predefined string of length L=50 and increasing sequence length T. 100 pre-trained SMRNNs with and without a replacement of their context weights (Wxx, Wyy) by the identity matrix were trained with eBiPTT for each sequence length T. The number of successfully trained networks (#suc of 100) is shown together with the mean value of the number of training epochs (#eps). Further, the mean accuracy of the successfully trained networks on the test set (ACC) and its standard deviation (STD) are shown.

There are several aspects that might explain this behaviour. First of all, no direct representation of the input data can be learnt during pre-training as predictor. As the task is to predict the following input, the networks fail to learn a distinct representation for the single inputs. Further, after pre-training, only the input weights of the layers (Wxx, Wyy, Wxy, Wyx) are used as starting points for the supervised training. The output weights of the layers (Wxx, Wyy) are discarded (cf. Fig. 7).

This is not a problem for an auto-encoder, as input and output are the same. The mapping to be learnt is for instance x(t) → x(t). Hence, input weights and output weights, Wxx and Wyy, are symmetric. In other words, the output weights carry no further information concerning the representation of the input.

This is different for the predictor. Here, the input and the output of a layer differ. The mapping to be learnt is for instance x(t) → x(t+1) (cf. Fig. 7). Therefore, input and output weights of a layer do not hold the same information regarding the representation of the input. That means, after pre-training as predictor, the information that was learnt in the output weights Wxx and Wyy is not available for the supervised training.

Additionally, in the experiment, the context weights Wxx and Wyy still inclined to zero. Thus, the hope, that the prediction task forces the network to learn the temporal relations between inputs,
was not fulfilled. Still, no temporal relation between inputs could be learned during pre-training.

4. Discussion

The SMRNN architecture implements the concept of a segmented memory to alleviate the vanishing gradient problem. Together with this architecture, eRTRL was proposed for the network training. Unfortunately, it has a very high computational complexity such that it is impractical for the training of large SMRNNs (cf. Fig. 5). Alternatively eBPTT was introduced (Section 2.2). It does not suffer from the high computational costs of eRTRL, but the comparison on the information latching problem showed that eRTRL was generally better able to cope with the latching of information over longer periods of time (cf. Table 2). Nevertheless, those networks that finally learned the task with eBPTT showed higher accuracies on the test set.

From the research in the field of deep neural networks, we know about the positive effects of an unsupervised pre-training. It is reasonable to apply this idea to recurrent neural networks [39] as well as to SMRNNs, as those can be regarded to be deep architectures. The SMRNN architecture is a stack of two SRNs such that it suggests itself to attempt a pre-training as a stack of auto-encoders. The evaluation of random initialised and pre-trained SMRNNs on the information latching problem showed 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 (cf. Table 3). Further, the pre-training showed no effect on the generalisation error of the supervised eBPTT training, which may be a consequence of the very low error that is already achieved with random initialised weights. On the other hand, this means that there is the possibility that pre-training improves the generalisation error of a supervised eRTRL training. However, we did not apply pre-training to eRTRL as this would further increase the already unreasonable long training time. From our point of view, this is possible but impractical, especially with respect to real world applications where rather big networks might be used.

The auto-encoder pre-training procedure mainly affected the direct forward connections $W^{du}$ and $W^{dv}$ of the SMRNN. A Gaussian-shaped distribution of the forward connections after pre-training (cf. Fig. 6a and c) suggests to test a Gaussian weight initialisation to increase performance and avoid pre-training time.

The context weights $W^{cu}$ and $W^{cv}$ tended to zero (cf. Figs. 6b and d). Even a complete reset of the context weights had no significant effect on the subsequent supervised training (cf. Table 4). This supports the conclusion that during pre-training only representations of the actual input vectors were learned and no temporal dependency between them. This is not a general behaviour of SRNs, but rather a consequence of the pre-training as auto-encoder. The SRN does not need to encode temporal relations between the inputs just to reproduce the input at the output. Therefore, the network learns a direct representation of the input in the hidden layer. Further, due to the complexity of the temporal dependencies between the inputs, the SRNs ‘learn’ that there is no temporal relation between them, i.e., the context weights tend to zero [34]. Hence, the temporal characteristic of the input sequences was learned during the supervised eBPTT training.

A replacement of the context weights by the identity matrix could further increase the number of successfully trained networks (cf. Table 5). It should be noted that the modification of the weights followed the discussion about the information latching problem. This implies prior knowledge about the nature of the task to be solved, which in most cases is not available. However, generally it might be a good idea to support the error propagation through time at the beginning of the supervised training by the manual setup of the recurrent connections.

The pre-training as predictor could hardly improve learning, compared to a random weight initialisation. The networks could not be forced to learn the temporal relations between inputs during pre-training. Furthermore, the prediction task made it more difficult to learn a direct representation of the data in the forward connections, which resulted in a poor performance in supervised training.

Concerning the question which learning algorithm to choose we can conclude that for learning tasks which require big networks, i.e., a large number of processing units, eRTRL is impractical due to its computational complexity. Thus, eBPTT is the only viable choice of a training algorithm in these cases. Further, pre-training extends the area of application of eBPTT to long(er)-term dependencies in sequence classification. Compared to eRTRL it guarantees better generalisation with less time consuming training.

Finally, even though the information latching problem is a widely used benchmark task, it is exclusively constructed to test a system’s ability to store information over some period of time. The authors in [40] use a simple 2D-version of the problem, i.e., sequences consisting of zeros and ones. They consider it to be a trivial task, as it may be solved by random weight guessing.

The information latching problem in our work is much more complex, as we used sequences of letters (1-of-26) that had to be mapped onto 10 hidden units. Further, the successful application of SMRNNs with eRTRL in protein secondary structure prediction [6] and speech processing [22] shows that the architecture is applicable to nontrivial tasks.

5. Future work

For the future, pre-trained SMRNNs with a supervised eBPTT training should be applied to real world problems like handwriting recognition or speech processing. The UCI Machine Learning Repository [41] provides a variety of sequential and time series data for classification and regression tasks. It is a good starting point to study the assets and drawbacks of SMRNNs, because the classification/regression results for some of the datasets are already published. Therefore, those datasets may constitute a useful benchmark.

Furthermore, advanced pre-training methods may be applied to SMRNN, e.g., a mixture of pre-training as auto-encoder for the forward connections and as predictor for the recurrent connections. The possibilities that come with an unsupervised pre-training should be further explored.

Additionally, e.g., a text (sequence of characters) has several levels of representation like characters, words, paragraphs, etc. Therefore, it is desirable to represent this knowledge about the structure in the classifier. Transferred to the idea of SMRNNs one could extend the network architecture to multiple levels of segments, which correspond to the assumed levels of the hierarchy. Every additional level could be realised by an additional SRN on top of the SRNs of the underlying levels. Even further, one could think about having multiple symbol level layers, or multiple segment level layers, or both.

References