Neurocomputing 141 (2014) 54–64

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Learning long-term dependencies in segmented-memory recurrent neural networks with backpropagation of error

Stefan Glüge [a,b,*], Ronald Böck [a], Günther Palm [c], Andreas Wendemuth [a]

[a] 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
[b] School of Life Science and Facility Management, Institute of Applied Simulation, Zurich University of Applied Sciences, Einsiedlerstrasse 31a, 8820 Wädenswil, Switzerland
[c] Faculty of Computer Science, Institute of Neural Information Processing, Ulm University, 89069 Ulm, Germany

ARTICLE INFO

Article history:
Received 25 June 2013
Received in revised form
23 September 2013
Accepted 8 November 2013
Available online 8 April 2014

Keywords:
Recurrent neural networks
Segmented-memory recurrent neural network
Vanishing gradient problem
Long-term dependencies
Unsupervised pre-training

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/

an average time complexity in the order of magnitude $\mathcal{O}(n^4)$, 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 $\mathcal{O}(n^2)$ [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 than eRTRL, i.e., higher accuracy on the test set. In a second step, we show that an unsupervised layer-local pre-training improves eBPTT's ability to learn long-term dependencies significantly preserving the good generalisation performance. This is further accompanied by experimental results and a discussion concerning the effect of the pre-training and different weight initialisation techniques.

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 $\mathbf{x}(t)$ is updated for each input $\mathbf{u}(t)$, while the segment level state $\mathbf{y}(t)$ 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

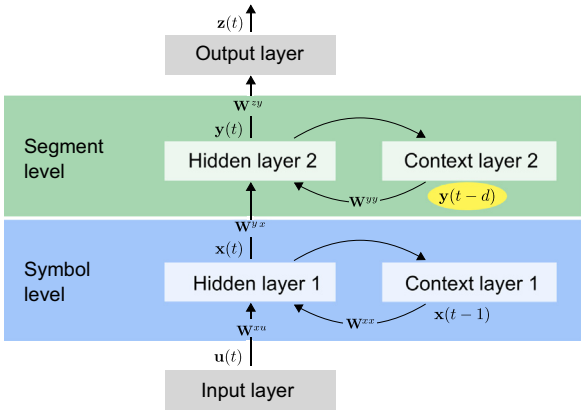

Fig. 1. SMRNN topology – two SRNs are arranged hierarchically. The parameter d on segment level makes the difference between a cascade of SRNs and an SMRNN. Only after a segment of length d the segment level state is updated.

indices to the single units. For example, W^{xu}_{ki} 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 is the transfer function of the network's units, and n_u, n_x, n_y, and n_z 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 $\mathbf{x}(0)$ 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

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

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

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

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

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

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.

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].