Introduction to Long Short-Term Memory Networks

A method description

Stefanie Urchs
University of Passau
Passau, Germany
urchs@fim.uni-passau.de

ABSTRACT

To enable networks to learn from experience recurrent neural networks were introduced. However these have problems with learning over long time steps due to vanishing and exploding gradients.
To answer the gradient problem Long Shot-Term Memory cells were introduced. These cells expanded the recurrent neural networks with gated units. Due to this units the error signals do not get backpropagated through the whole network and time. Thus the gradients do not tend to explode or vanish. This enables Long Short-Term Memory Networks (LSTM), as the expanded recurrent neural networks are called, to learn from long sequential data and to capture long time dependencies.
This paper discusses the approach of LSTM and explores their possible applications.

CCS CONCEPTS

• **Computer systems organization → Neural networks**;

1 INTRODUCTION

Recurrent neural networks (RNN) introduce recurrent layers to neural networks which make propagation of information through time possible. A problem of these propagated information enfolds when training the network. To compute the gradient in the network the information that was feed forward needs now to be propagated back in time. This leads to exploding or vanishing gradients, depending on the size of the weights [1]. Currently gated RNN like LSTM networks can handle the problems of exploding and vanishing gradients most effectively [8]. This paper explores how these long short-term memory networks are working. Furthermore, some applications, theoretical and in the real world, are explored.

2 RELATED WORK

The related work of LSTM include RNN and gated recurrent neural networks (gated RNN). LSTM are a evolution of RNN and gated RNN evolved from LSTM.

Recurrent Neural Networks have feedback connections that enable them to learn sequential patterns or time sequences. These kinds of networks are able to forecast financial data and electric power demand [13]. There are two kinds of architectures for recurrent neural networks. First the fully connected network. This one does not have a distinct input layer, all nodes get input from all other nodes. Furthermore, a node can send feedback to itself. Second the partially recurrent network. In this architecture distinct input and output layers exist. Some nodes are part of a feedforward structure, and there exist nodes that provide the sequential context. The weights of these nodes are processed with backpropagation. To provide the context they receive time-delayed feedback from other nodes. Data to train the network consist of inputs and the outputs of a desired successor [13].
Recursive neural networks are a generalisation of recurrent neural networks with a skewed tree structure. They work on directed acyclic graphs. The main use case for recursive neural networks is the sentence-level sentiment analysis [12].

Gated recurrent neural networks work with gated recurrent units (GRU). In contrast to ordinary recurrent units GRUs are able to process dependencies through time. A GRU is activated by a linear interpolation between the previous activation and the candidate activation. The candidate activation is processed similar to the ordinary recurrent unit. Furthermore, the GRU incorporates an update gate that is responsible for the amount of update in the activation or content. To reset the previously computed state forget gates exist [2].

Both discussed networks are suited to compute information over time. However,RNNs suffer from the problems of exploding and vanishing gradients [1]. Gated RNNs are less restricted than recursive neural networks. GRU also address the problem of vanishing gradients. However, it always exposes its memory content and uses only leaky integration. [1]. LSTMs address the gradient problems, do not always expose their memory content and do not depend on leaky integration.

3 ARCHITECTURE AND TRAINING

LSTM networks are a effective and scalable when working with sequential data, they can also effectively capture long time dependencies [10]. However, LSTM networks are extending RNN by at least two additional units (input gate and output gate) which increases the weights up to a factor of nine [11].

3.1 Architecture

In 1997 Hochreiter and Schmidhuber extended RNNs by an input gate and an output gate. They called the new unit a memory cell [11]. This initial LSTM network is called the vanilla LSTM and was later expanded by a forget gate and peephole connections.

3.1.1 Memory Cell. The core of a memory cell c_j consists of a "Constant Error Carousel" (CEC). This unit is linear and recurrently self-connected and has a weight of 1.0. The activation of this CEC is called the cell state s_{c_j}. The CECs are the solution to the vanishing gradient problem [4]. The CEC is the only part of the LSTM network whose errors can flow back forever, but they also permit the network

to bridge huge time lags [6].
The net input to c_j is computed as follows:

$$net_{c_j}(t) = \sum_i w_{c_j i} y^i(t-1) \tag{1}$$

i denotes any kind of unit (input units, gate unitsor conventional hidden units). The net input depends on the sum of all weights from the memory cell to an i times the activation of this i in the last time step [11].
The output of a memory cell to the time t is computed as

$$y^{c_j}(t) = y^{out_j}(t) h(s_{c_j}(t)) \tag{2}$$

where the "internal state" $s_{c_j}(t)$ is

$$s_{c_j}(0) = 0,\ s_{c_j}(t) = s_{c_j}(t-1) + y^{in_j}(t) g(net_{c_j}(t)) \text{ for } t > 0 \tag{3}$$

The internal state depends on the internal state of the last time step and the activation of the input gate. Furthermore, the cell input net_{c_j} is included. This cell input is "squashed" by the function g (a centred sigmoid with range[-1,1]. For calculating the output of a memory cell, the internal state is scaled by the function h (a centred sigmoid with range[-2,2] [7]) and multiplied by the activation of the output gate [11].
The cell output y^{c_j} from c_j in time $t$is computed as follows: [11]

$$y^{c_j}(t) = y^{out_j}(t) h(s_{c_j}(t)) \tag{4}$$

In 2005 Graves and Schmidhuber removed the output squashing function h from the LSTM architecture, because they found no empirical evidence that it was needed [6].

3.1.2 Input Gate. The input gate should protect the CEC from inferences of irrelevant input. It uses input from other memory cells to decide if it stores the current information in its memory cell or not. Simply said the input gate helps the net to decide whether to keep or to override the memory stored in the cell [11].
The activation y^{in_j} of the input gate in_j in time t is denoted with:

$$y^{in_j}(t) = f_{in_j}(net_{in_j}(t)); \quad net_{in_j}(t) = \sum_i w_{in_j i} y^i(t-1) \tag{5}$$

The activation of the input gate depends on the input net_{in_j} to the input gate [11] which is run through a logistic sigmiod function [7]. This input is the sum of the weights w from the input gate in_j to the units i times the activation function of this i in the last time step $(t-1)$ [11].
Figure 1 displays the architecture of a memory cell c_j as proposed by Hochreiter and Schmidhuber [11].

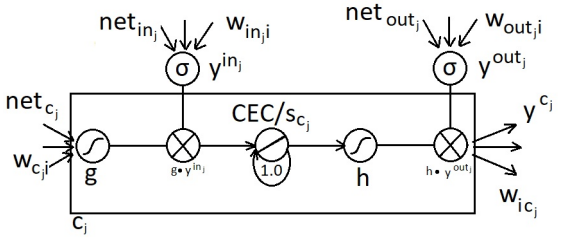

Figure 1: Architecture of a memory cell c_j and its input gate in_j and output gate out_j [11].

3.1.3 Output Gate. Like the input gate the output gate should protect against inferences. However, it should protect other units from saved memory content that is currently irrelevant. In other words the output gate can be used to decide whether to access the stored memory or not [11]. Furthermore, the output gate protects the CES from backward flowing error. [4].
The activation y^{out_j} of the output gate out_j is as follows [11]:

$$\begin{aligned} y^{out_j}(t) &= f_{out_j}(net_{out_j}(t)); \\ net_{out_j}(t) &= \sum_i w_{out_j i} y^i(t-1) \end{aligned} \tag{6}$$

Like the activation of the input gate the activation of the output gate depends on the input net_{out_j} to the output gate[11] which is run through a logistic sigmoid function [7]. This input is the sum of all weights w from the output gate to all i times the activation function of this i in the last time step [11].

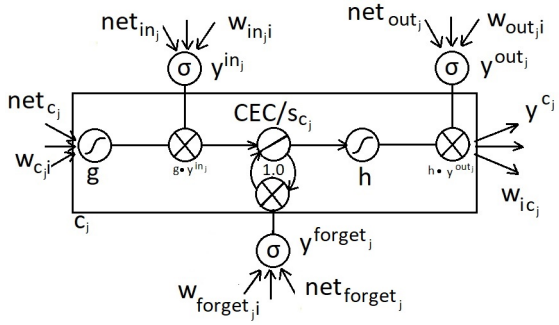

Figure 2: Architecture of a memory cell c_j and its input gate in_j, output gate out_j and forget gate $forget_j$ [7].

3.1.4 Forget Gate. Vanilla LSTM networks solve the vanishing gradient problem, but they suffer from learning problems if the input is streamed or very long. This input has to be a priori segmented into appropriate training subsequences whose beginnings and ends need to be clearly defined. If this is not the case the internal value of the cells may grow without bound. As a solution to this problem Hochreiter et al. proposed forget gates in 1999. These gates can reset memory cells once the information stored in them is no longer relevant. Forget gates replace the standard weight of 1.0 in the CECs (displayed in the middle of figure 1) [7].
The activation y^{forget} of the forget gate is calculated according to the other gates:

$$\begin{aligned} y^{forget_j}(t) &= f_{forget_j}(net_{forget_j}(t)); \\ net_{forget_j}(t) &= \sum_i w_{forget_j i} y^i(t-1) \end{aligned} \tag{7}$$

In the activation function the input from the network to the forget gate net_{forget_j} is squashed with a logistic sigmoid function. net_{forget_j} is calculated from the sum of the weights (from the forget gate to other units) multiplied with the activation function of these units (in the last time step) [7].

The calculation of the initial state of a c_j is updated to

$$\begin{aligned} &s_{c_j}(0) = 0, \\ &s_{c_j}(t) = y^{forget_j}(t)s_{c_j}(t-1) + y^{in_j}(t)g(net_{c_j}(t)) \text{ for } t > 0. \end{aligned} \tag{8}$$

The calculation of the internal state is expended by the activation of the forget gate. The biases of the input and output gates are initialised with negative values and the forget gate biases with positive ones [7].
Figure 2 displays the architecture of a memory cell c_j with a added forget gate as proposed by Gers, Schmidhuber and Cummins [7].

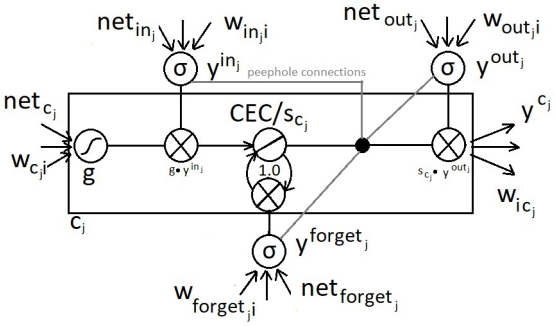

Figure 3: Architecture of a memory cell c_j and its input gate in_j, output gate out_j and forget gate $forget_j$. The grey lines show peephole connections [6].

3.1.5 Peephole Connections. To predict inputs that depend on inputs that were several time steps ago is a problem for vanilla LSTM. They can not handle long time lags (> 50 discrete time steps). To solve this problem Gers and Schmidbauer proposed peephole connections. These are weighted and connect the CECs to the input, output and forget gates of the same memory block. These connections enable the gates to shield the CECs from unwanted inputs or unwanted error signals. No error signals are backpropagated, from the gates to the CECs, during learning to keep the shielding intact [6].
Figure 3 displays the architecture of a memory cell c_j with added peephole connections as proposed by Gers and Schmidhuber [6]
These peephole connections changed the calculation of the input gate activation y^{in} and the forget gate activation y^{forget} to the following calculation, where f_l is a logistic sigmoid with a range of [0,1]:

$$\begin{aligned} &y^{l_j}(t) = f_{l_j}(net_{l_j}(t)) \text{ with } l \in \{forget, in\}; \\ &net_{l_j}(t) = \sum_{i_j} i_{l_j i} y_l^{i_j} \end{aligned} \tag{9}$$

i donates a unit in the network and i_j additionally includes the cells of the j-th memory block: $i_j = \{m, c_j\}$. The peephole connections are incorporated in this equation by including the CECs of the j-th memory block as source units. The calculation of the state of the memory cell $s_c(t)$ changed to the following:

$$\begin{aligned} &net_{c_j}(t) = \sum_i w_{c_j i} y^i(t-1), \\ &s_{c_j}(t) = y^{forget_j}(t)s_{c_j}(t-1) + y^{in_j}(t)g(net_{c_j}(t)),\ s_{c_j}(0) = 0 \end{aligned} \tag{10}$$

Furthermore, the output gate activation was updated to the following:

$$net_{out_j}(t) = \sum_{i_j} w_{out_j i} y_{out}^{i_j}; \quad y^{out_j}(t) = f_{out_j}(net_{out_j}(t)) \tag{11}$$

The output gate takes the CECs of memory block j with the cell states $s_c(t)$ into account [6].

3.1.6 Network Topology. Hochreiter and Schmidhuber propose a topology with one input, one hidden and one output layer in 1997. The hidden layer is self-connected an contains the memory cells and corresponding gate units, it may also contain conventional hidden units. All units in every layer are directly connected to the layer above [11]. In their 1999 paper Gers, Schmidhuber and Cummins lift the one hidden layer constraint by using several in their experiments [7].

3.2 Training

For the LSTM network discrete time steps are considered. In each step all units are updated (forward pass) and the errors signals for all weights are computed (backward pass) [7]. To avoid the problem of vanishing and exploding gradients Hochreiter and Schmidhuber use a variant of real-time recurrent learning (RLTL) and incorporate truncated backpropagation through time (tBPTT). The error signal does not get back propagated through time and the whole network. They only get propagated through earlier internal states of a memory cell, within a memory cell. The CECs at the core of a memory cell is the only part where errors can flow back indefinitely [11]. In vanilla LSTM only the gradient of the memory cell is backpropagated through time, the gradients for the other recurrent connections are truncated [10].
In 2005 Graves and Schmidhuber proposed a new learning method for LSTM networks. Instead of mixing RLTL and tBPTT they calculate the full error gradient for the LSTM network, which leads to a slightly better performance as the original method and makes LSTM directly comparable to RNN [9].

4 APPLICATIONS

LSTM networks emerged from RNN to provide scalable models for sequential data. Therefore tasks like handwriting recognition and -generation, language modelling and translation, acoustic modelling of speech, speech synthesis, protein secondary structure prediction, analysis of audio, and video data and many more can be solved [10]. In 2001 Gers and Schmidhuber discovered, that LSTM networks outperform RNN in learning context free languages. Furthermore, are LSTM networks the first RNNs that are capable to learn context sensitive languages[5]. An interesting "real world" application of LSTM networks is the composition of Blues music. Music is a type of signal stream and at the heart of the most music styles are long-term dependencies. A LSTM network with forget gates is able to learn from this stream and compose new music. In theory RNNs would also be able to compose novel melodies but they lack the abilities to correctly learn the global temporal structure. Therefore the music composed by RNNs is not very pleasing. These melodies lack thematic structure and are not musically coherent. This behaviour can likely be attributed to the vanishing gradient problem

of RNNs. LSTM networks are able to learn both the local structure of a melody and the long-term structure of a musical style [3].

5 SUMMARY

This paper introduces LSTM by discussing the architecture and the training of LSTM- Furthermore, examples of the application of LSTM are shown. This paper combines the current literature about LSTM and does not prove the claims made in the used papers. Due to a constraint on the length of the paper a discussion of the disadvantages of LSTM is omitted and the functionality of LSTM is not explained with examples or experiments.
Today LSTM networks are the state-of-the-art for the analysis of sequential data. Many variations of LSTM exist, that are tailored for special purposes. However, "normal" LSTM (vanilla LSTM with forget gates and peephole connections) are suitable for a broad range of tasks.

REFERENCES

[1] Nikhil Buduma. 2017. *Fundamentals of Deep Learning*. O'Reilly UK Ltd. http://www.ebook.de/de/product/24080206/nikhil_buduma_fundamentals_of_deep_learning.html

[2] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014). https://arxiv.org/pdf/1412.3555.pdf

[3] Douglas Eck and Juergen Schmidhuber. 2002. Finding temporal structure in music: Blues improvisation with LSTM recurrent networks. (2002), 747–756. ftp://ftp.idsia.ch/pub/juergen/2002_ieee.pdf

[4] Felix Gers. 2001. Long short-term memory in recurrent neural networks. *Unpublished PhD dissertation, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland* (2001). https://www.researchgate.net/profile/Felix_Gers/publication/2562741_Long_Short-Term_Memory_in_Recurrent_Neural_Networks/links/5759410a08ae9a9c954e77f5.pdf

[5] Felix A Gers and E Schmidhuber. 2001. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks* 12, 6 (2001), 1333–1340. ftp://ftp.idsia.ch/pub/juergen/L-IEEE.pdf

[6] Felix A Gers and Jürgen Schmidhuber. 2000. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, Vol. 3. IEEE, 189–194. ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf

[7] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999). https://pdfs.semanticscholar.org/1154/0131eae85b2e11d53df7f1360eeb6476e7f4.pdf

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2017. *Deep Learning*. The MIT Press. http://www.ebook.de/de/product/26337726/ian_goodfellow_yoshua_bengio_aaron_courville_deep_learning.html

[9] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18, 5 (2005), 602–610. ftp://ftp.idsia.ch/pub/juergen/nn_2005.pdf

[10] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2017. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems* (2017). https://arxiv.org/pdf/1503.04069.pdf

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780. http://www.bioinf.jku.at/publications/older/2604.pdf

[12] Ozan Irsoy and Claire Cardie. 2014. Deep recursive neural networks for compositionality in language. In *Advances in neural information processing systems*. 2096–2104. https://www.cs.cornell.edu/~oirsoy/files/nips14drsv.pdf

[13] LR Medsker and LC Jain. 2001. Recurrent neural networks. *Design and Applications* 5 (2001).