JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by

Clemens Kriechbaumer

Submitted at

Institute for Machine Learning

Supervisors

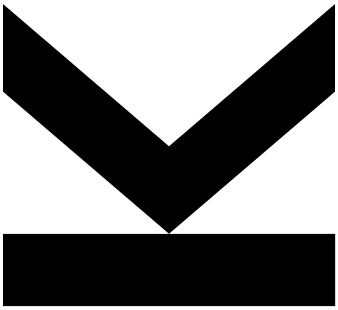
Dipl.-Ing. Daniel Klotz

Dipl.-Ing. Frederik Kratzert

February 2020

HIERARCHICAL MULTISCALE RECURRENT NEURAL NETWORKS

Junyoung Chung and Sungjin Ahn and Yoshua Bengio
arXiv:1609.01704, 2016



Seminar Bioinformatics

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

MOTIVATION

This seminar paper was written in the context of the topic “Multiscale streamflow forecasting using LSTMs”. It focuses on the novel approach of finding latent hierarchical structures proposed in the paper “Hierarchical Multiscale Recurrent Neural Networks” by Junyoung Chung, Sungjin Ahn and Yoshua Bengio published in 2016 [1] and provides a base for further project work.

This seminar paper is structured in describing the initial problem and terminology, present the proposed architecture in detail, show its results and performance and give a conclusion and outlook of possible future work and research direction.

Furthermore, this seminar paper provides the basis for further project work which will either focus on the novel architecture and its potential and/or gives an initial work direction in the field of hydrology in which recent research [2] has shown great performance when using Long Short-Term Memory (LSTM) models. In the context of hydrology especially the multiscale aspect, having different prediction resolutions like minutes, hours or days for streamflow forecast modeling and incorporate them in one combined model, would be the direction of the project work.

TABLE OF CONTENTS

1. Problem Introduction

To obtain hierarchical representations within data is one of the key principals of learning within deep neural networks. By stacking multiple hierarchical representation layers, deep neural networks can learn from data to optimize for a predefined objective and generalize to unseen examples. During the training phase of a deep neural network stacked layers pick up possible latent hierarchical structures within the data and learn their representations. The ability to learn hierarchical representations is a key component to the success especially in the context of spatial data and deep convolutional neural networks

The following Figure 1 shows a simplified example with three representation layers where each layer learned a hierarchical abstract concept of the underlying raw spatial data. The higher in the stack the layer is located, the more abstract the representation becomes.

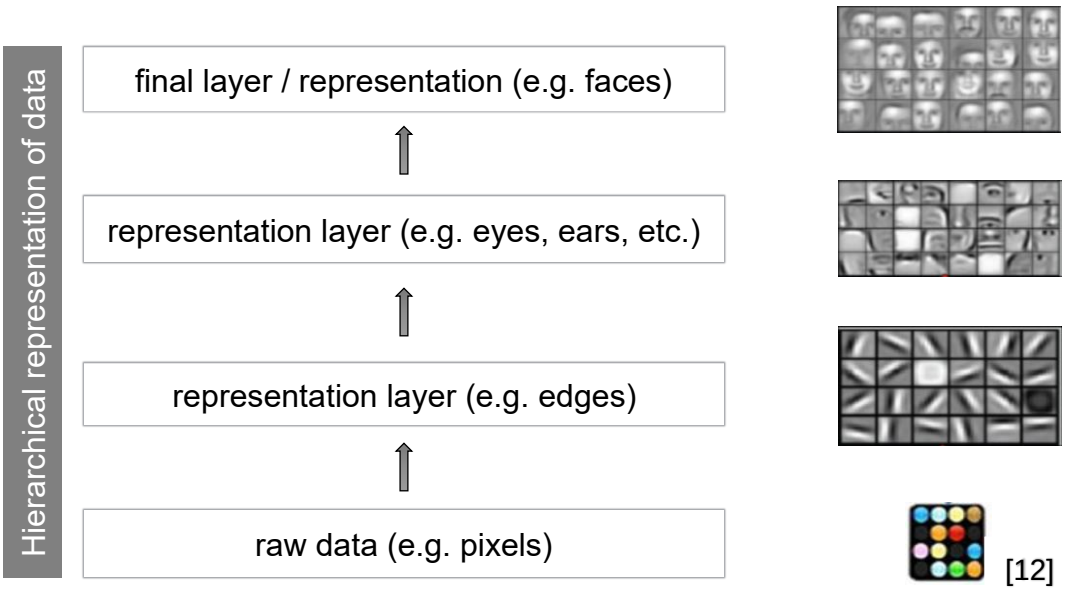


Figure 1 Hierarchical concept representations of data

It should be noted that the concepts (in this example edges, eyes, faces, etc.) which are learned by such a representation layer are not necessarily interpretable and might not refer to a human concept.

When looking at temporal data, which are typically the domain of recurrent neural networks, also hierarchical representations can be found. Beside stacked layers which represents hierarchical concepts, as mentioned in the example of Figure 1, for temporal data also the possibility of different scale within each hierarchical concept needs to be considered. As an example, in Figure 2 characters as a temporal sequence of data are shown. It is obvious that the raw data, in this example characters, can be represented on different layers of hierarchical abstractions as words, characters, and so on. Furthermore, it is shown that each layer has a different timescale and frequency of the objects it is representing. This fact is referred to as multiscale.

Beside language or text processing as shown in the previous example there many different domains where hierarchical abstract representations within temporal data are present and implicitly processed and interpreted by humans. Therefore, to be able to incorporate hierarchical abstract structures within recurrent neural networks, which are a typical model to use on temporal data, could potentially prove useful for a variety of

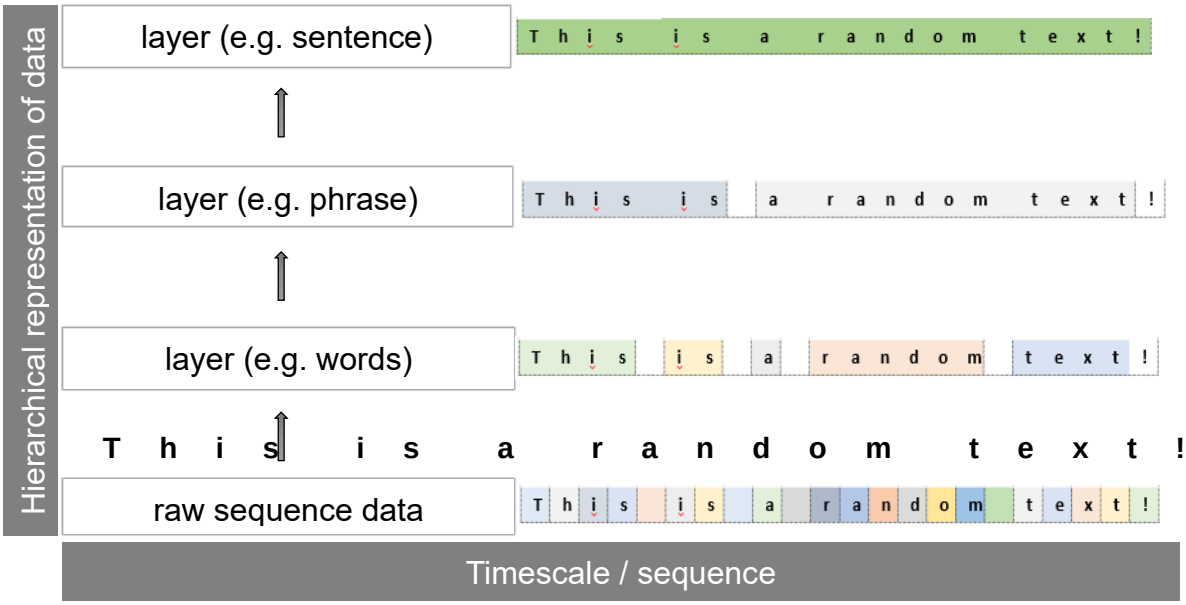


Figure 2 Hierarchical temporal data

applications and domains.

Chung et al. 2016 [1] proposed a model called “Hierarchical Multiscale Recurrent Neural Networks” (HM-RNN) to be able to incorporate latent hierarchical structures within temporal data. In the following chapters, this seminar paper describes the approach by the authors in detail.

2. Proposed Architecture

Previous approaches treated the timescale of temporal data as a dynamic variable that could be learned from the underlying data ([3], [4] or [5]). or set them as definable hyperparameter of the model([6], [7] or [8]) These approaches did not address the fact, that the timescale even within the same hierarchical representation layer can vary in length (e.g. a word or sentence). Other approaches did consider the variability of the timescale but only solved this by providing a fixed hierarchical boundary structure to the model ([9]). Such an approach cannot be a generic solution applicable to different domains and applications but is only tailored towards one specific problem and data structure.

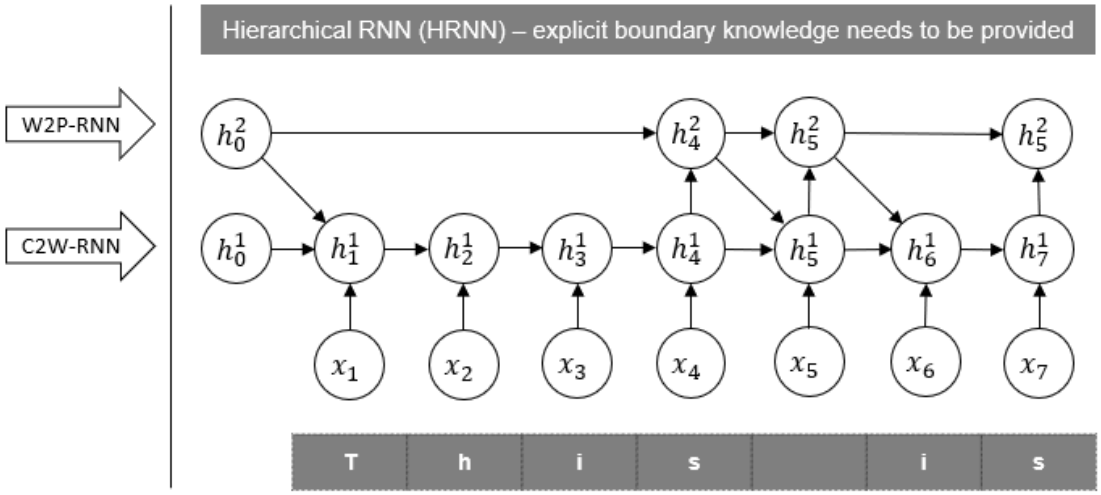


Figure 3 Hierarchical RNN with provided boundary structure

In Figure 3 a multi-layer RNN model for character sequence with a fixed provided hierarchical boundary structure is described. The first layer describes a character to word RNN (C2W-RNN) taking the characters as inputs and learns representations for words while the second layer learns phrase representations out of the underlying word representations (W2P-RNN).

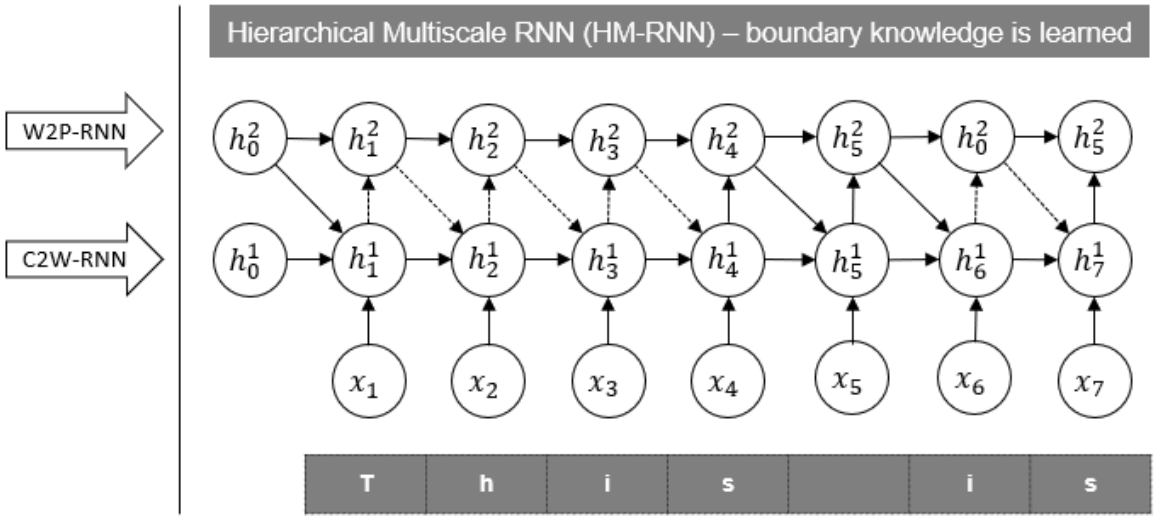


Figure 4 Hierarchical RNN - boundary knowledge is learned

On the contrary, in the example shown in Figure 4, the boundary information is learned during the training phase and can vary (multiscale). The result, the identified variability of timescale (e.g. length of words), is indicated with the dotted lines. Additionally, a more abstract/higher layer receives fewer updates as no new input information is propagated from the lower layers if the final representation has not yet been learned/found. This results in potentially fewer steps/parameters during backpropagation in the training phase.

2.1. Key Principals

The HM-RNN proposed by Chung et al. 2016 [1] is described based on the LSTM update schedule and has the following key principals.

2.1.1. Parametrized Boundary Detector

For each layer l at each time step t a binary variable $z \in \{0,1\}$ is introduced. This variable serves as a boundary detector. If $z_{t-1}^{l} = 1$ the model, consider this as the end of a segment (e.g. word) of the corresponding layer. Based on the state of z the following operations are performed.

2.1.2. Update Operation

The update operation describes the state that no boundary to the segment processing at the time has yet been found and input information can still flow from lower layers into the cell.

It is the case when the boundary state z of the previous time step of the same layer is 0 ($z_{t-1}^{l} = 0$) and z of the layer below at the same time step is 1 ($z_{t}^{l-1} = 1$). This results in the following calculation of the cell state c

$$c_t^l = f_t^l \odot c_{t-1}^l + i_t^l \odot g_t^l \tag{1}$$

and hidden state h.

$$h_t^l = o_t^l \odot tanh\,(c_t^l) \tag{2}$$

At this point all the necessary gates like forget gate f, input gate i, cell proposal g and output gate o need to be calculated just like the default LSTM update schedule.

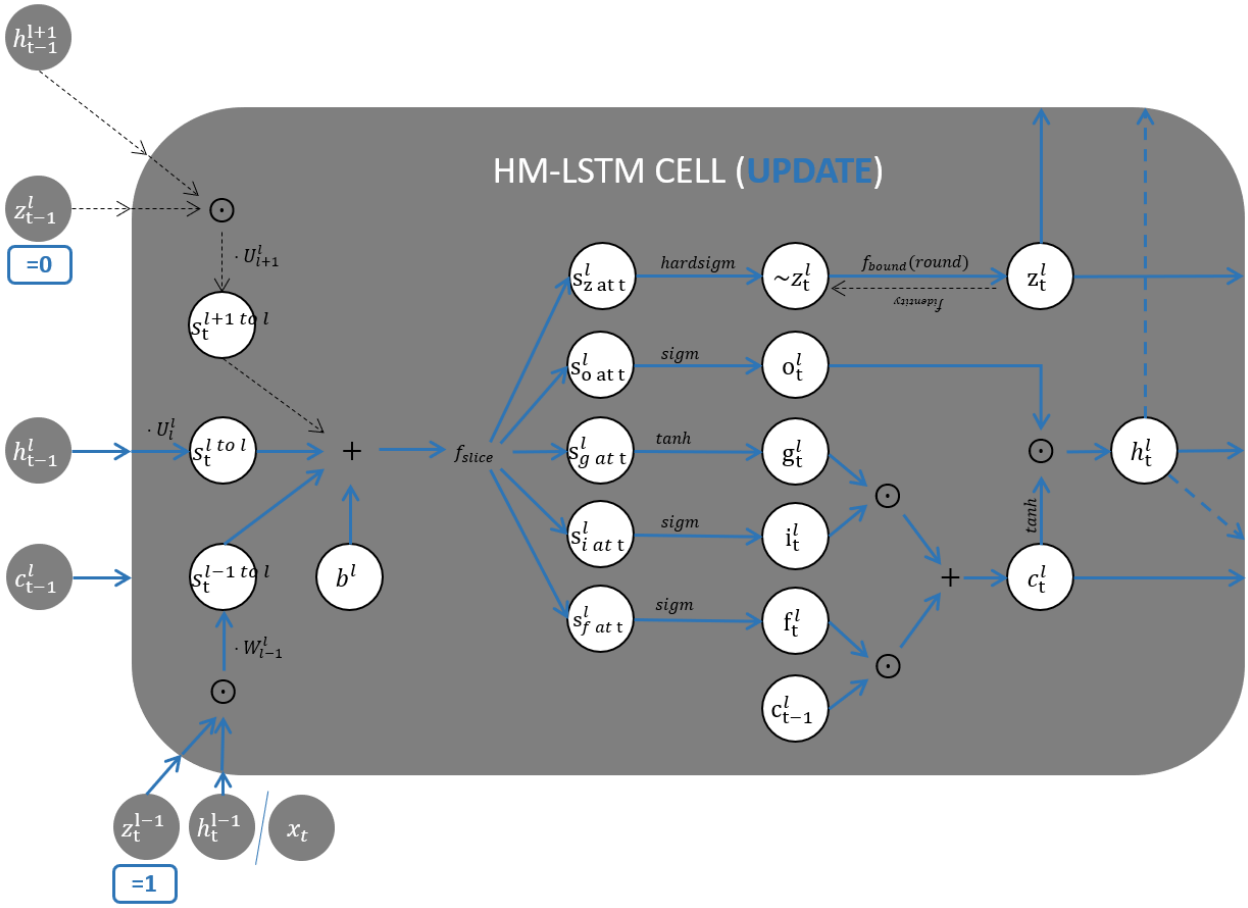


Figure 5 Update Operation

In Figure 5 the performed calculations for a HM-LSTM cell during the update operation are visualized. With the exception of the calculation of the boundary state z (see chapter 2.1.6.) the necessary calculations are the same as within a default LSTM cell. The coloured edges indicate the flow of information while the dotted black lines represent inactive connections during the operation.

2.1.3. Copy Operation

During the copy operation, no new information is expected from the layer below as new inputs and the state of the previous timestep are simply propagated/copied forward in time.

It is the case when the boundary state z of the previous time step of the same layer is 0 ($z_{t-1}^{l} = 0$) and z of the layer below at the same time step is also 0 ($z_{t}^{l-1} = 0$). This results in the forward propagation of the cell state c

$$c_t^l = c_{t-1}^l \tag{3}$$

and hidden state h.

$$h_t^l = h_{t-1}^l \tag{4}$$

It is obvious that at this point in time none of the gates (forget gate f, input gate i, cell proposal g or output gate o) within the cell needs to be calculated. This fact is shown in Figure 6. The coloured edges show the necessary calculations and information flow within the HM-LSTM cell during the copy operation. Compared to the update operation described in chapter 2.1.3. it becomes clear

that a lot less computations need to be performed. The only value which needs to be calculated

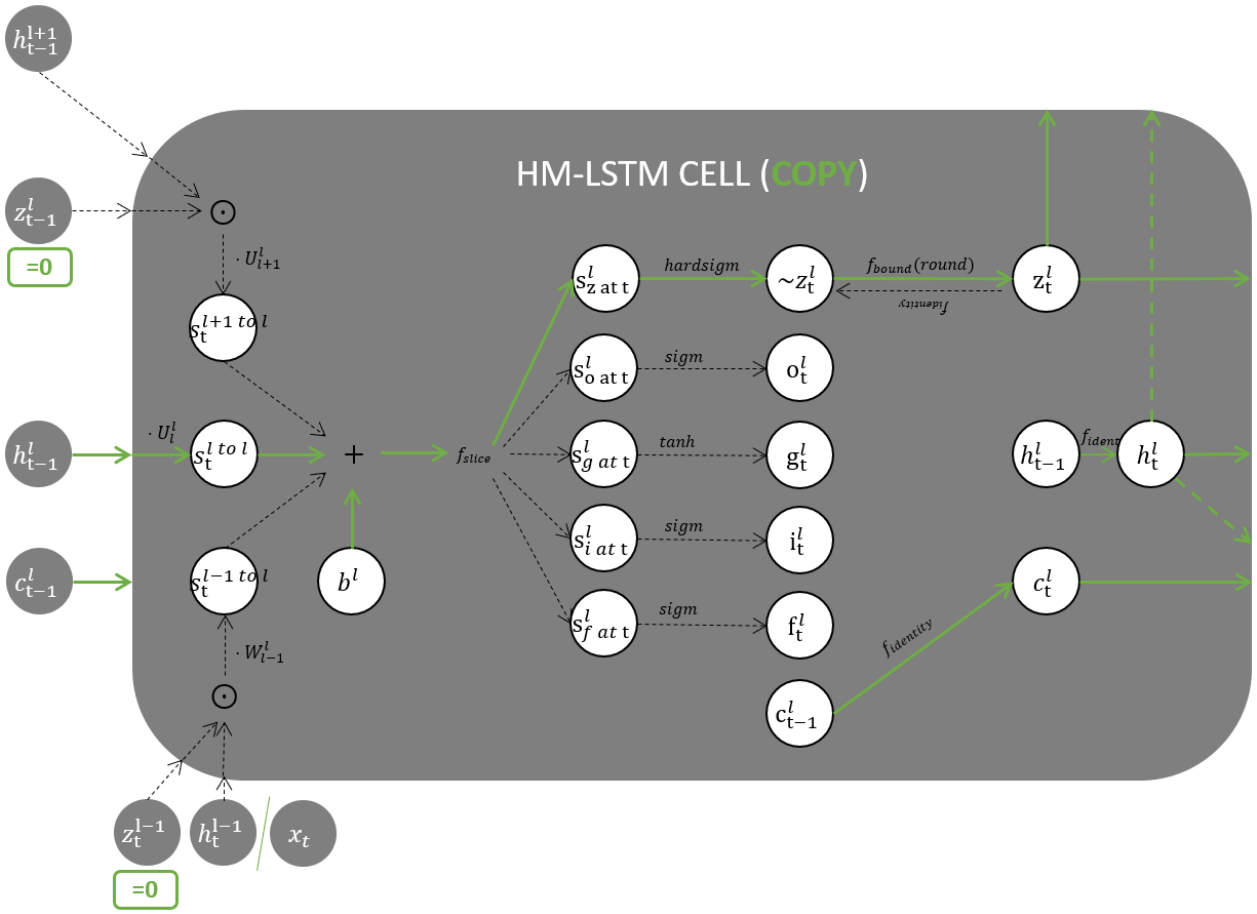


Figure 6 Copy Operation

is the boundary state z (see chapter 2.1.6.).

2.1.4. Flash Operation

In case the end of a segment has been reached the hidden state, representation of the context vector of the learned segment (e.g. a word) is implicitly forced as an input to the upper layer (refer to 2.1.2. resp. $z_t^{l-1} = 1$). At the same time the cell state is reset/flushed within the same layer.

It is the case when the boundary state z of the previous time step of the same layer is 1 ($z_{t-1}^{l} = 1$). This leads to the reset of the cell state c and recalculation of it using only new inputs

$$c_t^l = i_t^l \odot g_t^l \tag{5}$$

and to the new hidden state h.

$$h_t^l = o_t^l \odot tanh\,(c_t^l) \tag{6}$$

As demonstrated in Figure 7 the cell state h of the previous time step $t - 1$ is not considered during the calculations. Additionally, unlike a default stacked LSTM cell the HM-LSTM cell is initialized with long term information from the layer above $l + 1$ from the previous time step $t - 1$ therefore having a additional top-down connection (see chapter 2.1.5. . In addition, it should be mentioned that information from the layer below $l - 1$ are only incorporated if an end of an segment $z_t^{l-1} = 1$ has been found.

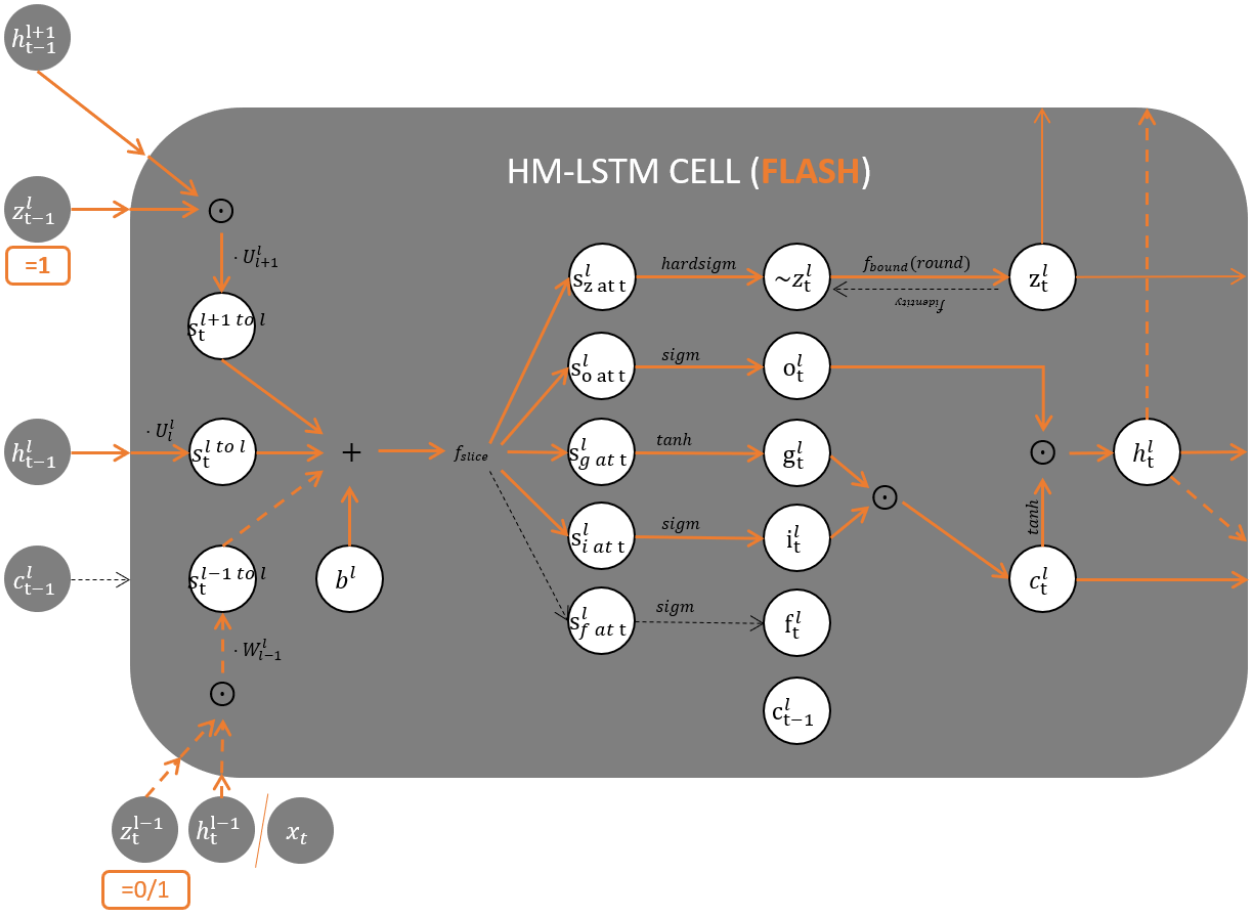


Figure 7 Flash Operation

2.1.5. Top-down Connection

Additionally, to the default update schedule for a stacked LSTM, the HM-LSTM has a top-down connection (from layer $l+1$ to l) when calculating the gate values. This is only applied if for the previous time step at layer l a boundary was detected ($z_{t-1}^{l} = 1$)

This leads to the calculations of the following pre-activations s.

$$s_t^{recurrent(l)} = U_l^l \, h_{t-1}^l \tag{7}$$

$$s_t^{top-down(l)} = z_{t-1}^l U_{l+1}^l \, h_{t-1}^{l+1} \tag{8}$$

$$s_t^{bottom-up(l)} = z_t^{l-1} W_{l-1}^l \, h_t^{l-1} \tag{9}$$

Here the matrix $W_{l-1}^l \epsilon \, \mathbb{R}^{(4\dim(h^l)+1)\times\dim(h^{l-1})}$ describes the linear transformation of the cell input layer $l-1$ to calculate the pre-activations $s_t^{bottom-up(l)}$ in layer l. The matrix $U_i^j \epsilon \, \mathbb{R}^{(4\dim(h^j)+1)\times\dim(h^i)}$, for the recurrent connection in an RNN usually denoted as R, models the recurrent and the top-down linear transformation for the pre-activations. For the recurrent connection, i and j refer to same layer l and leads to the calculation of $s_t^{recurrent(l)}$. For the top-down connection i refers to the layer and dimensionality above $l+1$ and j to the current layer l, the result of this applied linear transformation is denoted as $s_t^{top-down(l)}$.

The factor 4 in dimensions of W and U is based on the fact that the pre-activations are stacked and calculated for gates (f, i, g, and o) plus the boundary detector z ($+1$ in dimensions) within the same linear transformation operation. Finally, the activations of the gates are calculated as follows.

$$\begin{pmatrix} f_t^l \\ i_t^l \\ o_t^l \\ g_t^l \\ \widetilde{z_t^l} \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \\ hard\ sigm \end{pmatrix} f_{slice}\left(s_t^{recurrent(l)} + s_t^{top-down(l)} + s_t^{bottom-up(l)} + b^{(l)}\right) \quad (10)$$

The f_{slice} operation denotes slicing of the stacked pre-activations into the individual components for the gates (f, i, g and o) and the boundary detector. In contrast to the gates, the boundary state at this point is still pre-activated and shown with tilde symbol as $\widetilde{z_t^l}$. The actual calculation is described in the following chapter 2.1.6.

2.1.6. Calculation of Boundary Detector

Based on the recurrent, top-down and bottom-up linear transformed hidden states a hard-sigmoid function is applied to calculate the pre-activation $\widetilde{z_t^l}$. The operation herby is defined as follows.

$$hard\ sigm\ =\ max(0, min(1, \frac{ax+1}{2})) \quad (11)$$

Within the operation a denotes a slope variable. This variable is a hyperparameter of the HM-RNN model and plays a key role in the gradient calculation – refer to chapter 2.2. .

After the calculation of the pre-activation $\widetilde{z_t^l}$ the activation state as a binary boundary state is obtained by

$$z_t^l = f_{bound}(\widetilde{z_t^l}) \quad (12)$$

Here $f_{bound}: \mathbb{R} \rightarrow \{0,1\}$ refers to a deterministic step function (rounding based on a threshold)

$$z_t^l = \begin{cases} 1\ if\ \widetilde{z_t^l} > 0.5 \\ 0\ otherwise \end{cases} \quad (13)$$

or to sampling from a Bernoulli distribution.

2.2. Gradient Calculation

During the gradient calculation while doing backpropagation through time (BPTT) the HM-LSTM behaves similarly to a default LSTM model. Only the discrete variable of the boundary detector z needs to be treated differently.

As shown in equation 13 the boundary state is determined by a deterministic step function. This step function is non-differentiable. During the backpropagation, this deterministic step function is replaced by the identity function $f_{identity}$ with the gradient w.r.t to its input of 1. Furthermore, the gradient w.r.t the input of the hard-sigmoid function is defined as follows.

$$\frac{\partial}{\partial x}\ max(0, min(1, \frac{ax+1}{2})) = \begin{cases} \frac{a}{2}\ if\ -1 < ax < 1 \\ 0\ otherwise \end{cases} \quad (14)$$

It can be seen that the propagated gradient flow/delta propagation from future time steps is directly dependent on the slope variable a. The idea is to model the slope variable such that it mimics the deterministic step functions as good as possible – this is referred to as slope annealing trick.

In Figure 8 the non-continuous/non-differentiable step function (refer to equation 13) in blue is shown. The slope variable a is tuned and leads to good approximation of the step function (displayed in red).

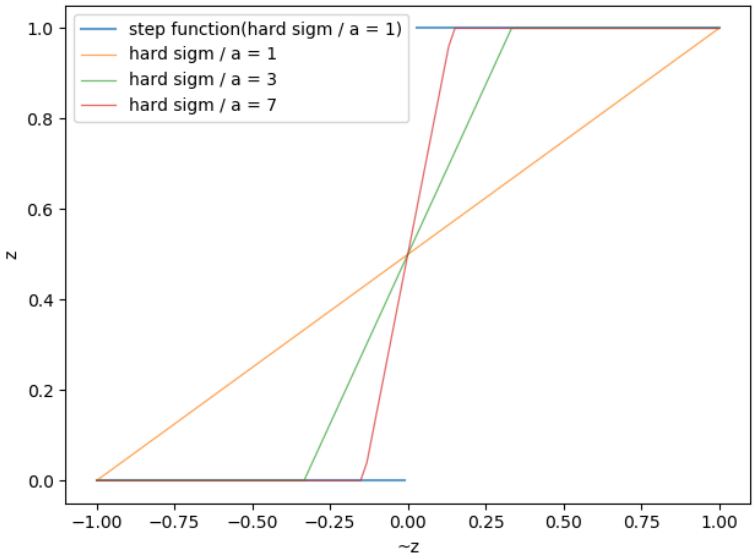


Figure 8 Slope annealing trick

2.3. Test Model

The proposed HM-LSTM architecture was tested by Chung et al. in 2016 [1] on character language modeling and handwriting sequence generation. The used test model consisted of three parts, an input embedding layer, the HM-LSTM model, and an output model. The output model used the hidden state of all layers and applied a weight parameter $w^l \epsilon \mathbb{R}^{(\Sigma_{l=1}^{L} \dim(h^l))}$ to calculate the scalar gating unit g.

$$g_t^l = sigm([h_t^l \dots h_t^L] w^l) \tag{15}$$

This approach allows an adaptive control of the importance of each individual layer. The final hidden state is further embedded using $W^e \epsilon \mathbb{R}^{(\Sigma_{l=1}^{L} \dim(h^l)) \times e)}$ and the activation function ReLU resulting in the embedded output hidden state h_t^e

$$h_t^e = ReLU(([h_t^l \dots h_t^L] \odot [g_t^l \dots g_t^L]) W^e) \tag{16}$$

Additionally, the output model uses two fully connected layers and if appropriate for the task a softmax layer at the end. In Figure 9 the process and calculations of the output model are described in detail.

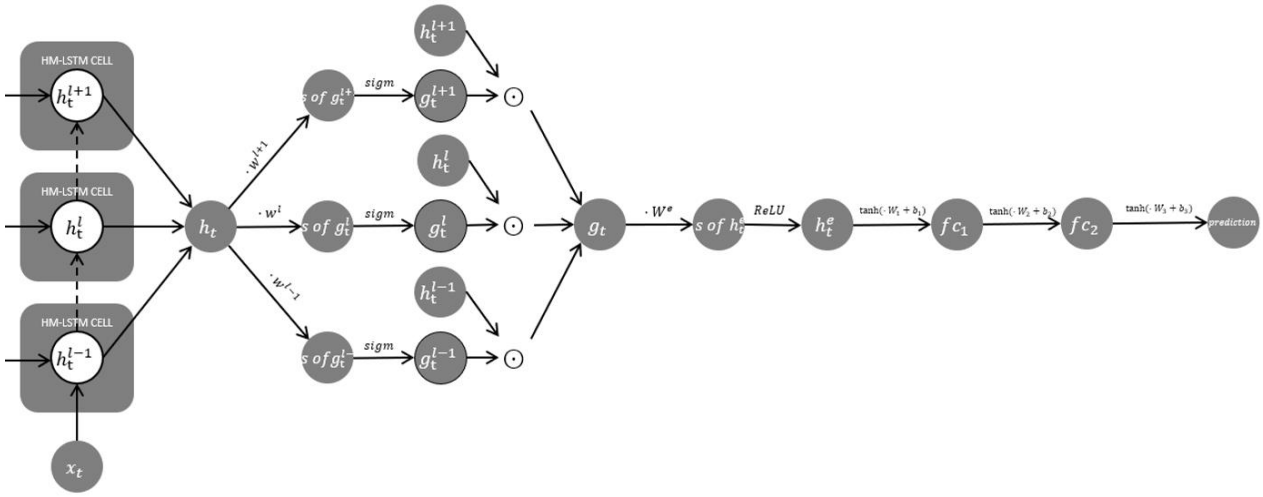


Figure 9 Output Model

3. Results

3.1. Benchmarks

The HM-LSTM architecture used in the described test model (see chapter 2.3.) was evaluated on multiple character-level language datasets/tests and a handwriting sequence generation task [10]

For the character-level language modeling evaluation, the architecture was tested on the Penn Treebank [11] dataset, Text8 [12] dataset, and the Hutter Price Wikipedia [13] (enwik8) dataset. Many at the time of publishing (2016) state of the art results were improved over time by different architectures and models but for the Text8 and enwik8 dataset according to paperswithcode.com [14], [15] the original results are still in the top 20.

If the number of model parameters is considered the performance can be rated even higher. As an example, according to paperswithcode.com [14], the LayerNorm HM-LSTM had 35 million parameters when trained on the Text8 dataset while a state of the art model used about 277 million parameters. In Table 1 and Table 2 the results for the Text8 and Penn Treebank dataset at the time of publishing together with the most recent results are shown.

Text8 (results from the original paper)	**BPC**
HF-MRNN (Mikolov et al., 2012)	1.54
MI-RNN (Wu et al., 2016)	1.52
Skipping-RNN (Pachitariu & Sahani, 2013)	1.48
MI-LSTM (Wu et al., 2016)	1.44
BatchNorm LSTM (Cooijmans et al., 2016)	1.36
HM-LSTM	**1.32**
LayerNorm HM-LSTM	**1.29**
Text8 (recent results paperswithcode.com [9])	**BPC**
12-layer Character Transformer Model (Al-Rfou et al., 2018)	1.18
64-layer Character Transformer Model (Al-Rfou et al., 2018)	1.13
Transformer-XL Large (Dai et al., 2018)	1.08
Transformer-XL + RMS dynamic eval + decay8 (Krause et al., 2019)	1.038

Table 1 Text8 Results

Penn Treebank (results from the original paper)	**BPC**
ME n-gram (Mikolov et al., 2012)	1.37
BatchNorm LSTM (Cooijmans et al., 2016)	1.32
Zoneout RNN (Krueger et al., 2016)	1.27
HyperNetworks (Ha et al., 2016)	1.27
LayerNorm HyperNetworks (Ha et al., 2016)	1.23
BatchNorm LSTM (Cooijmans et al., 2016)	1.36
LayerNorm CW-RNN	1.40
LayerNorm LSTM	1.29
LayerNorm HM-LSTM Sampling	**1.27**
LayerNorm HM-LSTM Soft	**1.27**

LayerNorm HM-LSTM Step Fn.	**1.25**
LayerNorm HM-LSTM Step Fn. & Slope Annealing	**1.25**
Penn Treebank (recent results paperswithcode.com [10])	**BPC**
3-layer AWD-LSTM (Merity et al., 2018)	1.175
Trellis Network (Bai et al., 2019)	1.159
Mogrifier LSTM (Melis et al., 2019)	1.12
Mogrifier LSTM + dynamic eval (Melis et al., 2019)	1.083

Table 2 Penn Treebank Results

3.2. Boundary State Visualization

Besides the predictive capacity as shown in character level modeling (as one example), the boundary state itself reveals information about the data and its hierarchical structure. In Figure 10 it can be observed that the boundary state is 1 (indicated in white block) at the end of a sequence and fires for layer 2 (z^2) consistently after each word and for layer 1 (z^1) infrequently in-between

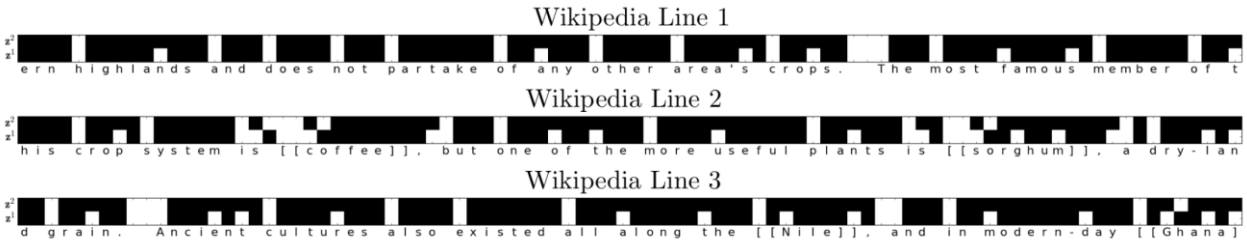


Figure 10 Boundary state visualized on enwiki8 [1]

Remarkable is that there was no constraint enforced on the model or data provided what to detect as the end of a segment. If the objective would be defined to find the hierarchical structure of data, the HM-LSTM would be able to do this in an unsupervised manner.

In Figure 11 additionally to the boundary state the l^2 norm of the hidden states in each layer is visualized in colour. As expected, when looking at higher layers the hidden state does not change as frequently as lower layers (indicated in non-changing of the colors). This behavior reflects the copy operation (refer to chapter 2.1.3.

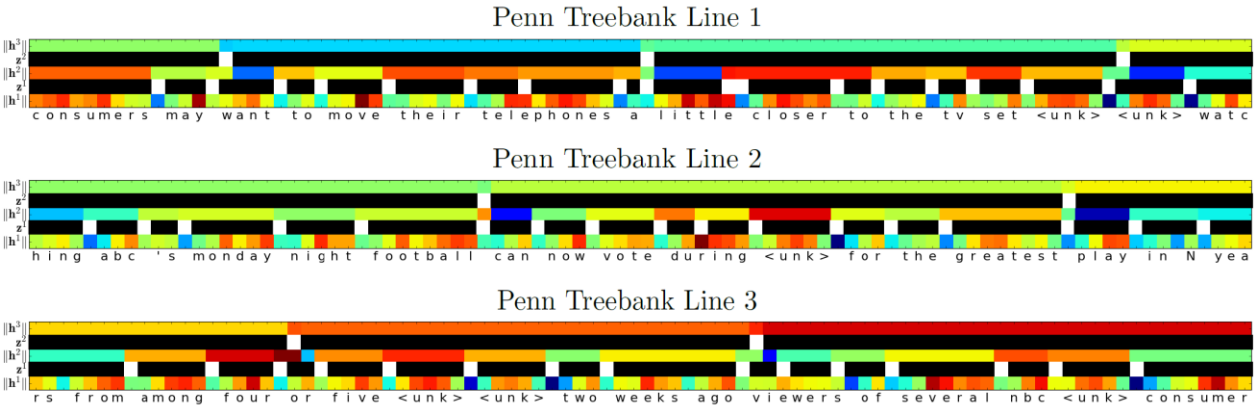


Figure 11 Boundary state visualized on Penn Treebank [1]

After a boundary is detected the l^2 norm of the hidden states is reset (indicated by the change in color) and reinitialized by the layer above of the previous time step plus possible input information from the layer below. This can be referred to as the flush operation (see chapter 2.1.4.

4. Discussion

The hierarchical multiscale recurrent network architecture introduced by Chung et al. 2016 [1] describes a novel approach to identify hierarchical structures within given data without providing constraints or further information about its content. It proves to deliver solid results in character-level language modeling tasks and handwriting sequence generation. Other benefits mentioned by the authors were the observed good generalization performance, possible computational benefits of the model and improved explainability of the model compared to a default RNN or LSTM.

Especially the ability to detect hierarchical structures in an unsupervised manner in a given dataset shows promising results and could be the base for further work and research. Here possible research questions could be:

- How the boundary detector behaves with many layers and are there any meaningful structures found especially on higher more abstract layers
- Do identified hierarchical structures refer to human concepts or are they at least interpretable
- Can those found boundaries be objectively compared and empirically measures
- How well does the model and its capacity to identify hierarchical structures performs in other domains beside language modeling.
- Are there even any application areas where the ability to identify hierarchical structures itself is the main task or problem and what or those
- Can the boundary detector and the defined operations especially copy, and flush be used to improve the explainability of what happens within a LSTM model and in particular what happens to the cell and hidden states.
- Do further investigations into the generalization performance. comparison with other generalizations methods and possibly empirical measured result show significant improvements
- Empirically investigate the computational saving potential of the proposed architecture while possible not losing accuracy in prediction capability.

5. List of Tables

6. List of Figures

7. References

[1] J. Chung, S. Ahn and Y. Bengio, “Hierarchical Multiscale Recurrent Neural Networks,” *arXiv:1609.01704,* 2016.

[2] F. Kratzert, D. Klotz, C. Brenner, K. Schulz and M. Herrnegger, “Rainfall--runoff modelling using Long Short-Term Memory (LSTM) networks,” *Hydrology and Earth System Sciences,* vol. 11, no. 22, pp. 6005-6022, 2018.

[3] J. Schmidhuber, “Neural sequence chunkers,” 1991.

[4] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, “Gated feedback recurrent neural networks,” *Proceedings of the 32nd International Conference on Machine Learning (ICML),* 2015.

[5] J. Chung, K. Cho and Y. Bengio, “A character-level decoder without explicit segmentation for neural machine translation,” *Association for Computational Linguistics (ACL),* 2016.

[6] S. El Hihi and Y. Bengio, “Hierarchical recurrent neural networks for long-term dependencies,” *In Advances in Neural Information Processing Systems,* pp. 493-499, 1995.

[7] J. Koutník, K. Greff, F. Gomez and J. Schmidhuber, “A clockwork rnn.,” *31st International Conference on Machine Learning (ICML 2014),* 2014.

[8] Bahdanau, Dzmitry, J. Chorowski, D. Serdyuk and Y. Bengio, “End-to-end attention-based large vocabulary speech recognition.,” *In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* pp. 4945-4949, 2016.

[9] A. Sordoni, B. Yoshua, H. Vahabi, C. Lioma, J. G. Simonsen and J.-Y. Nie, “A hierarchical recurrent encoder-decoder for generative context-aware query suggestion,” *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management,* pp. 553-562, 2015.

[10] M. Liwicki und H. Bunke, „Iam-ondb-an on-line english sentence database acquired from handwritten text on a whiteboard.,“ *In Eighth International Conference on Document Analysis and Recognition (ICDAR’05) IEEE,* pp. 956-961, 2005.

[11] M. P. Marcus and M. A. S. B. Marcinkiewicz, “Building a large annotated corpus of english: The penn treebank.,” *Computational linguistics,* vol. 19, no. 2, pp. 313-330, 1993.

[12] M. V. Mahoney, “Large text compression benchmark,” 2009. [Online]. Available: http://mattmahoney.net/dc/textdata.html.

[13] M. Hutter, „The human knowledge compression contest,“ 2012. [Online]. Available: http://prize.hutter1.net/.

[14] “Papers With Code - State of the Art Language Modelling - Text8 dataset,” [Online]. Available: https://paperswithcode.com/sota/language-modelling-on-text8. [Accessed February 2020].

[15] “Papers With Code - State of the Art Language Modelling - enwiki8 dataset,” [Online]. Available: https://paperswithcode.com/sota/language-modelling-on-enwiki8. [Accessed February 2020].

[16] Chaitanya, A. L. Katole and a. K. P. Y. a. A. K. B. a. S. S. K. a. M. Siva, “Hierarchical Deep Learning Architecture For 10K Objects Classification,” *arXiv:1509.01951,* 2015.