HIERARCHICAL MULTISCALE RECURRENT NEURAL NETWORKS FOR BIOLOGICAL DATA

Charlotte Kaae (S153171), Nikolas Thuesen (S152993) and Gustav Lindved (S153536)

Under supervision of Alexander Johansen

ABSTRACT

Hierarchical structures are found in several forms of temporal data such as language and amino acid sequences. These structures have been difficult to discover using recurrent neural networks, but hierarchical multiscale recurrent neural networks, proposed in a 2016 paper, have shown promising results. The model consists of lower-order layers, dealing with lower abstractions that update more frequently, and higher-order layers dealing with higher abstractions and updating on a slower timescale. In this paper, we construct models with 2 and 3 layers, examine the performance on the Penn Treebank data set and compare this performance with a standard LSTM model. The model was successfully implemented, however, the best hierarchical model (BPC of 1.47) did not outperform the standard LSTM (BPC 1.43). Furthermore, we attempted to use the model to predict hierarchical structures in protein data (amino acid sequences) - something which to our knowledge has not been tried previously. Although this attempt showed less promising results, we believe that the idea should not yet be abandoned, as knowledge of hierarchical structures are of huge value for biological and medical research.

1. INTRODUCTION

Learning hierarchies, ranging from low to high abstraction levels, remains a challenge in machine learning. This involves recognising features in an image from the individual pixels or summarising the content of a text. These very high-level abstractions build on several lower-layer abstractions. While convolutional neural networks have advanced the discovery of such hierarchies in spatial data, it has been difficult to achieve the same success with temporal data. Recurrent neural networks (RNNs) have performed well in capturing the temporal representation, but without really capturing the hierarchical structures that we know exist in a lot of temporal data [1]. One of the most important applications of deep learning on temporal data is natural language processing. Language has clear hierarchical structures with characters, words, and sentences. However, these structures remain difficult to uncover using machine learning techniques especially without predefining certain boundaries on the structures such as rules stating that words are separated by a space and sentences by a dot. One approach for solving this was presented in 2016 by Bengio et al who proposed a hierarchical multiscale LSTM (HM-LSTM) [2]. In their paper, they suggest an extended long short term memory neural network (LSTM) which uses multiple layers of hidden neurons to detect hierarchies without predefined boundaries. The proposed method gives three main benefits when compared to a normal LSTM. First, it reveals information about the hierarchical structures of the data, which is especially valuable for data where we do not know these structures in advance (such as biological data). Secondly, this detection of structures has also been shown to lead to better prediction of the next token in a sequence (which can be seen as a result of the model having a better "understanding" of the text). Thirdly, the model has the potential to be computationally more efficient, as there is no need to update all hidden units in every iteration due to the ordering of the hidden layers in the model. The model provides more control of the assignment of hidden units to different layers, each corresponding to a hierarchical level and gives better control of the update rate of each layer. Furthermore, this gained control of update rates means that less memory is leaked as a consequence of unnecessary updating, giving a better long-term memory [2].

This paper aims to implement this model in order to test the performance improvement and detection of hierarchical structures. The model is evaluated on character level language modelling, using the Penn Treebank data set [3]. Another type of temporal data, which is also is similar to language, is the amino acid sequences of proteins. Proteins are long chains built from 20 different amino acids, where each is assigned a letter (much like the alphabet). This sequence defines the protein's function, how it is folded and many other things such as solubility and stability. Like language, protein sequences are temporal data with hierarchical structures such as different domains as well as primary, secondary and tertiary structures [4]. However, since these structures are not yet completely understood directly from the sequence, we propose the hypothesis that these can be learned unsupervised like it has been shown that language structures can be by the HM-LSTM. Uncovering these structures would be a major breakthrough in computational biology, since predicting the relational structural relationships between the primary, secondary and tertiary structure of protein remains very difficult [2]. Therefore, we have also tested our model on a data set consisting of many different protein sequences from bacteria and archaea.

2. HIERARCHICAL MULTISCALE RECURRENT NEURAL NETWORK

Since this section describes the functionality of an HM-LSTM, it is heavily inspired by and largely similar to the corresponding section in the original article describing an HM-LSTM [2].

An overview of the structure of an HM-LSTM is shown in figure 1. In short, an HM-LSTM consists of multiple RNN's stacked on top of each other with only the bottom layer receiving information directly from the input data. When the model is properly trained, the lower layers will then predict lower-order structures (e.g. character level in language modelling) while higher-order layers are responsible for predicting higher-order structures (e.g. words, sentences etc. in language modelling). In figure 1 the model is shown to consist of cells with a hidden state for each layer and each time step $\mathbf{h}_t^l$, but for each hidden state the model also has a cell state $\mathbf{c}_t^l$ and a boundary state z_t^l. These are shown on the right hand side of figure 1.

The key-feature in an HM-LSTM, which makes the hierarchical order of layers possible, is the introduction of boundary detectors which determine whether or not a signal/representation should be passed on between two layers or two time steps. In figure 1 the boundary detectors are shown as circles on the connections between units either filled on a bold arrow (signal is sent) or empty on a dotted line (signal is not passed on). These boundary signals are specific to each layer (*l*) and timestep *t*, and referred to as z_l^t. When a boundary detector is turned off: $z_l^t = 0$ and when it is on: $z_l^t = 1$. A consequence of the hierarchical structure of the model is that the layers of higher-order do not need updating as frequently as lower layers. In language modelling this is a consequence of there being many characters in a single sentence. This means, that information is only passed on between two layers when a boundary in the input data is detected e.g. when a space between two words is detected. It is important to notice, that these boundaries are not prespecified but instead learned dynamically by the network while training. This feature is what makes the HM-LSTM special.

The equations governing exactly how information is passed on between cells are shown in equations 1-7. Equations 1-3 describe the three directions, from which a cell can receive signal (representation of the hidden state). These are from a previous timestep $\mathbf{h}_{t-1}^l$ called recurrent (rec), from a cell in the layer above and the previous timestep $\mathbf{h}_{t-1}^{l+1}$ here called top down (td) and from the same timestep and layer below h_t^{l-1} here called bottum-up (bu). $U_i^j \in \mathbb{R}^{(4\dim(\mathbf{h}^l)+1)\times\dim(\mathbf{h}^l)}$ and $W_i^j \in \mathbb{R}^{(4\dim(\mathbf{h}^l)+1)\times\dim(\mathbf{h}^{l-1})}$ are the state transition parameters from layer *i* to layer *j*.

$$\mathbf{s}_t^{\text{rec}(l)} = U_l^l \mathbf{h}_{t-1}^l, \tag{1}$$

$$\mathbf{s}_t^{\text{td}(l)} = z_{t-1}^l U_{l+1}^l \mathbf{h}_{t-1}^{l+1}, \tag{2}$$

$$\mathbf{s}_t^{\text{bu}(l)} = z_t^{l-1} W_l^{l-1} \mathbf{h}_t^{l-1}, \tag{3}$$

The boundary states are as mentioned learned dynamically, however in the bottom layer z_t^0 is always 1, as input should always be included and the top down connection from the bottom layer to the input layer is ignored. In order to calculate a cell state a number of gates are used. This is very similar to how an LSTM cell functions. The gates used are: the forget gate **f**, the input gate **i**, the output gate **o** and a cell proposal vector **g**. These are calculated based on the three signals in equation 1-3.

$$\begin{pmatrix} \mathbf{f}_t^l \\ \mathbf{i}_t^l \\ \mathbf{o}_t^l \\ \mathbf{g}_t^l \\ \tilde{z}_t^l \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \\ \text{hard sigm} \end{pmatrix} f_{\text{slice}}(\mathbf{s}_t^{rec(l)} + \mathbf{s}^{td}(l)_t + \mathbf{s}_t^{bu(l)} + \mathbf{b}^{(l)}) \tag{4}$$

Here $\mathbf{b} \in \mathbb{R}^{4\dim(\mathbf{h}^l)+1}$ is the bias term. The hard sigmoid function is defined as: hard $\text{sigm}(x) = \max(0, \min(1, \frac{ax+1}{2}))$. The reasoning behind the notation $\tilde{z}_t^l$ in equation 4 is, that the boundary state needs to be binary, while the output of the hard sigmoid function also takes values between 0 and 1. To get the boundary state z_t^l, the binarization/step function shown in equation 5 is used. However, other binarization functions could also be used [2].

$$z_t^l = \begin{cases} 1 & \text{if} \tilde{z}_t^l > 0.5 \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

Equation 6 shows another differing feature between the HM-LSTM network and the LSTM network. Namely, that the cell state can be updated in three distinct ways depending on the boundary states. The three options are UPDATE, COPY and FLUSH. The COPY operation copies the cell state and hidden state from one timestep to the next within the same layer. This is primarily used in higher layers, where the cell state and hidden state should remain the same for a number of timesteps until a boundary is detected. The UPDATE is similar to how an LSTM cell is updated resulting in a summary representation of the layer l. The FLUSH operation is used when a boundary is detected and consists of passing the current state to the layer above and resetting the cell state in the current layer. This is shown in figure 1 in layer 1 between time step 2 and 3.

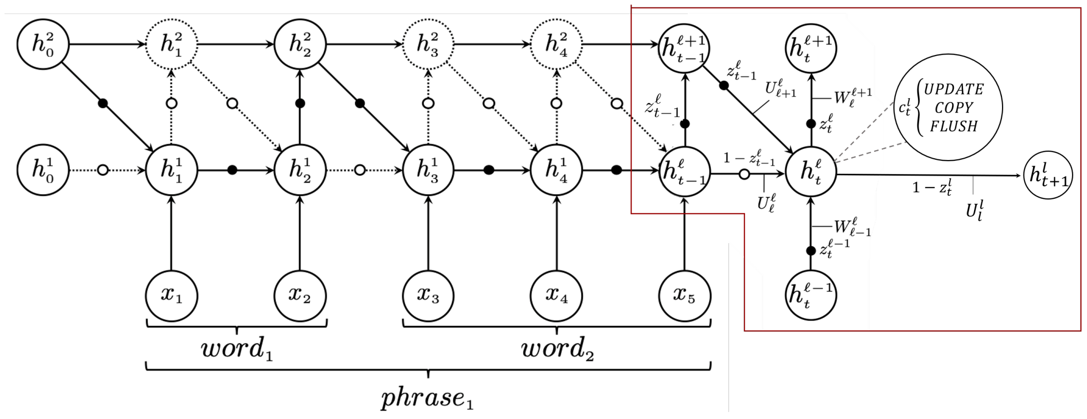

Fig. 1. The structure of a trained HMLSTM network. At each timestep (*t*) and at each layer information is either passed on (shown as a bold arrow) or not (dotted line). Information is passed on from higher layers when a boundary is detected. *This figure is adapted from figure 1 and 2 in the original article describing HM-LSTM [2].*

$$c_l^t = \begin{cases} \text{UPDATE:} & \\ f_t^l \odot c_{t-1}^l + i_t^l \odot g_t^l & \text{if } z_{t-1}^l = 0 \text{ and } z_t^{l-1} = 1 \\ \text{COPY:} & \\ c_{t-1}^l & \text{if } z_{t-1}^l = 0 \text{ and } z_t^{l-1} = 0 \\ \text{FLUSH:} & \\ i_t^l \odot g_t^l & \text{if } z_{t-1}^l = 0 \end{cases} \tag{6}$$

The hidden state of a cell is then calculated using equation 7.

$$\mathbf{h}_t^l = \begin{cases} \mathbf{h}_{t-1}^l & \text{if COPY} \\ \mathbf{o}_t^l \odot \tanh \mathbf{c}_t^l & \text{otherwise} \end{cases} \tag{7}$$

In summary, at each timestep, a boundary state, a cell state and a hidden state is calculated based on information passed on from multiple layers such that at layer *l* at time *t*, the three states are calculated using the following information:

$$\mathbf{h}_t^l, \mathbf{c}_t^l, z_t^l = f_{\text{HM-LSTM}}^l(\mathbf{c}_{t-1}^l, \mathbf{h}_{t-1}^l, \mathbf{h}_t^{l-1}, \mathbf{h}_{t-1}^{l+1}, z_t^{l-1}, z_{t-1}^l) \tag{8}$$

2.1. Backpropagation through the step function

The step function in equation 5 used to calculate the boundary state creates a problem with the backward pass since standard backpropagation requires differentiable functions. In order to solve this problem, a biased, straight-through estimator is used. This means, that the step function is only used in the forward pass and is replaced with the differentiable hard sigmoid in the backward pass. The bias originates from the difference between the step function and the hard sigmoid function. The bias can therefore be reduced by increasing a thus making the hard sigmoid more similar to the step function. As was also chosen in the original article, we start with $a = 1$ and slowly increase this value while training.

2.2. Implementation

We have based our implementation of the HM-LSTM on a PyTorch implementation of Bengio et al's original TensorFlow code [5]. The models in this paper were implemented using Google Colab and trained using the GPU provided by this service. As a comparison we have also implemented a standard LSTM based on lecture material from lecture 5 in course 02456 at DTU [6].

3. RESULTS AND DISCUSSION

3.1. Testing model performance on character level language modelling

In order to evaluate our implementation of the HM-LSTM, we test the model in language modelling on the Penn Treebank dataset using both a 2-layer and a 3-layer HM-LSTM. The input (51 distinct characters) was passed through a 128-dimensional embedding layer before being fed to the model. The predicted character was then found, using an output section consisting of an output embedding layer, which receives the hidden states of each layer in the HM-LSTM, and a softmax layer. The importance of each layer is controlled using scalar gating units with weight parameters that are learnt during training. The output embedding is hereafter calculated using the ReLU function [2] and the predicted character is found using the softmax function of this result.

The performance is measured in bits-per-character (BPC), which is the average of the cross-entropy loss using log2:

$$\text{BPC} = \mathbb{E}[-\log_2 p(x_{t+1}|x_{\leq t})] \tag{9}$$

The results achieved from the implementation of the models are shown in table 3.1. Hyperparameters such as the number of hidden units, embedding dimension and learning rate were chosen after experimenting with different settings, in order to optimise performance while not exhausting memory or running time. A batch size of 64 with a sequence length of 100 characters were used in all runs. The number of hidden units in the 2-layer HM-LSTM was chosen in accordance with the parameters chosen by Bengio et al. However, it was not possible to use 512 hidden units in the 3-layer HM-LSTM due to computational limitations. Furthermore an initial learning rate of 0.01 was used with a weight decay algorithm reducing the learning rate with 15-20% every second to fifth epoch.

	BPC	**Hidden Units**	**Epochs**
LSTM	1.43	1024	30
2L Treebank	1.58	[512, 512]	50
3L Treebank	1.47	[256, 256, 256]	30
3L Archaea	4.14	[256, 256, 256]	300
3L Bacteria	4.01	[256, 256, 256]	15

Table 1. The standard LSTM performed best (lowest BPC) in our trials when comparing the three models trained on Penn Treebank. The two models trained on biological data performed significantly worse. The models cannot be directly compared since the required resources and hyperparameters differ, but the results give an overview of the performance of each model.

3.2. Prediction and text generation for Penn Treebank

With average BPCs of around 1.5, it is clear that all the language models are quite efficient for predicting the next character in the sentence. It is however seen that the standard LSTM performs better than both a 2-layer and 3-layer HM-LSTM. There can be multiple reasons for this, but the most apparent is that because the standard LSTM model allowed a better optimisation of hyperparameters as this model was PyTorch-optimised, trained much quicker and took up less memory. The training time was much longer for the self-implemented HM-LSTM. This lack of optimisation and limited computational resources are possible reason as to why we did not achieve the same low BPC as in the original article (BPC = 1.24)[2]. The group has more computational resources available, can train with more hidden units and for many more epochs. Another reason could be, that our models were made without layer normalisation, and therefore could have a larger risk of overfitting the training data.

One interesting feature of the standard LSTM when looking at character prediction, is that it performs well in the end of a word but struggles when predicting the first couple of letters. This makes sense, as the model in the end of a word would recognise which word is about to be written. One of our hypotheses about the HM-LSTM was that this model would be better at predicting the beginning of words, as this is a task requiring knowledge of higher-order structure in the data. This tendency was however not observed, when comparing the output of our constructed models. Another observation about the models is, that although the models seem to have the ability to predict the next character in a sentence, they fail when trying to generate text themselves. When seeded with the start of a sentence, they often finish the sentence in a meaningful way, but then go into a loop repeating the same phrase over and over again such as ”the company said it will report the company ’s shares outstanding [eos] and the company said ...”. Intuitively we would blame the lack of long-term memory in the standard LSTM for this problem, unfortunately, the HM-LSTMs have the same problem.

Figure 2 shows an example output of the 3-layer HM-LSTM. Here it is seen that the model detects some form of hierarchical structure, with the first layer corresponding to characters, thus updating every time, and the second layer not directly corresponding to words, but at least updating in a way which is similar to the structure of words.

3.3. Biological data

The 3-layer HM-LSTM model was also was also trained on protein sequences from various species of archaea and bacteria. All sequences were of high quality, as they are Swiss-Prot results taken from Uniprot [7]. Since the basic language of protein sequences is identical between domains (bacteria, virus and eukarya), the model could likewise have been implemented on data from other groups of organisms.

For the archaea dataset a BPC of 4.14 was achieved. This is a quite low model performance, as there are only 20 different amino acids to choose between in each prediction. Nevertheless, it is not completely random, and the models have learnt to predict the data to some extent. This performance might be due to the limited size of the dataset (around 3000 sequences/2.1 MB) or it could be because these amino acid sequences might comprise of characters organised in a system, which is simply harder to predict than the characters/words in the Penn Treebank database.

To inspect if the model would improve given more sequences, the bacteria data set (43MB) was examined. This model achieved a lower albeit still high BPC of 4.01 and showed quite some problems with overfitting. Protein sequences always start with the same amino acid (M) and this boundary was correctly predicted by our model. However, the model was not able to predict other hierarchical structures in any of the two domains.

3.4. Potential errors, optimisation and future work

As previously mentioned it is clear that the HM-LSTM models work, learn and provides us with hierarchical structures

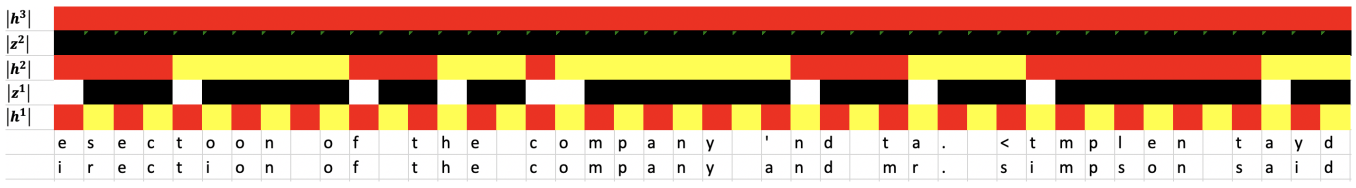

Fig. 2. Hierarchical multiscale structures and character prediction from the output of a 3-layer HM-LSTM. The bottom line shows correct sentence and the line above shows the model's prediction. To illustrate how the model predicts hierachies, the lines showing hidden states (***h***) change colour with the operations UPDATE or FLUSH, but remain the same colour with the operation COPY. The values of the boundary detectors z are also shown where black corresponds to 0 and white to 1.

of the input. But as also indicated, the results are nowhere as promising as those shown by Bengio et al. We think that this might be solved with further optimisation and increased computing power. We believe that some main focus points of improvement would be building a three-layer HM-LSTM with 512 neurons in each layer, optimising the learning rate and training for more epochs. Furthermore, the models are sometimes prone to overfitting. A proposed fix for this would be to implement layer-norm regularisation in the future as is also done in the original implementation [2].

4. CONCLUSION

In this paper, we have examined the efficiency of HM-LSTM models for detection and prediction of hierarchical structures in temporal data. We have successfully implemented both a 2-layer and a 3-layer HM-LSTM and evaluated their performance using character level language prediction on the Penn Treebank dataset and on protein sequences from archaea and bacteria. With the Penn Treebank datset and the 3-layer HM-LSTM model we achieved a BPC of 1.47, which is close to but higher than the benchmark of 1.43 found with a standard LSTM. Neither the standard LSTM or any of the HM-LSTMs showed great performance in text generation, but had tendencies to fall a repeating loop of words. This could be due to bad long-term memory. Performance was drastically worse on biological data, where we only obtained a BPC of 4.01 using a dataset of amino acid sequences from bacteria. Due to the promising results of Bengio et al and the large potential of hierarchical models in bioinformatics, we believe that this class of models deserve future work outside the scope of this project. Our recommendation for future optimisation is to 1) implement layernorm regularization to prevent overfitting, 2) optimize learning rate and decay to learn more efficiently and 3) to built a 3-layer model with 512 hidden units in each layer for better performance.

5. REFERENCES

[1] Jan Koutník, Klaus Greff, Faustino J. Gomez, and Jürgen Schmidhuber, "A clockwork RNN," *CoRR*, vol. abs/1402.3511, 2014.

[2] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio, "Hierarchical multiscale recurrent neural networks," *arXiv preprint arXiv:1609.01704*, 2016.

[3] Taylor A., Marcus M., and Santorini B., "The penn treebank: An overview," *Abeillé A. (eds) Treebanks. Text, Speech and Language Technology*, vol. 20, 2003.

[4] B. Kuhlman and P. Bradley, "Advances in protein structure prediction and design," *Nat Rev Mol Cell Biol*, vol. 20, pp. 681–697, 2019.

[5] Hanqing Lu., *Hierarchical. Multiscale Neural Networks Implementation in Pytorch*, Accesses 4-01-2020, `https://github.com/HanqingLu/MultiscaleRNN`.

[6] Ole Winther, *Deep learning (course 02456) colab exercise 5 (recurrent neural networks)*, Access 03-01-2020.

[7] Protein sequences for Archaea and Bacteria were provided by Jose Juan Almagro Armenteros.

Git repository

All code written for the project, as well as the examined biological data, can be found at our GitHub:

`https://github.com/DeepLearningProject-HMLSTM/HMLSTM`