Hyphenation using deep neural networks

Gergely Dániel Németh and Judit Ács
neged.ng@gmail.com, judit@aut.bme.hu

Budapest University of Technology and Economics
Department of Automation and Applied Informatics

Abstract. Hyphenation algorithms are the computer based ways of syllabification and mostly used in typesetting, formatting documents as well as text-to-speech and speech recognition systems. We present a deep learning approach to automatic hyphenation of Hungarian text. Our experiments compare feed forward, recurrenct and convolutional neural network approaches.
Keywords: hyphenation, machine learning, feedforward neural network, recurrent neural network, long short-term memory, convolutional neural network

1 Introduction

Hungarian children learn the rules of syllabification in their early teens. Hyphenation rules are clearly defined in [1], and after years of practice most people use it naturally and this is seen as part of the common knowledge.

In Hungarian hyphenation depends mostly on the word itself and less so on context. Polysemous words may have different hyphenations in different senses which can only be derived from the context as shown in the example *me-gint* (again) and *meg-int* (warn). Although it is a known issue, most hyphenation algorithms hyphenate words with no regard to context and assign only one possible hyphenation for a word.

Commonly used hyphenation algorithms are based on the methods defined in the first version of TEX [2]. It is a pattern based hyphenation algorithm with thousands of manually chosen patterns. It's employed in the online syllabification system[1] created and maintained by RIL HAS[2]. [3] describes the collection of patterns used in this system.

While we have enormous amount of words in corpora, we lack a gold standard corpus of pre-hyphenated words. One way to create hyphenated words is to use already available hyphenation algorithms.

2 Related Work

The world of open-source software *de facto* uses the TEX's hyphenation algorithm. The TEXcurrently uses the Hunspell's hyphenation algorithm which based on Liang's pattern matching algorithm.

[1] http://helyesiras.mta.hu/helyesiras/default/hyph
[2] Research Institute for Linguistics of the Hungarian Academy of Sciences

2.1 Liang's algorithm

The basic concepts of Liang's algorithm are the hyphenation patterns. The process of hyphenating is the following: the algorithm finds all the matching patterns. These patterns have numbers in them (skipped while matching). The odd numbers predict hyphens. The method chooses the maximum of the numbers between each letters (zero if not given).

The word *hyphenation*'s patterns are the following: `hy3ph, he2n, hena4, hen5at, 1na, n2at, 1tio, 2io` [4, page 37]. Placing the patterns in the right position and inserting the numbers in the patterns between the letters we got Figure 1.[3]

```
. h y p h e n a t i o n .
  h y3p h
        h e2n
        h e n a4
        h e n5a t
           1n a
            n2a t
               1t i o
                 2i o
.0h0y3p0h0e2n5a4t2i0o0n0.
  h y-p h e n-a t i o n
```

Fig. 1: The hyphenation of 'hyphenation' by Liang's algorithm

2.2 Hunspell

Hunspell's hyphenation algorithm is currently used in TEX and OpenOffice. It is based on Liang's work with Sojka's non-standard hyphenation extensions [6] and was published by Németh in 2006 [5]. Hunspell supports non-standard hyphenation patterns such as the hyphenation of the German word *Zucker* – *Zuck-ker* (the exact pattern is `c1k/k=k`.

Different languages use different non-standard patterns whose sizes and types vary significantly.

2.3 Errors in the Hunspell's hyphenation

Most of the hyphenation errors come from the fact that the Hunspell's creators wanted to create a typesetting algorithm so they decided to not hyphenate one letter long syllables the begining and the ending of the words. However, when it comes to compound words, these one-letter parts can be in the middle of the word. There are some examples of hyphenation errors in Table 1.

[3] The visualization method comes from Németh's article [5].

Hyphenation by Hunspell:	Correct hyphenation:	Error type:
au-tó-val	a-u-tó-val	one-letter
szem-üveg-gel	szem-ü-veg-gel	one-letter
has-izom	has-i-zom	one-letter
messze	mesz-sze	no hyphen
fölül	föl-ül	no hyphen
top-ikok	to-pi-kok	no hyphen[4]
vi-deó	vi-de-ó	one-letter
geo-dé-zia	ge-o-dé-zi-a	no hyphen
diszk-ri-mi-na-tív	disz-kri-mi-na-tív	wrong place

Table 1. Hyphenation errors in Hunspell

2.4 Neural networks for hyphenation

[7] is the only paper we found specifically on automatic hyphenation using neural networks. Due to technical constraints they only tested on a very small training dataset. Our first approach (feedforward neural net) is very similar to the one they present.

3 Neural networks

The following summary of neural networks is based on the *Deep learning in neural networks: An overview* by Schmidhuber [8].

3.1 Feedforward neural network

A feedforward neural network approximates any given function f as $y = f(x, T)$ where T represents those parameters with which the model can learn to achieve the best approximation. These networks are called feedforward because the information flows through the function from x to inner (hidden) parts and finally to y. There are no directed cycles or loops in the network.

The simplest model is a single-layer perceptron which has a weight W and bias b, so for an input x can compute the $\hat{y} = Wx+b$ function where the learning method optimizes the weight W and the bias b. Later on researchers showed that adding a non-linear *activation function* to it can fasten the learning method and makes it usable in non-linear functions [9]. So from the $\hat{y} = Wx + b$ the function changed to $z = Wx + b$ and the prediction became $\hat{y} = g(z)$.

The deep neural network's name comes from that instead of a single-layer perceptron the output of the above equation $g(z)$ now called as h_1 is used as the input of the next layer and so for many-many layers. So for the first layer it became $z_1 = W_1x + b_1$ and $h_1 = g_1(z_1)$ and to the second layer: $z_2 = W_2h_1 + b_2$ and $h_2 = g_2(z_2)$ and so on, until the last, nth layer where the a_n became the prediction $h_n = \hat{y}$. Figure 2. summarizes idea of feedforward networks.

[4] This word is hyphenated incorrectly by the online hyphenation tool of the Research Institute for Linguistics.

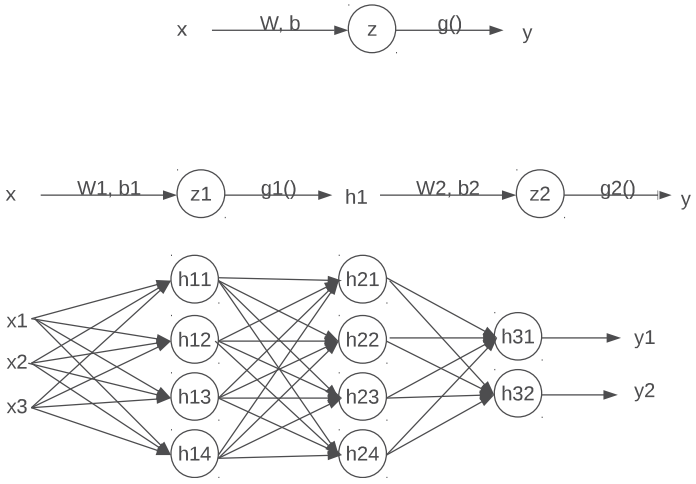

Fig. 2: Basics of Feedforward Neural Networks (F(F)NN)

A single iteration of training consists of a forward step where the model predicts $\hat{y}$, the evaluation step where the model compare the $\hat{y}$ and y with some type of gradient descent [10] and lastly a backpropagation step where the model updates the weights [11].

3.2 Convolutional neural network

Convolutional neural networks were introduced to solve image recognition problems [12]. A convolutional neural network has a filter (or kernel) which is sliding around the input (image) and multiplying the values in the filter (the weights of the filter) with the original input values (pixels). Summing up these values the network get a single number for every position of the filter. This will be the output of the layer. The size of the output depends on the filter size and the parameter *strides* which defines the steps of the filter's sliding.

Let a_{ij} be the cell (pixel) of the input (image) in the ith row and jth column and f_{ij} the cell of the filter, while h_{ij} the output. Thus the first cell of convolutional layer's output is

$$h_{11} = \sum_{x=1..k,y=1..l} a_{xy} f_{xy},$$

(where k and l are the height and width of the filter respectively), and assuming that the stride is 1, the h_{ij} is:

$$h_{ij} = \sum_{x=i..(i+k),y=j..(j+l)} a_{xy} f_{(x-i+1),(y-j+1)}.$$

Figure 3. illustrates a convolutional network with a $(3, 3)$ kernel.

Kim, Jernite, Sontag and Rush showed a way of using 1 dimensional convolutional neural networks and LSTM networks in character sequences [13], where

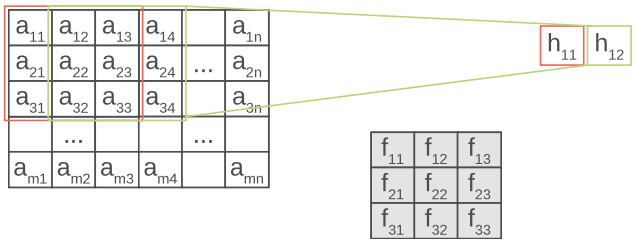

Fig. 3: Basics of Convolutional Neural Network (CNN)

the filter size says that how many character should be included into the convolution.

3.3 Recurrent neural network - LSTM

A recurrent neural network (RNN) is suited for modelling sequential phenomena. At each time step t, an RNN takes the input vector $x_t \in \Re^n$ and the hidden state vector $h_{t-1} \in \Re^m$ and produces the next hidden state h_t by applying the following recursive operation: $h_t = f(Wx_t + Uh_{t?1} + b)$. In theory, an RNN can store all information in h_t, however learning long-range dependences with it is difficult due to vanishing/exploding gradients [14].

Long short-term memory (LSTM) [15] addresses the problem of learning long range dependencies by augmenting the RNN with a memory cell vector $c_t \in \Re^n$ at each time step. Concretely, one step of an LSTM takes as input x_t , $h_{t?1}$, $c_{t?1}$ and produces h_t , c_t via the following intermediate calculations:

$$
\begin{aligned}
i_t &= \sigma(W^i x_t + U^i h_{t-1} + b_i) \\
f_t &= \sigma(W^f x_t + U^f h_{t-1} + b_f) \\
o_t &= \sigma(W^o x_t + U^o h_{t-1} + b_o) \\
g_t &= \tanh(W^g x_t + U^g h_{t-1} + b_g) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Here $\sigma(\cdot)$ and $\tanh(\cdot)$ are the element-wise sigmoid and hyperbolic tangent functions, $\odot$ is the element-wise multiplication operator, and i_t, f_t, o_t are referred to as *input*, *forget*, and *output* gates. At $t = 1$, h_0 and c_0 are initialized to zero vectors. Parameters of the LSTM are W^j , U^j, b^j for $j \in i, f, o, g$. See the visualisation of an LSTM unit in Figure 4.[5]

Bidirectional recurrent neural networks are based on the principle to split the neurons of a regular RNN into two directions, one for positive time direction (forward states), and another for negative time direction (backward states) [16]. In terms of characters in a word it means that the letters can affect their surroundings on both sides.

[5] Visualisation and more:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

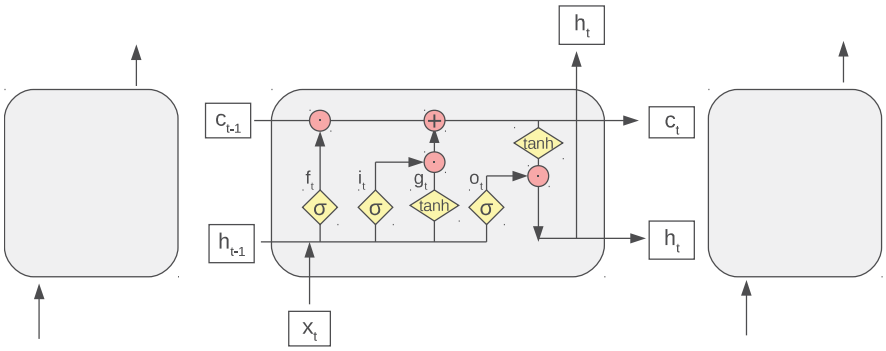

Fig. 4: Basics of Long short-term memory (LSTM)

4 Data and preprocessing

Hungarian Webcorpus is one of the largest Hungarian language corpora with over 600 million words and it is available in its entirety under a permissive Open Content license [17,18]. We used the 100000 most frequent words from the corpus.

4.1 Data preprocessing

The preparation methods used before the training:

Cleaning Numbers, punctuation characters are cleaned, setting all letters to lower-case.

Filtering non-standard hyphenation Because the models use character classification, they cannot handle letter addition or changing. The preprocessing method filters them however the models may find the position of the hyphenation.

Filtering special characters The models only accept the Hungarian alphabet and the padding characters.

Long word cut The CNN and LSTM networks wait for given length words so the longer words are dropped and the shorter ones are filled with padding characters.

Table 2. illustrates the preprocessing steps and the amount of dropped words.

	Origin	Cleaning	Non-standard	Special chars	Long words	Final
Words	100000	16322	1115	646	7	81910
% of the origin	100	16.32	1.11	0.65	0.01	81.91
% of the previous	-	16.32	1.33	0.78	0.01	
Words after	-	83678	82563	81917	81910	

Table 2. Data preprocessing

5 Experimental setup

We implemented and tested three neural network architectures described in the previous Section.

5.1 Character classification

Automatic hyphenation can be defined as a sequence labeling problem. We assign a binary label for each character:

- B: In the beginning of the syllables.
- M: Every other letter.

The word *leopárd* (leopard) hyphenated as `le-o-párd` and tagged as `BMBBMMM`.

There are more fine-grained labeling scemes such as the `BMES`, which denotes the middle of segments as well as single-character segments, they require a more complicated inference scheme so we decided to use the binary labeling system.

5.2 Feedforward neural network

In the feedforward neural network we want to decide to which class a specific character belongs. To do it we use the character and its surroundings coded in one-hot as the input layer of a fully connected network. The output is the class.

The network is summarized in Figure 5. The steps of the character classification.

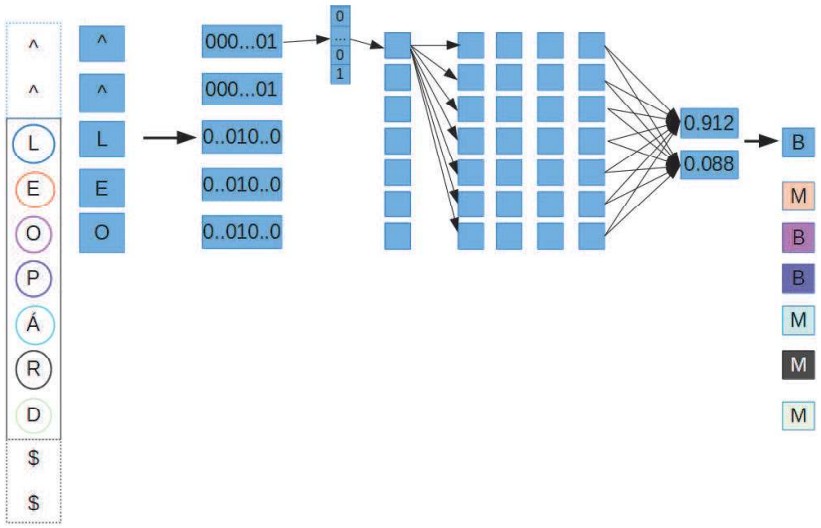

Fig. 5: Feedforward neural network

Learning from the surroundings (Windowing) In an n-length window we use $(n-1)/2$ characters before the given one, itself and $(n-1)/2$ characters after. For example, if the character is the letter l in the word *leopárd*, and the window-length is 5, the window will be `^^LEO`.

One-hot encoding We used one-hot encoding to represent characters as vectors. We define a 37 length array for each letter in the word: each element of the array means one letter of the Hungarian characters (35 letters) or the beginning or ending characters. One-hot means that there is only one 1 in the array, the others are 0. For the letter *a* it is the first one and the character *o* it is the 18th.

Thus if there are 5 characters in a window, a $(5, 37)$ shaped two-dimensional array is given.

Flattening The final step before the training is flattening. From the two-dimensional array we create a reshaped one-dimensional. This means simply putting the characters after each other. So if we had the $(5, 37)$ array, now we have the $5 \cdot 37 = 185$ long 0,1 sequence with only 5 ones in it. And this is the training data.

In summary, from the *leopá* window we got a 185 long $0, 1$ as the training input and $[1, 0]$ as the training output.

The neural network The neural network is a fully connected feedforward neural network. In the hyperparameter optimization along the window length, the number of hidden layers and the units in each hidden layer had been optimized. The last layer has a softmax activation to approximate the values as probabilities of **B** and **M** tags.

The training process used *sigmoid* activation functions and *adam* optimizer [19].

5.3 Convolutional neural network

For the CNN and LSTM networks the data preparation steps are the same. First, using the Hunspell hyphenation to define the labels, then filling the words to a fixed size with padding (the filling label is **M**) and finally using the one-hot encoding method (Figure 6.).

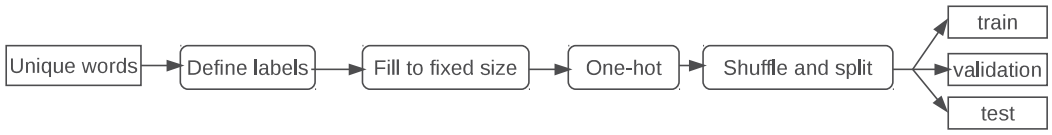

Fig. 6: CNN and LSTM data preparation

The model has two parts. The first one is the convolutional network and the second is a feedforward network. The convolutional layers have a stride one so it convolves all the characters. The kernel size (it is the same dimension as the window-length in the FFNN model), filter and hidden layer numbers were optimized. The Keras' built-in padding was used to prevent problems at the endings of the word. The model currently uses ReLU activation functions.

The feedforward part is a softmax layer for every character to get back the **BM** one-hot probabilities.

Figure 7. summarizes the CNN model.

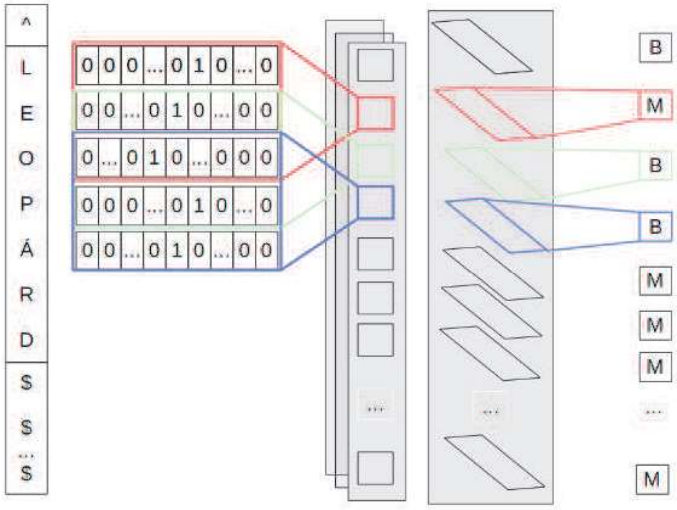

Fig. 7: CNN model summary

5.4 Long short-term memory

The LSTM network uses the same inputs as the CNN. The model uses BiLSTM and optimized by the unit and hidden layer numbers. At the output of the LSTM a softmax feedforward layer was inserted just as in the CNN one. The model summary of the LSTM network is in Figure 8.

5.5 Hyperparameter optimization

Hyperparameters were optimized via random search. Parameter values were uniformly sampled from the ranged listed in Table 3..

5.6 Results and error analysis

The best results are listed in Table 4., where the precision, recall and F-score values are for the **B** tagging while the word accuracy is the rate of successfully hyphenated words.

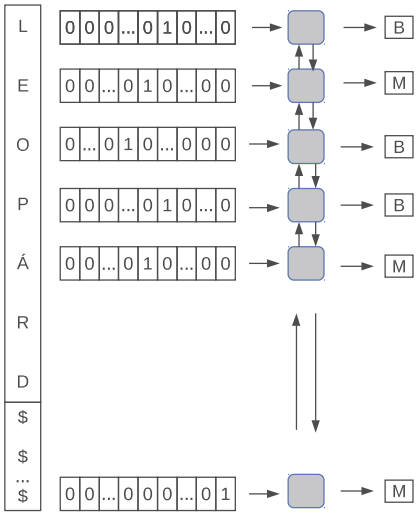

Fig. 8: Long Short-Term Memory network

	Window/Kernel	Hidden layers	Hidden units
FFNN	3-11	3-7	60-180 (step 10)
CNN	5-16	1-3	64-2048 (exponential scale)
LSTM	-	1-3	8-256 (exponential scale)

Table 3. Hyperparameter ranges

Model	Kernel	Layers	Hidden	Epochs	Trainable parameters	Precision	Recall	F-score	Word accuracy
FFNN	7	3	150	103	84,602	98.17%	99.11%	98.64%	93.57%
CNN	8	2	1024	12	8,695,810	98.37%	99.27%	98.81%	94.45%
LSTM	-	2	128	66	216,834	97.68%	99.16%	98.42%	93.13%

Table 4. The three model that performed the best.

All the errors of the three models were collected and 200 words were manually classified into the error categories. Figure 9. shows the category distribution of the errors for each model and among the errors. As we can see, over half of the errors are caused by using a Hungarian hyphenation algorithm on a non-Hungarian word.

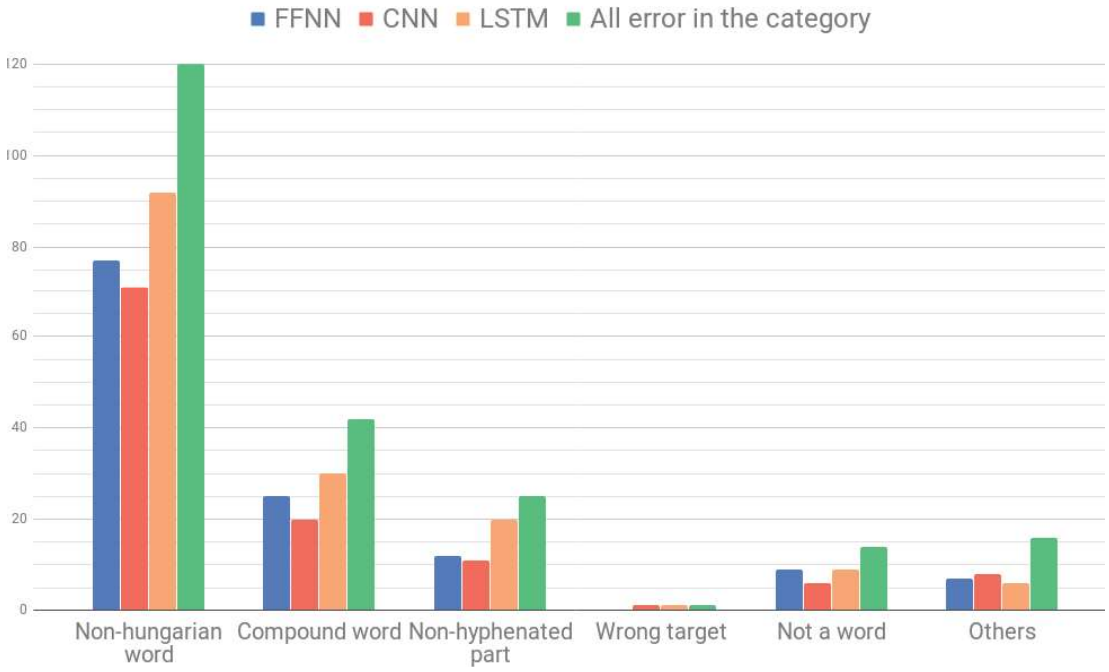

Fig. 9: Category distribution in the 200 wrong words

We manually checked and grouped the both correctly and incorrectly hyphenated words into the following categories:

Non-hungarian word is which was recognized as a word but not Hungarian.
Compound word
Non-hyphenated part has a hyphen missing because of Hunspell's typesetting goals.
Wrong target Hunspell misses the hyphenation.
Not a words are mostly mistyped Hungarian words like *elol*.
Other words not filling the above categories.

Table 5. shows category distribution among all the words (correctly predicted words as well as incorrect ones). Note that one word can be in multiple categories.

6 Conclusion

In this paper we introduced a series of experiments on Hungarian hyphenation using deep neural networks. We compared three architecutres with varying hy-

Category	Distribution	Example (Hunspell's hyphenation)
Non-Hungarian word	11	obsta-c-les
Compound word	21	ak-ció-film
Non-hyphenated part	8	ak-ció-film
Wrong target	1	diszk-ri-mi-na-tív
Not word	4	el-er-ni
Others	59	

Table 5. Word categories among 100 randomly chosen words

perparameters: feedforward, recurrent and convolutional neural networks. We trained and tested on the 100000 most frequent words from the Hungarian Webcorpus. All models achieve over 95% word accuracy, occasionally correcting the errors made by Hunspell. Our error analysis suggests that foreign and compound words are the most challenging for our systems. Our source code and the best models are available on GitHub.[6]

References

1. Akadémia, M.T., Budapešt', M.: A magyar helyesírás szabályai. Akadémiai kiadó (1959)
2. Knuth, D.E.: TEX and METAFONT: New directions in typesetting. American Mathematical Society (1979)
3. Miháltz, M., Hussami, P., Ludányi, Z., Mittelholtz, I., Nagy, Á., Oravecz, C., Pintér, T., Takács, D.: Helyesírás. hu. (2013)
4. Liang, F.M.: Word hyphenation by computer. Department of Computer Science, Stanford University (1983)
5. Németh, L.: Automatic non-standard hyphenation in openoffice. org. COMMUNICATIONS OF THE TEX USERS GROUP TUGBOAT EDITOR BARBARA BEETON PROCEEDINGS EDITOR KARL BERRY (2006) 32
6. Sojka, P., et al.: Notes on compound word hyphenation in tex. TUGboat **16**(3) (1995) 290–296
7. Kristensen, T.: A neural network approach to hyphenating norwegian. In: Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on. Volume 2., IEEE (2000) 148–153
8. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural networks **61** (2015) 85–117
9. Fahlman, S.E.: An empirical study of learning speed in back-propagation networks. (1988)
10. Hadamard, J.: Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées. Volume 33. Imprimerie nationale (1908)
11. Hecht-Nielsen, R., et al.: Theory of the backpropagation neural network. Neural Networks **1**(Supplement-1) (1988) 445–448
12. Fukushima, K., Miyake, S.: Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In: Competition and cooperation in neural nets. Springer (1982) 267–285

[6] https://github.com/negedng/deep-hyphen

13. Kim, Y., Jernite, Y., Sontag, D., Rush, A.M.: Character-aware neural language models. In: AAAI. (2016) 2741–2749
14. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks **5**(2) (1994) 157–166
15. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8) (1997) 1735–1780
16. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing **45**(11) (1997) 2673–2681
17. Halácsy, P., Kornai, A., Laszlo, N., Andras, R., Szakadát, I., Viktor, T.: Creating open language resources for hungarian. (2004)
18. Kornai, A., Halácsy, P., Nagy, V., Oravecz, C., Trón, V., Varga, D.: Web-based frequency dictionaries for medium density languages. In: Proceedings of the 2nd International Workshop on Web as Corpus, Association for Computational Linguistics (2006) 1–8
19. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)