BSc Thesis Task Description

Gergely Dániel Németh
candidate for BSc degree in Computer Engineering

Multilingual hyphenation using deep neural networks

Hungarian children learn the way of hyphenation in their early teens. The hyphenation rules are clearly defined and after years of practice most people use it naturally and it is seen as part of the common knowledge.
Automatic hyphenation currently operates with hyphenation pattern based algorithms which have been developed since the first edition of the TeX. These algorithms are good enough to be used by the Research Institute for Linguistics of the Hungarian Academy of Sciences in its online hyphenation portal.
With the rise of the deep learning paradigms increased the demand for solving natural language processing (NLP) problems by machine learning. The tremendous amount of the corpora found in the digital world is a good basis for generating training data. Since the hyphenation algorithm used in TeX works with few errors, it can be used to obtain practically unlimited learning data. One of the present study's purposes is to assess how the machine learning algorithms are able to adapt these rules that children can learn quite easily.
The student's task is to test different deep learning models in the field of hyphenation. The development of a multi-language usable algorithm can take forward the study of linguistics and palaeography or even the improvement of text-to-speech algorithms.
The deep learning models used in the thesis: feedforward neural network, convolutional neural network, recurrent neural network, sequence-to-sequence network.
Tasks to be performed by the student will include:

- Describe the various deep learning models.
- Describe the currently used hyphenation algorithms and analyze their accuracy.
- Develop a hyphenation algorithm based on the models.
- Analyze the effectiveness of these algorithms and compare the different models.
- Create a multilingual hyphenation tool using the discussed models.
- Publish the source code of the completed program as a Python package.

Supervisor at the department: Judit Ács, assistant lecturer

Budapest, 4 October 2017

Dr. Hassan Charaf
professor
head of department

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Multilingual hyphenation using deep neural networks

BSc Thesis

Author
Gergely Dániel Németh

Advisor
Judit Ács

December 8, 2017

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Gergely Dániel Németh*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2017. december 8.

Gergely Dániel Németh
hallgató

Kivonat

A szótagoló algoritmusok az elválasztás feladatának számítógépes megoldásai és legtöbbször dokumentumok tördelésekor használják. Mindemellett a szavak elválasztásának hatása van a költészetben vagy akár a szövegfelolvasó, szövegfelismerő alkalmazások fejlesztésében is.

A mélytanulásos módszerek előretörésével megnőtt az igény a nyelvtechnológiai problémák gépi tanulás alapú megoldására is. Az online elérhető corpus adatbázisok mennyisége elősegíti, hogy a módszereket kipróbálhassuk olyan problémákon is, mint a szótagolás.

Az itt következő dolgozatban a szerző egy új fajta szótagoló algoritmust ismertet. A jelenleg használt szótagoló algoritmusok rövid ismertetője és három nyelvtechnológiában elterjedt neurális háló bemutatása után rátér ezek alkalmazására a szótagolás terén.

A dolgozat ismertet továbbá egy negyedik hálót is, a nem standard elválasztási problémák megoldására, valamint bemutat egy többnyelvű elválasztó algoritmust.

Kulcsszavak: elválasztás, gépi tanulás, előrecsatolt neurális háló, rekurrens neurális háló, konvolúciós háló, sequence-to-sequence model

Abstract

Hyphenation algorithms are the computer based ways of syllabification and mostly used in typesetting and document formatting. In addition, the hyphenation of words affects the poetry as well as text-to-speech and speech recognition algorithms.

The rise of the deep learning paradigms increased the demand for solving natural language processing problems by machine learning. The amount of online available corpora facilitates the use of these paradigms in problems like hyphenation.

In the following thesis, the author shows a new hyphenation algorithm. After a short summary of the currently used algorithms, the study describes three neural networks used in Natural Language Processing and shows a way of their application in the field of hyphenation.

The thesis also implements a fourth model for non-standard hyphenations and introduces a multilingual hyphenation algorithm.

Keywords: hyphenation, machine learning, feedforward neural network, convolutional neural network, long short-term memory, sequence-to-sequence model

Introduction

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

In Hungarian, hyphenation depends mostly on the word itself and not on its surroundings. However, some words have different hyphenations based on the meaning which can only be derived from the context. For instance, *me-gint* (again) and *meg-int* (warn). Although it is a known issue, most hyphenation algorithms use only the word to insert hyphens and defines exactly one hyphenation for a word.

Commonly used hyphenation algorithms are based on the methods defined in the first version of TEX [14]. It is a pattern based hyphenation algorithm with thousands of manually chosen patterns. Nevertheless, its accuracy is good enough for the Research Institute for Linguistics of the Hungarian Academy of Sciences to collect its own patterns and use it in the online Hungarian hyphenation portal [18]. On this website[1] ,users can verify their syllabification.

In natural language processing (NLP) , corpus is a collection of linguistic data. Although this corpus consists of linguistic information and statistics of the words, not merely a group of words or sentences, in this thesis only the last two are used.

Despite the fact that one can find an enormous amount of words, it is hardly possible to find pre-hyphenated ones in the corpora. That is the reason why in this thesis, the author uses already available hyphenation algorithms to create hyphenated words.

[1] http://helyesiras.mta.hu/helyesiras/default/hyph

Chapter 1

Hyphenation algorithms

There are two main types of common hyphenation: rule-based (simple methods applied on the words) and dictionary-based (long files of pre-hyphenated words). The world of open-source software *de facto* uses the TEX's hyphenation algorithm.

The following is an appropriate summary of the early TEX hyphenation algorithm by Liang [17, page 3]:

> The original TEX hyphenation algorithm was designed by Prof. Knuth and the author [of his book] in the summer of 1977. It is essentially a rule-based algorithm, with three main types of rules: (1) suffix removal, (2) prefix removal, and (3) vowel-consonant-consonant-vowel (vccv) breaking. The latter rule states that when the pattern 'vowel-consonant-consonant-vowel' appears in a word, we can in most cases split between the consonants. There are also many special case rules; for example, "break vowel-q" or "break after ck". Finally a small exception dictionary (about 300 words) is used to handle particularly objectionable errors made by the above rules, and to hyphenate certain common words (e.g. pro-gram) that are not split by the rules. The complete algorithm is described in Appendix H of the old TEX manual.

This book of Liang was published in 1983 after the algorithm described in it had became the default hyphenation of the TEX82 version of TEX. The most important innovation of the new algorithm was that it used hyphenation *patterns* which are basically schemes that the program can look for in the words.

The latest version of TEX uses the Hunspell's hyphenation algorithm [19].This method is based on Liang's algorithm completed with *non-standard hyphenation extensions*.

The following sections show the basics of Liang's algorithm and the Hunspell [17, 19].

1.1 Liang's algorithm

The basic concepts of Liang's algorithm are the hyphenation patterns, defined and selected for every language manually. The process of hyphenating the word `hyphenation` is the following:

First of all, the algorithm checks if the word is in the exception list. These are essentially hard-coded hyphenations of full words which have rare or unique hyphenation. The word *hyphenation* is not in the exception list.

Secondly, the algorithm inserts a dot in both ends of the word as marker. This will be used when the algorithm checks for patterns used only in the beginning or the ending of words. The characters of the words are handled differently there as it cannot go to the previous or next one.

```
.hyphenation.
```

The next step is the pattern matching. The patterns used in Liang's algorithm consist of numbers and characters that can possibly occur in the given language. When it searches for matching patterns in a word, it skips the numbers and compares the letters with every same-length substring of that word.

Hyphenation's patterns according to the English pattern dictionary are the following [17, page 37]:

```
hy3ph, he2n, hena4, hen5at, 1na, n2at, 1tio, 2io
```

Placing the patterns in the right position of the word, and inserting the numbers given in the patterns we got Figure 1.1.[1]

After matching all the patterns, the algorithm inserts one number between every two letters of the word. This number will be the maximum of the matching numbers, otherwise zero.

In the final step the algorithm hyphenates at odd numbers and does not hyphenate in case it is even. Therefore, the hyphenation of *hyphenation*: `hy-phen-ation`.

The effectiveness of the algorithm depends on the choice of patterns. In 1982, Liang's goal was to achieve good error rate with relatively low use of disk space. The final program used around 4500 patterns, occupied 25K bytes of storage and found 89% of the hyphens

[1] The visualization method comes from Németh's article [19].

```
. 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
```

Figure 1.1. *The hyphenation of 'hyphenation' by Liang's algorithm*

in the dictionary it was tested on. The complete hyphenation dictionary of these words would be 500K bytes.

The choice of patterns was manual but with the newer machine learning mentality it would have been optimized with a deep neural network.

1.2 Hunspell

Hunspell's hyphenation algorithm is currently used in TEX and OpenOffice. It is based on Liang's work completed with Sojka's non-standard hyphenation extensions [23] and was published by Németh in 2006 [19].

During the creation of non-standard hyphenations, Liang's patterns were used, as well as an extension to character replacement. For example the German word *Zucker* with the non-standard hyphenation `c1k/k=k` gets a new *k* letter before the hyphen: *Zuck-ker*.

The different languages use different non-standard patterns whose sizes and types vary significantly. Here we summarize the list of Hungarian non-standart hyphenations.

1.2.1 Hungarian non-standard hyphenation patterns

The Hungarian language uses simplified forms to represent its double digraph and trigraph consonants ($sz + sz \rightarrow ssz$, $dzs + dzs \rightarrow ddzs$, etc.), but when the word is hyphenated into two part around these letters, it undoes the simplification creating a character addition (*sz-sz, dzs-dzs*). A classic example: *asszonnyal* $\rightarrow$ *asz-szony-nyal*.

Its difficulty comes from the fact that these digraph-simplifications can represent two letters (a monograph and a digraph) like *ggy* as g-gy in *meg-gyúj-tot-ta* or double digraph

as *meggyes* → *megy-gyes*. So the algorithm must know whether it is two digraphs simplified or not. Table 1.1. illustrates these extensions in the v20110815 version of the Magyar Ispell (Hunspell Hungarian pattern dictionary)[2].

Digraph	Example	No. of patterns
ccs	fröc5csen/cs=,4,1	96
ggy	meg3gyes/gy=,3,1	39
lly	gal5lya/ly=ly,3,3	20
nny	szen5nye./ny=ny,4,3	97
ssz	hos5szal./sz=,3,1	1727
tty	hat5tyú/ty=ty,3,3	18
zzs	z5zsel./zs=zs,1,3	4

Table 1.1. *The hyphenation of 'hyphenation' by Liang's algorithm*

The meaning of the patterns showed in the `fröc5csen/cs=,4,1` example is the following: the part before the slash character is the standard hyphenation pattern, after it comes the redefinition: `cs=,4,1` meaning that from the fourth character of the pattern for one letter long it changes the characters as *cs*= where the = represents the possible hyphen. In this case, the letter *c* will be replaced by the *cs*= so if it happens to be a hyphenation break, the algorithm will add a *s* character into the word (*fröcs-csen*).

1.2.2 Hyphenation errors of the Hunspell

Most of the hyphenation errors in Hunspell come from the fact that its creators wanted to develop a typesetting algorithmm. Therefore they decided not to hyphenate one letter long word parts at the beginning or the ending of the words, since a one letter long part in the end of a line in a text looks strange. 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 1.2.

[2]http://magyarispell.sourceforge.net/

[3]This word is hyphenated wrongly even in the online hyphenation tool of the Research Institute for Linguistics.

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[3]
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 1.2. *Hyphenation errors in Hunspell*

Chapter 2

Neural networks

The following summary of neural networks is based on the *Deep learning in neural networks: An overview* by Schmidhuber [20]. It explains many commonly used machine learning practice with their first usage and development path.

2.1 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.

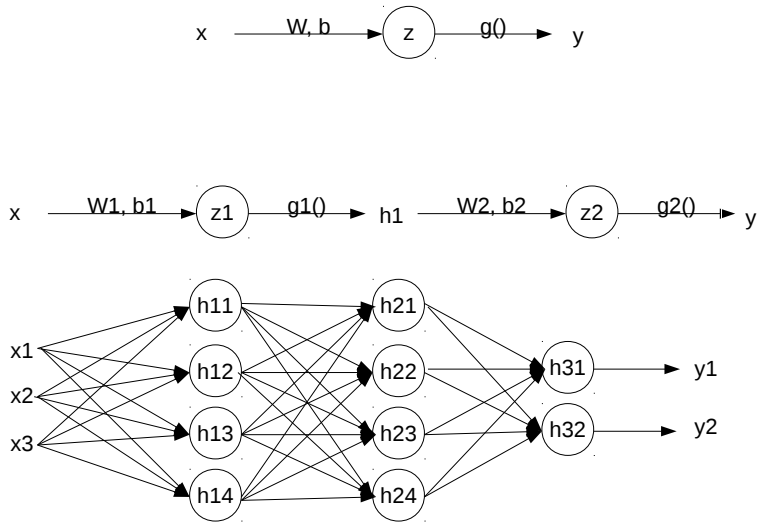


Figure 2.1. *Basics of Feedforward Neural Networks (F(F)NN)*

The simplest model is a single-layer perceptron which has a weight W and a 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 can make it usable in non-linear functions [5]. 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 $z_1 = W_1 x + b_1$ and $h_1 = g_1(z_1)$ and to the second layer: $z_2 = W_2 h_1 + b_2$ 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 2.1. summarizes idea of feedforward networks.

A single iteration of training consists of a forward step where the model predicts $\hat{y}$, an evaluation step where the model compares the $\hat{y}$ and y with some type of gradient descent [7] and lastly a backpropagation step where the model updates the weights [10].

2.2 Convolutional neural network

Convolutional neural networks were introduced to solve image recognition problems [6]. A convolutional neural network has a filter (or kernel) which is sliding around the input (image) and is multiplying the values in the filter (the weights of the filter) with the original input values (pixels). Summing up these values, the network gets 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 the 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 2.2. illustrates a convolutional network with a $(3,3)$ kernel.

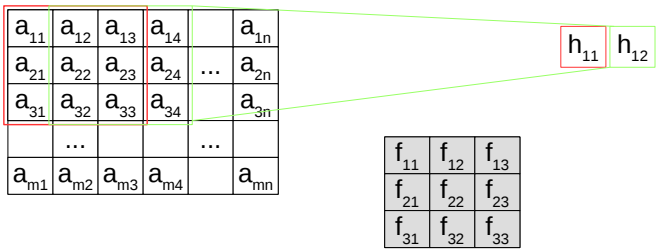


Figure 2.2. *Basics of Convolutional Neural Network (CNN)*

Kim, Jernite, Sontag and Rush showed a way of using one-dimensional convolutional neural networks and LSTM networks in character sequences [12], where the filter size says how many characters should be included into the convolution.

2.3 Recurrent neural network

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 or exploding gradients [2].

2.3.1 Long short-term memory

Long short-term memory (LSTM) [11] 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. More precisely, 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:

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b_i)$$

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b_f)$$

$$o_t = \sigma(W^o x_t + U^o h_{t-1} + b_o)$$

$$g_t = \tanh(W^g x_t + U^g h_{t-1} + b_g)$$

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

$$h_t = o_t \odot \tanh(c_t)$$

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 2.3.[1]

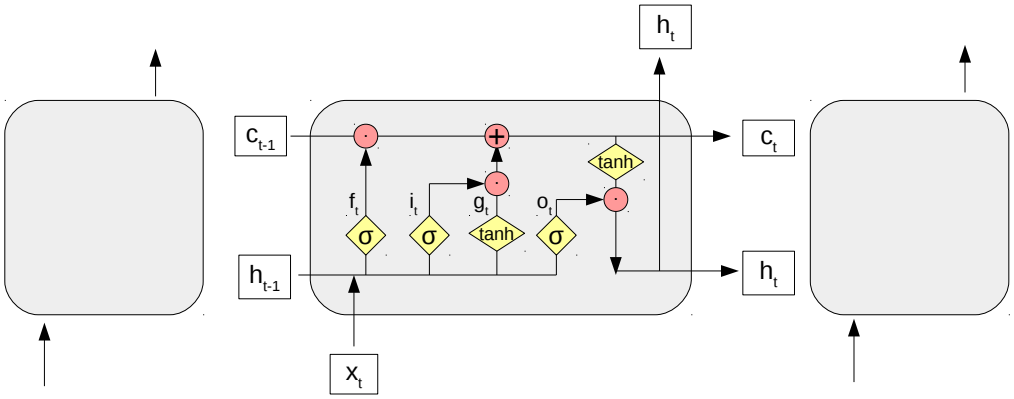


Figure 2.3. *Basics of Long short-term memory (LSTM)*

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) [21]. In terms of characters in a word it means that the letters can affect their surroundings on both sides.

2.4 Neural network APIs

When it comes to developing deep learning systems, recently the use of neural network APIs are rising both in industrial and scientific field. The two main reasons for this are the quickness of designing and the quickness of adaptation. The active community behind these APIs makes sure that the systems are up-to-date with the recent scientific results.

The most important APIs: TensorFlow[2], Torch7[3], Keras[4]. The programmings of this thesis are based on Keras[4].

[1] Visualisation and more : http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[2] TensorFlow: https://www.tensorflow.org/
[3] Torch7: http://torch.ch/
[4] Keras: https://keras.io/

2.4.1 Neural networks for hyphenation

The computer based hyphenation is considered as a solved problem with the high accuracy of Hunspell's hyphenation. Thus the modern machine learning based approach is not really tested on it. [16] is the only paper available specifically on automatic hyphenation using neural networks. It is an old paper about Norwegian hyphenation. They used a neural network very similar to the first approach of this thesis (feedforward neural net). However, they optimized only the character window and used only one and two hidden layers. Due to technical constraints they only ran tests on a very small training dataset.

Chapter 3

Data preprocessing

The following chapter summarizes the preprocessing methods used before the training. Data sizes mentioned here are generated by using the first 100 000 rows of the Hungarian Webcorpus's frequency list file.

Table 3.1. 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 prev.	-	16.32	1.33	0.78	0.01	
Words after	-	83678	82563	81917	81910	

Table 3.1. *Data preprocessing*

3.1 Hungarian Webcorpus

Hungarian Webcorpus is the largest Hungarian language corpus with over 1.48 billion words and it is available in its entirety under a permissive Open Content license [8, 15]. Due to having been collected from websites with a .hu domain, it contains not only Hungarian words but words from many different languages too. Although it has over a billion words, for the most of this study we will use only a smaller portion of Webcorpus' frequency shorted list (100 000 words).

3.2 Cleaning

In the first stage of the data preprocessing, three cleaning methods are being used to eliminate non-word elements, numbers or duplications:

All lowercase Setting the letters in lower case form. Words collected from the beginning of sentences are duplications of their lower-case form. The letter case is irrelevant in terms of hyphenation.

Filtering special characters Deleting the following punctuation characters from the texts (the *python* function `string.punctuation` defines them environment dependent so it can be different elsewhere):

! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~

Filtering numbers Deleting numbers from the texts. Grammatically correct Hungarian words do not contain numbers.

After the data had been cleaned, multiple occurrences were deleted (mostly caused by the upper case of sentence starting words), and the empty string data row (caused by deleting standing numbers). The remaining data size was 83678 words.

3.3 Filtering

There are some words that the models described here are not able to hyphenate. Thus, the words containing the following exceptions are deleted from the training data.

Non-standard hyphenation with character addition There are non-standard hyphenations where a character addition occurs (see more in Section 1.2.1). The following table shows these additions in the Hungarian language (Table 3.2.). Since the models use character tagging that may interfere, it was removed from the database. Another way of solving the problem would be to only remove these additions. Note that in the final model if we try to predict a word with this type of non-standard hyphenation it may find the place of the hyphen but cannot solve the addition.

There were 1115 words with this type of hyphenation in the dataset. There are words like *asszonnyal* which contains multiple sets of non-standard hyphenation so the total amount was 1117 and this number was used when the frequencies were calculated.

Special characters My models use a One-hot encoding for the Hungarian character set, which is:

a á b c d e é f g h i í j k l m n o ó ö ő p q r s t u ú ü ű v w x y z

In the previous step, we cleaned the punctuation characters from the data, but there are remaining letters like ô. With filtering these words we lost another 646 data row. Table 3.3.

Before	After	Number	Frequency(%)
ccs	cs-cs	21	1.88
ggy	gy-gy	21	1.88
lly	ly-ly	37	3.31
nny	ny-ny	210	18.80
ssz	sz-sz	813	72.78
tty	ty-ty	13	1.16
zzs	zs-zs	2	0.02

Table 3.2. *Hungarian non-standard hyphenation*

illustrates frequency of these special characters among the 646 words. Note that one word could contain multiple non-Hungarian characters.

Char	No.	Frequency(%)	Char	No.	Frequency(%)
ô	319	49.38	ń	13	2.01
ä	61	9.44	§	13	2.01
ă	37	5.73	â	9	1.39
Í	29	4.49	l’	9	1.39
d’	28	4.33	ż	9	1.39
î	26	4.02	ą	8	1.23
ß	25	3.87	ą	8	1.23
ë	18	2.79	ř	8	1.23
ę	18	2.79	ụ	8	1.23
ň	17	2.63	ž	8	1.23
ě	15	2.32	š	6	0.93
č	15	2.32	ć	6	0.93
ç	14	2.17	ý	5	0.77
ŕ	14	2.17	Other	69	10.68

Table 3.3. *Non-Hungarian characters in the data set*

3.4 Long word cut and padding

The CNN and LSTM models use fixed size words as input thus the shorter words have to be padded and the longer ones skipped. The FFNN model needs pre- and post-word padding too so they were defined as ^ and $ respectively. $ was chosen as the length standardizing filling character and the words’ end were filled with it.

The fixed size of the words was 30. There were only 7 words out of the range and all of them were gibberish (Table 3.4.).

Word	Possible origin
mailtomaiserszechenyinkzsasulinethu	mailto hyperlink
httpdelphiszechenyinkzsasulinethu	HTTP hyperlink
ftpftpszechenyinkzsasulinethudelphi	FTP hyperlink
httpwwwegyismertszerverhuképgif	HTTP hyperlink
orgapachecatalinacorestandardpipeline	?
standardpipelinevalvecontextinvokenext	?
httpwwwsomeunknownplacenetmypicturegif	HTTP hyperlink

Table 3.4. *Words out of the fixed length*

Chapter 4

Experimental setup

4.1 Character classification

The models introduced here are based on character classification. The letters of the words are inserted into groups according to their position between the hyphens. Two classification patterns are defined in this chaper, however, only the simpler one were actually used.

BM The BM classification uses two classes:

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

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

BMES

- B: In the beginning of the syllables.
- M: The middle of the syllables. Hyphens are neither before nor after this character.
- E: Ending character, a hyphen is placed after this.
- S: Single character: it is between hyphens.

The `le-o-párd` tagged as: `BESBMME`

4.2 Feedforward neural network

In the feedforward neural network we want to decide to which class a specific character belongs. To do so, 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 4.1. The blue boxes are the letters used to define the tag of *L*. In a five window length method it represents the five letter *^^LEO* in one-hot encoded way, then they are flattened to make input for the FFNN network. The outputs of the network are two numbers: the probability of *B* and *M*. The process chooses the larger one, *B*. The orange boxes are for the letter *E* and the green ones are for the *D*.

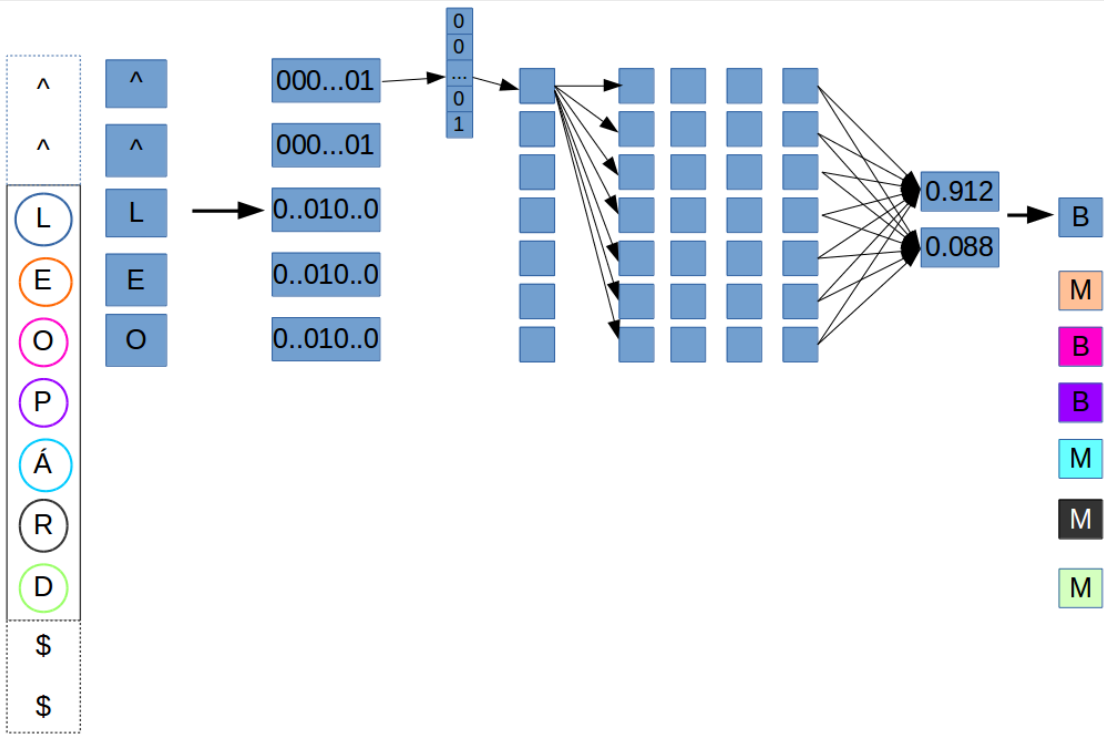


Figure 4.1. *Feedforward neural network*

4.2.1 Data preparation

The following section deals with the methods of creating the training data by showing an example. This example is the 5-length windowed BM classified preparation of the *o* character of the *leopárd* word. Figure 4.2. summarizes the data preparation.

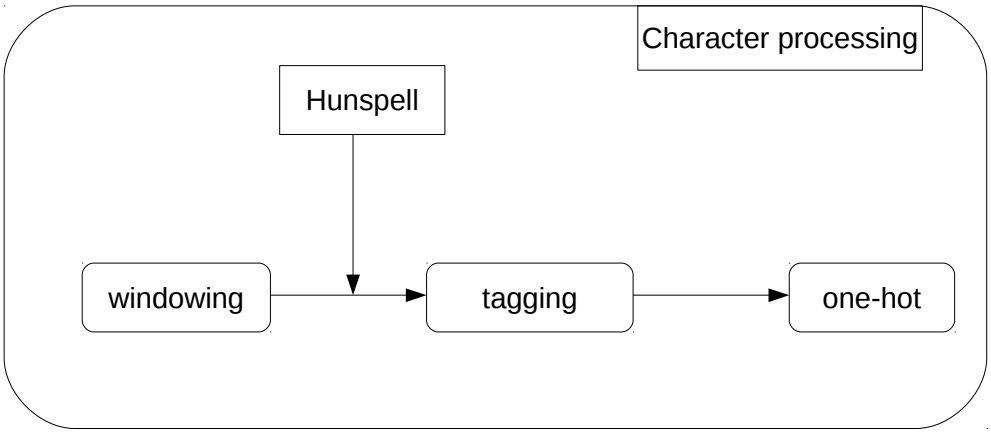


Figure 4.2. *Character preprocessing*

Learning from the surroundings (Windowing)

When we want to decide whether a hyphen is needed or not, we check the letters around its possible place. The 5-length window means that we use 5 characters around the *o* letter to classify it. The value 5 means that we will use two letters before the actual character, itself and two letters after it. Implementing it on *leopárd* the windows are from *^^l eo* to *ár d $$* where the window of letter *o* is: *le o pá*.

Note that if the windows are out of the word's range we add ^ character at the beginning of the word and $ character at the end of it. We can see the paddings in Figure 4.1.

Labelling

Hunspell's hyphenation algorithm were used to determine the labels of the training data. In this example it is the **BM** labelling, so the class of the letter *o* is B. At the beginning of every word, there is a **B** and there are as many more **B** as hyphens in the inserted form of hyphenation.

One-hot encoding

So far we have worked with characters, however, Keras trains on numbers so we have to represent the words as numbers. An effective way of doing so is the one-hot encoding, meaning that 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 we have 5 characters in a window, we have $(5, 37)$ shaped two-dimensional array.

The same encoding is used on the labels, B is the 0 and M is the 1.

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.

4.2.2 The model

The neural network is a fully connected feedforward neural network. In the hyperparameter optimization along the window length are the number of hidden layers and the units in each hidden layer. The last layer has a *softmax* activation to approximate the values as probabilities. Figure 4.3. is the Keras visualization of the layers.

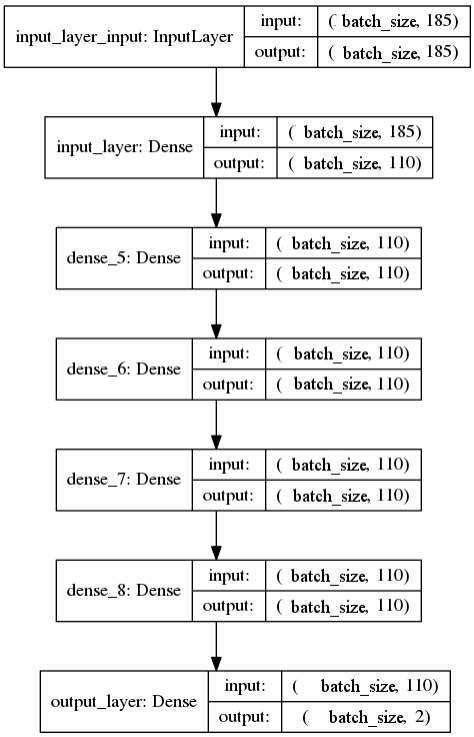


Figure 4.3. *The FFNN model*

The training process uses *sigmoid* activation function and *adam* optimizer [13]. Adam realizes the benefits of both AdaGrad and RMSProp. Instead of constant learning rate it uses adaptive value using the first and the second moments of the gradients. The training methods used the default parameters of the Keras Adam optimizer which are the same as described in the paper (learning rate $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-08}$, $decay = 0.0$).

The model was trained with a max. epoch number 1000 but it used Early Stopping method with patience 20.

It was trained in a NVDIA GeForce GTX 1070 and the training of different models described here and later lasted between 5 minutes and 1.5 hours. All codes are available at https://github.com/negedng/NLP

4.3 Train-, validation-, test data

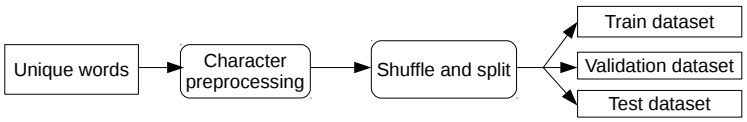
In machine learning, it is a well-known paradigm to separate the training, validation and test data. While the neural network fits on a training data, its performance during the training is measured on a different, validation data. If we use early stopping, validation data is used to halt the training when it stopped improving on the validation data. The third part is the test data. It has not been used in the training method thus it provides an independent evaluation data. The trainings use the common 70-20-10 separation rate.

Figure 4.4. shows the following 3 methods of splitting the data words into 3 separate sets.

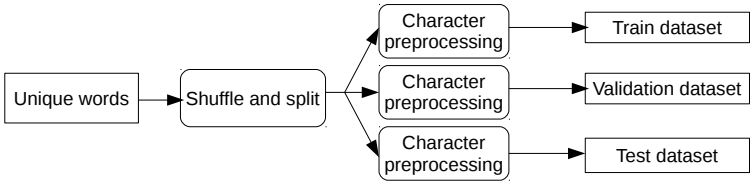
Original method The first method follows these steps: from the unique words of the dataset with the previously described way the train-ready data is created. Then shuffling it and splitting it into 3 parts. However it has two major problems.

Unique words Firstly, when we shuffle the windows *leopá* and *eopár*, they can be in different sets. The second method prevents this. In this method, the words are separated first, so every word is in one of the 3 datasets. So there are unique and separated words in the sets. But later we would create windows thus there are windows appearing multiple times, and in different datasets.

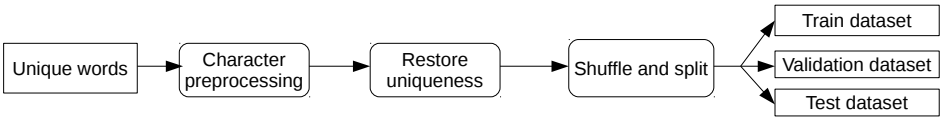
Unique characters The third method focuses on the uniqueness of the {training window, label} pairs. However, if a training window has an example when it is labelled with



1. Original



2. Unique words



3. Unique chars

Figure 4.4. *The 3 methods of splitting train-validation-test data.*

B and one with M and one of it went to the training set while the other was in the test set, it always makes an error at the evaluation.

Conclusion While the third method is the closest to an ideal setting, in a real situation it is quite unlikely to get an entirely new {training window, label} pair. However it is common to get a new word.

When we compare the FFNN network with CNN and LSTM ones, we separate the words first because the other two model use them.

4.4 Convolutional neural network

4.4.1 Data preparation

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 4.5.).

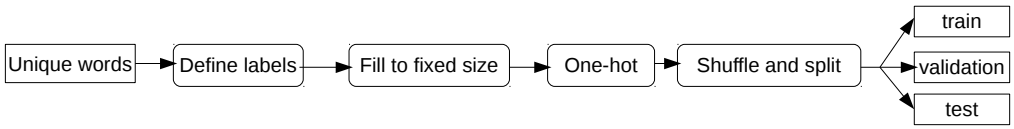


Figure 4.5. *CNN and LSTM data preparation*

4.4.2 The model

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. Optimization parameters are the kernel size (it is the same dimension as the window-length in the FFNN model), filter and hidden layer numbers. Keras' built-in padding is used to prevent problems at the endings of the word. The model uses ReLU activation functions and an early stopping patience 5.

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

Figure 4.6. summarizes the CNN model.

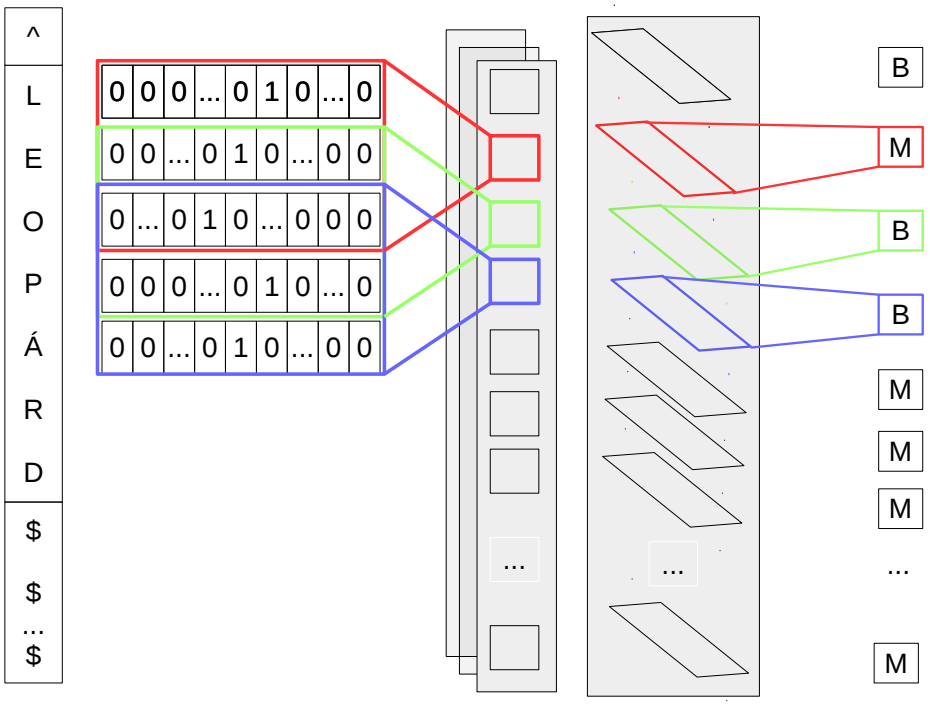


Figure 4.6. *CNN model summary*

4.5 Long short-term memory

The LSTM network uses the same inputs as the CNN. It is biLSTM (the one-way LSTM network was not efficient) and it is optimized by the unit and hidden layer numbers. At the output of the LSTM feedforward network were inserted just as in the CNN one. The model summary of the LSTM network is in Figure 4.7.

The LSTM layers use ReLU activation function and the Keras default *hard sigmoid* recurrent activation. It uses an early stopping patience 7.

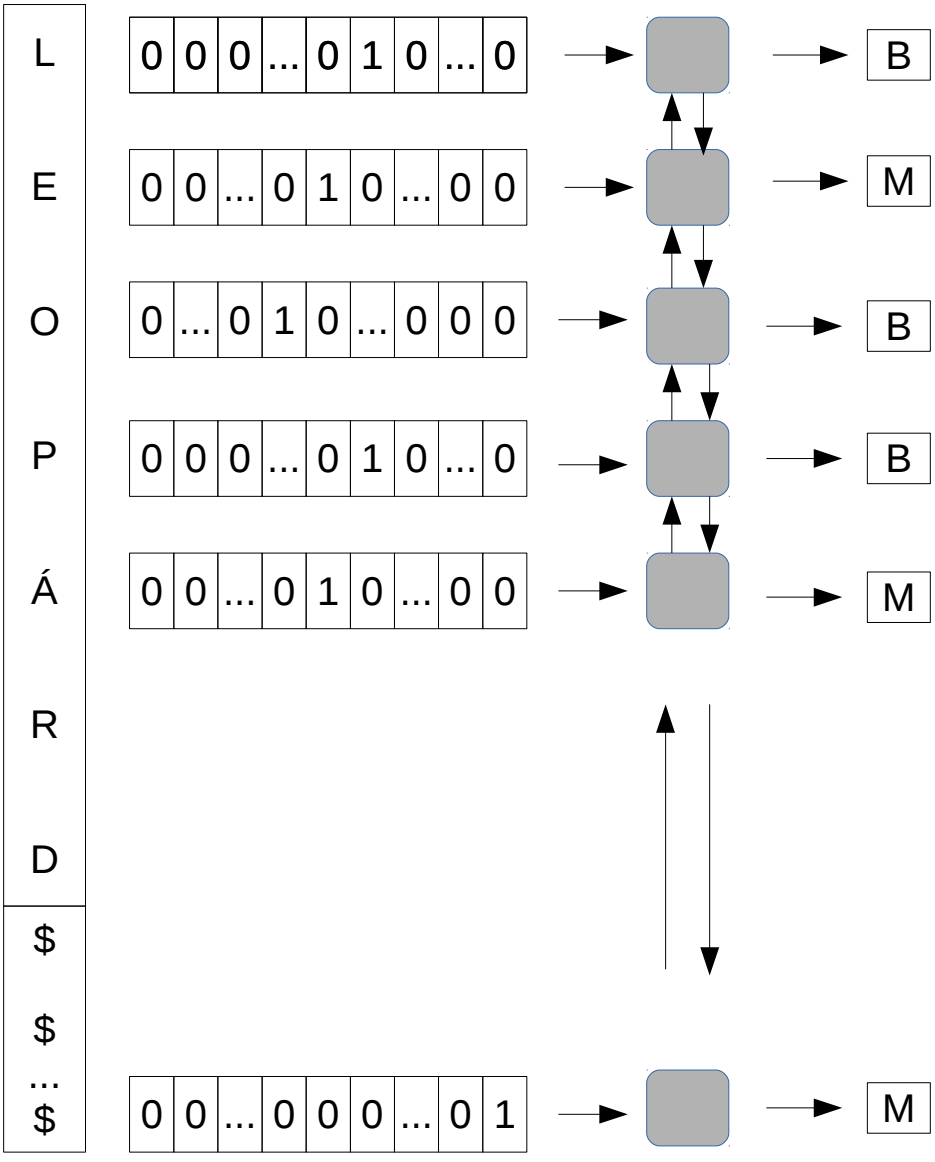


Figure 4.7. *Long Short-Term Memory network*

Chapter 5

Results

All three model hyphenated successful around 95% of the words. This chapter describes the evaluation values, the optimization methods and the best performed models. Table 5.1. shows examples of word hyphenation.

FFNN	CNN	LSTM	Target
szent-mi-sét	szent-mi-sét	szent-mi-sét	szent-mi-sét
egyé-ni-sé-gé-nek	egyé-ni-sé-gé-nek	egyé-ni-sé-gé-nek	egyé-ni-sé-gé-nek
vá-lo-gat-ták	vá-lo-gat-ták	vá-lo-gat-ták	vá-lo-gat-ták
ku-ta-tás-hoz	ku-ta-tás-hoz	ku-ta-tás-hoz	ku-ta-tás-hoz
meg-je-le-nít-he-ti	meg-je-le-nít-he-ti	meg-je-le-nít-he-ti	meg-je-le-nít-he-ti
idény	idény	idény	idény
hír-for-rás	hír-for-rás	hír-for-rás	hír-for-rás
vic-tor	vic-tor	vic-tor	vic-tor
tü-rel-mem	tü-rel-mem	tü-rel-mem	tü-rel-mem
mű-kö-dés-ben	mű-kö-dés-ben	mű-kö-dés-ben	mű-kö-dés-ben
köz-út	köz-út	kö-zút	köz-út

Table 5.1. *Hyphenation examples*

5.1 Evaluation values

Val_loss is the validation loss of the Keras model.

True positive sample is what has a **B** label in both the prediction of the model and the target (the Hunspell's hyphenation).

True negative sample has a **M** label as prediction and as target too.

False positive sample is predicted as **B** by the model but expected **M** by the dataset.

False negative is the opposite of false positive: the prediction is **M** and the target is **B**.

Precision is the rate of correct values among all positive results $P = t_p/(t_p + f_p)$.

Recall is the rate of predicted **B** values compared to all that should have been **B**, $R = t_p/(t_p + f_n)$.

F-Score is the harmonic mean of precision and recall: $F = 2\frac{P \cdot R}{P+R}$

Word accuracy is the ratio of correctly labelled words among the test data.

5.2 Hyperparameter optimization

There hyperparameter optimization was near 100 trainings. The ranges of the optimized values are showed in Table 5.2.

	Window/Kernel	Hidden layers	Hidden units
FFNN	3-11	3-7	60-180
CNN	5-16	1-3	64-2048
LSTM	-	1-3	8-256

Table 5.2. *Optimization parameters*

Feedforward neural network Optimization were around the window length, the layer number and the hidden unit numbers (Table A.1.). The window-length was tested between three and eleven character. The layer number and hidden unit numbers were only optimized for five and seven window-length and between 3 and 7 for the layer number and 80 to 130 for hidden units.

Convolutional neural network First optimization were the numbers of hidden units and the kernel size. After a few run, the optimum of the kernel size showed to be around 10 character so next the layer number and the numbers of hidden units were optimized only for 6-10 kernel size it (Table A.2.).

Long short-term memory The LSTM network was the most sensitive to the change of the LSTM layer number (Table A.3.).

5.3 The 3 model

Table 5.3. illustrates a rerun of the models with highest F-score. The hyper-parameters: **FFNN**: 3 layers of 130 units and a window length of 7. **CNN**: 2 CNN layers with 1024 hidden units and a kernel size of 8. **LSTM**: 2 layer of 128 units.

Model	Kernel	Layers	Hidden	Epochs	Precision	Recall	F-score	Word acc.
FFNN	7	3	150	103	98.17%	99.11%	98.64%	93.57%
CNN	8	2	1024	12	98.37%	99.27%	98.81%	94.45%
LSTM	-	2	128	66	97.68%	99.16%	98.42%	93.13%

Table 5.3. *The three model that performed the best*

Chapter 6

Evaluation

In Section 5.3. 3 models were trained in the same data. The following error analysis is based on those models. Figure 6.1. represents the models' overall performance on the 8191 test words. As we can see, if we uses all three models with the majority rule, 94.3% of the words can be hyphenated correctly (however the CNN network itself achieves a better performance with 94.45%).

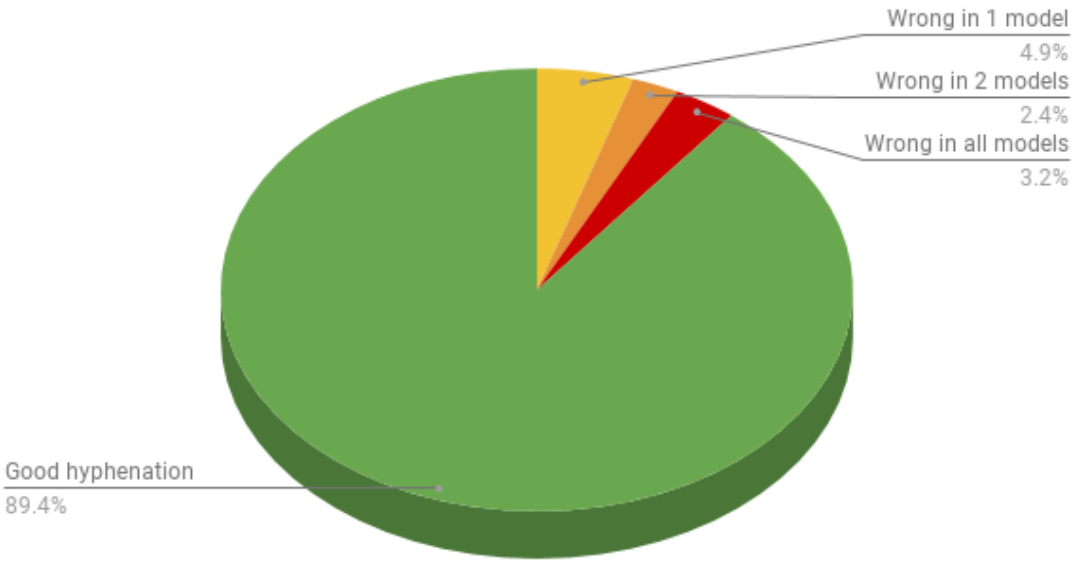


Figure 6.1. *Model performance*

6.1 Word categorization

After the rerun, the errors were manually tested and grouped into categories. These categories are:

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 word are mostly mistyped Hungarian words like *elol*.

Other words not filling the above categories.

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

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 a word	4	el-er-ni
Others	59	

Table 6.1. *Word categories among 100 randomly chosen words*

6.2 Model performance

All the errors of the three models were collected and manually inserted 200 words into the error categories. Figure 6.2. shows the category distribution of the errors for each model and among the errors.

The next figure (6.3.) visualizes the three models' performance in each categories. So in this figure we can see the percentage of missed words by the models, where the 100% is the amount of wrong word in the category. The last column shows the 3 models' distribution of hyphenation misses. As we can see, the Hunspell's typesetting error (*Non-hyphenated part*) had the worst effect on the LSTM network.

Note that there was only one word in the *wrong target* category. The word *topikok* was hyphenated wrongly not only with using Hunspell but with using the online hyphenation portal too. However te CNN and LSTM networks are hyphenated it correctly (Table 6.2.).

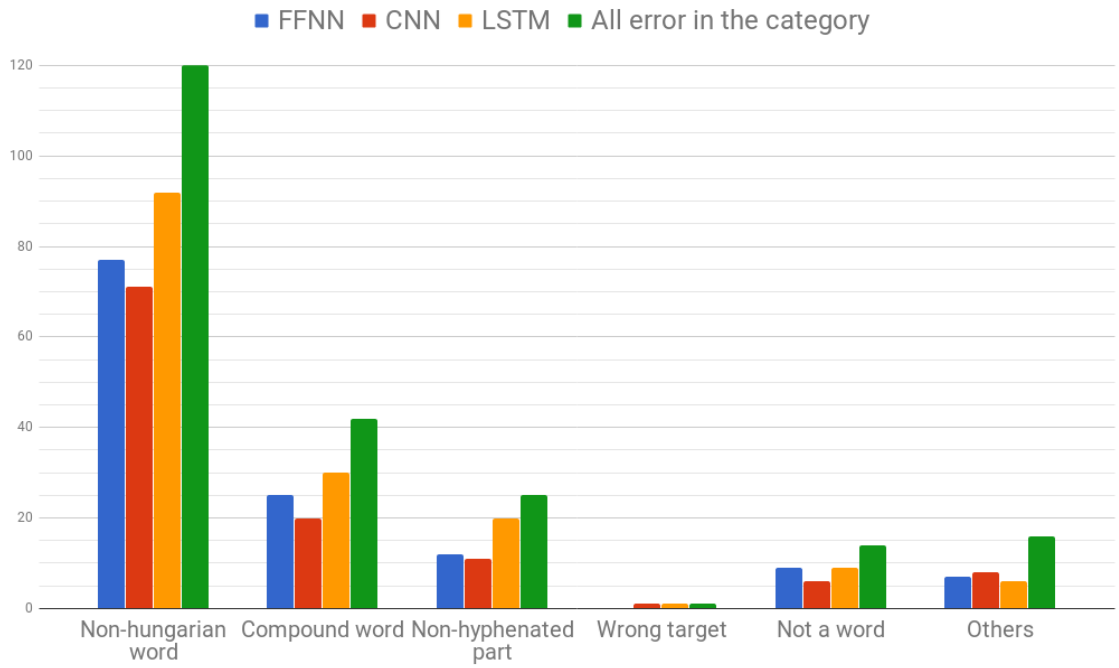


Figure 6.2. *Missed hyphenations by error categories (200 words)*

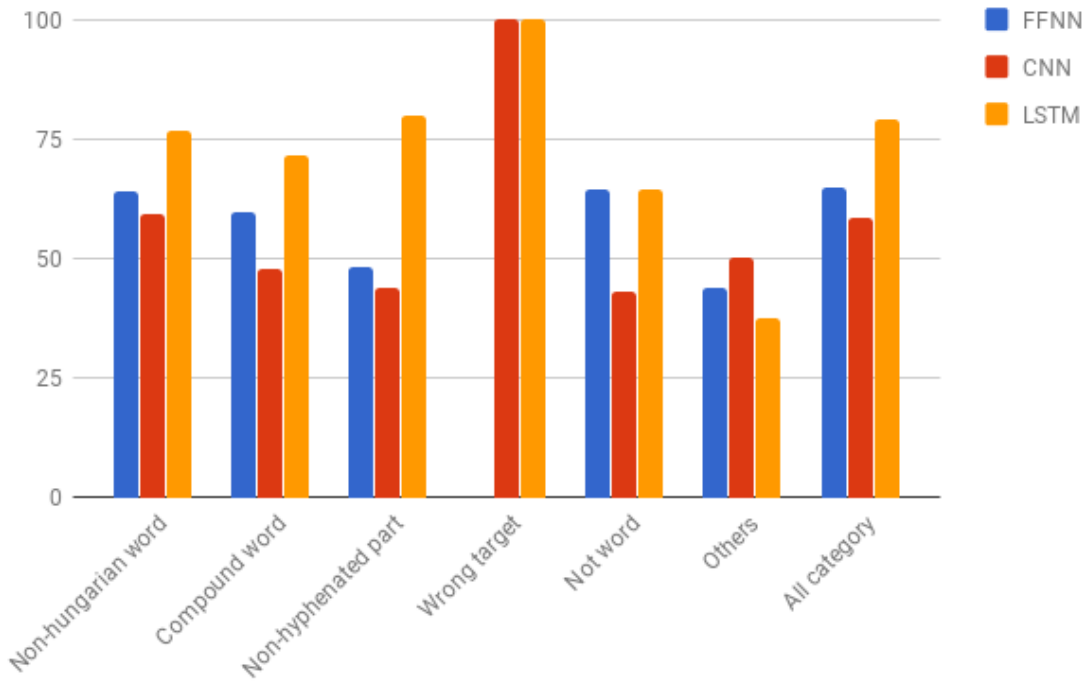


Figure 6.3. *Model performance by categories*

FFNN	CNN	LSTM	Target	Online	Correct
to-p-i-kok	to-pi-kok	to-pi-kok	top-ikok	to-p-ikok	to-pi-kok

Table 6.2. *Hyphenation of* topikok

6.2.1 Non-Hungarian words

As we can see in Figure 6.2., over half of the errors are caused by using a Hungarian hyphenation algorithm on a non-Hungarian word. While Table 6.1. says that only 11% of the words are non-Hungarian, half of the errors can be tracked back to them. If we compare it with the data of Table 5.3. where we saw that only 6% of the words are hyphenated badly we can conclude that half of this 6% is caused by non-Hungarian words so 3% of every word is miss-hyphenated non-Hungarian word. Cleaning the non-Hungarian words can achieve a 96-97% word accuracy model.

This experience leads us to the need of a multilingual hyphenation algorithm. In the world of globalization more and more multilingual text appears, the terminology of modern science is translated less and people use more foreign words in their everyday conversations. Chapter 7 describes the multilingual usage of the models.

6.2.2 Non-hyphenated part

As it was described in the Section 1.2.2, there is a typesetting difference between the Hunspell's hyphenation and the correct Hungarian hyphenation. However, the models learn to Hunspell, sometimes the error itself resulted because the model predicted the correct hyphenation. For example *testépítés* in Table 6.3.

FFNN	CNN	LSTM	Target	Correct
test-é-pí-tés	test-é-pí-tés	test-é-pí-tés	test-épí-tés	test-é-pí-tés
egyéb-i-ránt	egyé-bi-ránt	egyé-b-i-ránt	egyéb-iránt	e-gyéb-i-ránt
aka-ra-terő	aka-ra-t-erő	aka-ra-te-rő	aka-rat-erő	a-ka-rat-e-rő
szer-te-ága-zó	szer-te-á-ga-zó	szer-te-ága-zó	szer-te-ága-zó	szer-te-á-ga-zó

Table 6.3. *Words with non-hyphenated part*

Chapter 7

Multilingual hyphenation

The basic idea of multilingual hyphenation is to teach the models with multiple language corpora. The rules of hyphenation differ in every language and the Hunspell's patterns are chosen individually, however the machine learning details are the same. This leads us to the idea of multilingual hyphenation.

7.1 English hyphenation

This section summarizes the results of the models used as English hyphenation algorithms. Since the purpose of this experiment was to prepare a multilingual network, the algorithms were the same as for the Hungarian data. The optimization ranges were also the same due to the aforementioned reason.

7.1.1 UMBC WebBase corpus

The UMBC WebBase corpus is a dataset containing a collection of English paragraphs with over three billion words processed from the February 2007 crawl from the Stanford WebBase project. Compressed, it is about 13GB in size [9].

The corpus was chosen because it is similar to the Hungarian Webcorpus as it has been collected from English web pages.

7.1.2 English results

Using the same optimizing process as introduced earlier Table 7.1. shows the results of the English hyphenation. The optimization results show that the English hyphenation requires more layers or neurons than the Hungarian and it can be improved 1-2% if the

optimization range is expanded. However the following results are from the same range as it was for the Hungarian hyphenation.

The results of English hyphenation are lower than the Hungarian. The reason behind this is probably the difference between hyphenation rules. While the Hungarian hyphenation is based on the word's written form and morphological properties, the English hyphenation follows pronunciation based rules.

Model	Kernel	Layers	Hidden	Precision	Recall	F-score	Word accuracy
FFNN	7	6	150	94.71%	94.32%	94.51%	79.31%
CNN	7	4	1024	96.28%	95.42%	95.85%	84.25%
LSTM	-	2	256	97.41%	93.83%	95.58%	81.73%

Table 7.1. *Best models for English hyphenation*

7.2 Learning on bilingual database

In this section only the CNN model was trained as it had the best performance. The hyper-parameters are CNN layer number: 2, CNN layer dimension: 1024, kernel size: 8.

For the bilingual database, a slightly bigger portion of both the Hungarian Webcorpus and the UMBC WebBase corpus was used. There were 100000 actual trainings, 20000 validations and 10000 test words for both languages.

To create these words, 180000 words from the frequency lists of the corpora was inserted in the data preprocessing methods (see Chapter 3). Because of the noise in the Hungarian database, there are words appearing in both corpus and hyphenated differently (however there are some words with meaning in both language like *ember*). These words were deleted from the databases. With 144228 clean words in the Hungarian database and 162744 in the English one, the number of deleted words is 10553.

Comparing this number with the non-Hungarian word frequency from the Table 6.1. (11%) we could expect more. The reason behind this is that some words are hyphenated the same in both languages like *ember (em-ber)*. After this deletion the 100000, 20000, 10000 words datasets are created for both languages.

Table 7.2. shows the results of training the CNN model with different English-Hungarian word rate. The test words are the same for each model: 10000 - 10000 for both language. The train and validation words are chosen with the given ratio (HU**X**EN**Y** means $X\%$ Hungarian words and $Y\%$ English words). There were always 100000 train words and 20000 validation words.

Model	F-Score		Word accuracy (%)	
	Hungarian	English	Hungarian	English
HU0EN100	0.6643	0.9556	18.69	82.75
HU10EN90	0.9586	0.9477	80.90	79.88
HU20EN80	0.9675	0.9289	84.42	73.24
HU30EN70	0.9769	0.9374	89.56	76.18
HU40EN60	0.9803	0.9323	90.86	74.67
HU50EN50	0.9828	0.9205	91.05	71.04
HU60EN40	0.9853	0.9162	93.22	69.47
HU70EN30	0.9853	0.9033	93.25	65.74
HU80EN20	0.9875	0.8928	94.27	63.18
HU90EN10	0.9895	0.8648	95.2	56.96
HU100EN0	0.9930	0.7440	96.63	36.37
LangSelect	0.9946	0.9544	97.37	81.05

Table 7.2. *Word rate of different Hungarian-English dataset ratios*

HU90EN10 is the same ratio as the original Webcorpus and we got nearly the same result for the Hungarian word hyphenation success rate.

Figure 7.1. illustrates the word accuracies of the different models.

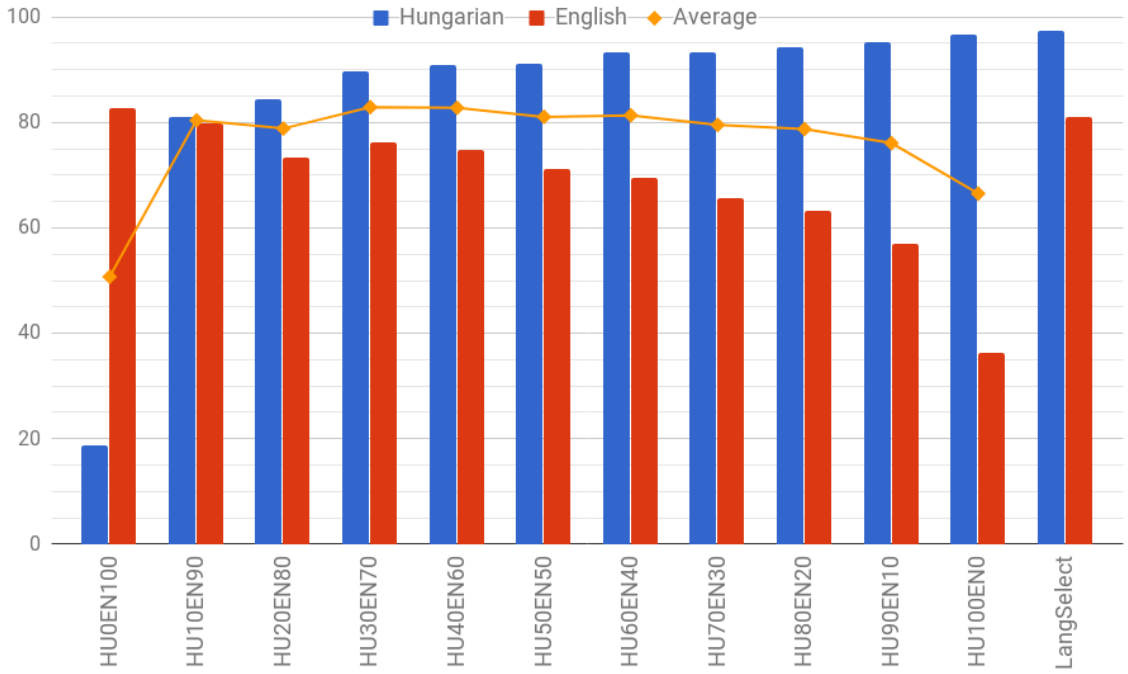


Figure 7.1. *Word accuracy(%) of bilangual models*

The LangSelect row represents a model where a language selector chooses a language for the words and the HU100EN0 or the HU0EN100 model hyphenates the word according to it.

The language selector uses the `langdetect` Python library which is based on the Lan-

guage Detection Library for Java [22]. It generates profiles from Wikipedia abstract xml and detects language of a text using naive Bayesian filter.

From the 20000 test words only 6883 was detected as Hungarian words and 1895 as English ones while 11222 words were detected as different from the two. The word accuracy in Table 7.2. is achieved only on this smaller test set.

In conclusion, the model works well as a bilingual hyphenation algorithm, way better than using a word-level language selector to choose the language of the hyphenation.

Chapter 8

Non-standard hyphenation

The character classification algorithms described in Chapter 4. are not usable when the task requires a non-standard hyphenation (1.2.1). Character additions, changes are excluded from the classification algorithms.

8.1 Sequence-to-sequence model

Sequence-to-sequence learning is about converting sequences from one domain to another [24, 3]. This summary and the Keras implementation of the sequence-to-sequence model are based on the following Tutorial: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

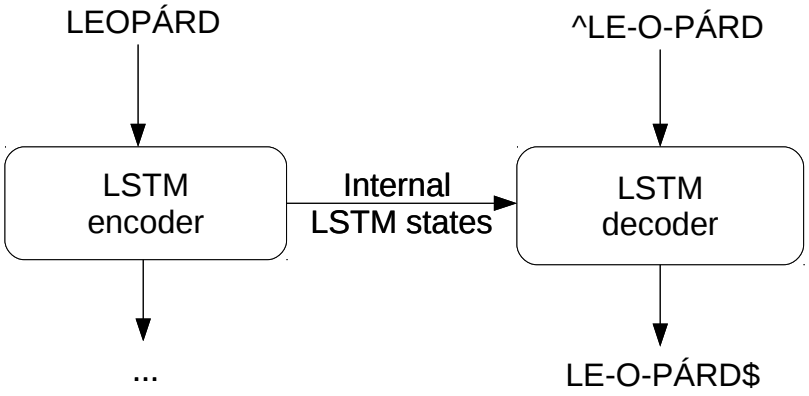


Figure 8.1. *Sequence-to-sequence training*

In general case, input sequences and output sequences have different lengths and the whole input is required to start predict output. The model has two parts: an **encoder** that

processes the input sequence and returns with its own states. These states will be the *context* of the second, **decoder** part. The decoder's input is the target sequence with a starting (^) character in the beginning, and a the output of the decoder is the target sequence with an ending($) character at the end of it.

The decoder learns as an LSTM, from the starting character one-by-one learns the next character of the word. The *context* of the encoder makes a difference between words. When it comes to prediction, only the encoder knows what the input sequence is, the decoder starts with a single starting character and predicts next characters until it comes to an ending character.

Figure 8.1. illustrates the training method while Figure 8.2. shows the prediction.

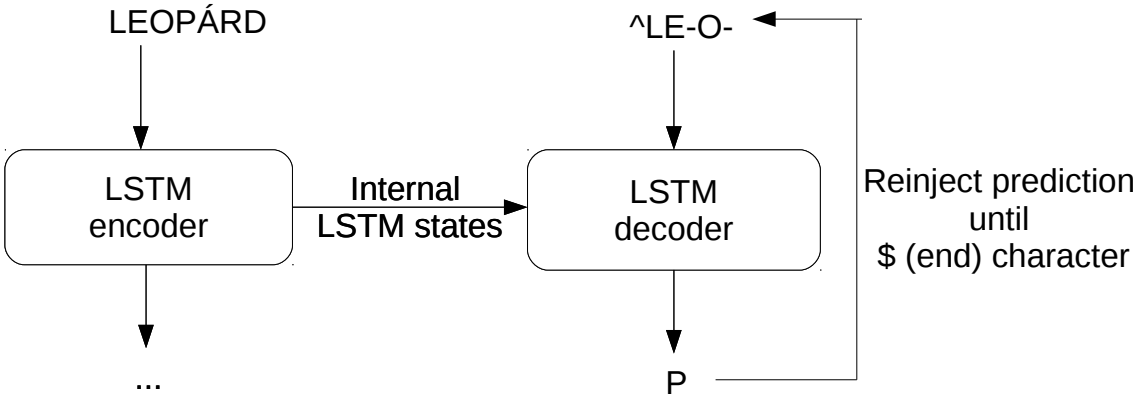


Figure 8.2. *Sequence-to-sequence prediction*

While this summary uses characters as objects of the input and target domain, more generally it can be anything from a pre-defined dictionary. For example machine translation uses words of the input and output language.

In the task of Hungarian hyphenation, the input domain is the Hungarian character set (35 characters) while the target has the - hyphen, ^ starting and $ ending characters added to it (38 characters).

8.2 Model comparison

The seq2seq model was optimized around the LSTM networks' dimension. In the evaluation of this model, the character level performance cannot be applied because of the difference of the models. Most of the mistaken words are not only hyphenated differently but the prediction resulted in an entirely new word. Table 8.1. shows examples of the prediction.

Word	Target	Prediction
kőre	kő-re	kő-re
munkaképes	mun-ka-ké-pes	mun-ka-ké-pes
csodálhatjuk	cso-dál-hat-juk	cso-dál-tat-juk
kommandó	kom-man-dó	kom-man-dó
érdemesnek	ér-de-mes-nek	ér-de-mes-nek
lakásban	la-kás-ban	la-kás-ban
traktor	trak-tor	trak-tor
vállalkozókkal	vál-lal-ko-zók-kal	vál-lal-ko-zó-ka-kat

Table 8.1. *Sequence-to-sequence model examples*

Comparing this model with the 3 previous ones, it has a way harder job: instead of predict a binary classification, it has to predict from 38 classes and the sequence length is not fixed.

The training parameters of the Seq2seq network was as it had described in the Keras Tutorial, only the latent dimension was optimized between 128 and 1024

Model	Kernel	Layers	Hidden	Epochs	Word accuracy
FFNN	7	3	150	103	93.57%
CNN	8	2	1024	12	94.45%
LSTM	-	2	128	66	93.13%
Seq2seq	-	-	1024	48	81.12%

Table 8.2. *FFNN, CNN, LSTM and Seq2seq results*

8.3 Training with non-standard hyphenation

In this section, the same sequence-to-sequence model was trained but the non-standard hyphenation had not removed from the training dataset. As Table 3.2. shows, there are sometimes only a few examples of a certain hyphenation. But filling up the dataset with manually collected data is not a possible way of improving the training: only 10 words was found with $zzs \rightarrow zs - zs$ hyphenation in it.

The model performed a 86.13% word accuracy and Table 8.3. non-standard hyphenation examples from the test words.

However at this rate, when only a few examples are in the training data, the noise of the Hungarian Webcorpus database influence a lot. There are a comparable amount of English words in the dataset with *lly* like *manually, gradually*. Furthermore, when only 4 *ggy* examples are in the test set, the miss-typed word *naggyon* changes a lot.

Word	Target	Prediction
CCS		
No example in the test set		
GGY		
naggyon	nag-gyon	nagy-gon
meggyek	megy-gyek	meg-gyek
LLY		
nyúllyuk	nyúl-lyuk	nyúl-log
végveszéllyel	vég-ve-szély-lyel	vég-ve-szély-lyel
süllyedt	süly-lyedt	sül-lety
SSZ		
összegű	ösz-sze-gű	ösz-sze-gű
visszafogott	visz-sza-fo-gott	visz-sza-fo-gott
asszonnyal	asz-szony-nyal	asz-szony-nyal
késszár	kés-szár	kész-szár
NNY		
ingatlannyilvántartási	in-gat-lan-nyil-ván-tar-tá-si	in-gat-lan-nyil-ván-tar-tá-si
ennyiből	eny-nyi-ből	eny-nyi-ből
TTY		
pottyant	poty-tyant	poty-tatn
szivattyú	szi-vaty-tyú	szi-vaty-tyú
ZZS		
rúzzsal	rúzs-zsal	rúz-szal
mézzsír	méz-zsír	méz-szír

Table 8.3. *Sequence-to-sequence non-standard hyphenation examples*

Conclusion

In this thesis a series of experiments on Hungarian hyphenation using deep neural networks were introduced. Four architectures were compared with varying hyperparameters: the Feedforward Neural Network (FFNN), the Convolutional Neural Network (CNN), the Long Short-Term Memory network (LSTM) and the Sequence-to-sequence model (Seq2seq).

It was trained and tested on the 100000 most frequent words from the Hungarian Webcorpus. The first three model achieve over 95% word accuracy, occasionally correcting the errors made by Hunspell, while the fourth model solves the problem of non-standard hyphenation. The error analysis suggests that foreign and compound words are the most challenging for our systems.

And a multilingual hyphenation algorithm was introduced to resolve the error of foreign words, tested on English and Hungarian words performed only 5% worse than the networks for the languages individually.

All codes are available at https://github.com/negedng/NLP

Acknowledgement

I would like to express my sincere gratitude to my consultant and advisor, Judit Ács for the continuous support for my thesis and her help in its appearance in the Scientific Students' Associations ("TDK") of the Budapest University of Technology and Economics and the Conference on Hungarian Computational Linguistics.

I would also like to thank Laura Seben for correcting most of my grammatical errors and motivating me during the process of writing.

List of Figures

List of Tables

Bibliography

[1] Magyar Tudományos Akadémia and Maď arsko Budapešť. *A magyar helyesírás szabályai*. Akadémiai kiadó, 1959.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[3] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[4] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[5] Scott E Fahlman. An empirical study of learning speed in back-propagation networks. 1988.

[6] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[7] Jacques Hadamard. *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées*, volume 33. Imprimerie nationale, 1908.

[8] Péter Halácsy, András Kornai, Nemeth Laszlo, Rung Andras, István Szakadát, and Tron Viktor. Creating open language resources for hungarian. 2004.

[9] Lushan Han, Abhay L Kashyap, Tim Finin, James Mayfield, and Jonathan Weese. Umbc_ebiquity-core: Semantic textual similarity systems. In ** SEM@ NAACL-HLT*, pages 44–52, 2013.

[10] Robert Hecht-Nielsen et al. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Donald Ervin Knuth. *TEX and METAFONT: New directions in typesetting*. American Mathematical Society, 1979.

[15] András Kornai, Péter Halácsy, Viktor Nagy, Csaba Oravecz, Viktor Trón, and Dániel Varga. Web-based frequency dictionaries for medium density languages. In *Proceedings of the 2nd International Workshop on Web as Corpus*, pages 1–8. Association for Computational Linguistics, 2006.

[16] Terje Kristensen. 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, pages 148–153. IEEE, 2000.

[17] Franklin Mark Liang. *Word hyphenation by computer*. Department of Computer Science, Stanford University, 1983.

[18] Márton Miháltz, Péter Hussami, Zsófia Ludányi, Iván Mittelholtz, Ágoston Nagy, Csaba Oravecz, Tibor Pintér, and Dávid Takács. Helyesírás. hu. 2013.

[19] László Németh. Automatic non-standard hyphenation in openoffice. org. *COMMUNICATIONS OF THE TEX USERS GROUP TUGBOAT EDITOR BARBARA BEETON PROCEEDINGS EDITOR KARL BERRY*, page 32, 2006.

[20] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[21] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[22] Nakatani Shuyo. Language detection library for java, 2010.

[23] Petr Sojka et al. Notes on compound word hyphenation in tex. *TUGboat*, 16(3):290–296, 1995.

[24] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Appendices

Window length	Num_layer	Num_hidden	Epochs	Val_loss
3	5	60	303	0.0916
3	5	70	303	0.0916
3	5	80	229	0.0916
3	5	90	252	0.0914
3	5	100	196	0.0928
3	5	110	249	0.0913
5	5	60	222	0.0406
5	5	70	202	0.0397
5	5	80	222	0.0421
5	5	90	214	0.0399
5	5	100	198	0.0395
5	5	110	190	0.0400
7	5	60	133	0.0385
7	5	70	131	0.0425
7	5	80	147	0.0443
7	5	90	149	0.0387
7	5	100	137	0.0381
7	5	110	136	0.0393
9	5	60	121	0.0390
9	5	70	119	0.0432
9	5	80	125	0.0413
9	5	90	118	0.0393
9	5	100	116	0.0383
9	5	110	126	0.0397
11	5	60	112	0.0454
11	5	70	111	0.0408
11	5	80	119	0.0457
11	5	90	99	0.0436
11	5	100	103	0.0401
11	5	110	98	0.0412

Table A.1. *FFNN window length optimization*

Num_layers	Num_hidden	Kernel size	F-score	Word accuracy
1	512	6	98.63%	93.30%
1	512	8	98.77%	94.07%
1	512	10	98.71%	93.70%
1	1024	6	98.78%	94.10%
1	1024	8	98.79%	94.08%
1	1024	10	98.86%	94.52%
1	2048	6	98.75%	93.85%
1	2048	8	98.85%	94.42%
1	2048	10	98.84%	94.40%
2	512	6	98.92%	94.68%
2	512	8	98.84%	94.41%
2	512	10	98.79%	94.23%
2	1024	6	98.96%	94.96%
2	1024	8	99.01%	95.28%
2	1024	10	98.90%	94.92%
2	2048	6	99.03%	95.25%
2	2048	8	98.99%	95.14%
2	2048	10	98.92%	94.91%
3	512	6	98.77%	94.10%
3	512	8	98.68%	93.71%
3	512	10	98.39%	92.14%
3	1024	6	98.88%	94.54%
3	1024	8	98.81%	94.24%
3	1024	10	98.64%	93.82%
3	2048	6	98.92%	94.64%
3	2048	8	98.76%	94.18%

Table A.2. *CNN hyper-parameter optimization*

Num_layers	Num_hidden	Precision	Recall	F-score	Word accuracy
1	8	50.33%	75.61%	60.43%	0.72%
1	16	50.32%	75.67%	60.45%	0.72%
1	32	50.33%	75.64%	60.44%	0.72%
1	64	50.33%	75.63%	60.44%	0.72%
1	128	50.34%	75.64%	60.45%	0.72%
1	256	50.33%	75.64%	60.44%	0.72%
2	8	42.55%	75.41%	54.41%	0.00%
2	16	97.09%	99.05%	98.06%	91.73%
2	32	95.95%	98.59%	97.25%	88.71%
2	64	97.52%	98.93%	98.22%	92.63%
2	128	97.99%	99.18%	98.58%	93.90%
2	256	97.77%	99.22%	98.49%	93.37%
3	8	54.23%	99.29%	70.15%	11.71%
3	16	64.65%	97.48%	77.74%	29.34%
3	32	99.94%	31.43%	47.82%	7.54%
3	64	54.95%	99.74%	70.86%	11.70%
3	128	60.53%	98.87%	75.09%	11.57%
3	256	55.52%	98.60%	71.04%	12.62%
3	512	43.75%	99.78%	60.83%	0.59%

Table A.3. *LSTM hyper-parameter optimization*