Hungarian Text Generator for Children's Novels

Szilárd Novoth, Páncsics Zsombor MSc
students
Faculty of Electrical Engineering and
Informatics
Budapest University of Technology and
Economics
Budapest, Hungary

***Abstract*— This work explores the field of Natural Language Generation (NLG) and develops a novel text generation algorithm for the Hungarian language using LSTM model. The work uses and prepares data collected from Magyar Elektronikus Könyvtár (MEK). Numerous language models and best practises will be introduced. As an introduction for Language modelling the classical, mathematical approaches will be presented. After that, the modern Deep Learning based approaches will be explained. In the end of the document the evaluation techniques will be shown, like perplexity and OOV.**

Keywords—Language Modelling, Natural Language Processing, Natural Language Generation, NLG, NLP, text generation, RNN, LSTM, Transformer

I. Introduction

Language modelling is a method which uses various techniques to determine the probability of a word/character sequence in a given sentence. For example, a sentence: "A dog will bark if you splash him" is more likely than "An iron will bark if you splash him"

Language Modelling is not used by itself but it is a crucial component of language processing problems. The method finding the most likely sentence can be used in Natural Language Processing for text translation, for text generation, for implementing virtual chatbot, in speech tagging, in speech recognition. Every time the Language Model module scores the sentences according to their likelihood, and choses the most likely one. For example, in speech recognition the LM (Language Model) module scores the generated translations (the output of acoustic module) and chooses the most likely one.

The Language Model determine word/character probability by analysing the text. It could be done according to classic statistical method or it could be done by Deep Learning approach. The goal is the same: By learning the characteristics of the text the model will be able to understand the context of a word or generate new ones.

Nowadays the importance of Language modelling and language processing is day to day growing, this is the input point of many online services like online shopping or online translating. In the future it will be used also for real time conversation-translation as well.

II. Preparation of input

Firstly, the input text should be prepared. The goal of this point is, to generate a such kind of text which is free of "non-linguistic items". and which has the minimum reachable entropy.

I will show this with an example. The goal of this part is to generate from the sentence: "The dog barks<>." that sentence: "the dog barks".

Therefore:

- Input texts should be merged into large text file
- Using dictionaries the punctuation marks and the abbreviations should be solved. (For example, "." will be converted to "<dot>"
- Each sentence of the text should be broken into different raw. All the remaining line break should be removed.
- Each character in the text should be lowercased
- The repetitive words (For example <dot><dot><dot>) should be decreased into one. It could be done by tokenising the raw into words and after the filtering joining the words again into text.

  - By deep learning one step is remaining. The texts should break up into different parts, to train, test, validation parts. (Taking into account that in each part somehow the texts should be mixed.) A good idea is, in case if we have some books from different authors, to mix the books immediately by the reading in and after that we can snip the already mixed text into parts while being sure that the parts will be heterogenic.

III. Classical Language Processing

There are different statistical techniques for language modelling. The common is that all of the statistical (Markov) models determine the probability of a sequence of word formally as follows:

$$P(w_1, w_2, \dots, w_i)$$

$$P(w_1, w_2, \dots, w_i) = P(w_1) * P(w_2|w_1) * P(w_3|w_1, w_2) * P(w_4|w_1, w_2, w_3) \dots$$

The problem could be also formulated: Which word is the most likely to follow the beginning-sequence.

For this question there are different options to calculate it. An option is to use probabilistic language models. Prob. language models states that the probability of i. word should be calculated with taking into account all the words before i. However, the memory is always a bottleneck.

UniGram:
UniGram model works at the level of individual words. It calculates the next word/character probability without taking the previous word/character into account

$$P(w_1, w_2, \dots, w_i) = P(w_1) * P(w_2) * P(w_3) \dots$$

BiGram

BiGram calculates the next word/character probability taking one of the previous words/characters before predicted word/character into account

$$P(w_1, w_2, \dots, w_i) = P(w_1) * P(w_2|w_1) * P(w_3|w_2)$$

N-gram

By N-gram model the parameter n limits the number of words (n-1) word which will take into account by estimating the n-th one. N-gram calculates the next word/character probability taking n-1 of the previous words/characters into account by n=3

$$P(w_1, w_2, \dots, w_i) = P(w_1) * P(w_2|w_1) * P(w_3|w_1, w_2) * P(w_4|w_2, w_3) \dots$$

How could be $P(w_1)$ and $P(w_2|w_1)$ and $P(w_3|w_1, w_2)$ calculated?

$$P(w_1) = \frac{C(w_1)}{N}$$

where C is the number of occurrences of w1 word
And where N is the number of words

$$P(w_2|w_1) \; = \frac{P(w_1 n \; w_2)}{P(w_2)}$$

where $P(w_2)$ could be calculated as $P(w_1)$ was
And where $P(w_1 n \; w_2)$ will be calculated as follows:

$$P(w_1 n \; w_2) = \frac{C(w_1 \; w_2)}{N}$$

where $C(w_1 \; w_2)$ means how many times is w1 with w2 followed (w1 w2) word pairs
And where N is the number of word pairs. (which is equal with number of word by huge amount of data)

$$P(w_3|w_1, w_2) = \frac{P(w_1 n \; w_2 \; n \; w_3)}{P(w_2 n \; w_3)} = \frac{C(w_1 \; w_2 \; w_3)}{C(w_1 \; w_2)}$$

where $C(w_1 \; w_2 \; w_3)$ counts how many times (w1 w2 w3) is a sequence

Generally:

$$\hat{p}(w_i = m|w_{i-k:i}) = \frac{count(w_{i-k:i+1})}{count(w_{i-k:i})}$$

Maximum entropy model

Maximum entropy models combine n-gram models with feature functions. Here there are more parameters than in n-gram model. The base principle of the model is entropy. The model chose that option which entropy has the maximum value. Because it has the maximum information. The equation is the following:

$$P(w_m| \; w_1, \dots, w_{m-1}) = \frac{1}{Z(w_1, \dots, w_{m-1})} * e^{(a^t * f(w_1, \dots, w_{m-1}))}$$

where Z is the partition function, where a is the parameter and f is the feature funct. (in simply case f is just an indicator of n-gram functions.)

EXAMPLE

technique	unigram	bigram	trigram
order of Markov m.	0	1	2
text	to, be, or, not, to, be	to be, be or, or not, not to, to be	to be or, be or not, or not to, not to be

After the choosing of the method a dictionary will be generated according to the chosen n-gram model. This is the so called "ARPA" Language model.

ARPA Language models describe probabilities (based on log10) of words according to the chosen n parameter.

An example looks like this:

```
\data\
ngram 1=5776
ngram 2=55566
\1-grams:
-4.7039757 <unk> 0
-4.554822 </s> 0
-1.7441652 the -1.075229
-2.4278247 book -1.6445116
-1.6198214 of -1.2298658
-3.932976 mormon -0.52456135
-2.7216232 an -0.6389431
-3.7242882 account -0.75844955
…
\2-grams:
-0.000042455064 . </s>
-1.9652246 <s> the
-0.5145896 of the
-1.5662498 mormon the
-1.7051648 written the
-0.30002946 by the
…
\end\
```

For example the last raw "-0.300 by the" should be understood as following: The probability of "the" provided that "by" was before "the" is log10(P)=-0.3 where P=0.74.

By unigram: for example in the first raw "-1.7441652 the -1.075229" the first number is the probability of "the" word. The last number is a back off value.

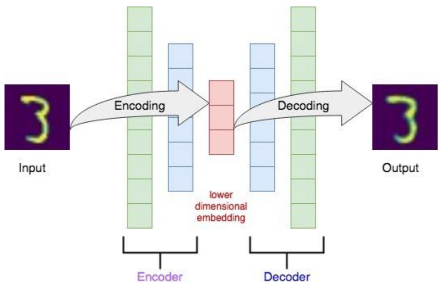
The ARPA text dictionary will be used as following: Provided 3-gram ARPA dictionary: the following word will be predicted firstly taking into account the 3-gram chapter of the dictionary. If there isn't any sequence with this word there it will be searched in 2-gram chapter. If there isn't any sequence ending with this word here either than it will be searched in the unigram chapter. By the processing smoothing algorithm is used, to generate sometimes different output than the most likely one. (This is important to avoid the repetitive, boring sentence, and it gives free range to creativity and it makes possible to create such kind of word-pairs which have not seen before in the input text.

The models above are different in significantly and in complexity. The simpler models are faster but not as accurate as the more complex ones. Because of Language Processing is a complex task the more complex models perform better. And here, we arrived to the next topic: the Deep Learning based models.

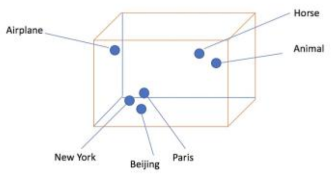
IV. Deep Learning based Language Models

The modern neural network-based models solve the complexity problem of traditional models. The strength of the Deep Learning based Language Models is, that they are so more complex than the models before that they parameters won't explode according their complexity. They are able to remember more far word sequence in the past and that's because they could understand the context and the reference of a word more than before. From this information (on the context and reference from the long past (and in some cases form the future as well)) these models are able to predict more accurate.

By using the Neural Networks, the problem transforms into classification problem. Therefore, as a first step the input must be converted into numbers. One method is one hot encoding, in this case each of the words becomes a different 0-1 combination where 1 is only once used. One hot encoding stores the difference in which place the number 1 is. like first: 100, second: 010, third: 001
More sophisticated encoding types are auto embedding techniques which are already NN-s. They get as input the word and their output is also the same word. But the goal of the network is to compress the word into a number using encoder end decoder module.

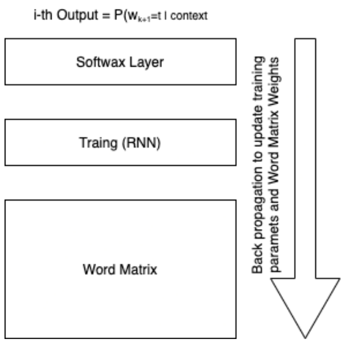


A frequently used example is the wordtovec encoding where each word becomes a vector as output so that the similar words will be closer to each other in this vector field than the different ones.

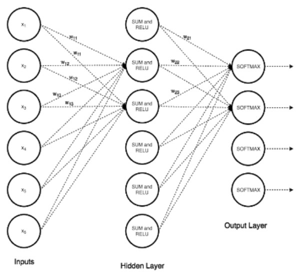


After the preparation of text the language model neural network should be prepared. A NN-s architecture is the following: It has an input layer, hidden layers, an output layer. Each layer consists of nodes and this nodes are connected with each other layer by layer.

The goal is to build such a structure that is able to learn and function as a language model by fine tuning the weights and the activation functions. Therefore, NN training algorithms are used, which updates the weights iteratively according stochastic grad descent method. There are different types of NN-s: Feedforward, Convolutional, Recurrent, etc.


i-th Output = P(w_{k+1}=t I context
Softwax Layer
Traing (RNN)
Word Matrix
Back propagation to update training paramets and Word Matrix Weights

As mentioned, for language modelling a classifier neural network should be built. This kind of NN-s have as many outputs as many words does contain the dictionary.

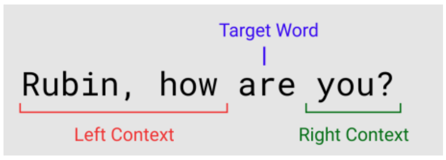


There are different types of neural networks (NN) which are used nowadays for language modelling. The goal is to predict the probability of the next word by classifying. This NN-s get as input a word sequence. The simplest ones are trained as n-gram models. Their input is n-1 word and they should predict a prob. distribution in the output as n. word according to the Language model and the input. Here fixed windows size is used to feed the NN with the prev. words.

$$P(w_t |\, w_{t-k}, \dots, w_{t-1})$$

The more sophisticated ones doesn't use only the words before the predictable word but they are using the words after it as well by the training phase. They are the bidirectional NN-s.

$$P(w_t |\, w_{t-k}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t-k})$$

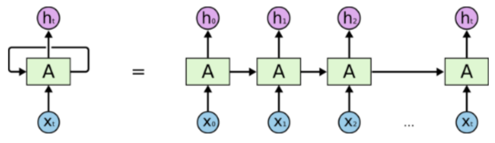


The, modern typed NN-s goal is to learn the context of a word. Therefore, they try to use every word and knowledge before the actual prediction. How is it possible?

The first one was published in 2003 called "A Neural Probabilistic Language Model". This was the base of the RNN (Recurrent neural network) (NN) model.

This approach changed significantly our perspective and RNN became a fundamental architecture. This was the first

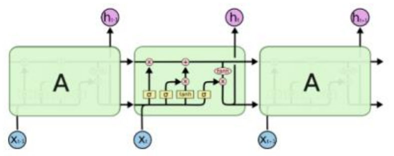
step to build a model which was able to remember all words from the earlier feeding as well. However RNN struggles of vanishing gradient problem that's because RNN is not able to remember all the word from the past. That's because LSTM (Long short term NN) is the nowadays commonly used NN for Language Modelling.
RNN (Recurrent Neural Network)
A major characteristic of neural networks is that they doesn't have memory. Each input is processed independently, only the weights of the NN is the same, (till the backward propagation process). If we would like to process a sequence of data, we have to think about the network as if it were there connected with themselves in the time continuum by every iteration. That is the key idea of RNN-s. RNN processes sequences by iterating through the sequence elements and reattach a state to the next iteration containing information about the prev. one.



An unrolled recurrent neural network.

If we do so, we will be able to remember all of the perv. inputs. (In theory) However RNN suffers from vanishing or from exploding gradient and it is not able to perform as well as it cloud. The RNN performs in Short terms well. But the problem occurs when the distance of the actual and the "reference" word is too large. That is the long-term problem. To solve Long Term problems LSTM is the solution. (Long-Short Term Memory NN)

LSTM (Long Short Term Memory network)
Are a kind of RNN which are able to "remember" long term as well.



The repeating module in an LSTM contains four interacting layers.

These networks has one cell and are working with three gates. This gates determines from which state how much information do we want to have in our "actual" cell state. This gates are the input gate, the forget gate and the output gate.
Input gate decides if we add and how much we add something to the present cell state from the input.
With Forget Gate we can decide what we would like to remove from the prev. state keeping only relevant information.
By tuning Output Gate: we can decide what we want to output from our cell state.

Now we can generate text. Cant we?
Generating text is not as easy as it sounds. While text generating, the choosing of the next character/word is the question. The naive approaches and the classical models say that the most likely next character/word should be chosen. But with this approach we will get repetitive, predictable sentences what doesn't looks like natural text.

A more sophisticated approach introduces noisier output with randomness by choice. This is called stochastic sampling. It allows even unlikely characters to be sampled some of the time, generating interesting sentences and sometimes creative ones. In order to control the stochasticity we can use SoftMax temperature parameter. This characterizes the entropy of the probability distribution used for sampling. (How surprising or predictable the choice of the next character will be.)

V. Preparing Data

Collecting a good set of data is an essential part of any Machine Learning (ML) task. Language models, trained on "good quality" text corpus achieve better results and work more reliably. However, being able to obtain data that lacks any noise is rarely possible. The goal of this section is to introduce the reader to the most common methods to prepare text before it can be used to train the ML model.

A. *Collecting the data*

For many tasks there are various datasets online, readily available to train the ML model. The goal of this work is to create a Hungarian text generator algorithm which is able to output children stories when supplied with the appropriate input sequence. There is however no dataset specifically for generating children's novels in Hungarian language. For this reason a sample of youth books were collected from the Hungarian database: MEK [1]. While the obtained files were relatively clean, still, there were steps needed to be taken, before the text corpus could be fed into our HLG algorithm. In the following the most common data-cleaning methods will be introduced and we will highlight the ones applied in our algorithm.

B. *Preprocessing*

Preprocessing the data is an essential step in Natural Language Processing (NLP) tasks. Its purpose it to transform the text so that the ML algorithms can perform better. Historically there are three main components: 1) splitting, 2) normalization and 3) noise removal. Splitting is a method for stripping text into smaller com-ponents. A book can be split into chapters, sentences and words. Even words can be further stripped to subwords and to characters. In the case of normalization, the words are converted into the same format: they are lower cased and in some cases punctuations are removed. Numbers can be converted into their textual representation and contractions can be expanded. Normalization typically refers to removing unwanted elements from the text, like extra whitespaces and HTML tags. The word-art frequency representation of the words in the used text corpus is shown in Figure 1.

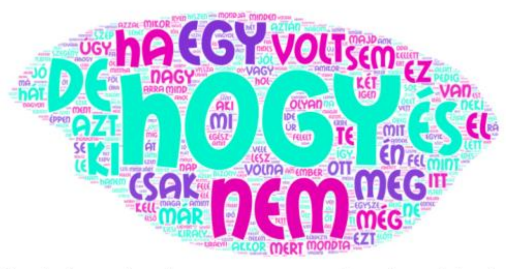


Fig. 1: A word-art frequency representation of words in the used text corpus [2]

VI. INPUT REPRESENTATIONS

Once the text has been preprocessed, it must be converted into numerical values before it can be fed into the ML algorithm. There are various ways the input can be generated and for each application the most appropriate method must be chosen. In the following three of the most popular methods will be presented: one-hot encoding, count vectorizer, word embedding.

A. One-hot encoding

One-hot encoding is useful when working with categorical features in a dataset. It uses sparse vectors where each element is zero, except those indices, which categorize the given data. In the case of text documents, tokens are replaced with their one-hot vectors. These vectors have the length of the vocabulary and contain zeros except at the index of the specific word where the value is one. As such, sentences can be turned into two dimensional matrices of size (n, m), n being the number of tokens and m the length of the vocabulary. While the method is intuitive, it has some disadvantages. We can see, that with increasing vocabulary size the vectors become increasingly long and sparse. This can be avoided by using character-level representation. In this case, the length of the vectors is limited by the number of characters present in the given language. Still, the length of the sentence is a function of the number of tokens it contains, making it less ideal for many of the ML models. The problem can be circumvented by truncating or padding the vectors. To make one-hot encoding robust against the appearance of unknown words (or characters for that matter), an artificially enlarged vocabulary can be used, which can be filled with out-of-vocabulary tokens, when necessarily.

B. Counter vectorizer

Using the count vectorizer method, sentences are replaced by their count vectors. The length of these vectors is equal to the length of the vocabulary and each number at the specific index specifies how many times the given word has appeared in the sentence. This technique reduces the size at which a given sentence is stored by representing it by a single vector instead of the large albeit sparse matrix present in the case of one- hot encoding. This however comes at a price: information about the sequential arrangement of the words is lost, only their count remains. This hinders the methods applicability to many NLP tasks, such as text generation.

C. Word embeddings

While representations like one-hot vector and count vec- tors are great for some problems like text classification, their usefulness in more complex NLP tasks is limited. One major disadvantage of one-hot vectors is that no comparison, no relationship between words can be made. Each word is represented by a vector which is orthogonal to all the other vectors in the multi-dimensional space, hence no distance or cosine-similarity can be calculated between them [3]. Word embeddings are capable of representing semantic meaning of words. Each word is characterised by a dense vector of length n. The size n of these vectors is dependent on how many corresponding features we are interested in, but usually ranges between 50 to 300. Each index corresponds to a specific feature like ”soft”, ”dark”, ”age”. The numbers at these indicies show how representative the given feature is.

VII. EVALUATION

Perplexity

Perplexity is one of the evaluation metric of language models. It shows how well a probability model predicts a sample. What makes a good language model? The model is good if it assigns high prob. to sentences/words which are correct and low prob. if they are incorrect. However larger datasets have more sentences therefore more uncertainty. So perplexity should be normalised with the amount of words (1/n)

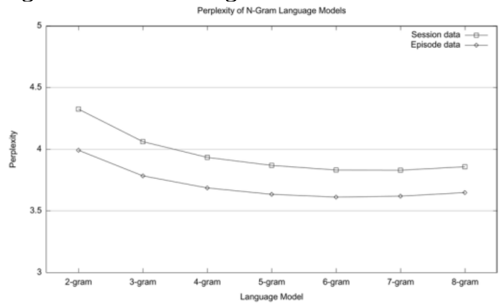
The formula is the following according to the definition as exponential cross entropy:

$$PP(W) = 2^{H(W)} = 2^{-\frac{1}{N}\log_2 P(w_1, w_2, \ldots, w_N)}$$

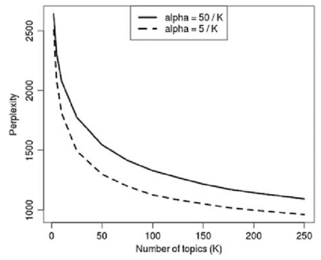
where the exponent is equal (in case bigram) with 1/n*(logpw1+logpw2|w1+logpw3|w2…)

The value of perplexity correlates with the accurate of the model. If the perplexity is 100, it means that the model is trying to guess the next word and it is as confused as if it had to pick between 100 words. (However, the correlation is only true until not too low numbers)

From this figure we can see the dependencies of perplexity according to the number of grams



And in the nex figure dependencies to the number of train words could be seen:



Out Of Vocabulary (OOV)

Out of vocabulary metric is also an important characteristic of language models. It shows the number of input words which are not represented in the dictionary. It is important because this shows in how many cases will our model mis predict the next word.

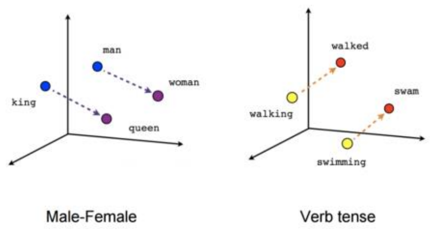
Traditionally statistic-based language models were used. But the deep learning approach brought new LM-s which outperforms the classical ones. The drawback of the statistical N-gram model are: limited complexity, limited performance. N-gram gives zero probability to words out of dictionary however NN-based are more clever. Nowadays GPT 3 has the most capacity to solve NLP problems.

VIII. Importance and uses of Language Modeling

Language modelling is essential in NLP applications. By using them next to the acoustical model the machines will be able to understand the meaning and the structure of the sentence. Without this module the machine will only predict the word according to the acoustical model.

Language modelling is used in various industries like tech, finance, healthcare, advertisements, government.
One of the usage field is Speech recognition and machine translation. Speech recognition uses language model after acoustical model to generate the most likely spoken sentence. The translation has further challenge. It should not translate the understood sentence word by word. It should generate a new sentence so, that the structure of it is understandable in the foreign language. Here it has the language model even more importance. In addition, Big Data information is also helpful in this situation. Research has proven that taking into account the frequently word-pairs will predict more accurate translation than taking into account only the language model.

Speech tagging: categorises word according their meaning. It is the base of auto encoding algorithm generates such an input for Deep Learning model where the words which has similar meaning are closer to each other than the words which has different meaning. This technique increases the accurate of the predicted words.



Spell checking and parsing of the text. Syntactical analysis is able to correct the sentences according to the spelling dictionary and furthermore according to the meaning of the sentence. The input of the method are the sentences, and each sentence will be analyzed and corrected according to the language model.

Feedback understanding and emotion detection: Sentimental analysis is used to understand huge amount of reviews faster. The method can determine the mood and the opinion of the writer when the review was written. And for example, in amazon it categorise the reviews.

Text recognitions on picture is a method used to convert the sentence from a picture or photo into text. Here instead of using acoustical model as pre-processing, image recognition module is used. After this module Language Model is again beneficial.

Writing a summary about a text. Here, the Deep Learning Neural Network is able to write a shorter summary from a long text input.

NLI Natural Language interface: The NLI interface makes easier to access data from database/ to get frequently asked but personalized information about a product. It transforms the question to a database query. In the second case it transforms the question to query and after the result if transforms back to natural language. Chatbots also uses this interface to generate their answer.

IX. References

[1] Margaret Rouse, Ben Lutkevich , "language modeling" searchenterpriseai., 2020. 03 https://searchenterpriseai.techtarget.com/definition/language-modeling

[2] Saurav Chakravorty, "Language Models", towardsdatascience, 2020.06. https://towardsdatascience.com/language-models-1a08779b8e12

[3] Duane Johnson, "Understanding ARPA and Language Models", medium, 2015.01, https://medium.com/@canadaduane/understanding-arpa-and-language-models-115d6cbc3893

[4] Chiara Campagnola, "Perplexity in Language Models", towardsdatascience, 2020.05, https://towardsdatascience.com/perplexity-in-language-models-87a196019a94

[5] Jurafsky, D. and Martin, J. H. "Speech and Language Processing" (2019).

[6] Ömer Faruk Tuna, "Introduction to Language Modelling and Deep Neural Network Based Text Generation", medium, 2020.01, https://medium.com/@merfaruktuna/introduction-to-language-modelling-and-deep-neural-network-based-text-generation-2bdfb1ab5088

[7] Yoshua Bengio , Réjean Ducharme , Pascal Vincent ,Christian Jauvin, "A Neural Probabilistic Language Model", Journal of Machine Learning Research 3 (2003)

[8] Christopher Olah Blog "Understanding LSTM Networks", Aug 27 2015

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, "Distributed Representations of Words and Phrases and their Compositionality", Part of Advances in Neural Information Processing Systems 26 (NIPS 2013)

[10] Dr. Mihailik Péter , "Perplexitás és nyelvi modell alapok", BME, online, 2020

[11] Olaszi Péter, "Magyar nyelvű szöveg-beszéd átalakítás: nyelvi modellek, algoritmusok és megvalósításuk", BME, 2002