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 generator algorithm for the Hungarian language using LSTM model. The work uses and prepares datas, collected from Magyar Elektronikus Könyvtár (MEK). Before NLG, NLM (Natural Language Modelling will be introduced and as an introduction for Language modelling firstly, the classical, mathematical approaches will be presented. After that, the modern Deep Learning based approaches will be explained. After the theory, an LSTM model will be implemented. In the end of the document, the model will be evaluated.**

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 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.

A. Preparation of input

To build a language model, firstly, the input text should be collected and 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<>." a sentence: "the dog barks".

Therefore:

- Input texts should be merged into one 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.

B. 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

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

Sz. Novoth, Zs. Páncsics with the Department of Electrical Engineering, Budapest University of Technology and Economics, Budapest, 1111, Hungary (email: novothsz@gmail.com, zsombor.pancsics@gmail.com)

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

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

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

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

Generally:

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

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: (generated with KenLM)

\data\
ngram 1=157898
ngram 2=792090
ngram 3=1461339

\1-grams:
-5.9373393 <unk> 0
-2.1503172 </s> 0
-3.77014 öcsi -0.31327975
-1.8583909 az -0.60119766
-1.475226 a -0.540149
-4.6322412 kisfiú -0.18078014
-4.2410226 aki -0.13330047
-3.3139088 mindig -0.23707722
…
\2-grams:
-3.3537679 akart húzódni -0.077821486
-2.6883066 mögé húzódni -0.077821486
-0.5435231 kétcsövű pisztolyát -0.17812906
-3.4830492 hét tatárt -0.077821486
-3.5945163 első tatárt -0.077821486
-5.842187 <vesszõ> lepuffantja -0.077821486
-5.646636 a szögletét -0.077821486
-4.121507 minden szögletét -0.077821486
…
\3-grams:
-0.80541956 a kocsikázás fáradalmait
-0.50439614 a diadalút fáradalmait
-2.1932456 <vesszõ> bizonyára kellően
-2.5905187 az állat képességeit
-0.5043997 nevű léghajójuk képességeit
-3.039432 azt az árat
-0.78494793 megérte az árat
-1.2620379 kezébe egy árat
-2.2785249 jó nagy árat
-2.348863 ha olyan árat
-0.78483415 gyűjtők minden árat
\end

For example the last raw's probability "-0.78483415" should be understood as following: The probability of word sequence "gyűjtők minden" is followed by "árat" is log10(P)= -0.78483415 where P=0.45.

By unigram and bigram: the first number is the probability of word and the last number is the a back off value (it is used if 3-gram was not found).

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.

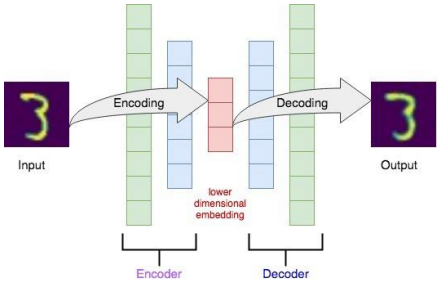
C. Deep Learning based Language Models

The modern neural network-based models solve the "lack of 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 sentences. They learn by the context, therefore they are able to generate more accurate sentence for a context, even if they never seen this word sequence before.

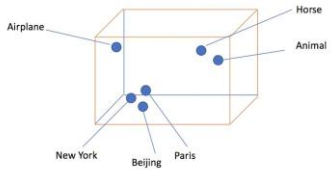
1) Input Generating

By using the Neural Networks, the problem transforms into discriminative 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 technique requires another NN-s. They get as input a word and their output is also the same word. Why is it good for us? The goal of an auto embedder network is to compress the words into numbers using encoder end decoder module.



1. Figure Auto embedder schematics

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.

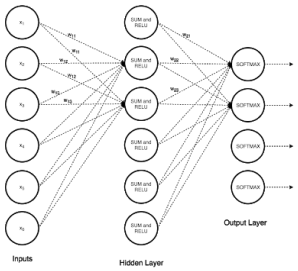


2. Figure Word2Vec output example

2) Types of Neural Network models

After the preparation of text the language model 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.. 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.



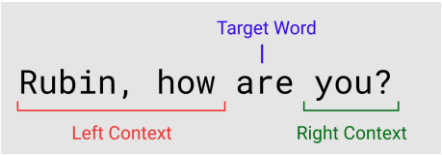
3. Figure Schematics of classifier Neural Network

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 for n. word according to the input. Here fixed windows size is used to feed the NN.

$$P(w_n | w_{n-k}, \dots, w_{n-1})$$

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.

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



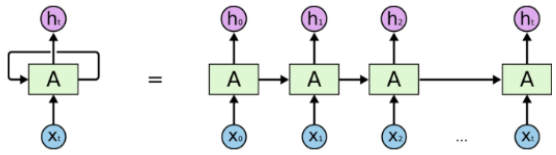
4. Figure Bidirectional Language model example

The goal is the same; to learn the context of the word. Therefore, if it is possible, we will use every word and knowledge before the actual prediction. How is it possible?

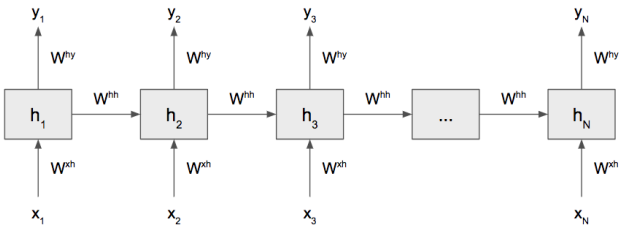
The first publication called "A Neural Probabilistic Language Model" in 2003 solved this problem with feedback. This was the base of the RNN (Recurrent neural network) (NN) model.

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 stable, (till the backward propagation process). If we would like to process a data sequence, we would like to have memory. Luckily we could think about the network as if it were "back leaded" and "re" connected to themselves in the time continuum by every iteration. That is the key idea of RNN-s. RNN processes the sequences by iterating through them and reattach the perv. state to the next iteration in this way saving information from the prev. one. If we use this technique, the NN will be able to remember all of

the perv. inputs. And so the output will not only depend on the actual input but the prev. ones as well.



5. Figure RNN working principle



6. Figure RNN weight structure

The RNN has an input X, a Wxh weight matrix for the input and it's actual (hidden) state is h. It has a recurrent Weight matrix Whh, an output and an output weight matrix Why.
The actual state is equal to an activation function(input*input weight matrix + prev. state*recurrent weight matrix)

$$s_t = sigmoid(W_{xh} * x_t + W_{hh} * h_{t-1}), h_0 = 0$$

The output state is equal with act.funct(actual state*output weight matrix)

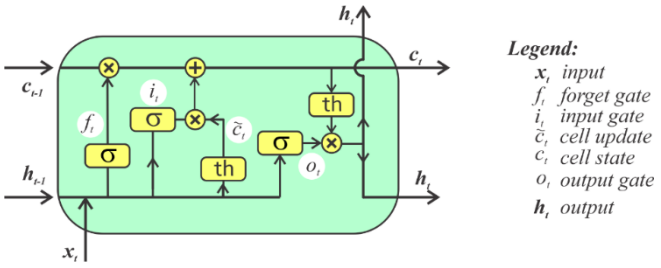
$$y_t = softmax(W_{hy} * s_t)$$

For the training of the network a modified backpropagation BPTT (BackProp Through Time) algorithm is used. To every sequence belongs a predicted output and an "y_hat" target value. The cost function (by classification the crossentropy) should be updated according to the output weight matrix AND according to the recurrent weight matrix as well.

The RNN performs in Short terms well. But problem occurs when the distance of the actual and the "reference" word is too large. As the distance is growing so is the calculation complicating. RNN suffers from vanishing or from exploding gradient problem and it is not able to perform as well as it could. 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 "back leaded" NN-s which are able to "remember" long term as well. Hochreiter,Sepp,Schmidhuber are the founders of the architecture (1997).

"This network is able to model the events and understand the context in much longer time continuum than RNN. Because it solves the vanishing gradient problem. The network is capable to process different sized inputs and outputs as well. As optimisations technique RMSprop or ADAM is used. "[12]



7. Figure LSTM cell units

These networks has one cell, where the actual state is stored and has three gates, which determines from which state how much information do we want to let in into our "actual" cell state. This gates are the input gate, the forget gate and the output gate.
The cell becomes on "Block input" channel the tanh(sum) of the actual input and the prev. state. The Input gate intended to decide if we add and how much we add to the "actual" cell state from the input (prev. state + act. input). The Forget Gate decides how much the the "prev. state" should take into account. The goal is to keep only the relevant information. Now the actual "cell state" is generated After activating the "actual cell state" with than, The Output Gate will decide what will be good to present as an output from "actual" 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.)

II. Building of the Model

A. 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.

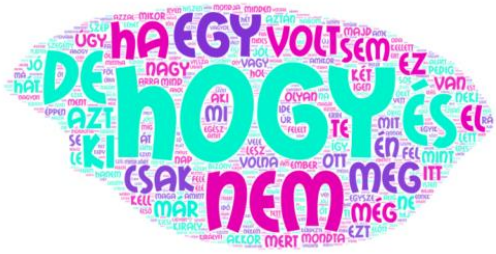
1) Collecting the data

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.

2) 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.



8. Figure A word-art frequent-representation of words of input text corpus

B. 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.

1) 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.

2) 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.

3) 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.

C. Buildig the model

We have tested the developed algorithm in Colab. First, we have created the train and validation dataset using the word2vec method. The chosen embedding dimension was 128 and the size of our vocabulary is 4095. Our model consists of a single LSTM unit followed by a Dense layer with output which has the size of the vocabulary. For training the dataset we feed in a sequence of the words with predefined length and try to predict the following word, which comes right after the input sequence. Both the input and the output are converted into a vector form naturally. The input vectors are created using the word embeddings and the output vector is a one-hot encoded version of the word.

The purpose of the project was to create a hungarian text generator using children novels. For this reason, we have taken the best performing model and used it to generate text. To start the generation only a starting input sequence is required and the algorithm automatically generated n number of words. After the first output each subsequent output is generated by deleting the first word in the input sequence but adding the newly generated output to the last position of the input sequence. At this point we have to mention, that in some cases such an algorithm can fall into endless loops, generating the same sequence of words over and over again. To avoid this, we have implemented a method, which re-samples the output using a random multi-modal sampling method. Using this, a little randomness can be added to the prediction making it robust against such occurrences. Finally, we would like to give an example for a prediction we were able to

obtain: "lépésnyire voltam sötétben ember akarom nem legyek fog érnek estig oda megyek tőle már lehet esze lesz gyakran és meg vesszõ mondja tehát majd nem leszünk gyuri ősz enyém kis meg voltam velem annak és ezt foga mikor kérdeztem asztal egymás elfogyott egész valami ne annak majd nézzen szobában volna"

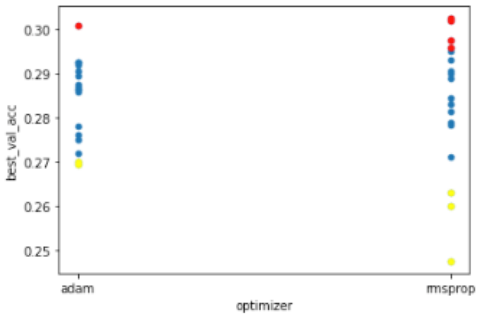
This clearly doesn't seem to be able reach such heights of writing skills as famous poets or authors do, but carefully looking and the words we can see, that their sequence does make some sense. Just one example: "lépésnyire", "voltam" "sötétben". There is a high chance, that the models has somewhere seen the combination of such words, for example when someone in one of the books has complained about barely being able to see in the darkness.

To summarize we have successfully created and trained an LSTM-based model to generate Hungarian word-sequences. The prediction is good, but not perfect and there are various ways to improve the methodology of our method. Key areas to improve the algorithm is choosing a better and possibly larger vocabulary based on frequencies of the occurring words in the corpus, gathering more data and training on larger datasets, increasing the size of our model and lastly by inspecting the probability of the predicted words.

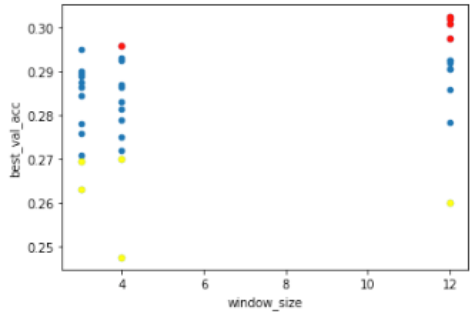
III. Evaluation Of the Model

To test the algorithm across multiple dimensions we have implemented hyperparameter optimization. With this method we have achieved performance increase compared to a baseline model with hand-tuned parameters. The hyperparameters were as follows: 1) the window size, which defines the the length of the input, 2) the size of the LSTM block, 3) the optimizer (adam and rmsprop) and 4) the dropout rate. Hyperparameter optimization can be achieved using pre-developed tools like hyperas and the new keras-tuner, however we have found that none of these are perfect for our setup. For this reason, we have coded our own hyperparameter optimizer and let it run for each hyperparameter-combination for 40 epochs. To ensure that the training of the model doesn't worsen once it has been properly trained, we have also implemented callbacks to prevent the validation loss to increase. The patience was set to a value of 10. We have found that it was triggered at around the 13th epoch for many of the hyperparameters.

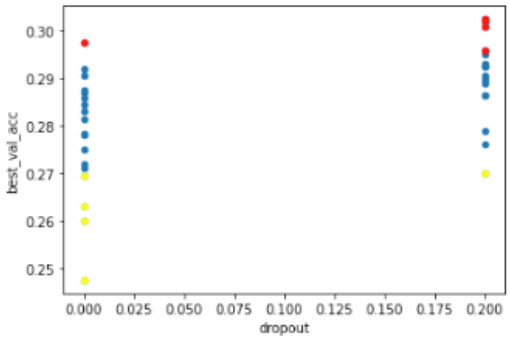
The metrics of each training was sawed to a .csv file and plotted. From the figures the following conclusions can be drawn: we have achieved good performance with LSTM blocks with size 32 and 512, but size 256 doesn't perform as well. We predict however, that when training the algorithms on the entire dataset (instead of a portion of it), increasing the size of the LSTM block will give us better performance. This makes sense, give that leading algorithms in the field are designed to be vastly larger than ours to better fit the data. Second, a window-size of 3, 4 and 12 all give good performance. It is also apparent, that dropout does improves performance. Lastly, we can see, that out of the optimization methods the best one is the rmsprop followed by adam (sgd was tested separately, but it performed significantly worse).



9. Figure Hyperparam. opt. according optimiser alg.

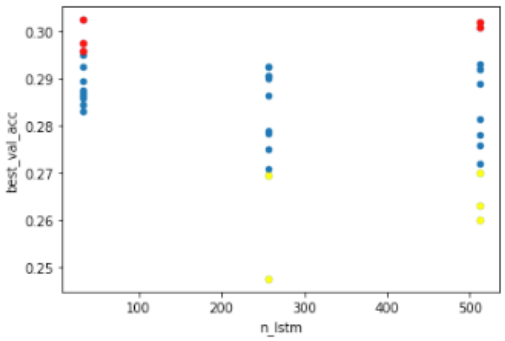


10. Figure Hyperparam. opt. accordingwindows size



11. Figure Hyperparam. opt. accordingdropout param.

Hyperparam. opt. according



12. Figure Hyperparam. opt. according member of LSTM units

	window_size	n_lstm	dropout	optimizer	best_val_acc
27	12	32	0.2	rmsprop	0.3025
35	12	512	0.2	rmsprop	0.3020
33	12	512	0.2	adam	0.3010
26	12	32	0.0	rmsprop	0.2975
15	4	32	0.2	rmsprop	0.2960

13. Figure Hyperparam. opt. result

According to the table of results the best model has

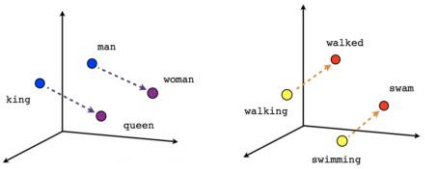
- window_size: 12
- 32 or 512 LSTM unit members
- 0.2 dropout parameter
- rmsprop optimizer

IV. 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.



Male-Female Verb tense

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.

V. 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

[12] Gyires-Tóth Bálint, Csapó Tamás Gábor, Zainkó Csaba, Moni Róbert, Hajgató Gergely DEEP LEARNING A GYAKORLATBAN PYTHON ÉS LUA ALAPON BME előadások és gyakorlatok anyagai 2020

[13] Jason Brownlee Text Generation With LSTM Recurrent Neural Networks in Python with Keras 2016

[14] Jay Alammar The Illustrated Transformer 2018

[15] Jay Alammar The Illustrated GPT-2 (Visualizing Transformer Language Models) 2019

[16] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, Alexander M. Rush Open-Source Toolkit for Neural Machine Translation 2017

[17] Jason Brownlee Gentle Introduction to Statistical Language Modeling and Neural Language Models 2017

[18] Jonathan Hui Speech Recognition — Acoustic, Lexicon & Language Model 2019

[19] Balázs Tarján State-of-the-art Natural Language Processing for PyTorch and TensorFlow 2.0 2020