National Research University Higher School of Economics

Faculty of Computer Science
HSE and University of London Double Degree Programme in
Data Science and Business Analytics

TERM PAPER

<u>Research Project</u>

Research of text normalisation methods for speech synthesis systems

Prepared by the student of Group 182, in Year: 3,
Tafintseva Albina Vadimovna
Term Paper Supervisor: Baliev Dmitry Leonidovich,
Sberbank RnD TTS Team Lead

Moscow
2021

Abstract

Nowadays, Natural Language Processing (NLP) is one of the most discussed topics in Artificial Intelligence (AI). NLP is aimed at teaching machines so that they can recognize, “understand” and pronounce text in the same way as humans can. Combining technologies such as machine learning and deep learning models, NLP makes it possible to implement this.

Many powerful models have been invented in NLP field to process our language. These models are already able to write essays, create pictures based on text and also normalize texts. In this paper I would like to investigate text normalization, how such a model works, and how it can be improved.

Text normalization is text processing and bringing it to “standards”. It is a rather difficult issue to teach the machine to identify the meaning of the text and change it leaving the meaning of the sentence behind. The task of text normalization is to bring the text into a form that will not contain any abbreviations, numbers, dates, etc. Text of this kind can be used for further training of speech synthesis systems. Thus, deep learning models help to automate text normalization for subsequent tasks applied in NLP.

Table of Contents

Main Issue

The aim of this project is to create a model that is able to normalize text to certain standards. This direction has an unlimited number of solutions, since our goal is still to bring the accuracy of the model to the highest point. Here, we will consider several approaches to the implementation of different tasks of text normalization. As a rule, each task is individual and requires its own solution to obtain the best result.

The main goal is to normalize entire sentences containing words that need to be normalized, as well as words that should remain unchanged. The model should be able to recognize such cases and process only those parts of sentence that are amenable to the normalization. Normalization tasks can be divided into simpler and more complex ones. To begin with, we decided to explore and try to create a model that works with simpler cases. Such inputs have no words that are not normalized, these sequences are normalized completely such as date abbreviations ("15 авг. 2003 г."). This example should be normalized as "15-08-2003". Next, the model can be modified and improved to handle complex cases where full sentences and sentence part recognition must be processed. For example, "1 мая я пошел на работу" should be convert to "первого мая я пошел на работу".

The main purpose of the work is to normalize texts containing numbers. That is, any numeric in the sentence should be disclosed in alphabetic form. In addition to numbers, you can normalize sentences containing abbreviations, parentheses, or anything else, that is not disclosed in full alphabetic form. However, our research is focused on numerals, where there are also many difficult cases. For example, there is a difficulty that not always numerals are converted to quantitative, the model is required to determine if it will be ordinal or not. Moreover, numerals can be expanded in different cases, depending on the context of the sentence. So, here are only a few problems that the model may encounter, below we will consider all in more detail.

Review of the Problem

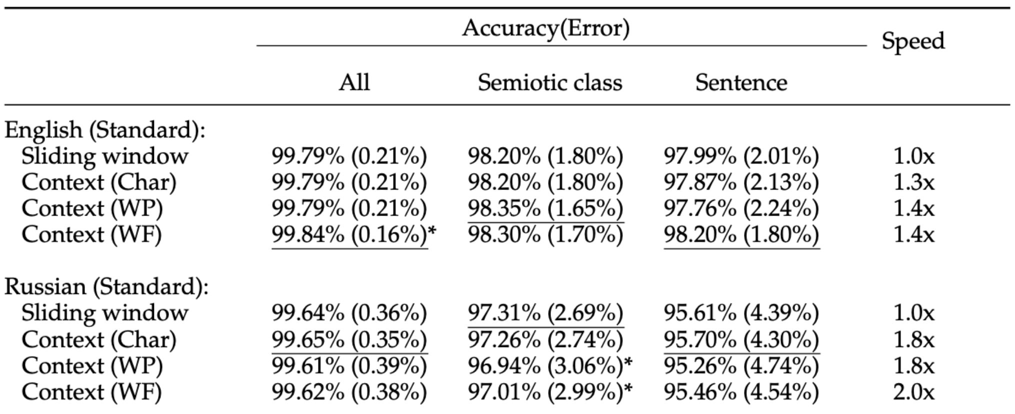
This research was based on scientific paper called "Neural Models of Text Normalization for Speech Applications" by Hao Zhang. This article discusses many methods of normalization for both English and Russian, since it is obvious that different languages require a different approach in solving the problem of normalization. For example, in English we do not encounter such problems as determining inflection, gender, and case, which in Russian is very critical. In Russian, side words are very important because they affect the disclosure of the numeral and the correctness of the normalization of the sentence. Let us consider normalization problem from the technical side as described in this article.

Here, the author considered the problem of text normalization as three parts: "Text normalization can be basically broken down into three notional components. The first is *tokenization*: How do we segment the text into tokens each of which is a word, a punctuation token, or an instance of a semiotic class? Second, what are the reasonable ways to *verbalize* that token? And third, which reading is most appropriate for the given context?" Tokenization is an important part of the model because it can greatly affect the final metrics. There it is needed to divide the text into tokens so that the model can recognize the parts of the sentence which should be changed and which should not. There are many options for tokenization: the text can be divided into characters, parts of words or whole words. The approach is individual for each normalization problem and requires consideration of different approaches to identify the best. That's why this paper discusses several types of tokenization that help identify the model with the highest accuracy.

As described above, there are many types of texts that lend themselves to normalization. For example, in this sentence, a model will work with an address: “John lives at 123 King Ave.” However, this sentence must be normalized to “John lives at one twenty three King Avenue.” First of all, this text should be tokenized to “John”, “lives”, “at 123 King Avenue” or to another tokens. Splitting into tokens also depends on the method you choose. Then, all tokens are verbalized in the way: “John” – immutable, “lives” – immutable, “at 123 King Avenue” – address. And already at the end, each token is converted to certain “standards”, that is, it is normalized. We have considered a general example of normalization from the paper, now it can be imagined how many different cases for normalizing text are there and in how many areas it can be applied.

Proposed Approaches

In the paper, mentioned above, several approaches were considered that led to quite good results. Mostly used Seq2Seq models but with different approaches to tokenization. Below you can see the results obtained on the accuracy of models with two languages: English and Russian.



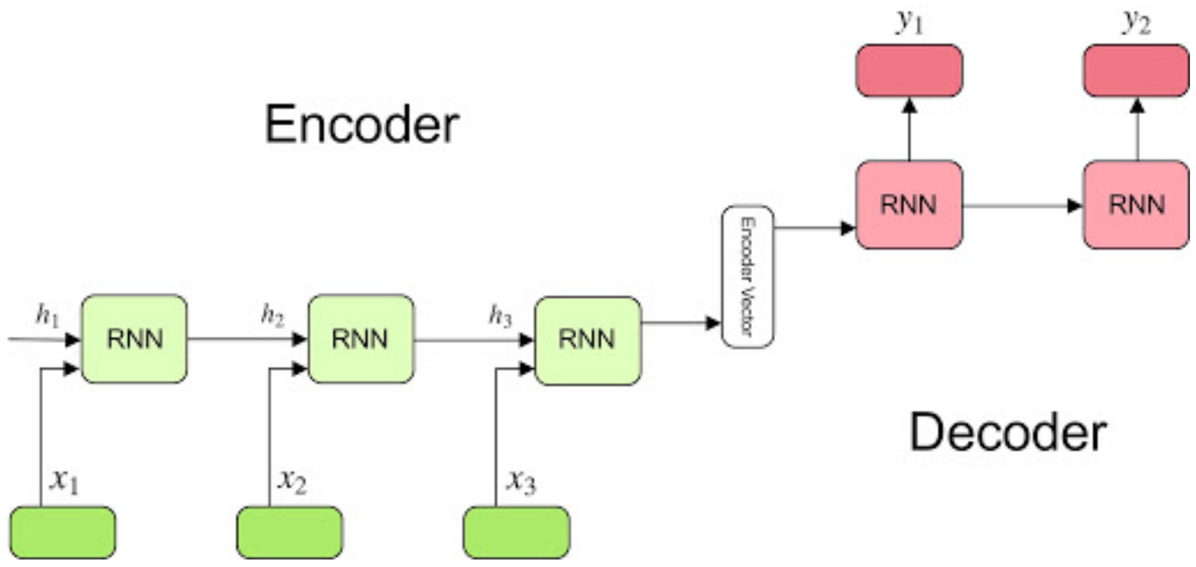
	Accuracy(Error)			Speed
	All	Semiotic class	Sentence	
English (Standard):				
Sliding window	99.79% (0.21%)	98.20% (1.80%)	97.99% (2.01%)	1.0x
Context (Char)	99.79% (0.21%)	98.20% (1.80%)	97.87% (2.13%)	1.3x
Context (WP)	99.79% (0.21%)	98.35% (1.65%)	97.76% (2.24%)	1.4x
Context (WF)	99.84% (0.16%)*	98.30% (1.70%)	98.20% (1.80%)	1.4x
Russian (Standard):				
Sliding window	99.64% (0.36%)	97.31% (2.69%)	95.61% (4.39%)	1.0x
Context (Char)	99.65% (0.35%)	97.26% (2.74%)	95.70% (4.30%)	1.8x
Context (WP)	99.61% (0.39%)	96.94% (3.06%)*	95.26% (4.74%)	1.8x
Context (WF)	99.62% (0.38%)	97.01% (2.99%)*	95.46% (4.54%)	2.0x

As it can be seen, several types of tokenization were used here, but for the English language, the model that was based on WF (word feature) tokenization was best. Word feature tokenization is when the sentence is split by spaces, that is, one token is one word. However, for Russian, the model based on Char was the best one. This can be explained by the fact that to normalize numbers in Russian we need to know the case, declension, and gender of neighboring words to convert a number (that is, the endings of neighboring words strongly affect the numerals), while in English we do not pay attention to that.

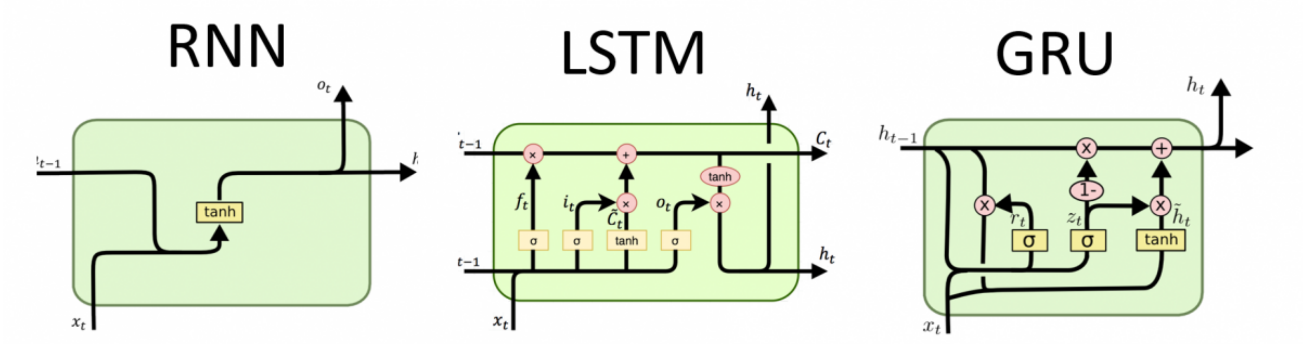
As stated above, we will first implement a model that will work with simple cases such as various forms of dates - these are short sentences that are fully processed by the model and must be fully normalized. For such a problem it was decided to apply char-level Seq2Seq model. But then, this model will be converted to word-level Seq2Seq model and several features for processing will be added in order to work with bigger texts where there are lots of immutable words.

Now we should discuss the Seq2Seq model and how it is set up. In this research it is the basic model used to normalize the text. Seq2Seq is also used in such tasks as text translation as it is a model that uses sequence-to-sequence approach. It consists in that we get a sequence of characters as input and expect also a sequence of characters as output, but not necessarily of the same length. There are different types of Seq2Seq, for example: 'one to one', 'one to many', 'many to one', 'many to many' (equal or different lengths). In our study we use 'many to many' type with different lengths.

As for the principle of operation of the Seq2Seq model, it is shown in the diagram below.



It is a standard Seq2Seq model that has layers such as Encoder and Decoder. The Encoder is responsible for processing the input data x_1, x_2, x_3 and x_4 (which correspond to 'how', 'are', 'you', '?' respectively). Each x is a token that was obtained from the tokenization of a sentence, for example x can be a word or a character. Then, we have h_1, h_2 and h_3 – it is the hidden state. So, at each step, x_i and h_i enters the Recurrent Neural Network (RNN). RNN is a class of artificial neural networks where the connections between the elements form a directional sequence. In contrast to multilayer perceptrons, recurrent networks can use their internal memory to process sequences of arbitrary length. There are also similar neural networks such as LSTM and GRU. In our research we decided to use GRU (Gated Recurrent Unit). GRU is very similar to LSTM, but it is newer, it has fewer parameters and mostly it is just as effective as LSTM. Below you can see the difference between these neural networks.



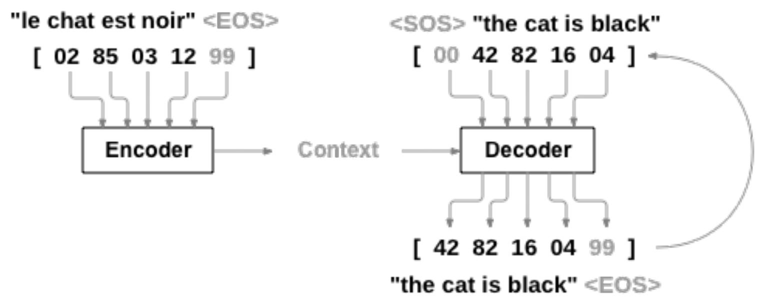
Initially, for $t = 0$, the output vector is $h_0 = 0$.

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
\hat{h}_t &= \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t
\end{aligned}
$$

Variables

- x_t: input vector
- h_t: output vector
- $\hat{h}_t$: candidate activation vector
- z_t: update gate vector
- r_t: reset gate vector
- W, U and b: parameter matrices and vector

Also, before entering the Encoder, the input is encoded according to the principle: each token is assigned its own id (as in the picture below). In addition, an embedding layer can be added to the input data x_i.



After all the input data have been processed, at the output of the last GRU cell we get the last hidden state which is called Encoder Vector and it goes to the Decoder. Decoder has different principles of operation and it depends on whether the model is being trained or validated. During training, each GRU cell receives a target (real) token, while during validation each GRU cell receives the previous value – the output of previous GRU cell. As a result, we get a sequence of ids and all we have to do is decode it back into characters or words. So, there is the sentence that the Seq2Seq model has predicted.

Implementation of Approaches and Results

At the beginning of the research, we only worked with dates that were without any context. In this case, the Seq2Seq char-level model was used since the input consisted only of the date, where both numbers and letters could be. As a result, it has been proven that with this type of data the Seq2Seq char-level model works excellently and achieves 100 percent accuracy on prepared sets:

Input: '18 июн. 1999'	*Output:* 1999-06-18
Input: 'вторник 1 ноября 1995 г'	*Output:* 1995-11-01
Input: '25-ое сентября 2014 г.'	*Output:* 2014-09-25

An excellent normalization result was obtained, so we checked that the model works and has good metrics. Based on this char-level Seq2Seq model we can move on to a more complex one for more complicated approaches.

This task was first step of the main task where it is required to process large texts and sentences where the machine does not know the part that should be normalized as it was before. The first attempt was decided to use the same model as before but for a different dataset consisting of large unnormalized texts including numerals. This run was unsuccessful because this approach was not suitable for the type of data such as long sentences, it was more important the context of the sentence to determine the correct case and gender of the numeral, as well as in what form it would be: ordinal or numeric.

So, after that we switched to the word-level Seq2seq model. We also used an attention vector in our model to help cover neighboring tokens next to the token being processed. The model's attention helps a lot to improve her performance, because in addition to finding the correct modifiable token in the sentence, the model must correctly determine what form the number should be in, while the attention mechanism allows the model to consider adjacent words to construct a new sentence without losing meaning. In this approach, the first step was to correctly tokenize sentences, so we removed all punctuation from sentences and then divided each sentence into tokens by spaces, so for example if there was a sentence like «это случилось 19.03.2005 после праздников», then we get these tokens: ['это', 'случилось', '19', '03', '2005', 'после', 'праздников']. Another variant of tokenization was also considered with the help of a python library called 'natasha'. But then tokens for this example looked like this: 'это', 'случилось', '19.03.2005', 'после'and 'праздников'. Such a tokenization where the date is considered as a whole was unsuccessful because in this case it was very hard for the model to reveal the correct numerals and we got very bad metrics.

However, another section of the library 'natasha' was used and it is called 'navec'. 'Navec' is a library with pretrained word embeddings for Russian language. The essence of these embeddings is that words close in meaning are closer to each other in the vector of all words. When processing the text, the model already has a set of words that are more likely to be adjacent in a sentence. Thus, the weights in the model were calculated also based on these embeddings.

Thus, we had a word-level Seq2Seq model with an attention mechanism and 'navec' embeddings. Not very impressive results were obtained by such a model, because, as we could trace from the poorly processed examples, the model often failed to recognize exactly those tokens that should be normalized, affected immutable tokens and often incorrectly interpreted the number into a letter format, confusing the numbers. This can happen because, in principle, there are so many numbers and there are no specific constraints so it is very difficult for model to learn and understand how numbers are expanded, especially it was often with big numbers (because small numbers, which is less 10, were met frequently and the model learns how they must be presented). Here is a good example of a sentence, which was processed by this model, to show its main problems:

['на', '5', 'мая', 'текущего', 'года', 'сумма', 'долга', 'составила', '81382', 'рубля', '93', 'копейки'] → ['на', 'пятое', 'мая', 'текущего', 'года', 'сумма', 'долга', 'составила', 'пятьдесят', 'четыре', 'тысячи', 'семьсот', 'пятьдесят', 'два', 'рубля', 'шестнадцать', 'копеек']

There on the left side is the sentence divided by the tokens which are supplied to the input of the model and on the right, we can observe the result that was obtained. As it can be noticed, there are three tokens in the proposal at once, which should be normalized. The first token was processed correctly: '5' → 'пятое', since this is a fairly easy case for the model - this is a small number and there are clear prerequisites that this is a date. Then comes a large number '81382' for which the model was not ready since such numbers were rare and it does not know how it is deciphered. So, we get a result like: 'пятьдесят', 'четыре', 'тысячи', 'семьсот', 'пятьдесят', 'два', which does not correspond to real token. And the third token was also incorrectly processed: '93' → 'шестнадцать'. However, it should be noted that the model has determined the correct form of the word 'копеек' because in the source sentence there is 'копейки' for 93 and in the target – 'копеек' for 16 which is absolutely correct according to Russian grammar.

It was clearly seen that the main problem was with the expansion of numbers, since it was difficult for the model to process numbers of any dimension. As a consequence, a method was devised to improve the performance of the model. The essence of the method is that the model remains the same, that is, unchanged, but we change the representation of the data before submitting it to the model. If earlier we applied for training a couple of source and target sentences like:

[['у', 'вас', 'платёж', '8000', 'рублей'],
['у', 'вас', 'платёж', 'восемь', 'тысяч', 'рублей']]

Now, we process the source sentence to receive a target one of the form:

['0', '0', '0', '11.1', '0']

How it is interpreted: each number matches the word from the source sentence (so its length is always equal to source length) and '0' means that the word from source sentence remains unchanged (immutable), another float numbers show the case, gender and type of number from source. For example, '11.1' means that the number '8000' must be in the nominative, feminine and not ordinal. Therefore, the model will be easier to learn, since it no longer has to learn how the numbers are revealed, it will be enough to correctly find the mutable tokens and correctly predict their case, gender, etc. based on the nearby words and the meaning of the sentence. At the output from the model, we will receive a sequence of numbers as presented above and using a python script we can decode each token back into words and for numbers there was a ready-made function that reveals the number in the desired case, gender and type.

As a result, the model trained on 13000 dataset and 2000 examples were for the test sample. Thus, the accuracy of the model was about 70 percent. There are still cases when the model predicts incorrectly, for example, it may not find the token that needs to be normalized or incorrectly predicts the case or gender. Of course, there are exceptions in the case when the dataset is not perfectly clean or as in this example it looks like the model predicted better than it was in the target sentence:

Source: *['а', 'у', 'меня', 'зарплата', '13']*
Target: *['а', 'у', 'меня', 'зарплата', 'тринадцать']*
Predicted: *['а', 'у', 'меня', 'зарплата', 'тринадцатого']*

Here it might be more appropriate 'тринадцатого' than 'тринадцать' as it is in target. So, we can say that the real accuracy of the model is higher than 70 percent. There are also examples of correct processing of the model:

Source: *['в', 'ближайшие', '3', 'дня', 'в', 'банк', 'оплата', 'поступает', 'на', 'счёт']*
Predicted: *['в', 'ближайшие', 'три', 'дня', 'в', 'банк', 'оплата', 'поступает', 'на', 'счёт']*

Source: *['78', 'дней', 'на', '19', 'марта', 'правильно']*
Predicted: *['семьдесят', 'восемь', 'дней', 'на', 'девятнадцатое', 'марта', 'правильно']*

Source: *['по', 'кредитной', 'карте', 'которая', 'оформлена', '9', 'июня', '14', 'года']*
Predicted: *['по', 'кредитной', 'карте', 'которая', 'оформлена', 'девятого', 'июня', 'четырнадцатого', 'года']*

There are many ways to improve the accuracy of the model, for example, we can make a cleaner sample for training and create a dataset balanced by classes of cases and genders. In our dataset, most examples are given in the nominative case and the neuter gender, because in the language we more often use this form with numerals. When the model receives a rarer form of numerical on the test, it is harder for it to determine the correct case, gender and type. Thus, when the sample has about the same number of examples with each case, type and gender, the model will handle the rarer case better.

Conclusion

In this research we considered such a direction in NLP as text normalization and methods of its realization. We have considered different model structures that can normalize numbers in the text so that they are represented in alphabetic form. As a result, char-level and word-level Seq2Seq models were implemented and compared. After comparing metrics, it was found that the word-level model performs better with large sentences where there may be immutable words and some numbers that must be normalized. Several features were also added to improve the performance of the model, thereby increasing its accuracy. Subsequently the normalized text can be used to train speech synthesis models. Sure, there are many other directions where text normalization will be useful.

List of sources

https://www.aclweb.org/anthology/J19-2004.pdf
https://towardsdatascience.com/text-normalization-for-natural-language-processing-nlp-70a314bfa646
https://www.ibm.com/cloud/learn/natural-language-processing