UNIVERSITY POLITEHNICA OF BUCHAREST

FACULTY OF AUTOMATIC CONTROL AND COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DIPLOMA PROJECT

Intelligent Linguistic System for the Grammar of the Romanian Language

Ioan-Florin-Cătălin NIȚU

Thesis advisor:

Conf. Dr. Ing. Traian-Eugen REBEDEA

BUCHAREST

2020

UNIVERSITY POLITEHNICA OF BUCHAREST

FACULTY OF AUTOMATIC CONTROL AND COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DIPLOMA PROJECT

Intelligent Linguistic System for the Grammar of the Romanian Language

Ioan-Florin-Cătălin NIȚU

Thesis advisor:

Conf. Dr. Ing. Traian-Eugen REBEDEA

BUCHAREST

2020

To Andreea, Vali, my sister, parents, family, teachers, colleagues and friends who helped me overcome the dark moments of life, because without your support in these for years (and not only) I would not have managed to get here today. I learned so much from you!

I Sinopsis

Domeniul prelucrării limbajului natural nu este la fel de puternic dezvoltat pentru limba română așa cum este pentru altele, cum este limba engleză. Faptul de a scrie corect texte a fost mereu o necesitate, iar dezvoltarea unor instrumente care să fie de folos în această nevoie este critică. Sistemul de corectare propus primește o propoziție cu greșeli gramaticale și o corectează, folosind tehnologii de ultimă oră pentru a realiza această operație cum sunt modelele neurale bazate pe atenție ca Transformatoarele cu Codificator-Decodificator. Acestea sunt o piatră de temelie în dezvoltarea de instrumente inteligente pentru prelucrări – traducerea, rezumarea sau corectarea – de texte și sunt fundația pentru proiectul de față. Lucrarea folosește RONACC, primul corpus pentru corecții gramaticale în română pentru modelarea, antrenarea, testarea și validarea proiectului. Folosind un set de date foarte mare cu peste un milion de exemple de învățare a fost obținut un scor BLEU mediu de 45.29 de puncte, într-un timp de antrenare destul de scurt (numai două ore pentru cinci epoci) executat pe mai multe GPU-uri. Totuși, chiar și un set de date redus, de numai cincizeci de mii de exemple cu un număr de o sută de epoci obține un scor BLEU mediu de 33.29 de puncte în trei ore. **Cuvinte cheie**: limba română, gramatică, transformatoare, atenție, codificare poziționlă.

II Abstract

The field of natural language processing is not as strongly developed for the Romanian language as it is for others, as is the English language. Writing texts correctly has always been a necessity, and the development of tools that will be useful in this need is critical. The proposed correction system receives a sentence with grammatical errors and corrects it, using state-of-the-art technologies to perform this operation such as attention-based neural models like Encoder-Decoder Transformers. These are a cornerstone in the development of intelligent tools for processing – translating, summarizing, or proofreading – texts and are the foundation for this project. The paper uses RONACC, the first corpus for grammatical corrections in Romanian for modeling, training, testing, and validating the project. Using a very large data set with over a million learning examples, an average BLEU score of 45.29 points was obtained, in a rather short training time (only two hours for five epochs) executed on several GPUs. However, even a small data set of only fifty thousand examples with as many as one hundred epochs achieves an average BLEU score of 33.29 points in three hours. **Keywords**: Romanian language, grammar, transformers, attention, positional encoding.

III Acknowledgments

I am grateful to Traian-Eugen Rebedea and Teodor-Mihai Coteț for the time, advice, inspiration and comments offered in carrying out this project. I thank Alexandru Meterez for the help, expertise, explanations and patience offered in the study of Machine Learning.

IV Contents

V List of Tables

VI List of Figures

I'm not saying that other languages, other tongues would not be wonderful and beautiful. But the language in which I was born is so peculiar, so familiar, so intimate, that I can consider it nothing but grass. We, in fact, have two coincident parts, once it is the homeland of earth and stone and once again it is the name of the homeland of earth and stone. The name of the homeland is also homeland.

A nameless homeland is not a homeland. The Romanian language is my homeland. That's why, for me, the mountain is called mountain, that's why, for me, the grass is called grass, that's why, for me, the spring springs, that's why, for me, life is lived.

– Nichita Stănescu

Chapter 1 Introduction

Never speak incorrectly; not only do you insult grammar, but you also make souls suffer.
– Socrates

Grammar (from ancient Greek γραμματική) is in linguistics the set of structural rules that establish in a natural language the relationships for the composition of clauses, sentences and words; but also aims at the study of such rules, including phonology, morphology and syntax, but also phonetics, semantics and pragmatics.[1]

Since antiquity, as described in Plato's dialogues, grammar has been an essential part of education (along with logic and rhetoric) as part of the trivium.[2] In the contemporary era, grammar is not missing from the education of young people, since the first years of school, they are introduced to the grammatical rules of the language spoken in the country where they live, the Romanian language being no exception to this custom.

Romanian is the official language of Romania (implicitly one of the official languages of the European Union) and of the Republic of Moldova (also referred to as the Moldovan language). Romanian is part of the sub-branch of the Eastern Romance languages of the Neo-Latin languages – a linguistic group that developed from dialects of Vulgar Latin, separate from the Western Romance languages between the 5th and 8th centuries – as part of the Balkan Romance languages: in addition to Romanian (Daco-Romanian), being also Aromanian (Macedonian-Romanian), Istro-Romanian and Megleno-Romanian. The Romanian language is divided into several sub-dialects (accents or graiuri) grouped as Nordic (Moldovan, Transylvanian, Banat) and southern (Muntenian, Dician – Dobrogean).[3]

1.1 Background

In Romania, the rate of functional illiteracy is 39% among students[4], and grammatical errors are frequently found in all categories of the population[5]. For English, various programs have appeared that help users write correctly, intelligibly and clearly, so that the message sent is rich in information, concise, but also easy to navigate. The Romanian language does not have such a computer program that can help a user to write documents correctly. Society is also

[1] Wikipedia, „Grammar," [Online]. Available: https://en.wikipedia.org/wiki/Grammar. [Accessed: 9 October 2019].
[2] Wikipedia, „Trivium," [Online]. Available: https://en.wikipedia.org/wiki/Trivium. [Accessed: 9 October 2019].
[3] Wikipedia, „Romanian language," [Online]. Available: https://en.wikipedia.org/wiki/Romanian_language. [Accessed: 9 October 2019].
[4] Digi24, „39% dintre elevii români sunt analfabeți funcțional. Cum îi va afecta pe viitor," 24 January 2019. [Online]. Available: https://www.digi24.ro/stiri/actualitate/educatie/39-dintre-elevii-romani-sunt-analfabeti-functional-cum-ii-va-afecta-pe-viitor-1070140. [Accessed: 9 October 2019].
[5] R. Paraschivescu, Interviewee *Cei care vorbesc corect vor forma un soi de trib, de sectă, care se va refugia undeva în păduri*, [Interview]. 26 July 2018. Available: http://www.contributors.ro/cultura/interviu-radu-paraschivescu-"cei-care-vorbesc-corect-vor-forma-un-soi-de-trib-de-secta-care-se-va-refugia-undeva-in-paduri".

moving at an increasingly fast pace, and people need quick and efficient solutions to their problems.

1.2 The Problem

Artificial intelligence has helped a lot in solving problems or in creating technologies that make people's work easier. The proposed application is based on artificial intelligence to be able to correct and proposes suggestions for changes to a text. Overall, the application provides a text field in which the user can add content (of course, in the future the features will be extended so that the user can work directly from his favorite applications), and the program will check for grammatical, lexical, punctuation, style or even tone of language, to ensure the creation of professional, correct and coherent texts. Mistakes can be highlighted by a red border, and any warnings (things that are not wrong in themselves, but can be improved) will be highlighted by a yellow border. The program will provide the opportunity to correct grammar or writing mistakes, improve vocabulary or improve readability.

1.3 Objectives

Improving the language of Romanians, but also helping citizens whose mother tongue is not Romanian must be a priority. Using a modern way, such as a computer program (which can be used on both personal computers and mobile phones, but also integrated with other existing applications – editors and word processors, e-mail programs, web browsers, etc.) can be used both for didactic purposes by teachers and for professional purposes – in industry or academia – to improve the language of documents or possibly even detect plagiarism.

1.4 Structure of the Paper

In Machine Learning, a model is a (mathematical) representation of a process with real-life applications, obtained by applying an algorithm to a data set. The paper must distinguish between the machine learning model and other models (such as the software model).

Chapter 1 (Introduction) presents the project theme, context, problem and objectives. Next, Chapter 2 (Requirements Analysis and Specification) presents the motivation for the project and the development requirements. The research on the existing approaches is done in Chapter 3 (Existing Approaches) in which a market study was made, an analysis of the existing methods and the state-of-the-art but also the presentation of the technology used. The architecture is described in Chapter 4 (Proposed Solution), and its implementation is described in Chapter 5 (Implementation Details); the two chapters are organized in subchapters with the same name, in the first being described the theoretical details, and in the second the ways of practical realization. Chapter 6 (Results Evaluation) presents the details of testing and evaluation of the project and the results obtained, and Chapter 7 (Conclusions) presents the current stage of development and concludes with possible further developments.

Chapter 2 Requirements Analysis and Specification

Correct writing is the bread of Romanian language teachers.
– Camil Petrescu

For better analysis and planning of the development strategy, the SWOT analysis from Table 1 SWOT Analysis was used to determine the strengths, weaknesses, opportunities and threats related to the development of the project.

SWOT Analysis	*Helpful* *to achieving the objective*	*harmful* *to achieving the objective*
Internal origin *(attributes of the organization)*	Strengths - can be used for self-taught purposes; - improves speech in all its aspects, so it can address a wide range of people.	Weaknesses - the process of creating data sets for the program is very laborious; - the training time of the model is directly proportional to the size of the training data set.
External origin *(attributes of the environment)*	Opportunities - the lack of such programs in Romanian; - the need to improve speech.	Threats - users' reluctance to use new tools (preferring the classic methods: asking other people or searching the internet); - people's lack of time; - people involved in the text editing and word processing are a possible competition.

Table 1 SWOT Analysis

Following the analysis, the requirements that the project must meet were determined. It must be simple and easy to use, as the user does not need technical training to understand how it works and how to use applications. For this, it is enough to be able to enter a text, with possible errors, and the received results to be another text in which the errors are correct.

2.1 Description of User Categories

The users of the application are the people who want to ensure the grammatical correctness of the texts written by them and not only. The broad spectrum includes the entire Romanian-speaking population. And the specific target audience is people who want to improve their writing skills, such as:

- Pupils and students of any level who want to improve their work for academic preparation;
- Professors, teachers and students who want to write scientific papers for specialized publications;
- Authors, journalists or correspondents working in the newsroom;
- Teachers and technical editors who will no longer be needed to correct texts when the application is launched;
- People who want to improve their writing skills;
- People who have to write texts or reports during the job (for example lawyers, police officers, jurists, economists).

2.2 System Requirements

To execute the source code (corresponding to the loading of the data set and the training of the model) one needs a computer that has installed a version of Python[6] newer than 3.5 and a list of frameworks, libraries and packages: TensorFlow[7], NumPy[8] and Matplotlib[9].

2.3 Functional Requirements

The application must allow users to enter texts (from the keyboard or from files) to be corrected. The application must allow users to save those texts.

2.4 Non-Functional Requirements

It must be possible to run the application without training the model if it has already been trained. The application must allow the model to be trained using new data.

2.5 System Models

The system model is based on the exchange of information between the application and users: users train the model and give the system input (a sentence to be corrected), and the application provides the system output (corrected sentence) using the trained model.

[6] Python. Available: https://www.python.org. [Accessed: 3 June 2020].
[7] TensorFlow. Available: https://www.tensorflow.org. [Accessed: 3 June 2020].
[8] NumPy. Available: https://numpy.org. [Accessed: 3 June 2020].
[9] Matplotlib. Available: https://matplotlib.org. [Accessed: 3 June 2020].

2.5.1 Actors and Use Cases

Actors: application users (those who want to correct texts).

Use Cases:

- Model training;
- Correcting a sentence.

2.5.2 Description of System Use Cases

The use cases presented are described below:

1. Model training:
    a. Precondition: The application is started.
    b. Basic flow:
        i. Data sets are loaded (to be processed);
        ii. The model is initialized;
        iii. The model is training.
    c. Alternative:
        i. Datasets do not exist or are invalid:
            ❖ Execution error.
        ii. Model parameters are incorrect:
            ❖ Execution error.
        iii. The model was not initialized:
            ❖ Execution error.
    d. Postcondition:
        i. The model is trained and the user can correct sentences.
2. Correction of a sentence:
    a. Precondition: The model is trained.
    b. Basic flow:
        i. The user enters a sentence to be corrected;
        ii. The user can also enter the correct version of the sentence for display and comparison.
    c. Alternative:
        i. The user did not enter a valid sentence:
            ❖ The correction of the sentence is ignored.
        ii. The user did not enter the correct sentence as a reference:
            ❖ Its display is ignored.
    d. Postcondition:
        i. The system displays the corrected sentence.

Chapter 3 Existing Approaches

The destiny of a nation depends on the state of its grammar. There is no great nation without the ownership of the language. – Fernando Pessoa

The problem of correcting texts and natural language, in general, is often encountered in computer science, artificial intelligence and machine learning. The study of natural language processing has brought technologies, models and products that can competitively address this issue. The solutions are usually approximate, but quite relevant. Most text and sequence manipulation problems (translation, proofreading, summarizing) can be modeled in the same way and technologies can be adapted from case to case.

3.1 Commercial Products

If for the English language there are a multitude of products, as can be seen below, for the Romanian language projects of this kind are small in number and are usually not as competitive as those for the English language (or for foreign languages where there has been considerable investment in research). The peculiarities of the Romanian language, compared to the English language, made the development of intelligent programs for correcting texts not as strong.

Microsoft Office Spell Checker[10] is one of the most used proofreaders. Its popularity lies in the fact that all applications in the Office suite use this feature that is available in several languages (English and even Romanian). From experience, the product is good for correcting small typos, but at least in Romanian, it is not detecting all mistakes or problems. Another disadvantage is that it does not offer solutions for understanding and improving the texts.

Grammarly[11] is an American international company that develops a very popular writing tool for English with the help of artificial intelligence and natural language processing. The product offered offers grammar check, spelling check, add suggestions for text clarity and conciseness, improve vocabulary, style and tone, all through the use of machine learning and deep learning algorithms. The main disadvantage is that the solution offered is only for English.

Qordoba[12] is another American artificial intelligence company whose technologies focus on software as a service. The main platform developed is used in content information, which includes a smart assistant along with other content quality tools. The wizard is useful to standardize the style, terminology and tone of documents written in a company, so an organization can achieve consistency on all content levels: marketing, products, legal and human resources policies, technical documentation, communication, knowledge bases for customers and more. The disadvantage of the solution comes from the fact that it is offered on the business side and not for ordinary users.

[10] Microsoft Office. Available: https://www.office.com. [Accessed: 20 June 2020].
[11] Grammarly. Available: https://www.grammarly.com. [Accessed: 20 June 2020].
[12] Qordoba. Available: https://qordoba.com. [Accessed: 20 June 2020].

Google Translate[13] is a free multilingual machine translation service based on statistics and neural networks used to translate texts and web pages. The service can correct texts received with small mistakes when it fails to understand the context or certain words. However, on a large scale, it is specialized in translations and not in grammatical corrections.

DEX Online[14] is a web platform organized as a collection of electronic dictionaries. The platform offers definitions for Romanian words, along with declensions, conjugations, synthesis, etymology and even examples of use in different contexts. The problem of this platform is that it does not offer any support for the automatic correction of texts, however by far being one of the few projects that considered the development of a free product (or a product in general) for the Romanian language.

The market for software products and services for writing and modeling texts is efficient and competitive in the English language (due to investments, but probably also because English is a fairly simple language in rules, which is why it is very popular). However, for the Romanian language, there are no products similar to those for the English language that allows the correction and improvement of the texts. Even if the Romanian language is much more complex in its grammatical rules, it does not mean that it should not be invested and researched in this field. To correct words with small mistakes some programs and modules allow them to be verified, but only at the word level, without checking the grammar or context.

3.2 Existing Methods

State-of-the-art technologies in the field of machine translation can also be used in other areas adjacent to natural language processing, such as generating abstracts for papers or proofreading texts. Below are the technologies developed over time in the field of natural language processing, both for translations and corrections. Most are based on machine learning techniques (especially neural networks and deep neural networks) and artificial intelligence: sequence-to-sequence learning, learning with recurring neural networks, encoders and decoders, learning with convolutional neural networks, and learning using attention, probably the most recent and strong evolution in the field. Other alternative methods do not use neural networks, for which the effort of collecting statistical and modeling data is very high (this does not mean that they cannot be at least as efficient as other methods).

Deep Neural Networks are powerful models that have performed excellently in difficult learning tasks. Although deep neural networks work well whenever large labeled training sets are available, they cannot be used to map sequence to sequence. A general end-to-end sequence learning approach can be used that makes minimal assumptions about sequence structure. The method uses a long short-term memory multilayer to map the input sequence

[13] Google Translate. Available: https://translate.google.ro. [Accessed: 20 June 2020].
[14] Dex Online. Available: https://dexonline.ro. [Accessed: 20 June 2020].

to a fixed-size vector, then another deep memory to decode the target sequence from the vector. Memory can also learn representations of sentences and sentences that are dependent on word order and are relatively invariant to the active and passive voice [1].

Another proposed neural network model, called the Deep Neural Network Encoder-Decoder, consists of two recurring neural networks. One network encodes a sequence of symbols in a fixed-length vector representation, and the other decodes the representation in another sequence of symbols. The encoder and decoder of the proposed model are jointly driven to maximize the conditional probability of a given target sequence with a source sequence. The performance of a statistical machine translation system proves to be improved empirically using the conditional probabilities of the phrase pairs calculated by the Deep Neural Network Encoder-Decoder. This model teaches a significant semantic and syntactic representation of linguistic sentences [2].

Neural machine translation is a recently proposed approach to machine translation. Unlike traditional statistical machine translation, neural machine translation aims to build a unique neural network that can be jointly tuned to maximize translation performance. Recently proposed models for neural machine translation often belong to a family of encoders-decoders and consist of an encoder that encodes a source sentence into a fixed-length vector and a decoder that generates a translation. Using a fixed-length vector can be a deadlock in improving the performance of this basic decoder architecture, and extending it would allow a model to automatically search for parts of a source sentence that are relevant to predicting a target word without having to type these parts as a segment explicitly. This new approach achieves a translation performance comparable to existing state-of-the-art sentence-based translation systems [3].

Another architecture is based entirely on convolutional neural networks, compared to recurring models, in which calculations for all elements can be completely parallelized during training, and optimization is easier because the number of nonlinearities is fixed and independent of the length of the entry. The use of closed linear units facilitates the propagation of gradients and each decoding layer is equipped with a separate attention module [4].

The best models connect the encoder and the decoder through an attention mechanism. The transformer is a simple network architecture, based exclusively on attention mechanisms, which is completely free of recurrences and convolutions. Experiments performed on two machine translation tasks show that these models are of superior quality, being parallelizable and requiring a significantly shorter training time. The transformer generalizes very well to other tasks and can be successfully applied for text analysis and the production of new texts [5].

In Table 2 Translation comparison (EN-DE, EN-FR) between Transformer and other state-of-the-art models [6] are shown the results of the Transformer in English to German (DE) or French (FR).

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet	23.75			
Deep-Alt + PosUnk		39.2		$1.0 \cdot 10^{20}$
GNMT + RL	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnkEnsemble		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Table 2 Translation comparison (EN-DE, EN-FR) between Transformer and other state-of-the-art models

In Table 3 Comparison analysis of the English language constitution between the Transformer and other state-of-the-art models [6] are shown the results obtained by the Transformer and how it generalizes for other tasks, not only for translations.

Parser	Training	WSJ 23 F1
Vinyals & Kaiser et al. (2014)	WSJ only, discriminative	88.3
Petrov et. al. (2006)	WSJ only, discriminative	90.4
Zhu et. al. (2013)	WSJ only, discriminative	90.4
Dyer et. al. (2016)	WSJ only, discriminative	91.7
Transformer (4 layers)	**WSJ only, discriminative**	**91.3**
Zhu et. al (2013)	semi-supervised	91.3
Huang & Harper (2009)	semi-supervised	91.3
McClosky et. al (2009)	semi-supervised	92.1
Vinyals & Kaiser et. al. (2014)	semi-supervised	92.1
Transformer (4 layers)	**semi-supervised**	**92.7**

Luong et. al (2015)	multi-task	93.0
Dyer et. al. (2016)	generative	93.3

Table 3 Comparison analysis of the English language constitution between the Transformer and other state-of-the-art models

An alternative type of spell checker uses only statistical information, such as n-grams, to recognize errors instead of correctly spelled words. This approach usually requires a lot of effort to obtain sufficient statistical information. Key advantages include the need for less storage space and the ability to correct errors in words that are not included in a dictionary.[15]

3.3 Technologies Used

This paper uses Python and TensorFlow to implement a Transformer-based solution proposed in the article by Vaswani et. al. "Attention Is All You Need" [6].

Recurrent neural networks and long short-term memory have been established as state-of-the-art approaches in sequence modeling and reasoning problems, such as language modeling and machine translation [2] [3]. Since then, many efforts have continued to push the boundaries of recurring language models and encoder-decoder architectures [7].

Attention mechanisms have become an integral part of persuasive sequence models and reasoning models in different tasks, allowing dependency modeling without taking into account their distance in input or output sequences [3]. In most cases, such attention mechanisms are used in conjunction with a recurring network. Transformers are introduced as a model architecture that avoids recurrences and relies on an attention mechanism to attract global dependencies between input and output. The transformer allows a significantly higher parallelization and can reach a new stage of technology in terms of the quality of translation or correction of texts after training [5].

[15] M. Kantrowitz and S. Baluja, „Method for rule-based correction of spelling and grammar errors". SUA Patent US6618697B1, 9 September 2003.

Chapter 4 Proposed Solution

What could be sadder than that? We came to be surprised by those able to respect the Romanian language.
– Andrei Ş.L. Evelin

Transformers are incredible models based on an encoder-decoder type structure [1] [2] [3] [4] taking the world of natural language by storm. They pushed the barriers of state-of-the-art technology and broke records in the NLP world. They have found their place in various applications, from translations and conversational agents to the improvement of search engines and are the latest craze in deep learning.

Below are presented their theoretical aspects and how they are used in natural language processing. The proposed solution uses transformers to process natural language and to correct texts written in Romanian.

4.1 Architecture

Competitive neural reasoning models have a coder-decoder type structure [1] [2] [3] [4]. The encoder transforms the symbolic input $x = (x_1, \dots, x_n)$ into a sequence of continuous representations $z = (z_1, \dots, z_l)$. Having z, the decoder generates the symbolic output $y = (y_1, \dots, y_m)$, one element at a time. The model is auto-regressive [8], using the previously generated elements as an additional input for generating the next element. The transformer is based on this architecture using attention and fully connected point-wise layers for the encoder and decoder [6].

4.2 Attention

Attention is a mechanism by which transformers can have infinite theoretical memory and thus can focus on words generated long before [6].

4.2.1 Scaled Dot-Product Attention

To gain attention, the entry is distinguished into three layers to create the query, key, and value vectors.

> *The key/value/query concepts come from retrieval systems. For example, when you type a query to search for some video on YouTube, the search engine will map your **query** against a set of **keys** (video title, description etc.) associated with candidate videos in the database, then present you the best matched videos (**values**).*[16]

The attention calculation is performed according to the model in Figure 1 Scaled Dot-Product Attention [5], where queries of size d_k are grouped in a matrix Q with l_k lines, keys of size d_k

[16] dontloo, „What exactly are keys, queries, and values in attention mechanisms?," [Answer] Stats StackExchange, 29 August 2019. [Online]. Available: https://stats.stackexchange.com/a/424127. [Accessed 19 June 2020].

are grouped in a matrix K with l_k lines and values of dimension d_v are grouped in a matrix V with l_k [6].

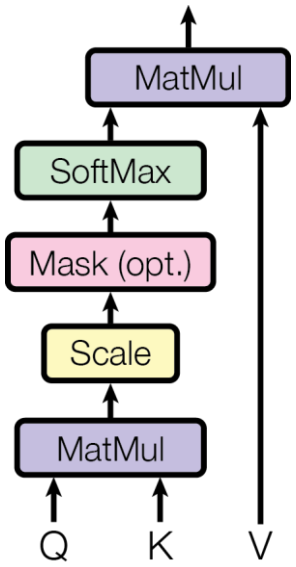

Figure 1 Scaled Dot-Product Attention

The calculations are performed simultaneously in matrix form, even if they were initially defined for vectors, as follows:

1. Scalar product between queries and keys: produces a score matrix that determines how relevant the other words are, relative to a word (the higher the score, the more relevant certain words are), and in this way the queries and keys are mapped;
2. Scaling: allows more stable gradients;
3. Masking: the assembly of a matrix of the same size where on the positions to be eliminated is $-\infty$ and on the rest of the positions 0; the masking result is observed in the next step, where when applying the softmax function the infinite values become zeros;
4. Softmax: attention weights are obtained that provide probabilities proportional to the scores; the model will be more confident about the words that must participate in the prediction;
5. Multiplication with values: the weights are multiplied by the values, and the higher scores will keep the words that the model learns as more important [9].

4.2.2 Multi-Head Attention

Multi-headed attention involves the linear design of queries, keys, and values with different learned linear projections. Each linear projection will be the input for the attention calculation, and the calculations can be performed in parallel, the results being concatenated at the end, as in Figure 2 Multi-Head Attention [5] [6]. The calculation relations are given by:

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O$$

$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

The projection parameters are $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W_i^O \in \mathbb{R}^{hd_v \times d_{model}}$ [6].

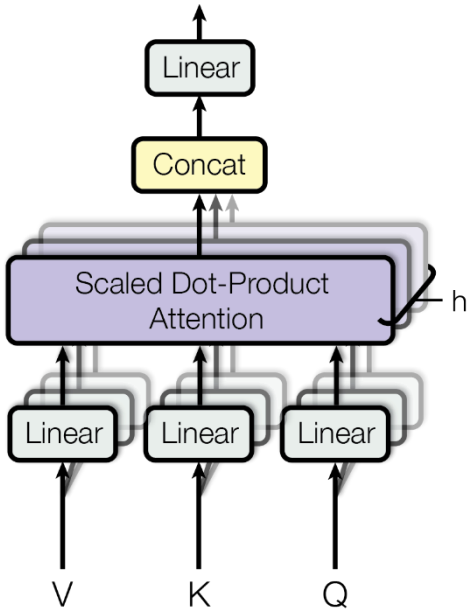

Figure 2 Multi-Head Attention

An alternative is that after learning the matrices are divided into h sub-matrices as in Figure 3 Multi-Head Attention with Split Matrices [5], the process will be carried out in the same way.

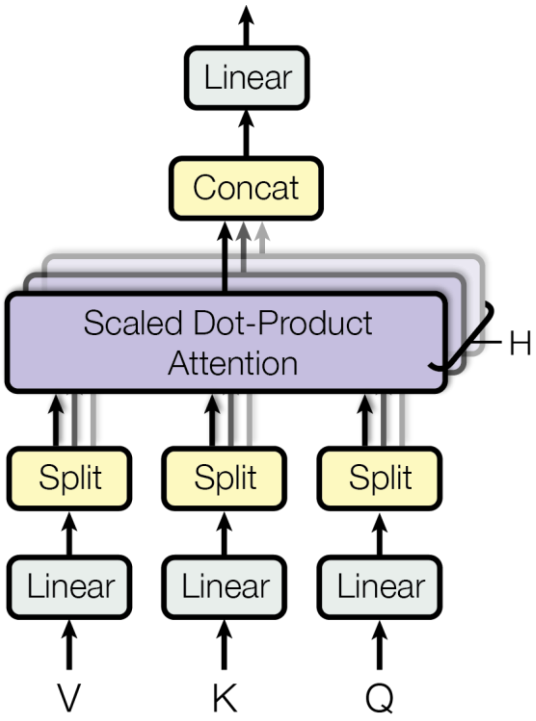

Figure 3 Multi-Head Attention with Split Matrices

4.3 Coding Layers

Each sub-layer of the coding layers has a residual connection [10] (the output is added to the input) which is then normalized [11]. Residual connections help the model to train, with gradients being able to pass directly through the network, while normalization is used to stabilize the network, which implies a significant reduction in training time [9].

4.3.1 Point-Wise Feed-Forward Network

The point-wise feed-forward network is used to project the attention of the output, which can give it a richer representation [9]. The network structure is given in Figure 4 Point-Wise Feed-Forward Network [5].

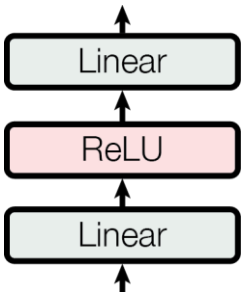

Figure 4 Point-Wise Feed-Forward Network

The network output is given by $FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$ and can be seen as the result of two convolutions with a kernel size of 1 [6].

4.3.2 Encoding Layer

The encoding layer consists of a multi-headed attention calculation sub-layer and a point-wise feed-forward network, as shown in Figure 5 Encoding Layer [5]. In the attention sub-layer all queries, keys and values come from the same place (previous layer in the encoder) [6].

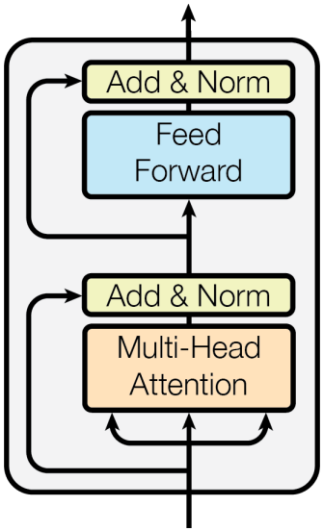

Figure 5 Encoding Layer

4.3.3 Decoding Layer

The decoding layer is composed of two sub-layers of multi-head attention calculation and a point-wise feed-forward network, as shown in Figure 6 Decoding Layer [5]. The attention layers allow each position to participate in all positions in the decoder up to this position. To preserve the auto-regression property [8], the positions beyond the current one must be masked (set to $-\infty$) [6].

In the attention layer that connects the encoder to the decoder (the second layer – the middle layer), the queries come from the previous decoder layer, while the keys and values come from the output of the encoder. This allows each position in the decoder to participate in each position in the input, similar to other attention mechanisms in sequence-to-sequence models, such as [3] [4] [6] [7].

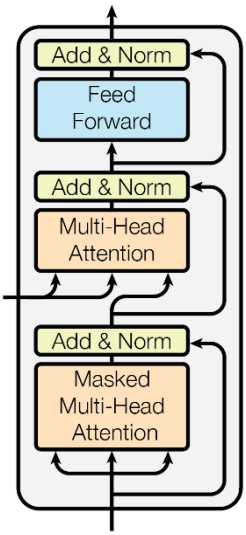

Figure 6 Decoding Layer

4.4 Coding

Learned embedding is used to convert input and output tokens into vectors of size d_{model}. After decoding, the output is converted to probabilities using the softmax function and a learned linear transformation. In the model, the weight matrix is divided between the two embedding layers and the pre-softmax linear transformation. In the embedding layers, the weights are multiplied by $\sqrt{d_{model}}$.

4.4.1 Positional Encoding

The transformer does not use recurrences and needs information about the positions in the input embedding [9]. For this it is used the positional encoding that has the same dimension d_{model} to be able to be added with the embeddings:

$$PE(pos, 2i + j) = \begin{cases} \sin \dfrac{pos}{1000^{\frac{2i}{d_{model}}}}, & j = 0 \\ \cos \dfrac{pos}{1000^{\frac{2i}{d_{model}}}}, & j = 1 \end{cases}$$

where pos is the position and i is the dimension. Each dimension of the positional coding corresponds to a sinusoid, the wavelengths forming a geometric progression $2k\pi$ [6].

These functions, hypothetically, allow the model to easily learn the values, because, for any k, $PE(pos + k, \cdot)$ can be represented as a linear function of $PE(pos, \cdot)$. The sinusoidal allows the model to extrapolate for longer lengths of the sequence than those present at the training [6]. There are other possibilities of positional encodings, such as those learned [4].

4.4.2 Encoder

The encoder architecture is shown in Figure 7 Encoder [5] and consists of:

- Input embedding;
- Positional encoding;
- Encoding layers [6] [9].

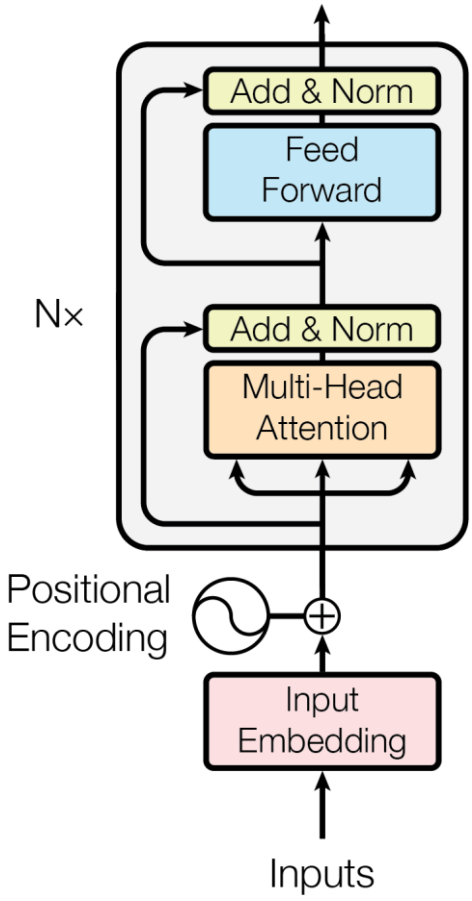

Figure 7 Encoder

4.4.3 Decoder

The decoder architecture is shown in Figure 8 Decoder [5] and consists of:

- Output embedding;
- Positional encoding;
- Decoding layers [6] [9].

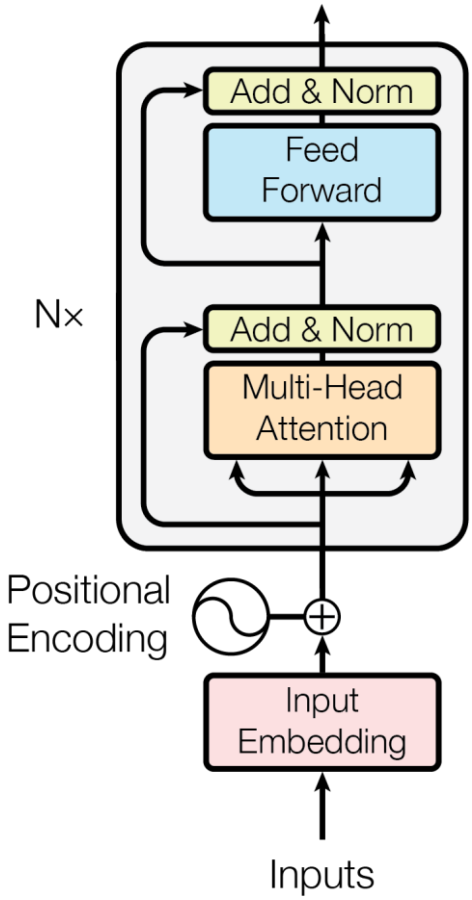

Figure 8 Decoder

4.5 Transformer

The transformer is composed of an encoder, decoder and a linear final layer the size of the target vocabulary, as in Figure 9 Transformer [5].

The encoder maps all input sequences into abstract continuous representations, which contain the learned information about the entire sequence. The decoder is auto-regressive, starts with a start token and receives the previous outputs as inputs along with the encoder outputs that contain information about the input attention. The output of the decoder passes through a linear final layer acting as a classifier (the number of classes is equal to the size of the vocabulary), and the predicted word is given by:

$$predicted_{id} = \underset{id \in \{0 \ldots vocab_{size}\}}{\operatorname{argmax}} softmax(id)$$

The obtained output is added to the decoder inputs and the procedure is continued until the final token is reached [9].

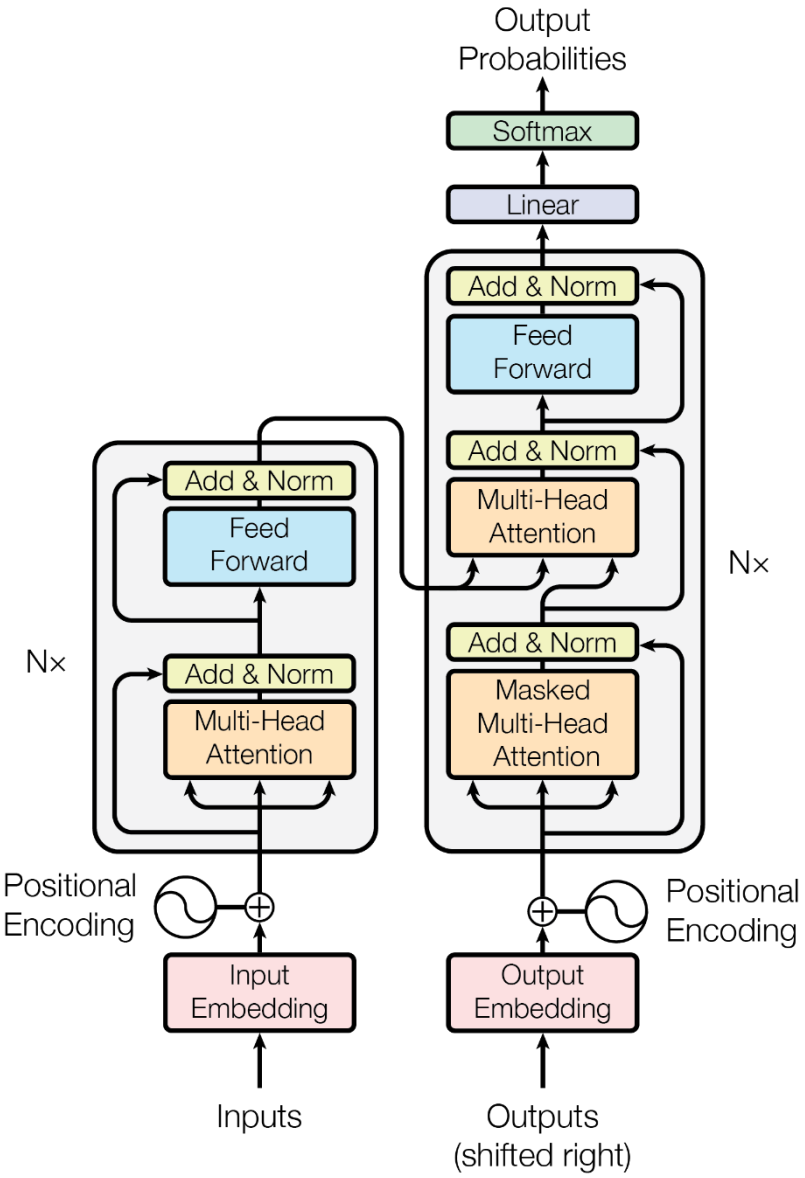

Figure 9 Transformer

4.5.1 Hyperparameters

The model uses the following hyperparameters:

- n: number of coding layers;
- d_{model}: the size of token vectors for input/output;
- dff: the size of the inner layers for the feed-forward network;
- h: number of heads for multi-head attention calculation.

4.6 Optimizer

The Adam optimizer [12] with the parameters $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$ is used together with a variable learning rate during training:

$$l_{rate} = d_{model}^{-0.5} \cdot \min\left(step_{num}^{-0.5}, step_{num} \cdot warmup_{steps}^{-1.5}\right)$$

The learning rate is represented as a linear increase corresponding to the first training steps up to $warmup_{steps} = 4000$ and then as a proportional decrease to the inverse square root of the step [6].

4.7 Loss

Loss is the optimized training metric to obtain a competitive model. The loss is calculated between two probability distributions (the predicted one and the real one). To compare the two distributions one computes their difference, calculates the Cross-Entropy or the Kullback-Leibler divergence [13]. Cross-Entropy was used in the implementation.

4.8 Training

Two adjustment methods can be used for training: [6]

1. Residual Dropout: $P_{drop_{out}} = 0.1$ applying the waiver to the output of each sub-layer, before being passed as input to another sub-layer or normalized; applying the waiver of the sum of incorporations and position encodings to both the encoder and the decoder [14].
2. Label Smoothing[17]: $\varepsilon_{ls} = 0.1$ the model learns to be more insecure but improves the accuracy and score of BL [15].

4.9 Evaluation

Other metrics used in training and evaluation, in addition to loss (Cross-Entropy) [13], are:

- accuracy;
- BLEU score.

BLEU (bilingual evaluation understudy) is a method of evaluating the quality of texts automatically translated from one natural language into another. The quality of texts is the correspondence between the result translated automatically and the result of a human translation. Scores are calculated on individually segmented texts (usually in sentences) by comparing them with a set of very good translation quality. The BLEU score is always between 0 and 1, interpreted as the similarity between the candidate text and the reference texts, where high values (close to 1) represent similar texts [15].

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, „Rethinking the Inception Architecture for Computer Vision," 2 December 2015. [Online]. Available: https://arxiv.org/abs/1512.00567. [Accessed 19 June 2020].

4.10 UML Diagram

Figure 10 UML Diagram shows the class diagram for the implementation of Transformer, Encoder, Decoder, Sub-layers and learning rate.

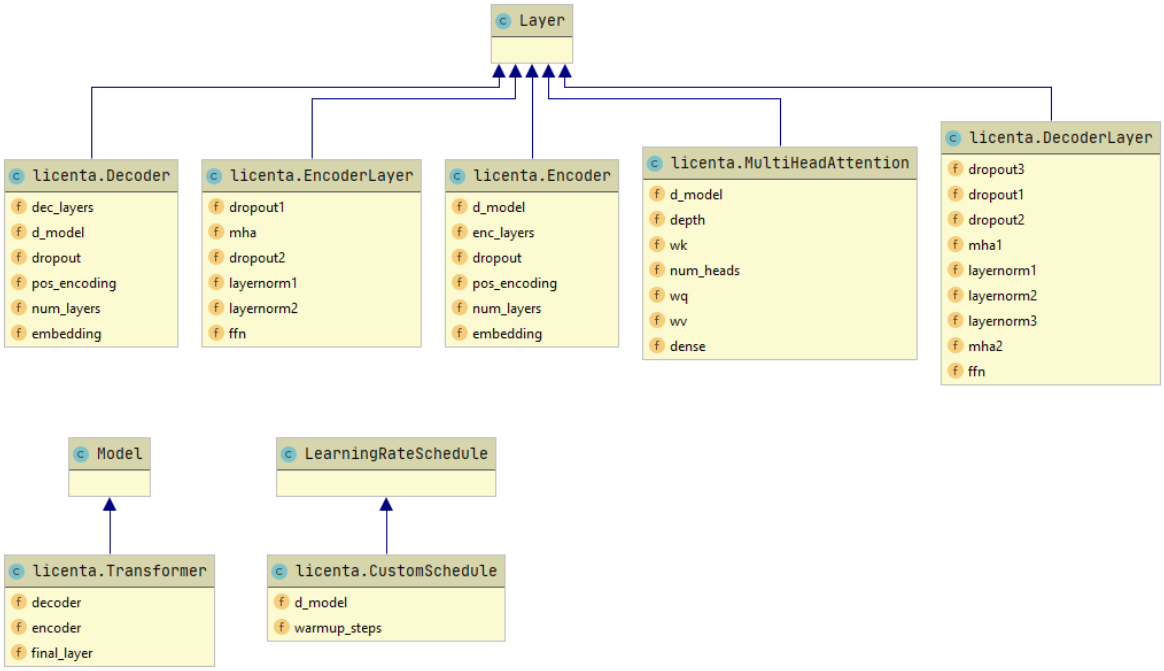

Figure 10 UML Diagram

The class diagram presents the description of the system and the classes that define its components (their attributes and the relationships between them). The diagram helps to realize and understand the structure of the implemented model.

Chapter 5 Implementation Details

If you want to make an apple pie from scratch, you must first create the universe.
– Carl Sagan, Cosmos

The implementation of the solution was done in a Jupyter Notebook[18], using Google Colab[19] for development. The code was written in Python3 using the TensorFlow [16] framework for model development and training and TensorFlow Datasets [17] for data set manipulation. The NumPy library [18] was also used for mathematical calculations and operations, in addition to the operations provided by TensorFlow, and the Matplotlib library was used for graphical representations.

5.1 Configuring the Input Pipeline

To solve the problem of correcting sentences it is necessary to configure the way the data is read, but also the way the data is manipulated and interact with each other.

The dataset consists of several TSV Tab-Separated Values files (not CSV Comma-Separated Values), each line in the file containing a pair of shapes (*correct sentence, wrong sentence*), where the *correct sentence* represents the expected reference or result in following the correction of the *wrong sentence*. The data sets, although divided into several files, are concatenated during running and are organized into three categories:

- Training/learning data sets;
- Validation data sets;
- Test data sets.

The loading of a data set is done using tf.data.TextLineDataset, then a map operation is applied to each line to divide the line with the pair in the two sentences, using tf.data.TextLineDataset with the separator '\t'. At the upload step, the Python os[20] the module is also used to find the path to the data set file. After loading the data sets, two tokenizers are used (one for the wrong sentences and the other for the correct sentences) using tfds.features.text.SubwordTextEncoder. Text encoders tfds.features.text.TextEncoder of this type are constructed from a corpus of sentences from which sub-words will be generated using the tfds.features.text.SubwordTextEncoder.build_from_corpus method. The result is a mapping of the words identified in the text to integers. These numbers can be used to encode words into numbers or to decode numbers into words, respectively.

To be able to train, validate and test the model all data sets must be encoded using the two tokenizers (to encode both the wrong sentences and the correct sentences). Besides, each sentence will have two special tokens, a start token (*sos*: Start Of String) and an end token

[18] Jupyter Notebook. Available: https://jupyter.org. [Accessed: 1 October 2019].
[19] Google Colab. Available: https://colab.research.google.com. [Accessed: 24 February 2020].
[20] Miscellaneous operating system interfaces. Available: https://docs.python.org/3/library/os.html. [Accessed: 1 June 2020].

(*eos*: End Of String). Each encoder is composed of a vocabulary with a size specified in the construction. As each word has a corresponding numerical identifier in the vocabulary, smaller than the vocabulary size, the values assigned to the two special tokens must be chosen so that they are greater than or equal to the vocabulary size.

Cea mai importantă este cea surprinsă asupra lunii Noiembrie.	
Token	**Sub-word**
194	Ce
3	a
19	mai
1638	important
7	ă
28	este
362	cea
613	sur
616	prin
10	să
983	asupra
1036	luni
8	i
1120	No
124	ie
863	mb
614	rie
1827	.

Table 4 Tokenization Example for Transformer

Words for which there was no vocabulary association were split into sub-words for which there was an association, or new associations were created where they did not exist. The vocabulary was built on data sets. Table 4 Tokenization Example for Transformer exemplifies the tokenization process.

5.1.1 Data Set

The data set used to train, validate and test the model is the RONACC[21] corpus. The initial data set consisted of pairs of lines, as in Table 5 Example Data Set, where the first line was the correct one and the second the wrong one, but it was preprocessed so that the pairs were on the same line (facilitate a load of data easier to achieve) separated using the tab character ('\t' for TSV files – the character ',' could also be used for CSV files but as a punctuation mark it should have been avoided in the text) as in Table 6 Modified Data Set. When reading from the file, the data is interpreted using the '\t' separator and the corresponding data sets of the form *(wrong sentence, correct sentence)* are created. The mistakes were highlighted, along with the correct variants by thickening.

Correct Sentence	Wrong Sentence
Acoperișul din șindrilă nu se mai păstrează, fiind înlocuită cu țiglă în 1936, fapt ce a necesitat sprijinirea acoperișului cu structuri **improvizate**.	Acoperișul din șindrilă nu se mai păstrează, fiind înlocuită cu țiglă în 1936, fapt ce a necesitat sprijinirea acoperișului cu structuri **inprovizate**.
Alte mărci comerciale utilizate vreme îndelungată sunt Löwenbräu, **deținătorii** căreia spun că este folosită din 1383, și Stella Artois din 1366	Alte mărci comerciale utilizate vreme îndelungată sunt Löwenbräu, **deținători** căreia spun că este folosită din 1383, și Stella Artois din 1366.
Cea mai importantă este **cea** surprinsă asupra **lunii** Noiembrie.	Cea mai importantă este **ceea** surprinsă asupra **luni** Noiembrie.

Table 5 Example Data Set

The paste[22] utility was used for the conversion:

```
paste -s -d "\t\n" original_dataset > dataset
```

Concatenated Sentences (tabs underlined)
Acoperișul din șindrilă nu se mai păstrează, fiind înlocuită cu țiglă în 1936, fapt ce a necesitat sprijinirea acoperișului cu structuri **improvizate**._____Acoperișul din șindrilă nu se mai păstrează, fiind înlocuită cu țiglă în 1936, fapt ce a necesitat sprijinirea acoperișului cu structuri **inprovizate**.
Alte mărci comerciale utilizate vreme îndelungată sunt Löwenbräu, **deținătorii** căreia spun că este folosită din 1383, și Stella Artois din 1366. Alte mărci comerciale utilizate vreme îndelungată sunt Löwenbräu, **deținători** căreia spun că este folosită din 1383, și Stella Artois din 1366.
Cea mai importantă este **cea** surprinsă asupra **lunii** Noiembrie._____Cea mai importantă este **ceea** surprinsă asupra **luni** Noiembrie.

Table 6 Modified Data Set

5.1.2 Masking

The masks used are those for filling and those for looking ahead.

[21] Romanian National Audiovisual Council Corpus. Available: https://nextcloud.readerbench.com/index.php/s/9pwymesT5sycxoM. [Accessed: 10 June 2020].
[22] Paste. Available: https://www.freebsd.org/cgi/man.cgi?query=paste. [Accessed: 10 June 2020].

Fill masks ensure that the model will not treat the filling as input, indicating where there are zero values. To create the mask, use tf.math.equal to mark zero positions.

The masks with look-ahead are used to indicate which tokens should not be used. For example, to predict the word i, only words $1, 2, \dots, i-1$ will be used, so the model is not allowed to see the expected output. For this, tf.linalg.band_part is used to create a triangular lower matrix of size $i \times i$ which is then complemented to a triangular upper matrix without the main diagonal.

The function of creating the masks deals with the masks for encoding (filling mask) and decoding (filling mask) used in the second attention block of the decoder to mask the outputs of the encoder, but also a combined mask (filling mask and masking mask with look-ahead) used in the first decoder attention block used both for filling and to mask future input tokens received by the decoder. The first two masks are created relative to the wrong sentence, and the combined one relative to the correct sentence.

5.2 Attention

Attention calculation involves a function that receives a query and a set of key – value pairs and returns a mapping as output [6].

5.2.1 Scaled Dot-Product Attention

Calculation of attention to a set (for simultaneous calculation) of queries of size d_k grouped in a matrix Q with l_k lines and keys of size d_k grouped in a matrix K with l_k lines and values of size d_v grouped in a matrix V with l_k lines [6] is made as follows:

1. The matrices Q and K^T are multiplied using tf.linalg.matmul;
2. The result of the multiplication is scaled by $\sqrt{d_k}$ using tf.math.sqrt;
3. If a mask was sent as a parameter, it is multiplied by a value close to $-\infty$ $(-1e9)$ and then added to the result;
4. Attention weights are calculated: $softmax(\frac{QK^T}{\sqrt{d_k}})$ using tf.nn.softmax;
5. The attention is given by the result of the weights multiplied by the matrix V using tf.linalg.matmul [9].

5.2.2 Multi-Head Attention

To calculate multi-head attention, the query, key, and value must be divided into num_{heads}vectors [9]. Using tf.keras.layers.Dense the weights are calculated for each one, then they are divided into several ends. Having the three matrices divided by several ends, the attention is calculated for each end and the results are concatenated using tf.reshape (the data are kept in a multi-dimensional structure). Finally, to get the output, the attention is passed through a layer like tf.keras.layers.Dense.

5.3 Coding Layers

The coding layers are composed using multi-headed attention layers, addition and normalization and feed-forward networks.

5.3.1 Point-Wise Feed-Forward Network

The Feed-Forward Network is implemented using tf.keras.Sequential with two linear layers tf.keras.layers.Dense with a rectified linear unit activation layer (ReLU) between them. The size of the inner layer is dff, and the input and output have the size d_{model}.

5.3.2 Encoding Layer

Each encoding layer consists of sub-layers of:

- Multi-headed attention (with filling mask);
- Point-wise feed-forward networks [19].

Each of these layers has a residual layer [10] around it followed by a normalization layer [11] of type tf.keras.layers.LayerNormalization. Residual connections help avoid the problem of the missing gradient in deep networks. The result of each sublayer is $LayerNorm(x + Sublayer(x))$ [19].

5.3.3 Decoding Layer

Each decoding layer consists of sub-layers of:

- Multi-headed masked attention (with a face mask and filling mask);
- Multi-headed attention (with filling mask). V (value) and K (key) receive the encoder output as inputs. Q (query) receives the exit from the sub-layer of multi-headed masked attention;
- Point-wise feed-forward networks [19].

Each of these layers has a residual layer [10] around it followed by a normalization layer [11] of type tf.keras.layers.LayerNormalization. Residual connections help avoid the problem of the missing gradient in deep networks. The result of each sublayer is $LayerNorm(x + Sublayer(x))$ [19].

Since Q receives the output of the first attention block of the decoder and K receives the output of the encoder, the attention weights represent the importance given to the input of the decoder based on the output of the encoder. In other words, the decoder predicts the next word, looking at the output of the encoder and performing auto-regression [8] at its output [19].

5.4 Coding

The coding is done by embedding the input/output, its positional encoding and several coding layers. The same number of layers is used for both encoding and decoding.

5.4.1 Positional Encoding

The positional encoding is implemented using np.sin and np.cos for the calculation of trigonometric functions. The calculations are made vectorially: the angle is calculated using np.power for raising to power, then two new vectors are created – for the even positions the sine is calculated, and for the odd positions the cosine is calculated. For each position, a vector of values corresponding to the word indices (vocabulary size) is created as in Figure 11 Positional Encoding.

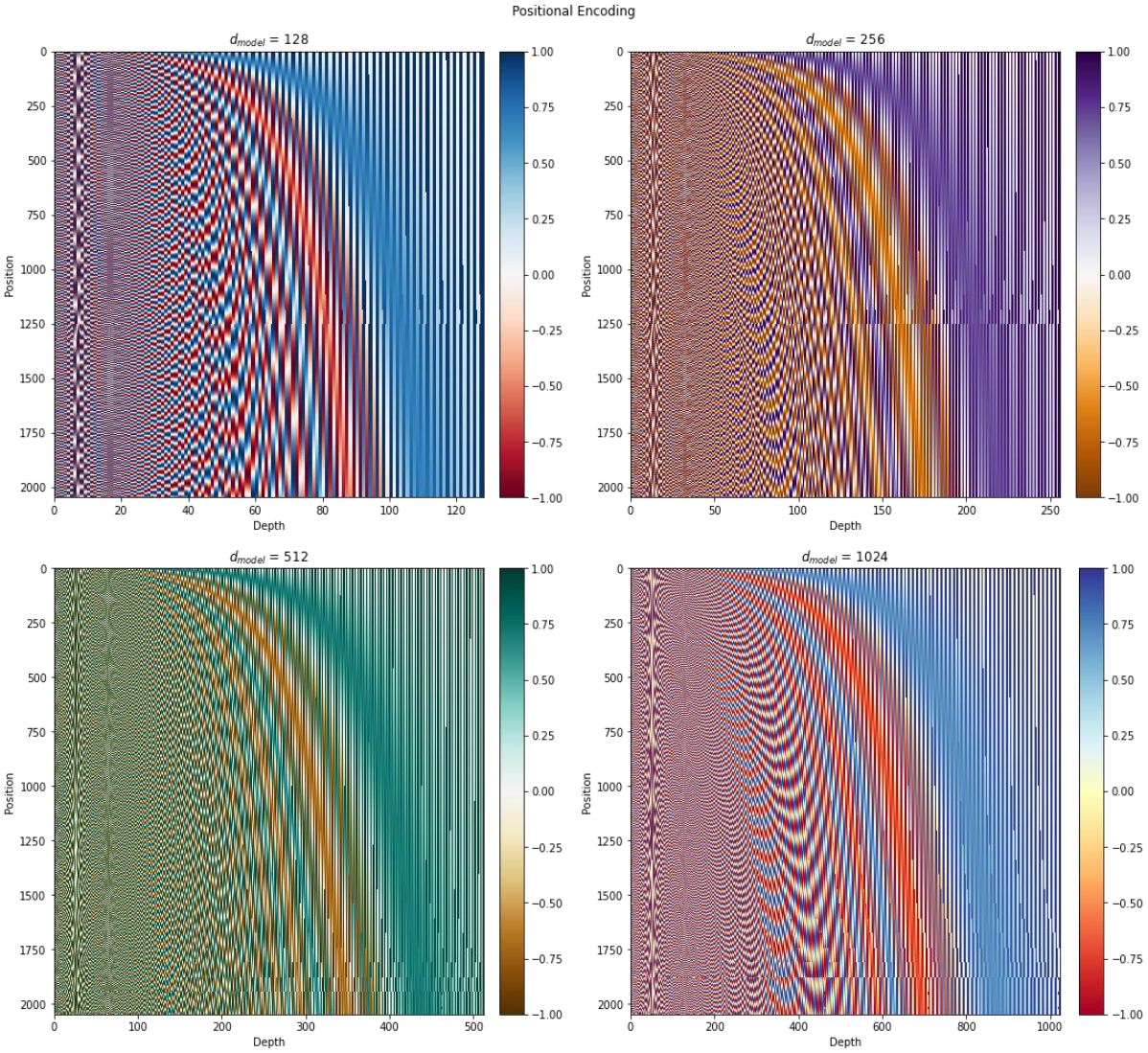

Figure 11 Positional Encoding

5.4.2 Encoder

The encoder consists of:

- Input embedding;
- Positional encoding;
- Encoding layers.

The input is put through an embedded tf.keras.layers.Embedding which is summed with positional encoding. The result of this summation is the input to the encoder layers. The output of the encoder is the input to the decoder [19].

5.4.3 Decoder

The decoder consists of:

- Output embedding;
- Positional encoding;
- Decoding layers.

The target is set by an embedded tf.keras.layers.Embedding and is summed with positional encoding. The result of this summation is the input to the decoder layers. The output of the decoder is the input to the final linear layer [19].

5.5 Transformer

The transformer consists of an encoder, a decoder and a linear final layer the size of the target vocabulary, implemented with tf.keras.layers.Dense.

5.5.1 Hyperparameters

The model uses the following hyperparameters:

- num_{layers}: the number of coding layers;
- d_{model}: the size of the token vectors for input/output;
- dff: the size of the inner layers for the feed-forward network;
- num_{heads}: the number of heads for calculating attention with multiple heads.

5.6 Optimizer

Adam [12] is the optimizer used, available as tf.keras.optimizers.Adam. The learning rate is implemented with tf.keras.optimizers.schedules.LearningRateSchedule:

1. Calculate $\frac{1}{\sqrt{step}}$ using tf.math.rsqrt;
2. Calculate $step \cdot warmup_{steps}{}^{-1.5}$ using Python operations;
3. Choose the minimum of the two values using tf.math.minimum;
4. The result is multiplied by $\frac{1}{\sqrt{d_{model}}}$ calculated with tf.math.rsqrt.

This learning rate is dynamic, increasing at first and then decreasing as learning more after the warm-up steps and depends on the size of the model, as seen in Figure 12 Variation in Learning Rate.

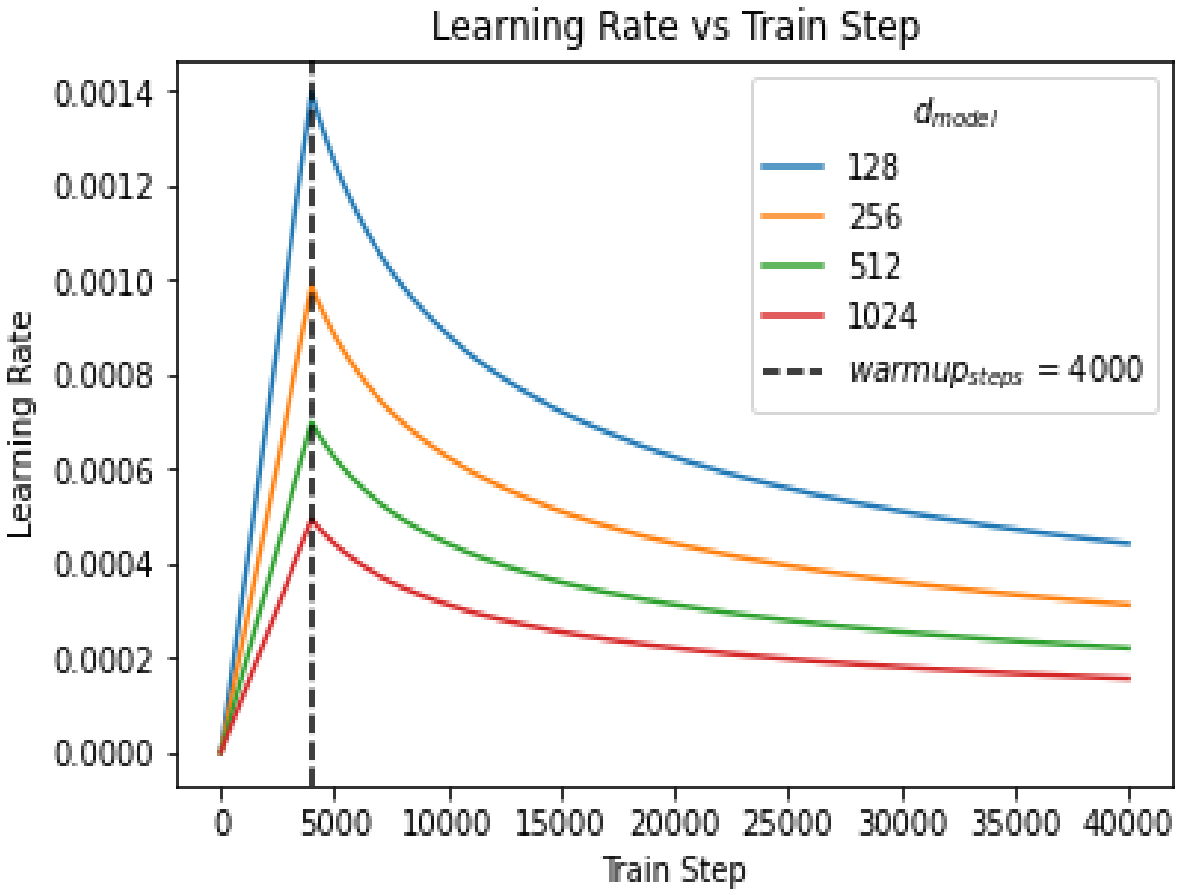

Figure 12 Variation in Learning Rate

5.7 Loss and Metrics

The loss is implemented using tf.keras.losses.SparseCategoricalCrossentropy and ignores, using a mask, the fill values.

Two other metrics are used for training (respectively evaluation): a loss implemented with tf.keras.metrics.Mean and an accuracy using tf.keras.metrics.SparseCategoricalAccuracy.

5.8 Training and Checkpoints

Training is done in several epochs. From time to time, after some epochs, a checkpoint is made to save the training done.

The training step is a function of type tf.function. The parameters of the function are the wrong sentence and the correct sentence. The function @tf.funciton trace-compiles train_step into a TF graph for faster execution. The function specializes in the precise shape of the argument tensors. To avoid retracing due to variable sequence lengths or variable batch sizes (the last batch is smaller), input_signature is used to specify more generic shapes [19].

Given that train_step is executed frequently, instead of the compilation process being executed on demand, at the beginning of each execution of the function (JIT - just-in-time), an optimization is introduced which consists of precompiling the function before execution, to accelerate the running (tracing just-in-time compilation), this based on the observations of the interpreter in connection with the most frequently performed functionalities.[23]

```
train_step_signature = [
                        tf.TensorSpec(shape=(None, None), dtype=tf.int64),
                        tf.TensorSpec(shape=(None, None), dtype=tf.int64),
                       ]

@tf.function(input_signature=train_step_signature)
def train_step(inp, tar):
    tar_inp = tar[:, :-1]
    tar_real = tar[:, 1:]
    # train step code
```

The transformer is an auto-regressive model [8]: it makes predictions one at a time and uses its output so far to decide what to do next [19].

During training, this example uses *teacher-forcing*. *Teacher-forcing* puts the correct output to the next stage, regardless of what the model predicts at the current stage [19]. To achieve this, two sentences are created from the correct sentence:

- The first sentence (the correct input sentence) does not have the last word;
- The second sentence (the correct reference sentence) does not have the first word; thus, for each index in the first sentence, the word to be predicted will be found in the second sentence.

As the transformer predicts each word, attention allows him to look at the previous words in the input sequence to better predict the next word [19].

The masks are then created using the wrong sentence and the correct input sentence, the predictions are obtained using the transformer and the loss between the correct reference sentence and the prediction is calculated. The gradients are computed using tf.GradientTape and then the optimizer is applied using tf.keras.optimizers.Optimizer.apply_gradients. It saves training loss and accuracy between the correct reference sentence and prediction.

To avoid overfitting [14], dropout is used with tf.keras.layers.Dropout.

5.8.1 Checkpoints

Checkpoints are files in which training results are saved. The checkpoints are created using tf.train.Checkpoint, and for their management a manager created with

[23] Wikipedia, „Tracing just-in-time compilation," [Online]. Available: https://en.wikipedia.org/wiki/Tracing_just-in-time_compilation. [Accessed: 20 June 2020].

tf.train.CheckpointManager is used that allows the restoration and saving of a checkpoint, but also the limitation of the number of saved control points.

5.8.2 Training

For each epoch, the states of loss and training accuracy are reset. In each era, one batch from the training set is taken one by one and a training step is taken. Loss and accuracy statistics are displayed after a certain number of batches. At the end of an epoch, loss, accuracy and training time are displayed (the time[24] module is used for this).

5.9 Evaluation

The evaluation is performed in a single epoch, one batch from the test set is taken in turn and an evaluation step is taken. Loss and accuracy statistics are displayed after a certain number of batches. At the end of the epoch, the loss, accuracy and evaluation time are displayed.

The assessment step is similar to the training step, except that the gradients are no longer calculated or applied.

5.9.1 Correction

The process of correcting a sentence involves encoding it using the tokenizer for wrong sentences and adding the two special tokens (the start and stop tokens). Using this input, it is expected to build the output using the transformer. To do this, initialize the output with the special stop token for the tokenizer for correct sentences. At each step the masks are created, predictions and weights for attention are obtained using the transformer, the last word in the predictions is chosen (the iteration is stopped if it is the final word) and it is concatenated at the output. If there is a limitation in the length of the output, the iteration stops; the final outputs and weights are returned. Weights are used to graphically represent the distribution of attention on different levels and blocks. The outputs are decoded using the tokenizer for correct sentences to get the prediction – the corrected sentence. Eventually, the reference sentence can also be displayed if it exists and is sent as a parameter.

5.9.2 Testing

For testing, several sentences with mistakes were tried to see the predictions of the model related to the correct variants using the correction function, but also to follow (optionally) the attention in different layers of the model.

[24] Time. Available: https://docs.python.org/3/library/time.html. [Accessed: 2 June 2020].

Chapter 6 Results Evaluation

Insanity is doing the same thing over and over again, but expecting different results.
– Rita Mae Brown, Sudden Death

Being a statistical model, testing the correctness of the system must be distinguished from assessing its correctness. Thus, the system can work correctly in the execution parameters, but still, it can produce a result that is not considered correct. This is due to the nature of artificial intelligence problems where one tries to approximate a solution, which means that the solution obtained is not optimal or necessarily correct. The evaluation part investigates the nature of the results obtained by the system.

6.1 Testing

Unit and integrity tests were used to test system functionality and correctness.

The main interaction of users with the system is achieved through the function of correcting a sentence, which receives as a parameter a wrong sentence and can optionally receive as parameters the correct variant as a reference and the name of a layer to display the weight of attention. To ensure the system on the input data, type tests were performed (the system must receive as input an incorrect sentence and possibly a correct one that are strings and nothing else). For this, assertion operations of type `assert isinstance(sentence, str) == True` were introduced. Next, unit tests were used to ensure the correct execution of the system with different inputs and to ensure that the system will receive correct input data from the user.

To ensure the integrity of the system, integrity tests were performed on the operations performed. For example, to ensure the integrity of multi-headed attention calculations, when matrices are split at multiple ends, the size of the model (corresponding to the input) must be divisible by the number of heads used by the system. This is done using `assert d_model % self.num_heads == 0`.

The tested functionalities of the system have been implemented correctly and completely. It is trained and evaluated on a data set (the data set must be formatted according to the TSV format) and produces concrete results. Even if a corrected sentence is not correct, the system can still work correctly, and the problem should be one of accuracy/precision of the predicted results. An example of a correct run in which the result is not as expected:

```
correct("Cea mai importantă este ceea surprinsă asupra luni Noiembrie.",
        real_sentence="Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.")
Input: Cea mai importantă este ceea surprinsă asupra luni Noiembrie.
Predicted correction: Cel mai importantă este cea surprinsă asupra lunii
Noiembrie.
Real correction: Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.
BLEU score: 0.8633
```

This situation is specific to machine learning problems and should be addressed in terms of system evaluation, not correctness testing.

To avoid possible errors and defects in the source code, code style conventions have been observed, such as alignment, lines shorter than 80 characters, modulation, functions and methods with a clear role and meaningful variable names.

6.2 Evaluation

The model was evaluated on several sets of training data and a test set with 1519 pairs of sentences: small: 7082 pairs of sentences; medium: 7082 + 50000 pairs of sentences; large: 7082 + 1000000 pairs of sentences.

In the end the results were validated using a validation set of 1518 pairs of sentences. The model was trained for 100 epochs on the small and medium data set, at each epoch being tested and measuring accuracy (Figure 13 Accuracy for the small data set and Figure 15 Accuracy for the medium data set) and loss (Figure 14 Loss for the small data set and Figure 16 Loss for the medium data set). From the graphs, it can be seen how these values converge, and also, the loss tends to zero.

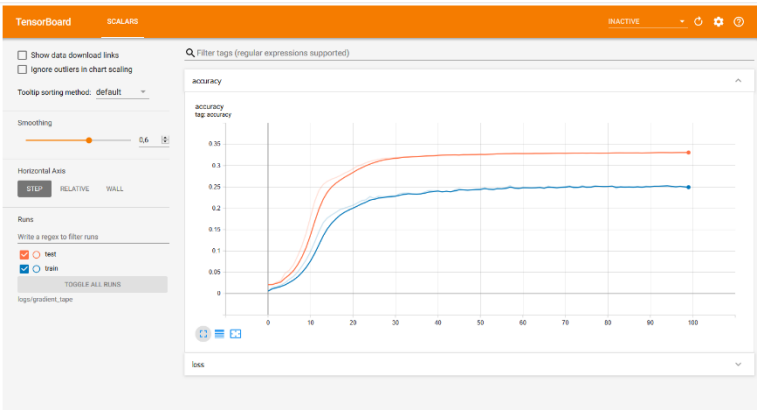

Figure 13 Accuracy for the small data set

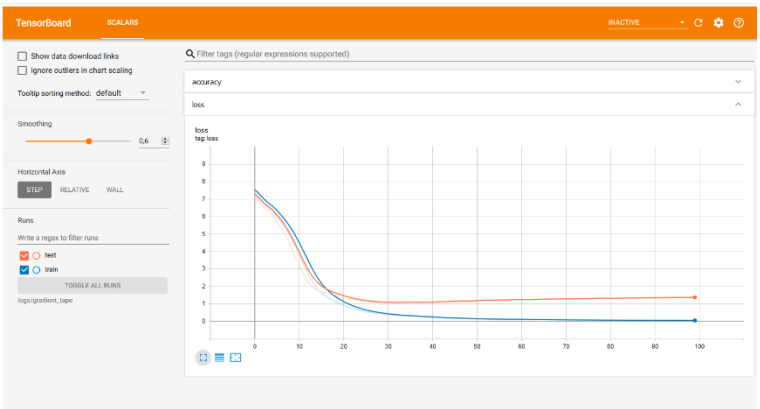

Figure 14 Loss for the small data set

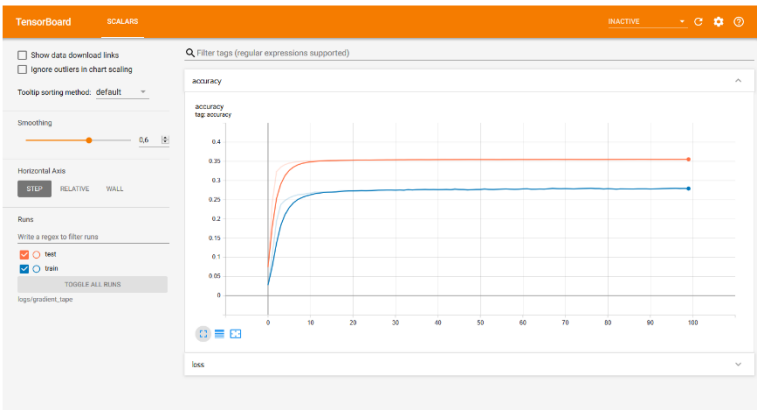

Figure 15 Accuracy for the medium data set

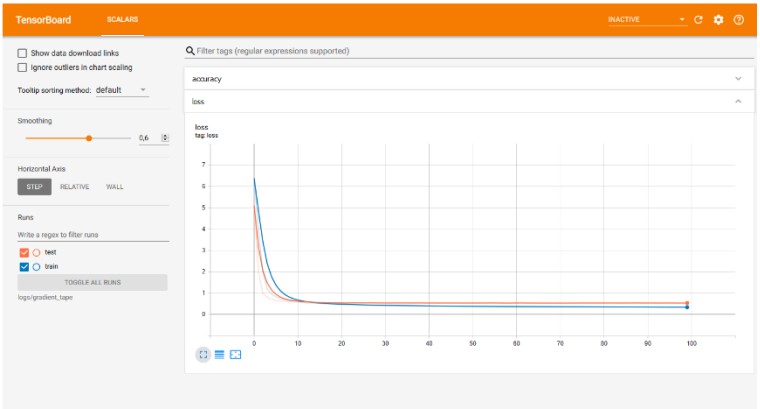

Figure 16 Loss for the medium data set

The model proposed in [5] obtains a BLEU score of 27.3, respectively 38.1 for the translation from English to German, respectively French for the Transformer with the base model and a BLEU score of 28.4, respectively 41.8 for the big Transformer, while the model implemented in the work present obtain a competitive BLEU score of 20.56 for the correction of texts in Romanian with a modest training set and a score of 33.29 with a larger training set, as can be seen in Table 7 Training results, which represents a substantial improvement. The data cannot be compared directly, as the article [5] used other data sets, worked on other issues and in other languages. However, it should be borne in mind that the field is currently quite limited for correcting texts in Romanian to make comparisons on other systems. In the table, it can be seen how a larger set of training data can substantially influence the score (45.29).

Data set	Epochs	BLEU Score	Training time
Small	100	20.56	9.66 seconds / epoch
Medium	100	33.29	99 seconds / epoch
Large	5	**45.29**	1032 seconds / epoch

Table 7 Training results

The calculation of the BLEU score [15] was implemented using nltk.translate.bleu_score.sentence_bleu and nltk.translate.bleu_score.SmoothingFunction with method4 for the case of short sentences [20]. Another important metric is the attention that words pay to other words, as can be seen in the examples in Appendix A (Attention Weights).

6.2.1 Qualitative Evaluation

In Appendix A (Attention Weights), some examples of running the model on different input sentences are presented. If the quantitative results obtained were on average satisfactory, in terms of the BLEU score, for example, for which improvements were observed, qualitatively things are slightly different due to the particularities of each type of grammatical error.

An example that the model can easily correct is the case of what - that, as is the case with the sentence *„Cea mai importantă este ceea surprinsă asupra luni Noiembrie.”* for which the system produces the prediction *„Cea mai importantă este cea surprinsă asupra lunii Noiembrie.*" which is also the correct variant expected at the output. Instead, the case **n-am – nam (lack of hyphen)** it is not as well learned by the model, for example *„„Nici odată nam văzut cartea așa” a mărturisit el.*" for which the model does not reach the correct variant *„„Niciodată n-am văzut cartea așa”, a mărturisit el.”* being a more complicated case because one lexical part is misspelled (by nam instead of n-am) and another is misspelled by parting (nici odată instead of niciodată). The problem of correction also comes from the intersection of the two cases.

Another example for which the model does not generalize as well as that of words it does not know (they do not appear in the output dictionary) and does not know how to make associations. For example, in the case of misspelled words, if they have not been learned they will not be able to be corrected. For the sentence *„În prezent, satul are 3.897 locuitori, prepoderent ucraineni.*" the model will not produce the expected output *„În prezent, satul are 3.897 locuitori, preponderent ucraineni.*", because the model does not know the association **prepoderent – preponderent** and does not know that there is a mistake of this type. For this example, the BLEU score is misleading, somewhere around 70%, which may seem to be a fairly fair result.

Such cases must be considered when evaluating the model. Possible solutions to such problems are to increase the number of training epochs and to use training sets (with the disadvantage of increased training time) with an extensive number of learning examples to cover different cases and mistakes frequently encountered in the language.

However, it should be noted, as can be seen in Appendix A (Attention Weights), that for a sentence that is spelled correctly the model will most likely not try to change it, as the words are found in the dictionary and the links between them produce large weights of attention.

Chapter 7 Conclusions

Finis coronat opus. (The end crowns the work.)
– Ovid, Heroides

The result of this project is a text corrector with reduced functionality. In its realization, the needs of the users were taken into account, as well as the scientific and technical progress in the field, but also the technologies available for the implementation of the Transformer.

Compared to the model proposed in [5] which was trained twelve hours on eight P100 GPUs, the implemented model was trained, on the medium data set for almost three hours, and on the large set for an hour and a half (only five epochs). Only one super dataset was used for modeling and no comparisons could be made with other datasets and also, unfortunately, the development in the field did not make it possible to compare the implemented model with other models. Furthermore, the implementation is a starting point in the development of the field and can be used as a starting point in the development and modeling of high-performance and competitive technologies for correcting texts in Romanian. The generalization of the model is not as expected, but there are cases in which it has managed to accurately predict a correct sentence starting from a sentence that had grammatical errors (see Appendix A Attention Weights).

7.1 Further Developments

The development and improvement of the model (or at least of the field of text correction in Romanian) are urgently needed, and this can be done using more training data, a larger number of training epochs or an extensive set of validation data. As technology evolves, the model needs to be updated and improved with the latest technologies. An example of such technology is BERT (Bidirectional Transformer Encoder Representations) [21].

In addition to improving the model, further application developments require the use of an interface for user-program interaction. The application must be able to be used from different operating systems, such as Android[25], iOS[26], macOS[27], Linux[28] or Microsoft Windows[29]. Also, an accessible version can be developed for web browsers like Mozilla Firefox[30] or Google Chrome[31]. Another important step is to integrate the software with other word processing applications such as those in the Microsoft Office[32] suite.

The implementation code is available at the web addresses in Appendix B (Source Code).

[25] Android. Available: https://www.android.com. [Accessed: 20 June 2020].
[26] iOS. Available: https://www.apple.com/ios. [Accessed: 20 June 2020].
[27] macOS. Available: https://www.apple.com/macos. [Accessed: 20 June 2020].
[28] Linux. Available: https://www.linuxfoundation.org. [Accessed: 20 June 2020].
[29] Microsoft Windows. Available: https://www.microsoft.com/windows. [Accessed: 20 June 2020].
[30] Mozilla Firefox. Available: https://www.mozilla.org/firefox/new. [Accessed: 21 June 2020].
[31] Google Chrome. Available: https://www.google.com/chrome. [Accessed: 21 June 2020].
[32] Microsoft Office. Available: https://www.microsoft.com/microsoft-365. [Accessed: 22 June 2020].

Bibliography

[1] I. Sutskever, O. Vinyals and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," 10 September 2014. [Online]. Available: https://arxiv.org/abs/1409.3215. [Accessed 20 June 2020].

[2] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," 3 June 2014. [Online]. Available: https://arxiv.org/abs/1406.1078. [Accessed 20 June 2020].

[3] D. Bahdanau, K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," 1 September 2014. [Online]. Available: https://arxiv.org/abs/1409.0473. [Accessed 20 June 2020].

[4] J. Gehring, M. Auli, D. Grangier, D. Yarats and Y. N. Dauphin, "Convolutional Sequence to Sequence Learning," 8 May 2017. [Online]. Available: https://arxiv.org/abs/1705.03122. [Accessed 21 June 2020].

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention Is All You Need," 12 June 2017. [Online]. Available: https://arxiv.org/abs/1706.03762. [Accessed 10 June 2020].

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention Is All You Need," in *Neural Information Processing Systems (NIPS)*, Long Beach, CA, USA, 2017.

[7] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes and J. Dean, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," 26 September 2016. [Online]. Available: https://arxiv.org/abs/1609.08144. [Accessed 20 June 2020].

[8] A. Graves, "Generating Sequences With Recurrent Neural Networks," 4 August 2013. [Online]. Available: https://arxiv.org/abs/1308.0850. [Accessed 20 June 2020].

[9] M. Phi, "Illustrated Guide to Transformers: Step by Step Explanation," 30 April 2020. [Online]. Available: https://www.michaelphi.com/illustrated-guide-to-transformers. [Accessed 19 June 2020].

[10] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 10 December 2015. [Online]. Available: https://arxiv.org/abs/1512.03385. [Accessed 21 June 2020].

[11] J. Lei Ba, J. R. Kiros and E. G. Hinton, "Layer Normalization," 21 July 2016. [Online]. Available: https://arxiv.org/abs/1607.06450. [Accessed 21 June 2020].

[12] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 22 December 2014. [Online]. Available: https://arxiv.org/abs/1512.03385. [Accessed 6 June 2020].

[13] J. Alammar, "The Illustrated Transformer," 27 June 2018. [Online]. Available: http://jalammar.github.io/illustrated-transformer. [Accessed 20 June 2020].

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* vol. I, no. 15, pp. 1929-1958, June 2014.

[15] K. Papineni, S. Roukos, T. Ward and W.-J. Zhu, "Bleu: a Method for Automatic Evaluation of Machine Translation," in *Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA, 2002.

[16] Google Brain Team, "All symbols in TensorFlow 2," TensorFlow, 9 November 2015. [Online]. Available: https://www.tensorflow.org/api_docs/python. [Accessed 16 June 2020].

[17] Google Brain Team, "All symbols in TensorFlow Datasets," TensorFlow, 9 November 2015. [Online]. Available: https://www.tensorflow.org/datasets/api_docs/python. [Accessed 16 June 2020].

[18] NumPy, "NumPy Reference," NumPy, 24 May 2020. [Online]. Available: https://numpy.org/doc/stable/reference. [Accessed 16 June 2020].

[19] TensorFlow, "Transformer model for language understanding," 22 August 2019. [Online]. Available: https://www.tensorflow.org/tutorials/text/transformer. [Accessed 10 June 2020].

[20] Team NLTK, "NLTK documentation," Natural Language Toolkit, 2001. [Online]. Available: https://www.nltk.org/. [Accessed 21 June 2020].

[21] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 11 October 2018. [Online]. Available: https://arxiv.org/abs/1810.04805. [Accessed 23 June 2020].

Appendix A Attention Weights

The following examples were obtained after training the transformer on the medium data set. Attention was measured at the output (second substrate) of the last decoder layer of the decoder. The examples presented are also fully corrected cases, partially corrected sentences, but also sentences with errors or that could not be corrected.

```
correct("Cea mai importantă este ceea surprinsă asupra luni Noiembrie.",
        real_sentence="Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.",
        plot="decoder_layer2_block2")
Input: Cea mai importantă este ceea surprinsă asupra luni Noiembrie.
Predicted correction: Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.
Real correction: Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.
BLEU score: 1.0000
```

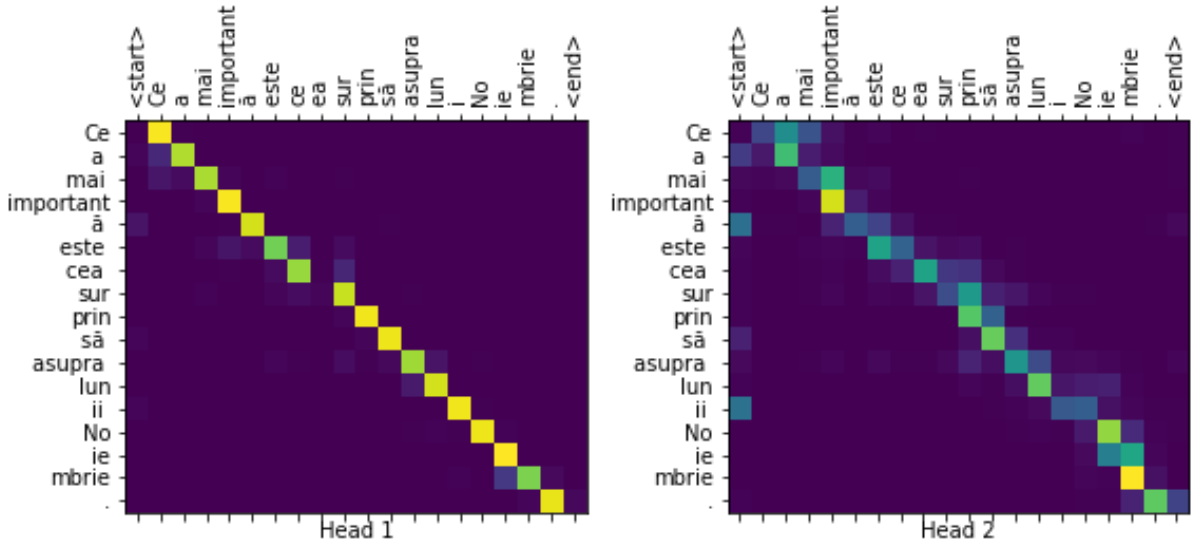

Figure 17 Example 1: Attention

```
correct("„Nici odată nam văzut cartea așa” a mărturisit el.",
        real_sentence="„Niciodată n-am văzut cartea așa”, a mărturisit
el.",
        plot="decoder_layer2_block2")
Input: „Nici odată nam văzut cartea așa” a mărturisit el.
Predicted correction: „Nici odată nam văzut cartea așa” a mărturisit el.
Real correction: „Niciodată n-am văzut cartea așa”, a mărturisit el.
BLEU score: 0.2741
```

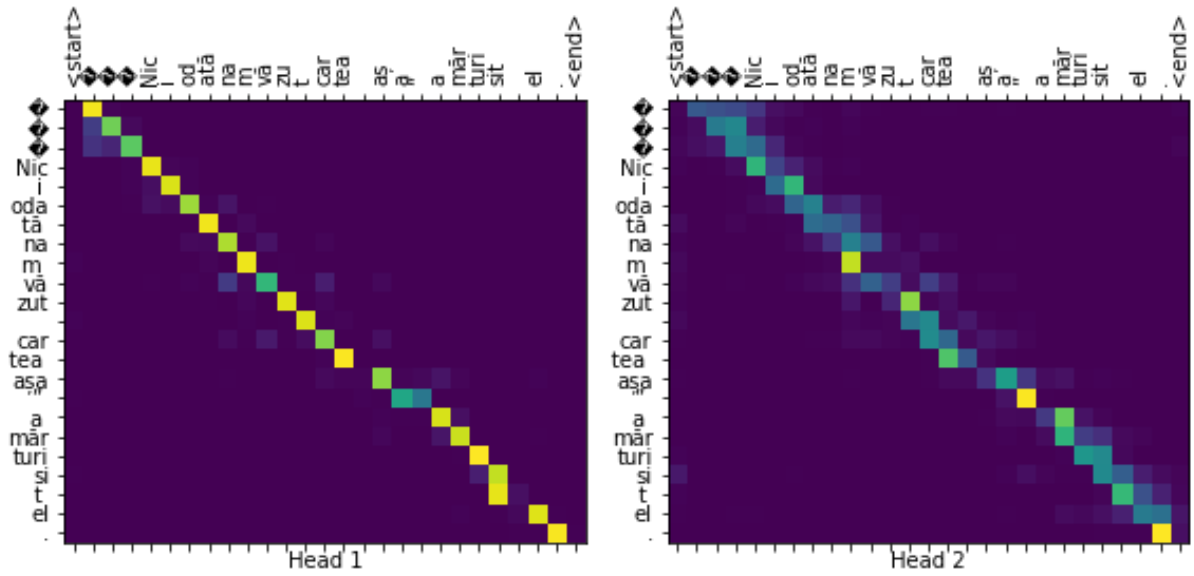

Figure 18 Example 2: Attention

```
correct("În prezent, satul are 3.897 locuitori, prepoderent ucraineni.",
        real_sentence="În prezent, satul are 3.897 locuitori, preponderent
ucraineni.",
        plot="decoder_layer2_block2")
Input: În prezent, satul are 3.897 locuitori, prepoderent ucraineni.
Predicted correction: În prezent, satul are 3.897 locuitori, prepoderent,
ucraineni.
Real correction: În prezent, satul are 3.897 locuitori, preponderent
ucraineni.
BLEU score: 0.7071
```

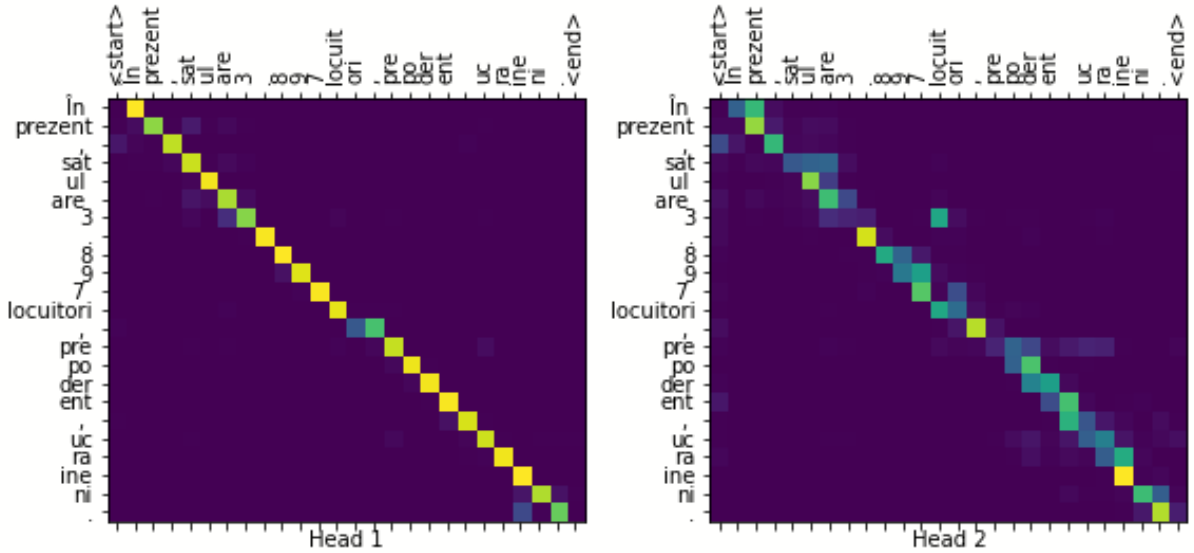

Figure 19 Example 3: Attention

Appendix B Source Code

The source code used in the development of the project, along with other documents, is available at the following addresses:

- https://gitlab.cs.pub.ro/ioan_florin.nitu/l4-diploma-project
- https://github.com/nitu-catalin1998/diploma-project

The two repositories contain the same data (duplicates to have a backup). Their use has facilitated version control and tracking of changes during software development. In addition to controlling source versions, the repository contains important resources for project development: datasets, model configurations, graphs and figures, references to articles and tutorials used in project development. The code is written in Python and two usable versions are available. One version is a Jupyter Notebook that can be run using Google Colab, and the other version is the simple Python code obtained by converting the notebook.