Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Master's Thesis

Development of rephrase system using Natural Language Processing models

Ivan Tsvietkov

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Ivan Tsvietkov

Systems Engineering and Informatics

Informatics

Thesis title

Development of rephrase system using Nature Language Processing models

---

Objectives of thesis

The main objective of this thesis is to describe and explain process of development of a rephrase system using various natural learning processing, machine learning and deep learning libraries in Python. The second and final objectives is to deploy the rephrase system which performs substitution of words on a given text without losing context.

Methodology

The methodology of the thesis is based on analysis of technical and scientific sources focusing on artificial intelligence, machine learning, deep learning and natural language processing. Based on the synthesis of the knowledge gained, a prototype application will be implemented to rephrase text. The application will be written in Python using libraries for data science as NLTK, Pandas, NumPy and many others. The application will be tested and assessed in terms of performance and replacement quality.

The proposed extent of the thesis

60-80 pages

Keywords

Deep learning, NLTK, Natural language processing, Machine learning, TensorFlow, TextBlob, Pytorch

Recommended information sources

BENGFORT, Benjamin, Rebecca BILBRO a Tony OJEDA. Applied text analysis with Python: enabling language-aware data products with machine learning. 3rd ed. Sebastopol, CA: O'Reilly Media, 2018. ISBN 978-1491963043.

RAO, Delip and Brian MCMAHAN, 2019. Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning. O'Reilly Media. ISBN 978-0321898388.

RUSSELL, Stuart J. and Peter NORVIG and Ernest DAVIS. Artificial intelligence: a modern approach. 4th ed. Upper Saddle River: Prentice Hall, 2020. ISBN 1292401133

Expected date of thesis defence

2021/22 WS – FEM

The Diploma Thesis Supervisor

Ing. Petr Hanzlík, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 1. 3. 2022

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 7. 3. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 14. 04. 2022

Declaration

I declare that I have worked on my master's thesis titled "Development of rephrase system using Nature Language Processing models" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the master's thesis, I declare that the thesis does not break any copyrights.

In Prague on 11.04.2022 _____

Acknowledgement

I would like to thank my thesis supervisor Ing. Petr Hanzlík, Ph.D. for his advice, support and time which helped me a lot with writing this thesis, and my good friend Bc. Illia Prazdnyk for motivation.

Development of rephrase system using Natural Language Processing models

Abstract

The thesis is focused on development of rephrase system using Nature Language Processing models in Python. The theoretical part starts with origins Artificial Intelligence field. After this it shortly defines Machine learning and Deep learning, and describes what is it neural network, how it works, why neurons need activation function and types of it. The theoretical part goes on to describe main NLP techniques, types of neural networks which are used to process human language like RNN and Transformers and ends with brief description of tools for NLP engineer.

The practical part shows to ways of creating rephase system. The first way shows how to change sentence using synonyms with libraries like NLTK and spaCy. The second way shows more sophisticated way, it takes already pre-trained model of transformer and additionally trains it on specific corpus created for rephase tasks.

Keywords: Deep learning, NLTK, Natural language processing, Machine learning, TensorFlow, TextBlob, Pytorch

Vývoj systému přeformulování textu používají modely zpracování přirozeného jazyka

Abstrakt

Tato diplomová práce je zaměřená na vývoj systému přeformulování textu používají modely zpracování přirozeného jazyka v Pythonu. Teoretická část se začíná s popsaní vytvoření oboru Umělá inteligence. Pak stručně charakterizuje strojové a hluboké učení, a přechází o popsaní co je neuronové sítě, jak to funguje, proč neurony potřebují aktivační funkce a jaké jsou její druhy. Pak teoretická část pokračují v popsaní základních NLP metod, druhů neuronových sítí, který se používají při zpracovaní přirozeného jazyka jako RNN a Transformers a se ukončuje definicemi nástroji pro NLP vývojáře.

Praktická část vysvětluje dvě cesty vývoje systému přeformulování. První cesta ukazuje, jak je možné změnit větu používají synonyma a knihovny NLTK a spaCy. Druhá část směruje na vice sofistikovaný přistup a ukazuje, jak vzít natrénovaný model transformeru a dodatečně natrénovat pro specifický úkol přeformulování textu.

Klíčová slova: Hluboké učení, NLTK, zpracování přirozeného jazyka, strojové učení, TensorFlow, TextBlob, Pytorch

Table of content

List of pictures

List of abbreviations

NLP – Nature Language Processing

AI – Artificial Intelligence

POS – Part-of-speech

RNN – Recurrent Neural Network

1 Introduction

The modern era is the era of computers and ubiquitous digitalization. Thanks to them, we were able to solve many applied problems that would otherwise have taken hundreds of years to solve. Thanks to this opportunity, people started to change the world with help of computers. Nowadays we basically can't live without computer technology and our dependence on this technology will grow even more in the future.

The development of computer technology had led us to the creation of artificial intelligence. Thanks to it, we can automate almost any process that requires the presence of a person, such as systems used for evaluation and control. Since the field of Artificial Intelligence today is quite large and includes many sub-sciences, this thesis will focus only on a selected subset - the human language processing, i.e., natural language processing.

2 Objectives and Methodology

2.1 Objectives

The main objective of this thesis is to describe and explain the process of development of a rephrase system using various natural learning processing, machine learning and deep learning libraries in Python. The second and final objectives is to deploy the rephrase system which performs substitution of words on a given text without losing context.

2.2 Methodology

The methodology of the thesis is based on analysis of technical and scientific sources focusing on artificial intelligence, machine learning, deep learning and natural language processing. Based on the synthesis of the knowledge gained, a prototype application will be implemented to rephrase text. The application will be written in Python using libraries for data science as NLTK, Pandas, NumPy and many others. The application will be tested and assessed in terms of performance and replacement quality.

3 Literature Review

3.1 Artificial Intelligence

For the last two decades, many systems, services, and functions that use AI technology have been developed: recommending a cafe based on individual preferences, building the best route, deciphering medical data – the list is almost endless. All of this has been made possible thanks to the efforts of scientists from around the world. Following sections of this thesis will therefore focus on origins of Artificial Intelligence and its fundamental concepts, without which current achievements would not be possible.

3.1.1 Creation of a field of artificial intelligence

In 1936, Alan Turing, now considered the founder of AI theory and the American mathematician Alonzo Church, independently of each other made the claim that there is no algorithm that decides whether a given statement is deducible from some set of mathematical axioms, now known as the "Church-Turing thesis". (Penrose, 1989)

In "On Computable Numbers, with an Application to the Entscheidungsproblem", Turing developed the notion of an abstract digital computing machine, later called a Turing machine, capable of simulating (with an appropriate program) any machine whose actions consist of of moving from one discrete state to another, to solve a given task. (Penrose, 1989)

In his next paper "Computing Machinery and Intelligence" published in the journal Mind in 1950, Turing set forth a mental experiment (later called the Turing test) in which two players, talk to a third player separated from them by a wall through a channel that excludes voice. The aim of third player is to determine, by indirect questions, the sex of each of players he is talking to. Turing changed rules of the game -he assumed what if a machine took part instead of one player and now the task for the third player will be to determine who is machine and who is not. The question what arising simultaneously with such change of the rules – "Can a machine think?" or more specifically, "Can machines do what we (as thinking entities) can do? (Penrose, 1989)

The next and final step in creation of the new field was The Dartmouth Seminar. It was a conference on Artificial Intelligence held at Dartmouth College in the summer of 1956. The

conference was important for the new-founded science: it introduced scientists interested in modelling the human mind to each other, approved the emergence of a new field of science, and gave it a name – Artificial Intelligence. (Stuart, et al., 2002)

3.1.2   Subsets of Artificial Intelligence

With time, two subfields of Artificial Intelligence were created:

- Machine Learning
- Deep Learning
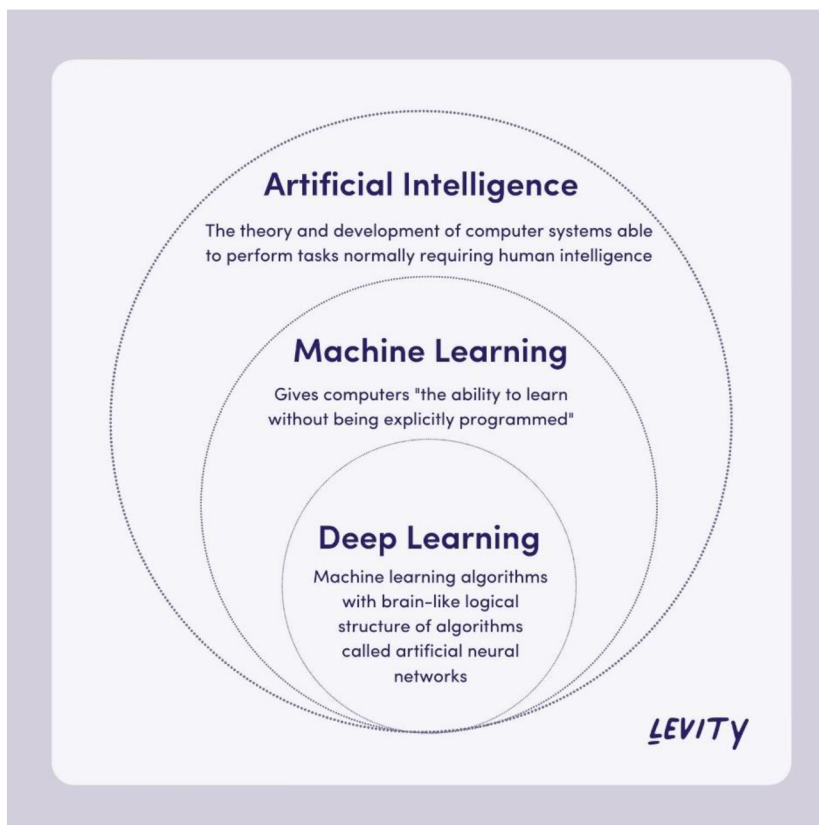
Figure 1 - Subsets of Artificial Intelligence (Wolfewicz, 2021)

3.1.3   Machine learning

Machine Learning refers to a variety of mathematical, statistical, and computational methods for developing algorithms that can solve a problem not by a direct way, but by searching for patterns in input data. The solution is calculated not by an explicit formula, but by an established dependence of the results on a particular set of features and their values.

3.1.4   Deep learning

Deep learning is a machine learning technique that involves independently building (training) general rules in the form of an artificial neural network, which trains on examples of data during the learning process.

3.1.5   Neural networks

When scientists started thinking about Artificial Intelligence, the first thing they did was to study the nature around them. One of the main components of any system in nature is a neural network. They are found everywhere. Their main function is to control various parts of the body in response to changing environmental conditions. As an example, we can consider the mechanism of pupil contraction and dilation depending on the level of light.

Our eye has sensors that pick up the amount of light entering through the pupil at the back of the eye. They convert this information into electrical impulses and transmit them to the attached nerve endings. This signal then travels through a network of neurons which decide whether this amount of light is harmful to the eye, whether it is sufficient for the eye to clearly recognize visual information, and whether, based on these factors, the amount of light needs to be reduced or increased.

At the output of this network are the muscles responsible for pupil dilation or constriction, and they actuate these mechanisms depending on the signal received from the neural network. And there are a huge number of such mechanisms in the body of any living being with a nervous system.

Scientist tried to replicate this system using mathematics and mathematical functions.
So, we have an input layer of neurons, which, in fact, are the sensors of our system. They are needed to get information from the environment and pass it further to the neural network. We also have several layers of neurons, each of which receives information from all the neurons in the previous layer, processes it in some way, and passes it on to the next layer. Finally, we have output neurons. Based on signals coming from them, we can judge the decision made by the neural network. Simple version of a neuron is called a perceptron.
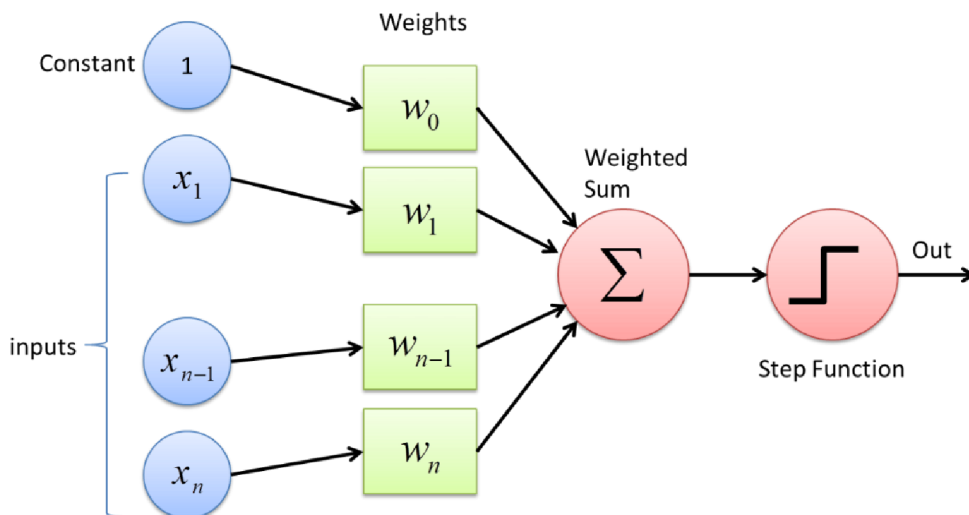
Figure 2 - Perceptron model (Sharma, 2017)

All neurons are essentially the same and decide how strong the signal to transmit next using the same algorithm. This algorithm is called an activation function . At the input, it receives the sum of the values of the input signals, and at the output it transmits the value of the output signal.

But in this case, it turns out that all neurons of any layer will receive the same signal and give the same value. Thus, we could replace our entire network with one neuron. To eliminate this problem, we will assign a certain weight to the input of each neuron. This weight will indicate how important the signal received from another neuron is for each particular neuron.

To make the neural network work correctly, we must first train it. Neural network training is the process of selecting input weights for each neuron in such a way that the output signal is as close as possible to expectations.

One of possible approach is to assign the random weights to the feature inputs, after this feed datato the input of the neural network, for which the result are known. Next, the result that the neural network gives is compared with the expected result, the error is calculated, and the weights of the neurons adjusted in a way which minimizes this error. This action is repeated for a needed number of times for a large number of input and output data, so that the network understands which signals on which neuron are more important to it, and which ones are less important. This approach is called supervised learning. If the problem is well represented by dataset – the bigger that dataset is – the better. If the neural network would encounter more observations – it could learn more information.

The structure of neural network generally consists of 3 types of layers:

- Input layer – accepts data inputs
- Hidden layers – perform mathematical calculations with input data
- Output layer – gives output
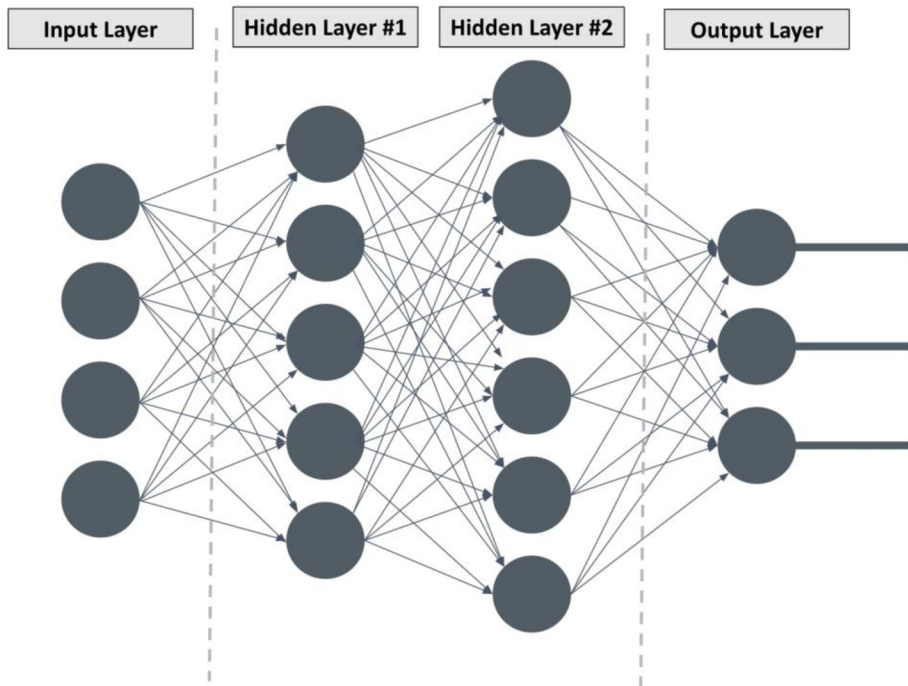
Figure 3 - Simple Neural Network (Wolfewicz, 2021)

Neurons in neural networks differ from each other by activation function.

3.1.6 Activation functions

Today exist many different activation functions. The activation function of a neuron can be any function that exists on the entire range of values produced by the output of the neuron and the input data.

The first one is sigmoid function.

$$\phi(z) = \frac{1}{1 + e^{-z}}$$
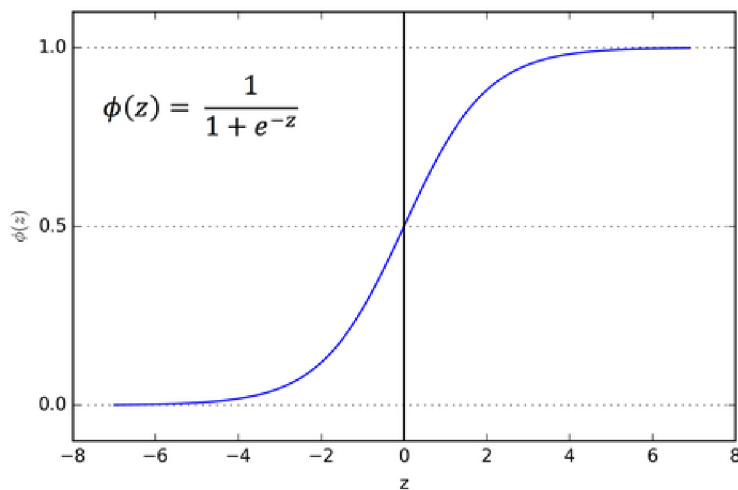
Figure 4 - Sigmoid function (Sharma, 2017)

This activation function works well only in binary classification and it is practically not used in practice nowadays.

The second one is an improved version of sigmoid function – Tanh or hyperbolic tangent activation function. This function is mainly used in classification tasks with only two possible classes.
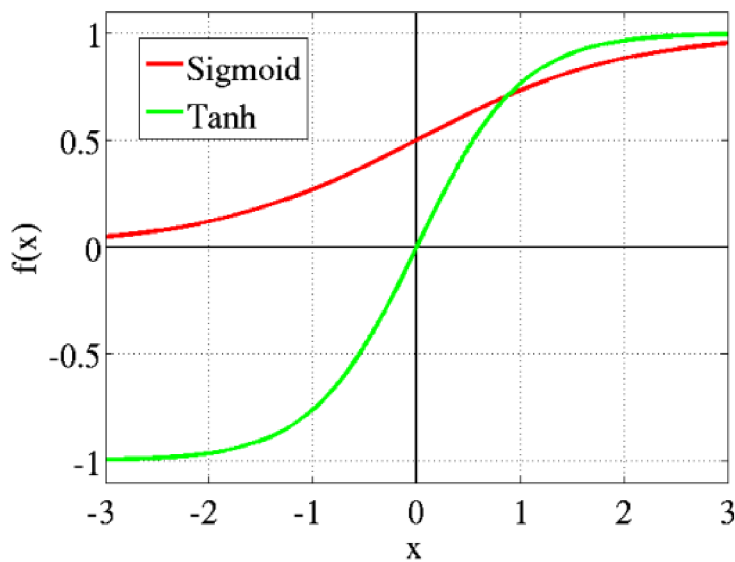
Figure 5 - Tahn activation function compared to the sigmoid function (Sharma, 2017)

The next one is ReLU activation function. This function is very popular today and used in deep learning.
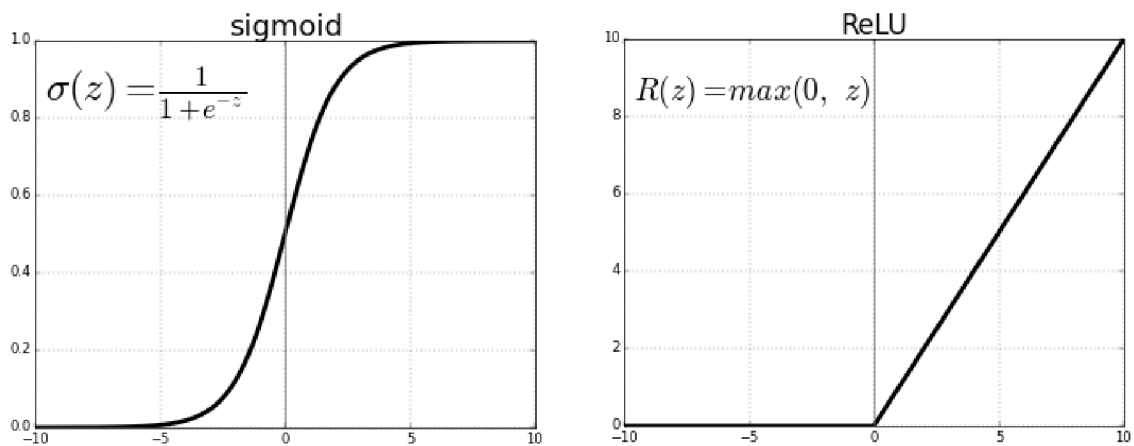
Figure 6 - ReLU function compared to Sigmoid (Sharma, 2017)

The ReLU activation function has one problem which known as a dying neuron or a dead neuron problem. I the input to a ReLU neuron is negative, the output would be zero. (Lu, et al., 2019)

To deal with this problem, the Leaky ReLU activation function was developed.
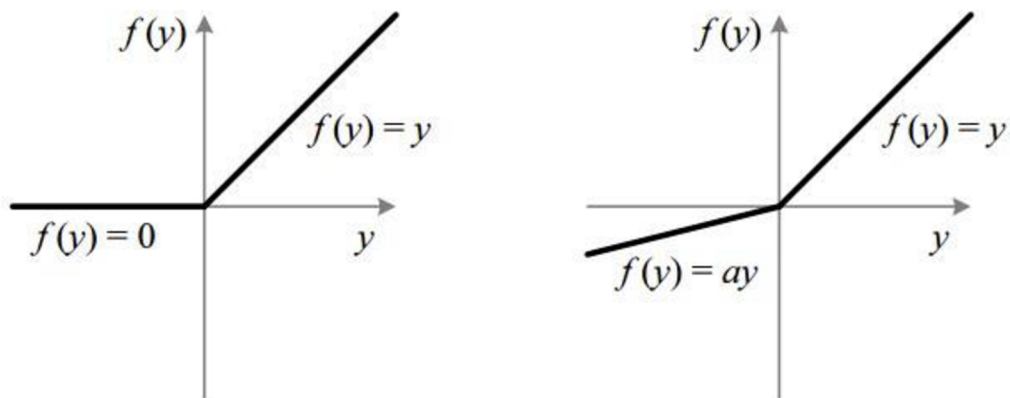
Figure 7 - Leaky ReLU (right) compared to ReLU (Sharma, 2017)

3.2   Natural Language Processing

To create a program which can understand human language and obtain meaning from a text is needed to use Natural Language Processing. Therefore the next part of this thesis will define what is NLP and describe the main techniques which it utilizes.

"NLP refers to a set of techniques involving the application of statistical methods, with or without insights from linguistics, to understand text for the sake of solving real-world

21

tasks. This "understanding" of text is mainly derived by transforming texts to useable computational representations, which are discrete or continuous combinatorial structures such as vectors or tensors, graphs, and trees." (McMahan, et al., 2019)

The common techniques of NLP include:

- Tokenization –splitting the text into tokens, or words
- Removing the stop words – removing words which are used only to build a sentence and do not carry a semantic meaning
- Embeddings – transformation the text into numerical data
- Part-of-speech tagging – every tokenized word can be tagged as a part of speech: a noun, verb, or adjective
- Word and phrase frequencies – mostly used to analyze large blocks of text; in this technique is checked how often appear every word or phrase of interest
- N-grams – dividing a text into sequences of words of a set length:
    - unigram – a single word
    - bigrams – two words
    - trigram – three words
    - n-grams – any number of words
- Noun phrase extraction – in most sentences a noun is a subject of phrase, so noun extraction is common task in NLP when attempting to understand the meaning of a sentence
- Sentiment analysis –analyzes how positive or negative is a sentence or text
- Inflection – enables to get the singular or plural form of the word
- Lemmatization – a lemma is the root or headword for a set of words

3.2.1 Tokenization
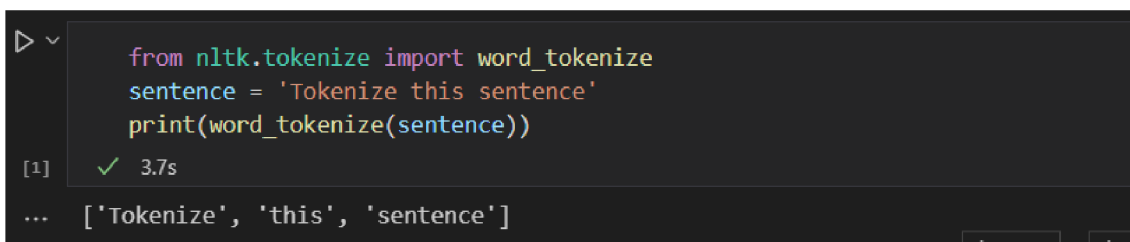
Tokenization is technique which divide input text into small chunks which could refer to words, sentences and called tokens. With tokenization and tokens, we can better understand the context of text and interpret it the right way. For instance, the text "I read book" can be broken into following tokens: "I", "read," "book."

We can apply tokenization to separate words or sentences. If the text is split into words, it is called word tokenization and when it is split to sentences – sentence tokenization.

There are different tokenization techniques which can be applied to the text based on the aim and language of text:

- White Space Tokenization – one of the simplest techniques. It divides text into tokens based on white spaces between worlds. It is effective with languages in which has white spaces between words, like English, Czech, or German.

- Dictionary Based Tokenization – in this technique a pre-made dictionary is available based which is the tokenization is performed., If some world is not found in dictionary, then program applies special rules to tokenize it.

- Rule Based Tokenization – in this method is created a set of rules, and the tokenization is performed based on the rules.

- Regular Expression Tokenizer – this method uses regular expression to perform the tokenization process.

- Penn Treebank Tokenization – is tokenization which is done based on Treebank. Treebank is a parsed text corpus that annotates syntactic or semantic sentence structure.

- Subword Tokenization – the idea of this relies on the principle that frequently used words should not be split into smaller sub words, but rare words should be decomposed into meaningful sub words. Therefore, algorithm divides words by frequency. If the word appears quite often, it does not divide the word and gives to it unique id, but if the word appears rarely it tries to divide it into sub words. For instance, if the word "refactoring" appears rarely in text algorithm divide it for next sub words – "re," "factor," "ing." This helps the model to find out similar words and better understand the meaning of text.

Today if we want to tokenize text, we have many various approaches to do it. For example, below you can see rule based tokenization using NLTK library in Python


```
from nltk.tokenize import word_tokenize
sentence = 'Tokenize this sentence'
print(word_tokenize(sentence))
```
[1] ✓ 3.7s
... ['Tokenize', 'this', 'sentence']

Figure 8 - Tokenization in Python using NLTK library

3.2.2   Removing the stop words

Stop words are frequently used words that do not add any additional information to the text. Words like "the", "is", "a" have no value and only add noise to the data. Another benefit of stop word removal is that it reduces the size of the data set and the time taken in training the model. If we want to remove stop words from this text:

"The film Arrival is one of those tapes that analyses an already hackneyed topic from a side that no one has approached yet. The film is based on the story of how twelve alien ships flew to Earth and landed at different places on the planet. After that, the film does not turn into a story about the occupation of the Earth or about the rapid exchange of technologies and the development of mankind. No, the emphasis is on trying to contact aliens and understand their purpose. The central figure in the plot is a linguist played by Amy Adams, who is assigned to study the language of aliens who are not clichéd humanoids, which is why speech is radically different.," we can use the Python snippet listed in Figure 9. The result of removing stop words you can see below:

```python
from nltk.corpus import stopwords

stop_words = stopwords.words('english')

text = 'The film Arrival is one of those tapes that analyses an already hackneyed topic from a side that no one has approached yet. The film is based on
text_tokenize = word_tokenize(text)
remove_stop_words = [word for word in text_tokenize if not word.lower() in stop_words]
print(remove_stop_words)
```

```
✓ 0.3s                                                                                                              Python

['film', 'Arrival', 'one', 'tapes', 'analyses', 'already', 'hackneyed', 'topic', 'side', 'one', 'approached', 'yet', '.', 'film', 'based', 'story',
'twelve', 'alien', 'ships', 'flew', 'Earth', 'landed', 'different', 'places', 'planet', '.', ',', 'film', 'turn', 'story', 'occupation', 'Earth', 'rapid',
'exchange', 'technologies', 'development', 'mankind', '.', ',', 'emphasis', 'trying', 'contact', 'aliens', 'understand', 'purpose', '.', 'central',
'figure', 'plot', 'linguist', 'played', 'Amy', 'Adams', ',', 'assigned', 'study', 'language', 'aliens', 'clichéd', 'humanoids', ',', 'speech', 'radically',
'different', '.']
```

Figure 9 - Removing stop words in Python using NLTK library

3.2.3   Embeddings

Because computers cannot understand data in a non-numerical format, we need to somehow transform our non-numerical data to numerical form.

"An embedding is a mapping of a discrete — categorical — variable to a vector of continuous numbers. In the context of neural networks, embeddings are low-dimensional, learned continuous vector representations of discrete variables. Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space." (Koehrsen, 2018)

Depending on the task, different approaches can be applied.

First one is One-Hot encoding. It is an unsupervised technique which helps to represent categorical non-numerical variables in word of digits. For instance, we have the category "Color" where there are three colors: red, green, and yellow. To transform this information to digits, One-Hot encoding needs to be applied. It is performed via mapping, creating a vector the same size as a quantity of categories. Example of it can be seen below:

```python
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore')
category = [['red', 'green', 'yellow']]
category = enc.fit_transform(category)
print(category)
```

✓ 0.7s
```
(0, 0)        1.0
(0, 1)        1.0
(0, 2)        1.0
```

Figure 10 - Simple encoding of words in Python using scikit-learn library

The next and final method of embedding is word embedding. This is an approach to represent text in natural language processing. It allows algorithms to understand the meaning of words.
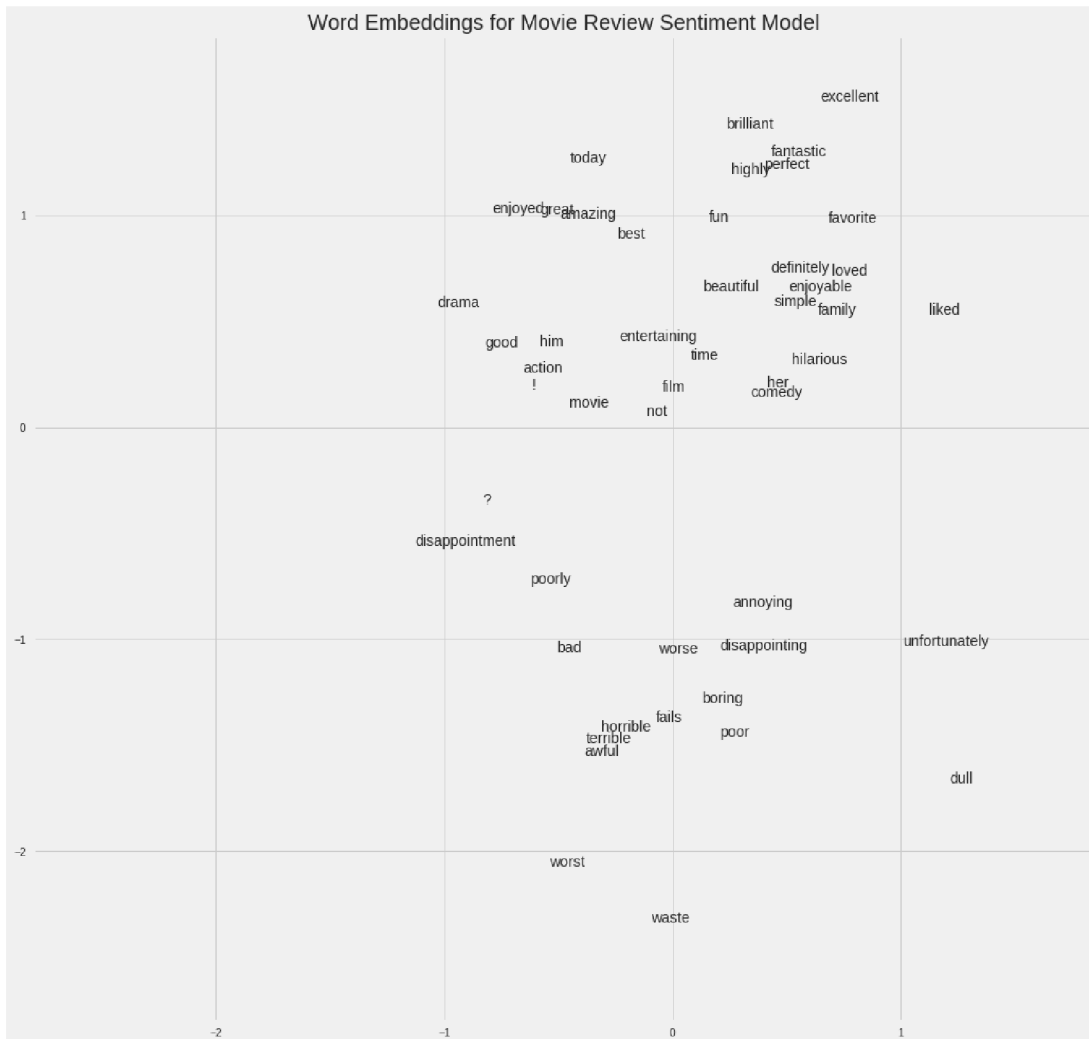
Figure 11- Example of word embedding (Koehrsen, 2018)

Nowadays the most effective tool to perform word embedding is Word2vec. It implements two main architectures – Continuous Bag of Words (CBOW) and Skip-gram. The corpus of text is fed as input, and a set of vectors of words is obtained as output.

The principle of operation is finding links between word contexts under the assumption that words in similar contexts tend to mean similar things, i.e., be semantically related. More formally, the problem is: maximizing the cosine proximity between vectors of words (the dot product of vectors) that appear next to each other and minimizing the cosine proximity between vectors of words that do not appear next to each other. Near each other in this case means in close contexts.

26

For example, the words "analysis" and "research" often appear in similar contexts, such as "Scientists have analyzed algorithms" or "Scientists have conducted research on algorithms." Word2vec analyzes these contexts and concludes that the words "analysis" and "research" are close in meaning. Since Word2vec draws such conclusions based on a large amount of text, the conclusions are quite adequate. Based on the example above, it can be concluded that to train a good quality Word2vec model a very large corpus of text is needed.

3.2.4   Parsing and part-of-speech tagging

Part-of-speech (POS) tagging is a stage of automatic text processing, the task of which is to determine the part of speech and grammatical characteristics of words in the text (corpus) with the assignment of appropriate tags to them. POS tagging is one of the first stages of computer text analysis.

```python
import spacy
nlp = spacy.load("en_core_web_sm")

text = 'Soft skills like sharing and negotiating will be crucial.'
doc = nlp(text)
for token in doc:
    print(token.pos_, token.text, token.tag_)
```
✓ 0.4s
```
ADJ Soft JJ
NOUN skills NNS
ADP like IN
VERB sharing VBG
CCONJ and CC
NOUN negotiating NN
AUX will MD
AUX be VB
ADJ crucial JJ
PUNCT . .
```

Figure 12 – Part-of-speech tagging in Python using spaCy library

3.2.5   Word and phrase frequencies

This approach finds the words which appear more frequently in a document and determines the importance of such words. But it has a problem – the words with the

highest frequency have the highest score. These words however may not have as much information gain for the model as the less frequent words. One way to fix the situation is to lower the score of a word that occurs frequently in all similar documents. This is called term frequency - inverse document frequency (TF-IDF).

TF-IDF is a statistical measure for evaluating the importance of a word in a document that is part of a collection or a corpus.

The TF-IDF scoring increases in proportion to the frequency with which the word appears in the document, but this is offset by the number of documents containing the word.

For example, if we have three documents of text:

1. *The elephant sneezed at the sight of potatoes.*
2. *Bats can see via echolocation. See the bat sight sneeze!*
3. *Wondering, she opened the door to the studio.*

And applying to them TF-IDF scoring, for the third document more significant will be words *studio*, *door,* and *wonder*.
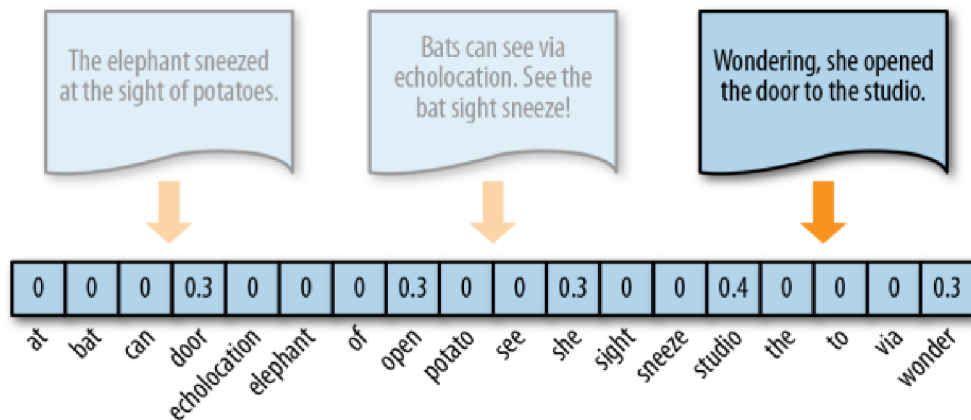
Figure 13- Example of applying TF-IDF scoring for multiple texts (Bengfort, et al., 2018)

3.2.6 N-grams

An N-gram is a sequence of entities (words, letters, numbers, digits, etc.). In the context of language corpora, an N-gram is usually understood as a sequence of words. A unigram is one word, a bigram is a sequence of two words, a trigram is three words, and so on. The

28

number N denotes how many grouped words are included in an N-gram. The model does not include all possible N-grams, but only those that appear in the corpus.

3.2.7 Sentiment analysis

This technique can be used to determine whether the text is positive, negative, or neutral. Sentiment analysis helps to understand the emotional undertones of language. This, in turn, helps to automatically sort out the opinions behind reviews, social media discussions, comments, etc.

3.2.8 Lemmatization and stemming

Typically, texts contain different grammatical forms of the same word, and may also contain homonymic words. Aim of lemmatization and stemming is to bring all occurring word forms to the same, normal dictionary form.

Lemmatization and stemming are special cases of normalization. Stemming is a crude heuristic process that cuts off the "extra" from the word root, often resulting in the loss of word-formation suffixes. Lemmatization is a more sophisticated process that uses dictionary and morphological analysis to bring a word to its canonical form – the lemma.

3.3 Neural networks for NLP

All techniques discussed above stand for preparational part of text processing from letters to digits. But how can computers obtain some information from this structured array of digits?

This is where neural networks come in. Today, many different types of neural networks exist each of which is designed to solve a special type of problem. Among them all, two main types of neural networks for NLP problem solving can be highlighted:

- Recurrent Neural Network, and
- Transformer.

3.3.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) are a type of neural networks that specializes in sequence processing. They are often used in NLP tasks because of their effectiveness in text analysis.

One of the nuances of working with neural networks, is that they work with predefined parameters. They take input data with fixed dimensions and output the result, which also has fixed dimension. The plus side of RNN is that they can work with sequences with variable lengths for both input and output.
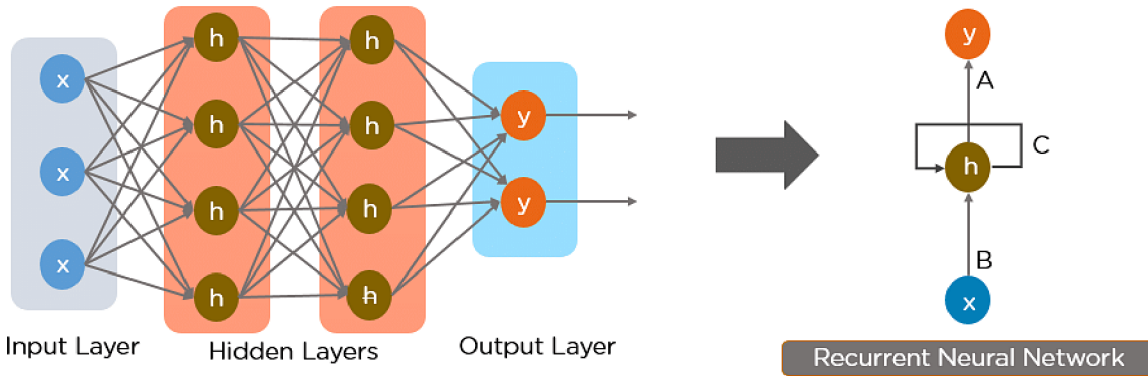
Figure 14 – RNN compared to a simple neural network (Biswal, 2022)

Due to the structure of recurrent neural network, a neuron in it receives some data about the previous state of the network in addition to incoming data. In this way the network implements "memory", which fundamentally changes the nature of its operation and allows to analyze any sequence of data, like audio recordings, text, or stock quotes, in which is important the order of values.
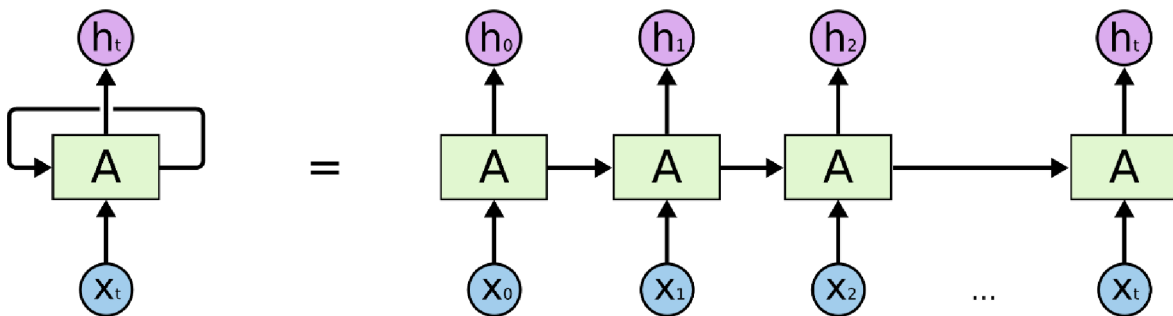
Figure 15 - An unrolled recurrent neural network (Olah, 2015)

Until 2017, engineers used deep learning to understand text using recurrent neural networks.

For example, when translating a sentence from English into Czech, an RNN would take an English sentence as input, process the words one by one, and then sequentially produce their Czech analogs. The key word here is "sequential". In a language, the order of words is important, and you cannot just mix them up.

This is where RNNs encounter several problems. First, they try to handle large sequences of text. By the time they get to the end of a paragraph, they "forget" the content of the

30

beginning. For example, an RNN-based translation model may have trouble remembering the gender of a long text object. Second, RNNs are difficult to train. They are known to be prone to the so-called vanishing/exploding gradient problem. (Or, 2020) Third, they process words sequentially, a recurrent neural network is difficult to parallelize. This means that it is impossible to accelerate learning using more GPUs. Consequently, it cannot be trained on a large amount of data.

To solve these problems, a new type of neural network was created – Transformer

3.3.2 Transformer

Transformer is a relatively new type of neural network aimed at solving sequences with easy processing of long-range dependencies. Today, it is the most advanced technique in the field of natural language processing.

They can be used to translate text, write poems and articles, and even generate computer code. Unlike RNN, transformers do not process sequences in order. For example, if the input data is text, they do not need to process the end of a sentence after processing the beginning. Due to this, such a neural network can be parallelized and trained much faster. Transformer was first described by engineers at Google Brain in "Attention Is All You Need" in 2017. (Vaswani, et al., 2017)

One of the main differences from existing data processing methods is that the input sequence can be transmitted in parallel, so that the graphical processor can be used efficiently, and the learning speed can be increased. cannot

The main components of transformers are an encoder and a decoder.
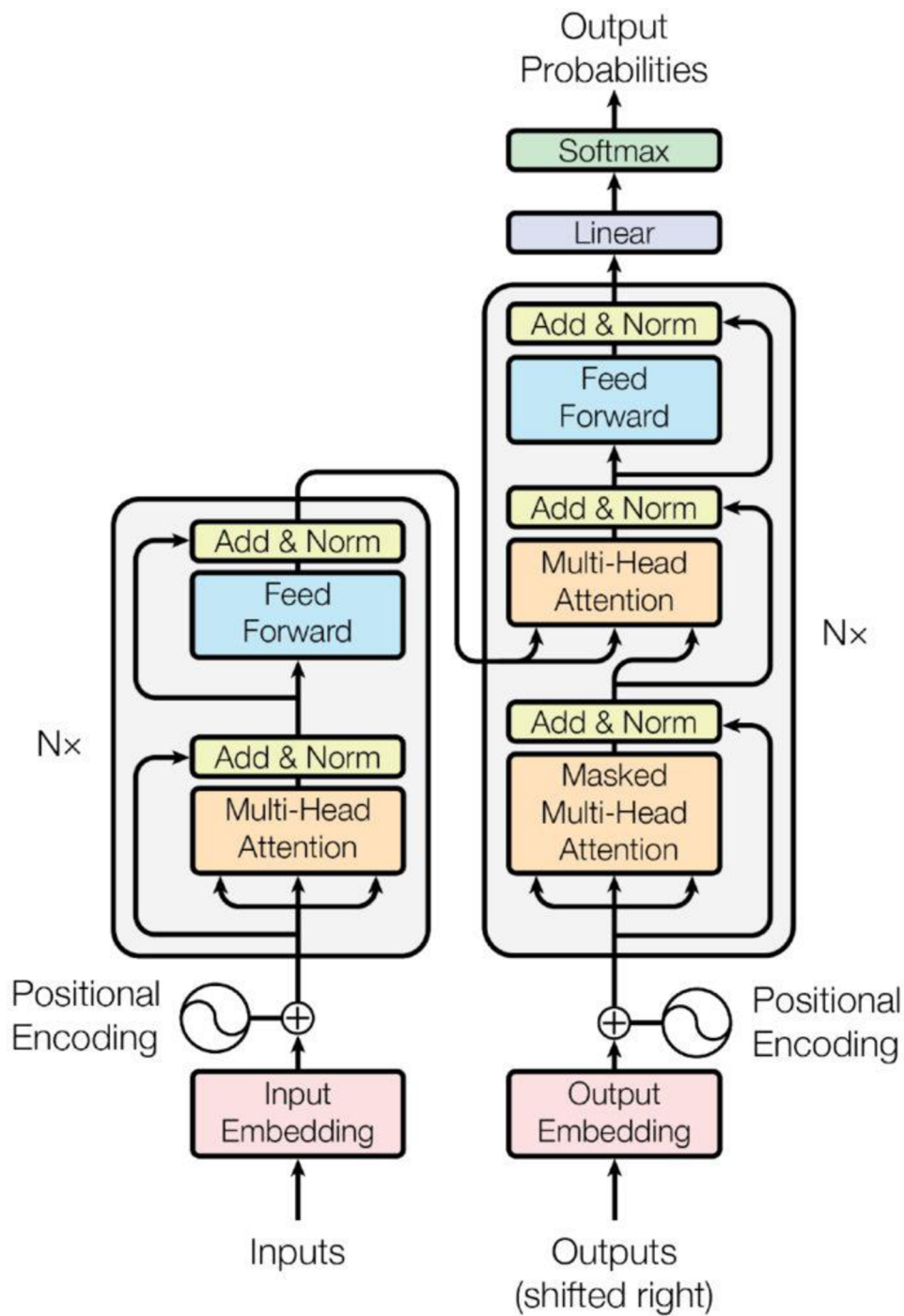
Figure 16 – Model of transformer (Vaswani, et al., 2017)

The encoder converts the incoming information (e.g., text) and converts it into a vector (a set of numbers). The decoder, in its turn, decodes it as a new sequence (e.g., the answer to a question) of words in another language, depending on what purpose the neural network was created for.

Other innovations underlying transformers can be summarized in three main concepts:

- Positional Encodings
- Attention
- Self-Attention

Position encoders take all the words in the input sequence, in this case an English sentence, and add a number to each in its order. Therefore, the transformer network works with following input:

[("Today", 1), ("is", 2), ("a", 3), ("good", 4), ("day", 5), ("to", 6), ("swim", 7)]

Conceptually, this can be seen as shifting the process of understanding word order from the structure of the neural network to the data itself.

At first, before transformers learn from any information, they do not know how to interpret these positional encodings. But as the model sees more and more examples of sentences and their encodings, it learns to use them effectively.

The structure presented above is given in an oversimplified way –the authors of the "Attention Is All You Need" used sinusoidal functions to produce positional encodings rather than the prime integers, but the idea is the same. By keeping the word order as data rather than structure, the neural network is easier to train. (Vaswani, et al., 2017)

Attention is a neural network structure introduced into the context of machine translation in 2015. (Bahdanau, et al., 2015) Imagine that we need to translate a phrase into French: *The agreement on the European Economic Area was signed in August 1992.*
The French equivalent of the phrase is as follows: *L'accord sur la zone économique européenne a été signé en août 1992.*

The worst translation option is a direct search for English words in French, one by one. This cannot be done for several reasons. First, some of the words in the French translation are reversed: "European Economic Area" versus "la zone économique européenne." Second, the French language is rich in gender words. To match the feminine object "la zone," the adjectives "économique" and "européenne" must also be put in the feminine

gender. Attention helps to avoid such situations. Its mechanism allows the text model to "look" at each word in the original sentence when deciding how to translate them. (Bahdanau, et al., 2015) The visualization from the original article demonstrates this:
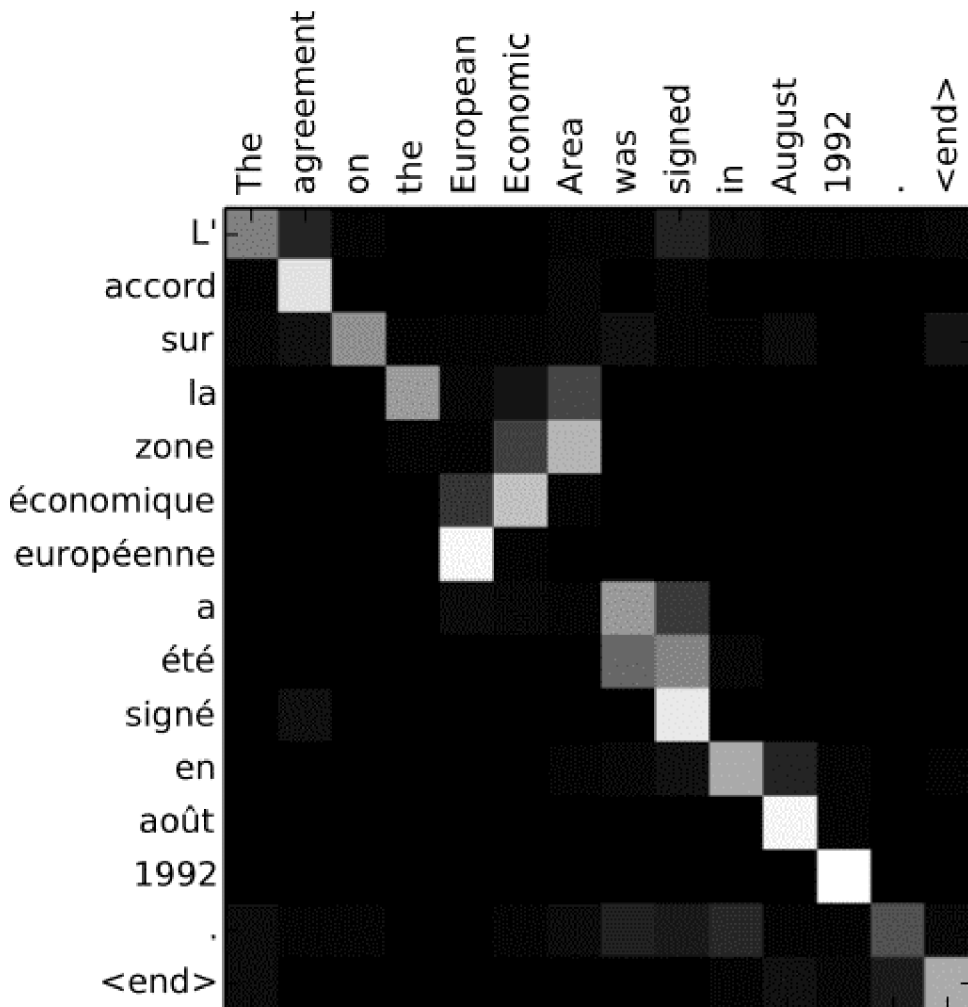
Figure 17 – Visualization of how model the understand similarity in meaning between two sentences (Bahdanau, et al., 2015)

This heat map in Figure 17 shows what the model "pays attention to" when it translates each word in a French sentence. As you might expect, when the model outputs the word "européenne," it largely considers both input words, "European" and "Economic.

Learning which words to "pay attention to" at each step helps the model learn from training data. By observing thousands of English and French sentences, the algorithm learns interdependent word types. It learns to consider gender, plurality, and other rules of grammar.

The last part of the transformers is a **self-attention**. While attention helps aligning words when translating from one language to another, self-attention allows the model to understand the meaning and patterns of language. For example, consider these two sentences:

"The animal did not cross the street because it was too tired."

"The animal did not cross the street because it was too wide."

The word "it" here refers to two different subjects, which we humans, knowing the situation, can understand. Self-attention allows the neural network to understand a word in the context of the words around it. Therefore, when the model processes the word "it" in the first sentence, it can refer it to "animal" and understand that too tired was animal not a street and understand what in second sentence "it" what refers to the street.

3.4   Tools for NLP engineer

After successfully introducing some of the algorithms and principles of NLP, the next part focuses on tools and environment which are necessary to develop applications in NLP area.

3.4.1   Python

"Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, It is used for both learning and real-life programming. It is easy to learn as most of the commands are similar to normal word used by humans (e.g., To print any statement or any value we simply write "print (statements)"). Many of the famous application are developed using python (e.g., Instagram, Dropbox.)." (Rana, 2019)

Following advantages of Python helped it to become popular and one of the main tools for development NLP systems:

- Easy to write and read code – Python code looks like simple English words and, in most cases, is set in the usual text-reading order. No semicolons or brackets are used, and indents define a block of code.
- Huge number of modules and packages – in addition to the standard Python libraries, there are simply countless additional modules and libraries available to

everyone. There are libraries for working with images, databases, unit testing, data analysis, data engineering, Artificial Intelligence, and many other features.

- Portable and versatile – Python is portable in the sense that the same code can be used on different machines. Suppose you write Python code on a Mac. If you want to run it on Windows or Linux later, you do not need to make any changes to it. That way you do not have to write the program multiple times for multiple platforms.

- Extensible – programming language is called extensible if it can be translated into other languages. Python code can also be written in other languages, such as C++, making it a very extensible language.

To create a program in Python, you first need to determine which packages will be used to create it, so next I would like to describe Python packages that allow you to solve NLP tasks.

3.4.2 NumPy

NumPy is a Python library that is used for mathematical calculations: from basic functions to linear algebra. This library has several important features that have made it popular. First, its source code is freely available on GitHub, which is why NumPy is called an open-source module for Python. Second, the library is written in C and Fortran. This makes computing much faster and more efficient. This makes this library indispensable for Artificial Intelligence tasks.

3.4.3 Scikit-learn

Scikit-learn is one of the most widely used Python packages for Data Science and Machine Learning. It allows to perform many operations and provides many algorithms. Scikit-learn also offers excellent documentation about its classes and methods , as well as descriptions of the algorithms used.

Scikit-learn supports:
- data preprocessing
- dimensionality reduction
- model selection
- regressions

- classifications
- cluster analysis

Scikit-learn does not have functionality directly for Deep Learning tasks, but it can help with data preprocessing.

3.4.4 Pandas

Pandas is a statistical data processing package close in functionality to SQL and R. It includes functionality for working with databases and Excel tables.

3.4.5 NLTK

NLTK (Natural Language Toolkit) is the leading platform for creating NLP programs in Python. It has easy-to-use interfaces for many language corpora, as well as text processing libraries for classification, tokenization, stemming, markup, filtering, and semantic reasoning.

3.4.6 Textblob

Textblob is a text processing library written in Python. It can be used for part-of-speech labeling, parsing, sentiment analysis, spelling corrections, and translation. Textblob relies on the Google Translate API in translation tasks. This means that it requires an active Internet connection to perform translations.

3.4.7 SpaCy

SpaCy is an open-source library for NLP written in Python and Cython. It is analogous to NLTK. But unlike NLTK, which is widely used for teaching and research, spaCy focuses on providing software for development.

3.4.8 TensorFlow

TensorFlow is a machine learning library developed by Google to build and train neural networks. TensorFlow is excellent for automatic image retrieval and classification, as the quality of recognition is close to human perception. TensorFlow can run in parallel mode on multiple processors: both CPUs and GPUs. The GPUs use CUDA for general purpose computations. This provides high speed training and operation of trained models. The TensorFlow library includes various APIs for building at scale deep learning architectures such as CNN or RNN.

4 Practical Part

4.1 Designing rephase system

The initial step in creation any program, application or system is to define the final goal, algorithm of working and the list of technologies to be used to help accomplish the tasks at hand.

4.1.1 The final goal of system

Two systems will be created in this thesis. First system is based on searching synonyms to words in phrase, which allows to save meaning and simultaneously change the given sentence. The second system is based on modern approach to dealing with NLP tasks – transformers. The given pre-trained model will be additionally train on new special corpus of text which will help it to better understand the paraphrase task and give more meaningful outputs.

4.1.2 Defining needed technologies and algorithm

Chosen technologies and their purpose for the first project:
- Python as language in which system will be written
- Visual Studio Code as IDE for writing Python code
- NLTK library for finding word antonyms and text processing
- Spacy library for finding word synonyms and text processing
- Datamuse API is API which allows to find similar words, so it will be used for their direct purpose – to find synonyms
- Urllib library – to send requests to the API
- JSON library – since the Datamuse API sends JSON file, will be needed to use the JSON library for Python to extract the necessary information from it

Algorithm of first system:
1. Obtain input sentences from txt file
2. Tokenize words in sentence
3. Get synonyms from Datamuse API
4. Get synonyms from NLTK library
5. Get score of similarity between initial word and synonyms

6. Choose best synonym by score

7. Change words in sentence

8. Print new sentence

9. Repeat step 2

Chosen technologies and their purpose for the second project:

- Python as language in which system will be written

- Google Collaboratory as IDE for writing Python code

- Transformers, torch, tqdm, collection and random libraires for process of training model

- Pandas, NumPy and csv libraries for data manipulating

- Os library for saving model

- Opusparcus_v1as corpus, on which model will be trained, this corpus is consisting of paraphrase pair of subtitles

- Rut5-base-multitask is already pre-trained transformer, which will be additionally trained to handle with rephase task

For the second system the algorithm of work will not be provided as it will rely on neural network.

4.2 Creating first system

4.2.1 Installing packages for the first system

The first phase is to install all needed packages, for this task, pip package manager will be used.

```
pip install nltk, spacy
```

Figure 18 - Installing needed libraries

The second phase is to download all datasets from NLTK library, which helps to find antonyms for text.

```
import nltk
nltk.download()
```

Figure 19 - Installing needed packages from NLTK library

After running this code in new window is needed to select and download packages.
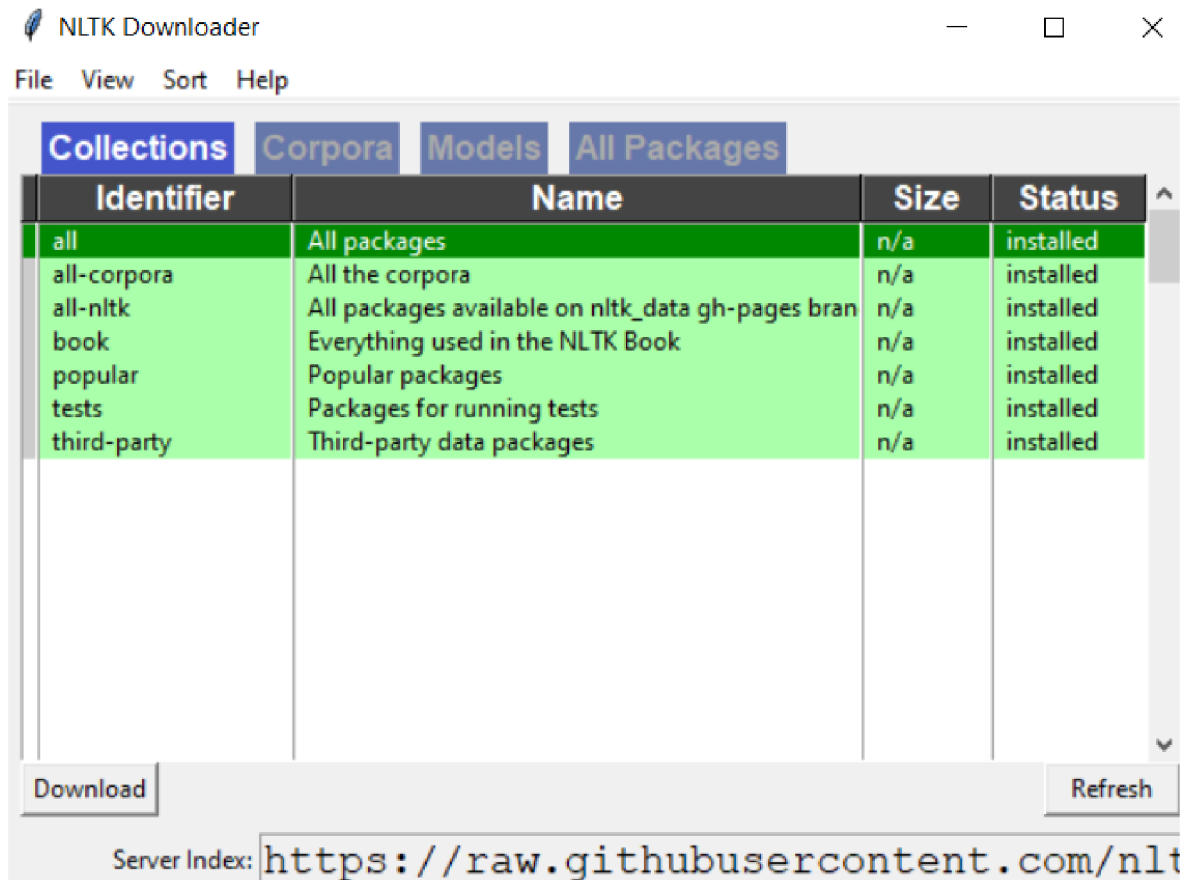
Figure 20 - Downloading window for packages

Now is needed to download trained model from SpaCy library which helps us to find synonyms for words.

Figure 21 - Downloading needed package from spaCy library

4.2.2   Implementing first system

Firstly, we need to import all needed libraries.

```python
from nltk.corpus import wordnet as wd
import spacy
import urllib.request
import json


nlp = spacy.load('en_core_web_lg')
```

Figure 22 - Importing libraries and downloading the model from spaCy

After this step is completed in directory in which script is placed will be placed txt file with phrase which needed to be rephase. The script needed to open this txt file and read all information from it.

```python
with open('text.txt') as f:
    contents = f.readlines()
```

Figure 23 - Opening and reading the file with text

After its script runs loop which takes every phrase from file and tries to rephase it.

```python
for content in contents:
    new_content = Rewrite(content).work()
    print(new_content)
```

Figure 24 – Starting to rephase every sentence from text file

```python
class Rewrite:

    def __init__(self, sentence):
        self.sentence = sentence

    def work(self):

        rewrite_types = [u'NN', u'NNS', u'JJ', u'JJS']
        pos_tokenizer = nlp(self.sentence)
        words = []

        for token in pos_tokenizer:
            if token.tag_ in rewrite_types:
                words.append(token.text)
        rewritten_sentence = self.sentence

        for word in words:
            word_syn = Synonym(word).compare()[1]
            rewritten_sentence = rewritten_sentence.replace(word, word_syn)
        return rewritten_sentence

    def __del__(self):
        self.sentence = False
```

Figure 25 - First class which defines parameters of rephasing

At the start need to be defined which words by token script can take for changing and performing tokenization using spaCy functionality. I have defined which words can be changed by next tokens:

- NN – noun, singular

42

- NNS – noun, plural

- JJ – adjective

- JJS – adjective, superlative

```python
def work(self):

    rewrite_types = [u'NN', u'NNS', u'JJ', u'JJS']
    pos_tokenizer = nlp(self.sentence)
    words = []
```

Figure 26 - Choosing which words can be changed by their token

After its script creates list of changeable words.

```python
for token in pos_tokenizer:
    if token.tag_ in rewrite_types:
        words.append(token.text)
rewritten_sentence = self.sentence
```

Figure 27 - Adding word what will be changed into list

And sends it to another class

```python
for word in words:
    word_syn = Synonym(word).compare()[1]
    rewritten_sentence = rewritten_sentence.replace(word, word_syn)
return rewritten_sentence
```

Figure 28 - Ruining class which will obtain synonyms from Datamuse API

In this class is going comparison between words. In first, script is sending words to Datamuse API with aim to obtain from its synonyms to these words.

```python
def __init__(self, word):
    self.word = word
    self.best_score = 0.0
    self.best_choice = ""


def compare(self):
    words_list = self.synonym_list()
```

Figure 29 - Initializing score of similarity

43

```
def synonym_list(self):
    path = "https://api.datamuse.com/words?ml=" + self.word
    output = urllib.request.urlopen(path)
    data = output.read().decode("utf-8")
    json_data = json.loads(data)
    words_list = []
    for i in json_data:
        words_list.append(i['word'])
    return words_list
```

Figure 30 - Getting synonyms from Datamuse API

After obtaining the list of similar words, we want to find another synonymous from NLTK datasets and compare these words and find the best matching by score.

In first script tries to obtain synonyms from using NLTK library.

```
def compare(self):
    words_list = self.synonym_list()
    for syn_word in words_list:
        use_nltk = True
        try:
            nltk_raw_word = wd.synsets(self.word)[0]
            nltk_syn_word = wd.synsets(syn_word)[0]
        except:
            use_nltk = False
```

Figure 31 - Finding synonyms for base word and already synonymized by Datamuse API

After it obtains synonyms from NLTK, it computes the similarity score between initial word and synonym from Datamuse using spaCy library and similarity score for synonymic words obtaining from NLTK using NLTK functionality.

```
spacy_score = spacy_raw_word.similarity(spacy_syn_word)

if (use_nltk == True):
    nltk_score = nltk_syn_word.wup_similarity(nltk_raw_word)
    if (nltk_score == None):
        nltk_score = 0
    score = (nltk_score+spacy_score)/2
else:
    score = spacy_score
```

Figure 32 - Computing the score of similarity

The next step is comparing by score. Scripts compares scores and finds the best match which return to initial function.

```
if (score > self.best_score):
    self.best_score = score
    self.best_choice = syn_word
result = [self.best_score, self.best_choice]
return result
```

Figure 33 - Comparing the score

After it, script replaces words in text by their synonyms.

```
rewritten_sentence = rewritten_sentence.replace(word, word_syn)
return rewritten_sentence
```

Figure 34 - Returning the rewritten sentence

The final step is just to print the rephased sentences.

```
for content in contents:
    new_content = Rewrite(content).work()
    print(new_content)
```

Figure 35 - Printing rephased sentences

4.3   Creating second system

For the second system as IDE, I have chosen Google Collaboratory This choice was made to use pretrained transformer model and additionally train it to rephrase task on new dataset. Because the script will run on servers which already have the necessary basic programs to run the neural model training process., Setting up IDE and installing packages for the second system.

First initial step is setting up IDE. Google Collaboratory allows to choose on which hardware the neural network can be trained, the faster way is to train it on GPU. To do it you need to go to runtime and change runtime type to GPU.
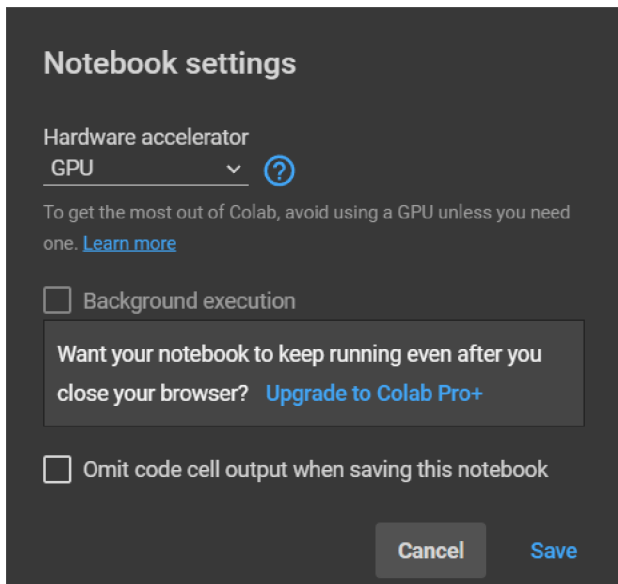
Figure 36 - Changing runtime type

After it is settled, we mounted up drive in order to create checkpoint for saving model.
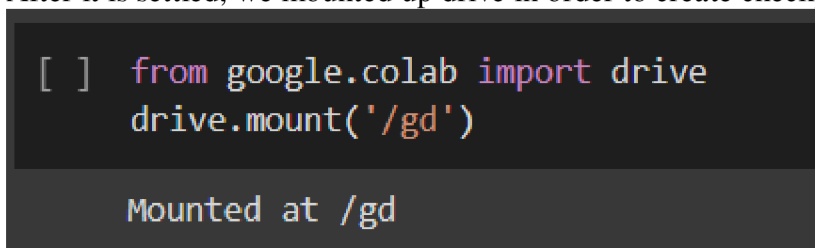
Figure 37 - Creating checkpoint

Next step is installing needed packages to the environment.
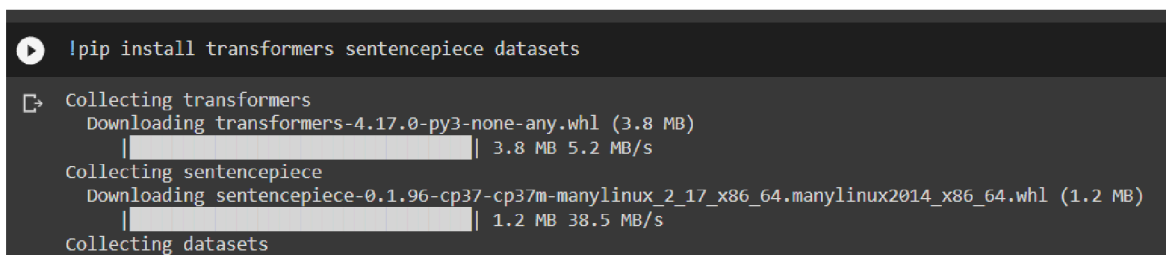
Figure 38 - Installing packages

4.3.1 Implementing second system

The first step, as always, is to import all needed libraries.

```
from transformers import (
    AdamW,
    T5ForConditionalGeneration,
    T5Tokenizer,
    get_linear_schedule_with_warmup
)
import torch
#from datasets import load_dataset
from tqdm.auto import tqdm, trange
from collections import Counter, defaultdict
import random
```

```
import pandas as pd
import os
import csv
import numpy as np
pd.options.display.max_colwidth = 300
```

Figure 39 - Importing libraries for the second system

After it, the script downloads corpus of text on which model will be trained and unzippes
it.

```
!wget https://korp.csc.fi/download/opusparcus/opusparcus_en.zip
!unzip opusparcus_en.zip
!bzip2 -dk opusparcus_v1/en/train/en-train.txt.bz2

--2022-04-03 09:53:42--  https://korp.csc.fi/download/opusparcus/opusparcus_en.zip
Resolving korp.csc.fi (korp.csc.fi)... 195.148.22.239
Connecting to korp.csc.fi (korp.csc.fi)|195.148.22.239|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 536916707 (512M) [application/zip]
Saving to: 'opusparcus_en.zip'

opusparcus_en.zip    100%[===================>] 512.04M  17.6MB/s    in 31s

2022-04-03 09:54:14 (16.3 MB/s) - 'opusparcus_en.zip' saved [536916707/536916707]
```

Figure 40 - Downloading and unzipping the corpus of data

At first, we need to see the size of dataset to understand how many sequences it has and for
additional verification that everything was downloaded successfully

```
opus = pd.read_csv('opusparcus_v1/en/train/en-train.txt', sep='\t', header=None)
opus.columns = ['idx', 'text1', 'text2', 'pmi', 'nalign', 'nlang', 'edit_distance']
print(opus.shape)

(40653996, 7)
```

Figure 41 - Dataset size

47

And the distribution of dataset.


```
opus.pmi.sample(1000).sort_index().plot()

<matplotlib.axes._subplots.AxesSubplot at 0x7fca0e1b0e10>
```
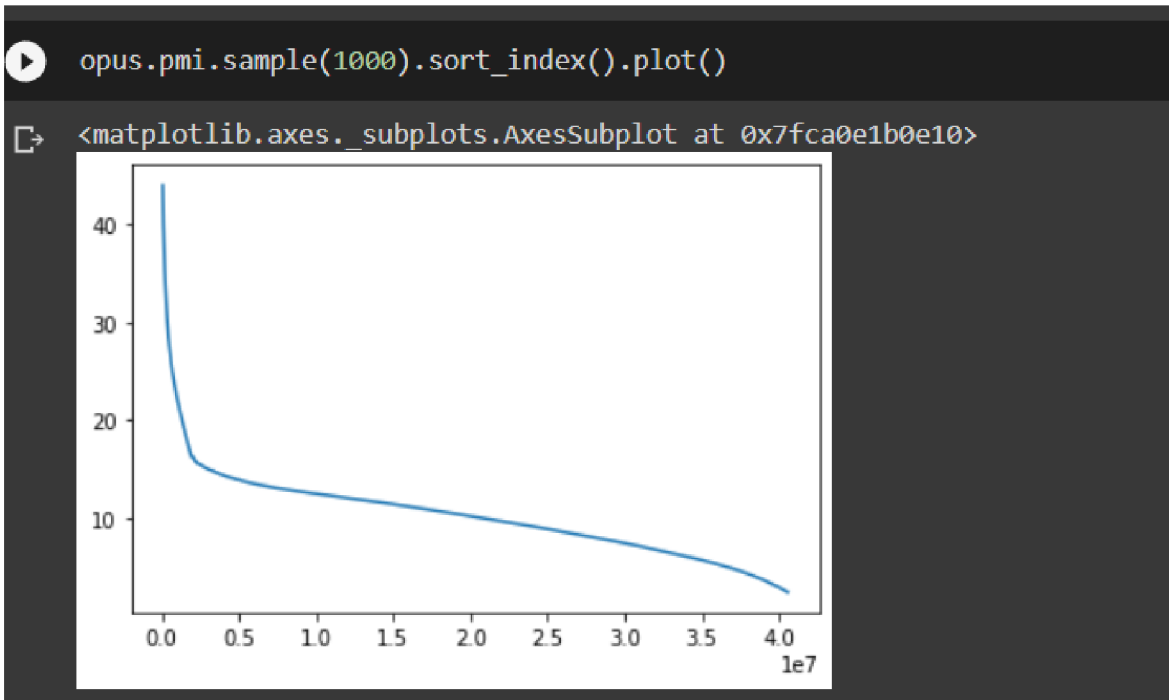
Figure 42 - Distribution of dataset

We do not need all this data, so next step is to choose best parts of corpus.

```
opus = opus[opus.pmi > 13].copy()
print(opus.shape)

(7530453, 7)
```

Figure 43 - Choosing best parts of corpus

Lest see how now looks the corpus

```
opus.sample(10)
```

	idx	text1	text2	pmi	nalign	nlang	edit_distance
4017611	en-N4017611	You 're missing some ?	You feeling deprived ?	14.3218	0.250000	1	14
5304109	en-N5304109	I do not want children , okay ?	What if we 're making a mistake , right ?	13.7264	0.083333	1	22
1321785	en-N1321785	Was there anything else ?	Is there anything else ?	19.9318	0.092332	2	1
6386712	en-N6386712	Did you hit him at all ?	No passage beyond this point without passports .	13.3349	0.029412	1	16
7345399	en-N7345399	Did you steal that ?	Were you the one who stole .	13.0546	0.066667	1	12
1082554	en-N1082554	Hi , son .	Howdy , son .	21.5710	0.378573	2	2
1706572	en-N1706572	He 's working for me .	Works for me .	17.4810	0.314102	2	4
4140083	en-N4140083	I mean , they " re , like , perfect .	They 're perfection .	14.2654	0.142857	1	9
556830	en-N556830	I go with you .	I 'm going with you .	26.4697	0.055935	3	3
2764749	en-N2764749	Can 't be many .	Couldn 't be many .	15.1259	0.666667	1	2

Figure 44 - sample of corpus

The next part is initializing a batch on which model will be trained

48

```
def get_batch(mult=1):
    """ Batch size is 10 x mult """
    xx = []
    yy = []
    idx = np.random.randint(opus.shape[0], size=2 * mult); xx.extend(opus.text1[idx]); yy.extend(opus.text2[idx])
    return xx, yy


xx, yy = get_batch(mult=3)
print(len(xx))

6
```

Figure 45 - Creating batch

Now it is time to download the model in notebook.

```
# take the model from this folder
raw_model = 'cointegrated/rut5-base-multitask'
# save the trained model to this folder
MODEL_NAME = '/gd/MyDrive/models/mt5-base-rephase'
if os.path.exists(MODEL_NAME):
    print('load from local checkpoint')
    raw_model = MODEL_NAME
model = T5ForConditionalGeneration.from_pretrained(raw_model)
tokenizer = T5Tokenizer.from_pretrained(raw_model)

load from local checkpoint
```

Figure 46 - loading model

Because the model trains for a huge amount of time, checkpoints are crucial. So, for this it checks if the model has been trained before.

Now, it is time to setting up the train condition for model.

```
#device = torch.device('cuda')
device = torch.device('cpu')
model.to(device)

        )
        (1): T5LayerCrossAttention(
          (EncDecAttention): T5Attention(
            (q): Linear(in_features=768, out_features=768, bias=False)
            (k): Linear(in_features=768, out_features=768, bias=False)
            (v): Linear(in_features=768, out_features=768, bias=False)
            (o): Linear(in_features=768, out_features=768, bias=False)
```

Figure 47 - Choosing hardware type

Next step is a choose optimizer for model and see how long text samples in data is.

```
optimizer = torch.optim.Adam(params = [p for p in model.parameters() if p.requires_grad], lr=1e-5)

qq = [0.5, 0.75, 0.9, 0.99, 1]
print(pd.Series(len(tokenizer(get_batch(mult=3)[0], padding=True)['input_ids'][0]) for _ in range(10000)).quantile(qq))
print(pd.Series(len(tokenizer(get_batch(mult=3)[1], padding=True)['input_ids'][0]) for _ in range(10000)).quantile(qq))

0.50     15.0
0.75     17.0
0.90     20.0
0.99     30.0
1.00     56.0
dtype: float64
0.50     15.0
0.75     17.0
0.90     20.0
0.99     29.0
1.00    102.0
dtype: float64
```

Figure 48 - Choosing optimizer and checking the length of samples in part of corpus

Adam is the best optimizer right now, because it combines the best properties of the adaptive gradient and root mean square propagation algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. Therefore, I have chosen it. Length will help in future to tune model.

To prevent hitting in the model some previous computations, next step is to clear cache.

```
import gc

def cleanup():
    gc.collect()
    torch.cuda.empty_cache()

cleanup()
```

Figure 49 - Cleaning cache

Also, we need to prevent training of layers from pre-trained model.

```
optimizer.param_groups[0]['lr'] = 1e-5
```

Figure 50 - Excluding pre-trained layers from training process

The next step is setting up the training process and its parameters.

- *Mult – this variable is needed to easily perform a changes in model, by default is equal to 1*

- *Batch_size – defines the size of input layer, for optimization purposes sets by 8*

- *Max_len – defines the maximal length of tokenized text. In dataset mostly of sentences have length less than 102 symbols, transformer performs subword tokenization therefore is needed to set max_len at least in one and a half times*

50

more than length of sentences. But in case a larger sentence is possible is required to make it higher than 161 symbols, therefore is equal to 384

- *Epochs, tq – set the number of training cycles. Because the training process on such a big corpus takes huge amount of time (using Google Collaboratory to train model on all corpus takes approximately one month) was selected 500 000 cycles*

- *Accumulation_steps – is responsible for how often will be updating parameters of model. During the tests, model worked the best than a value of accumulation_steps was equal to 32*

- *Save_steps – is responsible for how often model will be saving into Google Drive. Any number is possible here. But since the model made four thousand steps per hour in average, a value of four thousand was chosen for this variable.*

```python
model.train();
mult = 1
batch_size = mult * 8
max_len = 384
epochs = 5
accumulation_steps = 32
save_steps = 4000

window = 100
ewm = 0
errors = 0

tq = trange(int(1_00000 * epochs / mult))
cleanup()

for i in tq:
    xx, yy = get_batch(mult=mult)
    try:
        x = tokenizer(xx, return_tensors='pt', padding=True, truncation=True, max_length=max_len).to(device)
        y = tokenizer(yy, return_tensors='pt', padding=True, truncation=True, max_length=max_len).to(device)
        y.input_ids[y.input_ids==0] = -100
```

Figure 51 - First part of model

```
        loss = model(
            input_ids=x.input_ids,
            attention_mask=x.attention_mask,
            labels=y.input_ids,
            decoder_attention_mask=y.attention_mask,
            return_dict=True
        ).loss
        loss.backward()
    except RuntimeError as e:
        errors += 1
        loss = None
        cleanup()
        continue

    w = 1 / min(i+1, window)
    ewm = ewm * (1-w) + loss.item() * w
    tq.set_description(f'loss: {ewm}')

    if i % accumulation_steps == 0:
        optimizer.step()
        optimizer.zero_grad()
        cleanup()
```

Figure 52 - Second part of model

```
        if i % window == 0 and i > 0:
            print(ewm, errors)
            errors = 0
            cleanup()

        if i % save_steps == 0 and i > 0:
            model.save_pretrained(MODEL_NAME)
            tokenizer.save_pretrained(MODEL_NAME)
            print('saving...', i, optimizer.param_groups[0]['lr'])
```

```
loss: 2.031547614658195: 6%        32252/500000 [11:57:46<164:27:09, 1.27s/it]
2.087347303768993 0
2.0612074379464906 0
2.08807061637395 0
2.1257722850557035 0
2.0709691562900563 0
2.094296000537735 0
```

Figure 53 - Third part of model

The final part is evaluating the model.

```
model.eval();

def paraphrase(text):
    x = tokenizer(text, return_tensors='pt', padding=True).to(model.device)
    max_size = int(x.input_ids.shape[1] * 1.5 + 10)
    out = model.generate(**x, encoder_no_repeat_ngram_size=4, do_sample=False, num_beams=10, max_length=max_size, no_repeat_ngram_size=4,)
    return tokenizer.decode(out[0], skip_special_tokens=True)

for text1, text2 in zip(xx, yy):
    print(text1)
    print(paraphrase(text1))
    print(text2)
    print()
```

```
It 's only going to upset you .
However, it would only upset you.
It 's only gonna upset you .

Taste some .
These are some of the best.
Would you at least try this ?
```

In output we can see the initial sentence, rephased sentence and pair for initial sentence.

5 Results and Discussion

5.1 Evaluating both systems

This part focuses on testing the systems and comparing their outputs. For testing purposes, I have created two text files. One text file includes random sentences and the second random sentences from dialogues.

```
During the class we saw a short animated movie.
The movie was about a boy and his mother.
They were very poor.
What time are you going on duty?
The cow was too old to produce milk.
On the way to the market boy saw a mysterious looking man who wanted to buy his cow for the price of a bean.
He convinced the boy that the bean is not just a normal bean. It is a magical bean.
Boy sold his cow to the man and returned home.
When his mother asked him that how much did he got for the cow.
He showed up the bean. His mother yelled at him for being fool.
```

Figure 54 - Txt file with random sentences

```
Hey, I am starving.
How about a grilled cheese?
Walter, your dad and I have something we need to discuss.
I have always done business with certain local manufacturers.
So are you gonna be at work tomorrow?
I'd really look into some counselling.
Best of luck to you folks.
```

Figure 55 - Txt file with sentences from dialogues

These two text files have created because when assessing the quality of work rephase system there is no algorithm to confidently evaluate quality of output and is needed to use heuristic method. One of them contains random sentences and is aimed to understand how second system will handle types of sentences she has not faced in additional training on corpus of paraphrase pair of sentences from subtilties. And second one contains group of sentences which is generated from subtitles to evaluate how second system will handle types of sentences which it is familiar.

5.1.1 Testing the first system

At first let us test the first system.

```
with open('text.txt') as f:
    contents = f.readlines()

for content in contents:
    new_content = Rewrite(content).work()
    print(new_content)
```

Figure 56 - Processing first txt file to the first system

```
During the social class we also saw a short and sweet animated film.

The film was about a kid and his mom.

They were very needy.

What clock time are you going on obligation?

The moo-cow was too past to produce soya milk.

On the way of life to the marketplace kid saw a cryptic looking for guy who wanted to buy his moo-cow for the price tag of a sieva bean.

He convinced the kid that the sieva sieva sieva bean is not just a mean sieva sieva sieva bean. It is a magical power sieva sieva sieva bean.

Boy sold his moo-cow to the guy and returned home.

When his mom asked him that how so much did he got for the moo-cow.

He showed up the sieva bean. His mom yelled at him for being fool away.
```

Figure 57 - Rephased sentences from the firs txt by first system

Judging from the output, the first system acceptably rephased only one sentence. And only one sentence looks like it was written by human. So, it can be said what for the random sentences systems based on algorithm of finding synonyms and changing by them words in sentence performs badly.

Second phase of testing is to test the first system on the second text file, which includes simpler sentences taken from dialogues.

```
with open('text2.txt') as f:
    contents = f.readlines()

for content in contents:
    new_content = Rewrite(content).work()
    print(new_content)
```

*Figure 58 - Processing second txt file to the first system*

Figure 59 - Rephased sentences from the firs txt by first system

The first system performed a little bit better on the second text file. The first system made acceptable rephase of two sentences.

5.1.2 Testing the second system

Let us see how performs second system.

```python
def paraphrase(text, beams=5, grams=4):
    x = tokenizer(text, return_tensors='pt', padding=True).to(model.device)
    max_size = int(x.input_ids.shape[1] * 1.5 + 10)
    out = model.generate(**x, encoder_no_repeat_ngram_size=grams, num_beams=beams, max_length=max_size)
    return tokenizer.decode(out[0], skip_special_tokens=True)

with open('text.txt') as f:
    contents = f.readlines()

for content in contents:
    print(paraphrase(content))
```

```
We saw a movie.
This movie was about him and his mother.
They were poor.
What time do you go on duty?
The cow's too old to make milk.
I saw a man looking for a cow.
He convinced him that the bean's not a normal.
They sold him to the man, and he returned home.
When she asked him how much he got for a cow.
He's a fool.
```

*Figure 60 - Processing first txt file to the second system*

The result of second system is better than results produced by the first system.
The second system makes almost all rephased sentences in human-like style, but it did not rephase forth sentence and almost all sentences lost some detail and have a more general description of the context.
Let us see how it deals this the second text file.

```
def paraphrase(text, beams=5, grams=4):
    x = tokenizer(text, return_tensors='pt', padding=True).to(model.device)
    max_size = int(x.input_ids.shape[1] * 1.5 + 10)
    out = model.generate(**x, encoder_no_repeat_ngram_size=grams, num_beams=beams, max_length=max_size)
    return tokenizer.decode(out[0], skip_special_tokens=True)


with open('text2.txt') as f:
    contents = f.readlines()


for content in contents:
    print(paraphrase(content))

I'm starving.
What about a cheese?
Your dad and I need something to discuss.
I always work with local manufacturers.
You're going to work tomorrow?
I'm gonna look into counselling.
Well, good luck to you guys.
```

Figure 61 - Processing second txt file to the second system

For the second task seconds system performs well. It did not adequately rephase third sentence and we can see strong trend to generalize the meaning of sentences, but other sentences are rephased quite good.

5.1.3   Comparison of the two systems

Based on the results of the two systems shown, we can say that the approach of a trivial search for synonyms and replace the originals with them works rather poorly. This is due to the fact that often when a word is changed to its synonym, the structure of the sentence also changes. Therefore, if the rephrase system uses only algorithm to replace words in a sentence with their synonyms, it cannot produce good outputs. Out of seventeen sentences it was able to rephase only three of them correctly. Therefore, it is safe to say that the idea of simply replacing words with synonyms will not give the desired effect even for very simple phrases.

The second system, based on the pre-trained Transformer and additionally trained on a corpus of paraphrased subtitles from movies, works much better. Out of seventeen sentences it was able to rephase fifteen of them. It can also be noted that the system has a clear tendency to generalize sentences, which leads to loss of context and generalization of meaning. This is due to the fact that the system was trained on relatively short sentences. To improve the performance of such a system it is necessary to train it on a more diverse corpus of text.

57

6 Conclusions

This thesis aims to show possible approaches to text rephrasing using NLP methods in Python. The theoretical part of the thesis is devoted to Artificial Intelligence, neural networks, and basic NLP methods. The theoretical part also pays attention to a more detailed description of RNNs and Transformers and briefly describes the main tools of NLP engineer.

The practical part demonstrates two possible approaches to the creation of rephrasing systems. The first one was created using the usual algorithm using neural networks only for embedding words. This approach showed its great inefficiency and proved that even for simple tasks in NLP, the usual algorithm is simply not enough. The second approach used a relatively new development in the field of NLP – Transformers.

The second system was based on an already pre-trained Transformer and additionally trained on a corpus of text, which contained pairs of paraphrased sentences taken from subtitles. This approach turned out to be effective, but it should also be noted that additional training of transformers on only one dataset, which includes similar sentences with similar structure, is problematic. To create a better Transformer, it is necessary to choose a more diverse dataset.

As of today, the potential of transformers is still unexplored. They have already proven themselves in word processing, but recently this kind of neural network is being considered in other tasks, such as computer vision.

At the end of 2020, CV models showed good results in some popular benchmarks, such as object detection on the COGO dataset or image classification on ImageNet. In October 2020, researchers from Facebook AI Research published an article describing the Data-efficient Image Transformers model based on Transformers. According to the authors, they found a way to train the algorithm without a huge set of marked-up data and obtained a high accuracy of image recognition – 85%. (Touvron, et al., 2020)

In May 2021, Facebook AI Research presented DINO, an open-source computer vision algorithm that automatically segments objects in photos and videos without manual markup. It is also based on transformers, and segmentation accuracy has reached 80%. (Bojanowski,

et al., 2021) We can conclude what in addition to NLP, transformers are increasingly finding use in other tasks as well.

7 References

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. 2017. *Attention Is All You Need.* 2017.

Benjamin Bengfort, Rebecca Bilbro, and Tony Ojeda. 2018. *Applied Text Analysis with Python.* 2018. 978-1-491-96304-3.

Biswal, Avijeet. 2022. Recurrent Neural Network (RNN) Tutorial: Types, Examples, LSTM and More. *Simplilearn.* [Online] 21. 02 2022. [Citace: 27. 03 2022.] https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn.

Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. 2015. *Neural Machine Translation by Jointly Learning to Align and Translate.* 2015.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, Hervé Jégou. 2020. *Training data-efficient image transformers & distillation through attention.* 2020.

Koehrsen, Will. 2018. Neural Network Embeddings Explained. *Towards Data Science.* [Online] 2. 10 2018. [Citace: 20. 3 2022.] https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526.

Lu, Lu, a další. 2019. *Dying ReLU and Initialization: Theory and Numerical.* místo neznámé : Global Science Press, 2019. Sv. 28. DOI: https://doi.org/10.48550/arXiv.1903.06733.

McMahan, Delip Rao and Brian. 2019. *Natural Language Processing with PyTorch.* s.l. : O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2019.

Olah, Christopher. 2015. Understanding LSTM Networks. *colah's blog.* [Online] 27. 8 2015. [Citace: 29. 3 2022.]

Or, Barrak. 2020. The Exploding and Vanishing Gradients Problem in Time Series. *Towards Data Science .* [Online] 10. 10 2020. [Citace: 10. 03 2022.] https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22.

Penrose, Roger. 1989. *The Emperor's New Mind: Concerning Computers, Minds and The Laws of Physics.* 1989. 01401.45346.

Piotr Bojanowski, Mike Rabbat, Armand Joulin, Nicolas Ballas, Mathilde Caron, Mahmoud Assran. 2021. Advancing the state of the art in computer vision with self-supervised Transformers and 10x more efficient training. *MetaAI.* [Online] 30. 04 2021. [Citace: 1. 4 2022.] https://ai.facebook.com/blog/dino-paws-computer-vision-with-self-supervised-transformers-and-10x-more-efficient-training/.

Rana, Yogesh. 2019. Python: Simple though an Important Programming language. *International Research Journal of Engineering and Technology .* 2019, Sv. 06, 02.

Russel, Stuart a Norvig, Peter. 2002. *Artificial Intelligence: A Modern Approach (2nd Edition).* místo neznámé : Prentice Hall, 2002. 0-13-790395-2.

Sharma, Sagar. 2017. Activation Functions in Neural Networks. *Towards Data Science.* [Online] 06. 09 2017. [Citace: 12. 1 2022.]

—. **2017.** What the Hell is Perceptron? *Towards Data Science.* [Online] 09. 09 2017. [Citace: 10. 01 2022.]

Wolfewicz, Arne. 2021. Deep learning vs. machine learning – What's the difference? *Levity.* [Online] 09. 11 2021. [Citace: 12. 03 2022.] https://levity.ai/blog/difference-machine-learning-deep-learning.

8 Appendix

First_system.py

Text.txt

Text2.txt

[Link to Google Collaboratory](#)

[Link to the Transformer](#)

[Link to the training corpus](#)