Adam Mickiewicz University in Poznań

Faculty of Mathematics and Computer Science

Master's Thesis

Computer-assisted writing of English texts for non-native speakers

Komputerowe wspomaganie pisania tekstów w języku angielskim dla nierodzimych użytkowników języka

Krzysztof Jurkiewicz

452088

Field of study: Computer Science

Major: Intelligent Systems

Supervisor:

dr Filip Graliński

Poznań, 2020

Poznań, dnia 25.05.2020 r.

Oświadczenie

Ja, niżej podpisany Krzysztof Jurkiewicz, student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt. "Komputerowe wspomaganie pisania tekstów w języku angielskim dla nierodzimych użytkowników języka" napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom ani nie odpisywałem tej rozprawy lub jej części od innych osób. Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[ ]* – wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM

[ ]* – wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

....

(czytelny podpis studenta)

Contents

Abstract

This thesis is devoted to the problem of assisting non-native speakers in writing English texts which focuses on the development of algorithms for Grammatical Error Detection. Grammatical Error Detection problem covers the "problem of writing" for non-native English speakers and this thesis investigates how to solve it. The Proposal introduces the oddballness measure which includes two novel algorithms. They will be studied to see how theoretical aspects of the problem translates to an empirical comparison between the related solutions. The goal of this thesis is to explore how oddballness can be applied to assist the writing of English texts. First, the fundamentals of Neural Language Processing are introduced. They are extended with the recent state-of-the-art models, which were the foundation for this thesis.

A Master's Project has been created to accompany this thesis. It is a scalable project with a Vue frontend, Django backend, and Huggingface's implementation of the GPT-2 engine. The project allows users to create an English text while detecting and correcting their errors automatically.

There have been several experiments conducted, the aim of which was to compare oddballness Grammatical Error Detection models with the state-of-the-art UEDINMS model as well as with Languagetool. The results are included in this thesis.

Streszczenie

Niniejsza praca dyplomowa poświęcona jest problemowi pomocy obcokrajowcom w pisaniu tekstów w języku angielskim. Problem ten koncentruje się na opracowaniu algorytmów do wykrywania błędów gramatycznych. Problem z wykrywaniem błędów gramatycznych obejmuje "problem pisania" dla osób, które nie są rodzimymi użytkownikami języka angielskiego, a ta teza bada, jak go rozwiązać. W pracy wprowadzona jest miara oddballness, która obejmuje dwa nowe algorytmy. Zostaną one zbadane, aby zobaczyć, jak teoretyczne aspekty problemu przekładają się na empiryczne porównanie powiązanych rozwiązań. Celem tej pracy jest zbadanie, w jaki sposób można zastosować oddballness, aby pomóc w pisaniu angielskich tekstów. Po pierwsze, wprowadzono podstawy neuronowego przetwarzania języka. Zostały one rozszerzone o najnowocześniejsze modele, które były podstawą na której opierały się wnioski badane w tej pracy.

W celu uzupełnienia tej pracy powstał projekt magisterski. Jest to skalowalny projekt z frontendem w Vue, backendem w Django i implementacją silnika GPT-2 przez Huggingface. Projekt pozwala użytkownikom tworzyć tekst w języku angielskim, automatycznie wykrywając i poprawiając błędy.

Przeprowadzono kilka eksperymentów, których celem było porównanie oddballness w modelach wykrywania błędów gramatycznych z nowoczesnym modelem UEDINMS, a także z systemem Languagetool. Wyniki badań zawarte są w tej pracy.

Introduction

Purpose and scope of the thesis

The text has always been the most popular form of saving information. Languages around the world evolved with time and with them, spoken words got transformed into written form. Along with the need for writing the correct form, there came a need to correct words that were ill-written. There are multiple rule-based systems like "Rule-based System for Automatic Grammar Correction Using Syntactic N-grams for English Language Learning (L2)" [28] or "LanguageTool" [21] that are designed to help people correct and detect mistakes in their text. The goal of this master's thesis is to explore the methods of computer-assisted writing of English texts for non-native speakers. The theoretical foundations are primarily Neural Language Processing (later NLP) algorithms and methods.

The main focus of this Thesis is on exploring fields of Grammatical Error Correction (later GEC) and Grammatical Error Detection (later GED). The choice of those fields was carefully deliberated given the effectiveness of computer-assisted writing. Moreover, they were chosen with regards to the rapid development of Machine Learning [7] — especially NLP in recent years and the potential within it.

Structure of the thesis

This master's thesis has been written between February 2019 and June 2020. All sources and code used to write this thesis were collected or acquired in this period. The structure of the master's thesis consists of five chapters.

In the introduction, the purpose and scope of the work are justified. It begins by stating the most important aspects which will be discussed in this thesis.

In the first chapter, basic concepts of NLP are described. They are essential for an understanding of later chapters. In the early sections methods such as tokenization, stemming and normalization are described. Next, not only the machine learning paradigms but also BLEU and F-score measures are discussed. Those aspects combined with Levenshtein distance, word embeddings, and Byte Pair Encoding lead to a more complex description of BERT and GPT-2 models.

In the second chapter, the most prominent programs and tools used in GEC tasks are described. Those include state-of-the-art UEDINMS system which successfully used Transformer architecture for Grammatical Error Correction as well as Languagetool which gained a lot of community trust as a reliable rule-based tool for error detection and correction.

The following chapter is devoted to describing the setup for the evaluation of the master's thesis. First, the choice of the operating system is justified. The next section aims to describe the preparation of the test set. Finally, the platform for evaluation — Gonito.net — is described.

The fourth chapter contains an explanation of the algorithm using an innovative measure called oddballness. It dives into the mathematical formula for the oddballness and extends the concept with bidirectional oddballness.

The penultimate chapter describes the whole process of evaluation. Starting from the architecture used — the frontend, the backend, and the engine are described. This chapter brings closer the aspects of fine-tuning and inference steps. The final section discusses the results of the experiment.

Last but not least conclusions are presented, granting the reader a firm recapitulation of masters thesis.

Chapter 1

Introduction to Neural Language Models

1.1. The Beginning

Philipp Koehnn said in his book “Statistical Machine Translation” [17] that

> Efforts to build machine translation systems started almost as soon as electronic computers came into existence.

Those efforts also mark the beginnings of the inclusive field of Natural Language Processing (NLP).

The first important milestone could be considered as the 1954 Georgetown-I.B.M. experiment [14] that translated more than sixty Russian sentences into English having had only a few grammar rules and 250 lexical items in its vocabulary. This experiment made the impression that the field of machine translation was about to be solved very soon, and thus started the first golden era for NLP experiments. A few years later in 1966, the ALPAC report [23] pointed out challenges in machine translation process and costs thus being the cause of the reduction in funding that caused a significant slowdown of NLP research.

After several decades at the beginning of the current century interest in NLP increased again with the Conference on Natural Language Learning (CoNLL) shared-tasks. Papers such as Lafferty et al. proposing Conditional Random Fields (CRF) [18] or Bengio et al. “A Neural Probabilistic Language Model” [3], who was the first to introduce “dense vector representation” usage, laid the foundations of the current age state of the art NLP algorithms.

1.2. Theoretical Fundations in NLP

GED is the task that has the main focus in detecting different types of errors in a written text. Those errors could be one of the following:

- spelling errors
- typographical errors
- unwanted words
- missing words
- prepositional errors
- punctuation errors
- grammatical errors
- etc.

An important part of the GED is the language model which is a probability distribution over the sequence of words:

$$P(w_1, w_2, \ldots, w_n) \tag{1.1}$$

Ideally, the probability of a good language model would represent the likelihood for that sequence to be uttered by a random speaker. In 1951 Shannon introduced “Shannon’s guessing Game” [27] that consisted of a simple task. A player had to predict the next letter in the sentence given the preceding text, and the number of tries has been recorded.

> A perfect model of English will have a ranking of the predictions for the next letter based on their probabilities. By telling this model that the next letter is, say, the 5th guess, it can reconstruct the text. [17]

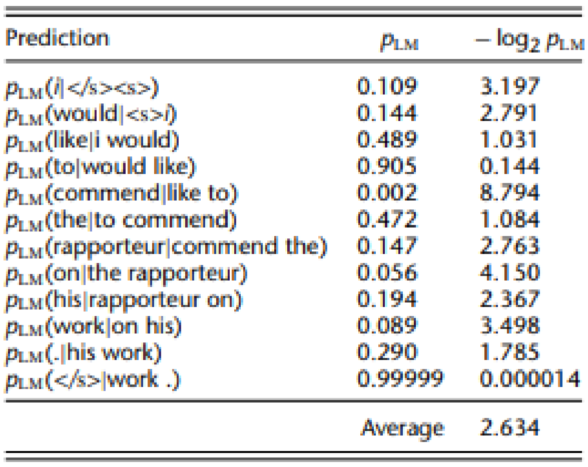
The perplexity of a probability distribution p is based on cross-entropy and is defined as:

$$PP = 2^{H(P_{LM})} \tag{1.2}$$

where $H(P_{LM})$ is a cross-entropy:

$$H(P_{LM}) = -\frac{1}{n}\sum_{i=1}^{n} logP_{LM}(w_i|w_1,\dots,w_{i-1}) \tag{1.3}$$

Perplexity could be used as a way to measure a model's quality. An Entropy is a math-



Prediction	p_{LM}	$-\log_2 p_{\mathrm{LM}}$
p_{LM}(*i*|</s><s>)	0.109	3.197
p_{LM}(would|<s>*i*)	0.144	2.791
p_{LM}(like|i would)	0.489	1.031
p_{LM}(to|would like)	0.905	0.144
p_{LM}(commend|like to)	0.002	8.794
p_{LM}(the|to commend)	0.472	1.084
p_{LM}(rapporteur|commend the)	0.147	2.763
p_{LM}(on|the rapporteur)	0.056	4.150
p_{LM}(his|rapporteur on)	0.194	2.367
p_{LM}(work|on his)	0.089	3.498
p_{LM}(.|his work)	0.290	1.785
p_{LM}(</s>|work .)	0.99999	0.000014
	Average	2.634

Figure 1.1. Computation of perplexity for the sentence "I would like to commend the rapporteur on his work." Koehn [17]

ematical derivation of perplexity. It is related to a number of bits that are necessary to encode some information. Mathematically it is defined as:

$$H(P) = -\sum_{x} p(x)log_2p(x) \tag{1.4}$$

The closer entropy gets to the entropy of the English language the better the model is. While Shannon's upper bound for the entropy of English language was:

$$H(P) \leq -\sum_{i=1}^{27} \hat{q_i}^N log_2\hat{q_i}^N \tag{1.5}$$

where $\hat{q_i}^N$ is defined as the relative frequency of times the subject needed exactly i guesses to discover the correct letter. Cover and King [6] proposed a convergent gambling estimate of the entropy of English to be about 1.25 bits per character.

> One naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: 'This is really written in English, but it has been coded in some strange

symbols. I will now proceed to decode.'*[Weaver,1947]*

If we were to define the task of GEC we could think of it in the similar terms great scientist Warren Weaver talked about translating a sentence from Russian to English. The text which is the subject of the GEC task is really written correctly(without errors), but has been coded with some errors. This is the idea behind the **noisy-channel model** which in this case would combine language model with a correction model.

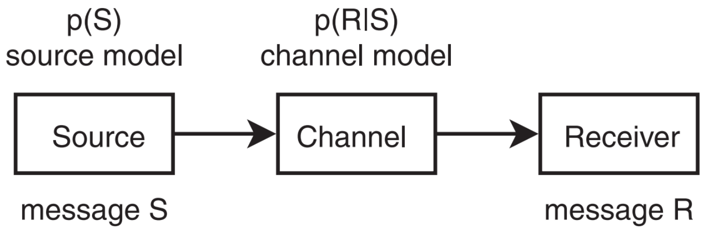


Figure 1.2. Noisy-channel model: The message that is transmitted by a noisy channel to the receiver. Message can be changed (contain errors) after this step. The text is then reconstructed with the usage of a source model $p(s)$ and a channel model $p(R|S)$ Koehn [17]

1.3. Text manipulation methods used in NLP

There are multiple ways that the text which enters the model may be transformed. Process of applying transformations to the text before it enters the language model is called **preprocessing**. Preprocessing often leads to text **normalization**. Text normalization is defined as a process of transforming text into a consistent, homogeneous, and canonical form. It is often said that text normalization helps to train language models by simplifying the data.

1.3.1. Lowercasing

Lowercasing is one of the simplest text manipulation methods in NLP. Given the example sentence:

"Penguins are associated with UNIX."

after lowercasing is applied, each letter is transformed to its lowercase form, as shown in a table 2 and coded in a listing 1.1. The output sentence would be:

"penguins are associated with unix."

```python
animal='PenGuiN'
print(animal.lower())
# Out[]: penguin
```

Listing 1.1. An example of lowercasing in python

Similar in nature transformation is **uppercasing** which transforms all letters to their capital form.

1.3.2. Tokenization

Tokenization is the process of the splitting sentence into words, or more generally **tokens**. A token is defined as a string of characters between two spaces or space and a punctuation mark. For Latin languages process of tokenization is fairly easy and consists of splitting on punctuation marks and spaces.

Although in English there are some peculiarities such as whether or not to split tokens that are a conjunction of two words. An example of such tokens might be "Don't" or "£1". Should they be left in the provided form, or split into two words as seen on the image 1.3?

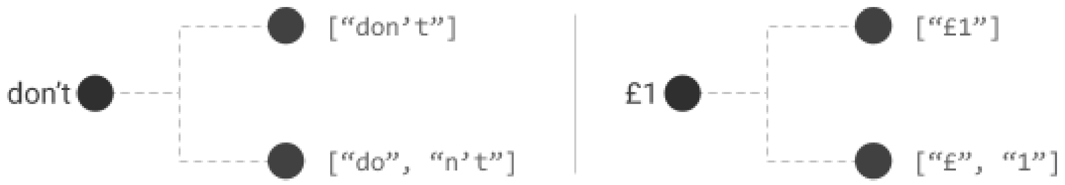


Figure 1.3. There are multiple ways to split a string of characters into a token.

In python's nltk library this could be achieved with the following bit of code on a listing 1.2.

1.3.3. Detokenization

In some tasks, like the "BEA-2019 Shared Task on Grammatical Error Correction" [5], the text is tokenized by default. Thatis because they use the ERRANT scorer, which requires

```python
import spacy

nlp = spacy.load("en_core_web_sm")
sample_string = "Isn't it  interesting ,  that  I  don't  have a  single  £1?"
tokenized  = nlp(sample_string)
print( list (repr(x)  for  x  in  tokenized))
# Out[]: [Is ,  n't ,  it ,  interesting ,  ,,  that,
# I,  do,  n't ,  have,  a,  single ,  £,  1,  ?]
```

Listing 1.2. Tokenization with the SpaCy module. The sentence is divided into tokens, while words "Isn't", "don't" and "£1?" has been disjointed

text to be tokenized. However, it may be beneficial to reverse the tokenization process, to let the text be processed by any language model. The technique that joins tokens to create a sentence is called detokenization. The detokenization method from listing 2 which reverses SpaCy tokenization was created as part of this master's thesis. The core idea is to join only those tokens, that would be split by SpaCy during tokenization. Unfortunately, it is not always possible to get exactly the same sentence after detokenization as it was before tokenization.

1.3.4. Stemming

Another transformation applied in NLP during preprocessing is stemming. Stemming involves cutting off the suffix of a word and therefore leaving a **stem** created of a root word. An example of stemming transformations are shown on the table 1.1

Before		After
Penguins	→	Penguin
are	→	are
fishing	→	fish

Table 1.1. Stemming transformation on a few examples

Stemming is just an inadvertent process of cutting off ends of words. It arrives with the assumption it will do it correctly most of the time, but not always.

1.3.5. Lemmatization

Lemmatization may seem similar to stemming, as it groups grammatically inflected forms to a singular base form. The difference comes with the details. The base form of a word is also called the **lemma**. Lemmatization tries to do things properly with the use of morphological analyses of the word. The example could be seen at the table 1.2.

Before		After
Penguins	→	Penguin
are	→	be
fishing	→	fishing

Table 1.2. Lemmatization transformation on a few examples

1.3.6. Normalization

Normalization is the process of transforming raw text into a sequence of tokens in canonical form. It often inquires doing any of the mentioned methods: lowercasing, tokenization, stemming, lemmatization, etc. An example of normalization could be:

"Penguin is making me smile." → ["penguin", "be", "make", "me", "smile", "."]

1.4. Machine Learning paradigms

Machine learning is gaining more and more popularity due to a variety of applications in a wide area of scientific and business projects. In order to become familiar with the models used in this project, it is worth becoming aware of several techniques used during machine learning.

1.4.1. F-score

F-score, also known as the F-beta score is the metric that allows evaluating the performance of an algorithm. Because it is the harmonic mean of precision and recall, F-beta score can be used to fine-tune models that have to optimize both precision and recall, without the drawbacks of either of them.

$$precision = \frac{true\_positive}{true\_positive + false\_positive}$$

$$recall = \frac{true\_positive}{true\_positive + false\_negative}$$

$$F_\beta = (1+\beta^2)\frac{2 \cdot precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

1.4.2. BLEU

BLEU is a method of evaluating machine learning [22] results, that can be used to measure the quality of the GEC task. It measures the similarity between two documents, by computing averaged product of n-gram coverage over an entire corpus. It takes values between 0 and 1.

1.4.3. Softmax

The softmax function is a function that normalizes a vector of real numbers into a probability distribution. It is often used in neural networks to map the vector to a probability distribution. It can be written as the following expression:

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$$

1.4.4. ReLU

The first AI winter was caused because at that time neural networks were not able to break the linearity of the models. Simple functions such as XOR proved to be impossible to solve. One of the functions that break the linearity of the neural network is ReLU — Rectified Linear Unit which can be written as a simple function:

$$ReLU(x) = max(0, x)$$

1.4.5. Supervised vs unsupervised learning

Supervised learning is a method in which the data requires to be labeled. On the opposite, unsupervised learning models can learn from data that is not labeled. The latter data is easier to acquire and is less costly. This fact may explain why we have a lot of hope for the future of unsupervised learning models.

1.5. Other methods used in NLP

There are plentiful other methods used in NLP. They can adjust the text for future processing by language model, or measure important metrics. Few of them are being described in this section.

1.5.1. Levenshtein distance

Levenshtein distance is also known as edit distance. It is defined as the minimum required number of edit operations needed to transform word A into word B, where we assume that single edit operation has a cost of 1 [19]. Edit operations are insertion, deletion, and substitution.

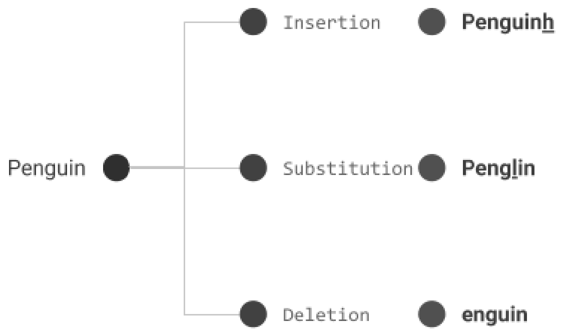


Figure 1.4. Example of three edit operations - insertion, substitution, and deletion.

Let's assume we define ϵ as an empty string, and a,b as any characters such as $a \neq b$, then:

- Insertion is defined as $\epsilon \to a$.
- Deletion is defined as $a \to \epsilon$.
- Substitution is defined as $a \to b$.

For example, given two words:

A = "english" and B = "penguin", Levenshtein distance between them is 4, as shown in figure 1.5.

This metric is useful for finding the best candidate as a correction for a word, or for aligning two sentences so that the distance between words is minimal.

```
0. penguin
             ←deletion (p→ε)
1. enguin
             ←substitution (u→l)
2. englin
             ←substitution (n→s)
3. englis
             ←insertion (ε→h)
4. english
```

Figure 1.5. Example of transforming “penguin” into “english”

Damerau-levenshtein distance is a modification of Levenshtein distance. It’s edit operations are extended with an operation of transposing two adjacent characters (or words).

1.5.2. Alignment

For the two sentences - original with possible errors *Orig* and second corrected *Corr* - we can define word alignment formally as a function that minimizes the distance between pairs of words in two sentences. Given words $w_1^{orig}, w_2^{orig}, \ldots, w_n^{orig}$ in the sentence *Orig* and words $w_1^{corr}, w_2^{corr}, \ldots, w_n^{corr}$ in the sentence *Corr*, for each i, j we match w_i^{orig} with w_j^{corr} so that sum of distances between matched words is minimized.

Felice M. et al. in “Automatic Extraction of Learner Errors in ESL Sentences Using Linguistically Enhanced Alignments” [8] showed that most natural alignment is when we use Damerau-levenshtein as a distance function. Example code for alignment is shown on a listing 1.3

```python
from errant.scripts.rdlextra import WagnerFischer
from errant.scripts.align_text import AlignText

orig_toks = "Penguin lives on Africa".split()
cor_toks  = "Penguins live in Antarctica".split()
orig = applySpacy(orig_toks, nlp)
cor = applySpacy(cor_toks, nlp)
alignments = WagnerFischer(orig_toks, cor_toks, orig, cor,
                           substitution=AlignText.token_substitution)

def applySpacy(sent, nlp):
    # Convert tokens to spacy tokens and POS tag and parse.
    sent = nlp.tokenizer.tokens_from_list(sent)
    nlp.tagger(sent)
    nlp.parser(sent)
    return sent
```

Listing 1.3. An example of alignment code with SpaCy tokenizer.

1.5.3. word embeddings

Word embeddings are defined as vector representations of words that are learned by neural network models. In 2003, Bengio et al. [3] showed the NLP model for learning word embeddings. Uday Kamath [16] summarizes word embeddings as follows:

> Instead of sparse, high-dimensional representations, the Bengio model proposed representing words and documents in lower-dimensional continuous vector spaces by using a multilayer neural network to predict the next word given the previous ones.

1.5.4. Word2Vec

Word2Vec [20] is a method of representing words as vectors. Each word is being assigned with some n dimensional vector of real numbers. By creating a vector representation of words in a continuous n dimensional space, it can take into account the context in which it occurs.

Thanks to the vector representation of words, that was learned with the help of neural networks, arithmetic operations on words are possible, for example:

$$\text{vector(“queen”)} = \text{vector(“king”) - vector(“man”)} + \text{vector(“woman”)}$$
$$\text{similarity(vector(“penguin”), vector(“book”))} = 0.54$$

Word2Vec method also allows to define a vector describing the whole document — for example by averaging vector representations, which were earlier learned with Word2Vec, in a context of the whole document, we can get a vector that describes that exact document:

$$\bar{D} = \frac{1}{N}\sum_{i=1}^{N} \bar{w_i}$$

The authors of the Word2Vec method suggest two different architectures for the neural network. Both models are being taught the use of *SGD (Stochastic Gradient Descent)* — the numerical algorithm, which aims to find minimum in some continuous, differentiable function — as well as *back propagation* which is a method of refreshing weights in the neural network.

1.5.5. Byte Pair Encoding

Sennrich R. et al showed in [26] that Machine Translation models can improve on the WMT15 translation tasks by over 1 Bleu score, when they are used with Byte Pair Encoding (later BPE).

The first to suggest the concept of BPE was Gage P. in 1994 in his article "A New Algorithm for Data Compression" [9]. It is a data compression algorithm that relies on replacing common pairs of neighboring bytes with an unused character.

Authors of "Neural Machine Translation of Rare Words with Subword Units" modified BPE to suit the task of creating an open-vocabulary by encoding uncommon and unusual words as a sequence of subword units. Further references of BPE will refer to subword unit implementation. They also present an easy and short python code (listing 1.4) for learning BPE operations.

```python
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

Listing 1.4. Algorithm for learning BPE operations, Sennrich R. et al in [26].

1.6. Recent models and methods in NLP

1.6.1. n-gram model (Markov Chain)

One of the simplest language models is probably an n-gram model. N-gram model is a model that assigns a probability to a word based on n-previous words. It differs from word-level representations because it can make use of a context. Predicting a single word based on every word that was before would quickly prove to be computationally expensive. That's why usually people constrain history by using some constant value of n proceeding words.

A simple unigram model, often called bag-of-words, takes exactly 0. If we assume that L is the length of the sentence, we can mathematically write it as:

$$P(w_1, w_2, \dots, w_L) = \prod_{i=1}^{L} P(w_i)$$

While it can be useful, there are far better approaches. Taking consecutive tokens into the account is called the bigram model. The generalized equation for any n-gram model looks like this:

$$P(w_1, w_2, \dots, w_L) = \prod_{i=n}^{L} P(w_i | w_{i-1}, w_{i-2}, \dots, w_{i-n})$$

Jurafsky and Martin in [15] classified this model as a one based on Markov assumption: " The assumption that the probability of a word depends only on the previous word Markov is called a Markov assumption. Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past."

1.6.2. Transformer

One of the most popular methods of giving a model the ability to focus on specific parts of the input is **attention**. This concept was first introduced in the paper "Neural machine translation by jointly learning to align and translate" by Bahdanau D. et al. [2]. It allows improving the quality of predictions as well as creates valuable intuition regarding the network, by viewing which parts of the input had most of the focus.

An interesting approach to attention was described by Vaswani A. et al. in “Attention is all you need” [29]. Authors introduced state of the art Transformer model which resigned entirely from recurrence to focus on sole attention. This model was able to improve by over 2 BLEU on existing best results from WMT 2014 English-to-German translation task, reducing the training costs, and increasing the generalization of the model.

Transformer - the model architecture

The Transformer is composed of two main blocks — encoder and decoder as shown on a figure 1.6. The last output from the encoder is the first input of the decoder. The encoder

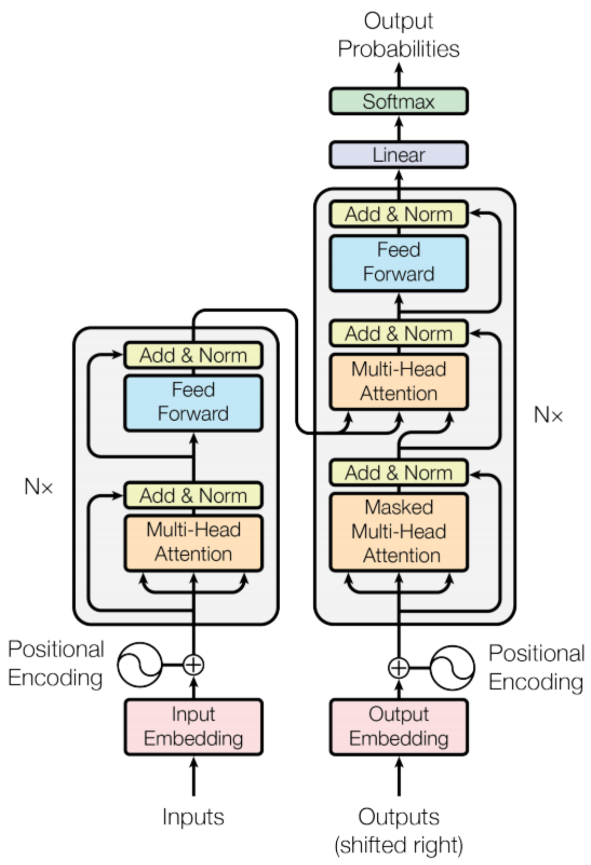


Figure 1.6. The Transformer - model architecture (Vasvani A. et al. in [29]).

aims to summarize the input text in the form of a vector and the decoder to decode the new domain.

The encoder in the Transformer consists of a *Multi-Head Attention* sub-layer and *Feed Forward* sub-layer. Those two sub-layers create a layer, which is stacked six times on top of itself. The output from each layer is normalized:

$$LayerNorm(x + Sublayer(x))$$

The decoder layer differs from the encoder because it has a third sub-layer, which is a *Masked Multi-Head Attention.* A mask in the attention layer by clever matrix multiplication prevents the flow of information of tokens in subsequent positions.

Attention

Attention is a mechanism that gives the model the ability to focus on different parts of the input. It enhances the flow of information the output at a particular time step. The attention used in Transformer is called "Scaled Dot-Product Attention". The input is composed of three matrices — queries (Q), keys (K), and Values (V). Those three matrices allow computing the attention as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Where d_k is the dimension of matrices Q and K, while $\sqrt{d_k}$ is a scaling factor, that accounts for the variance of multiplication QK^T.

Multi-head Attention

Authors of the paper implemented several attention layers that were running parallel to each other, therefore creating Multi-Head Attention (figure 1.7). Each Attention layer concentrates on different positions of the input space. This is effect is shown in figure 6 in the appendix.

1.6.3. BERT

BERT or "Bidirectional Encoder Representations from Transformer" is a model, which appeared shortly after the first version of GPT model [24]. The assumptions are, that BERT trains a big model on the available data, and then finetunes that model to do specific tasks without changes to the neural network. An important feature of BERT is the fact that it is a bidirectional — model not only tries to predict words that occur after the questioned word, but also takes into consideration context from the left and the right

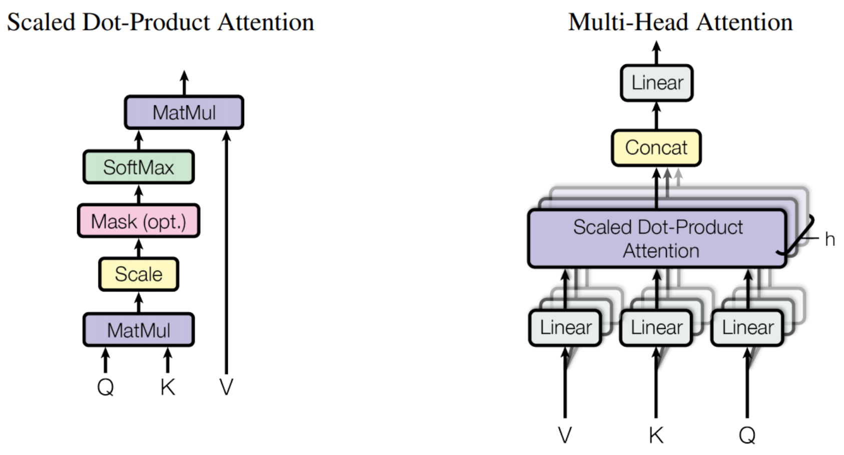


Figure 1.7. On the left — Scaled Dot-Product Attention, on the right — Multi-Head Attention architecture. [29]).

side. In order to enhance the bidirectionality of the model, BERT is being taught on two auxiliary tasks.

The first of the tasks is learning on a masked language model. It is about randomly choosing 15% of tokens, and changing them with a special [MASK] token. Among all the [MASK] tokens, 10% has been changed to a random word, 10% are changed back to the original word and the remaining 80% prevail without changes. This kind of action helps to generalize the model.

The second task is about training binary classifier, which aims to answer, whether the sentence B is a continuation of a sentence A, for example:

A) “Antarctic penguin”

B) “is an edible plant that grows in Africa”

The data has been chosen the way that in half of the cases sentence B is the continuation of a sentence A, and in the second half - it is not.

1.6.4. GPT-2

In 2019 State of the art model has been described by Radford et al. in [25]. It is called GPT-2 which stands for Generative Pretrained Transformer 2, named after the earlier model GPT [24]. GPT-2 model was a great success in the field of unsupervised task learning, on which many future models will build upon.

Dataset

The GPT-2 model was trained on a huge dataset consisting of 40GB of webpages crawled from the internet. Radaford (2019) note that: “Our approach motivates building as large and diverse a dataset as possible in order to collect natural language demonstrations of tasks in as varied of domains and contexts as possible”. The dataset is called WebText and was created only with data filtered by humans. Authors of the paper extracted all the text from the pages linked in every www.reddit.com post, that got at least a 3 stars karma rating, thus ensuring the quality of scraped corpora.

Input representation

Radford et al. decided to use a slightly modified version of BPE to create the vocabulary from WebText. Regular BPE created on WebText would have to allocate over 130,000 vocabulary slots for the sole purpose of storing every Unicode symbol. The authors also noticed the need to restrain BPE from combining pairs from different character categories.

> “We observed BPE including many versions of common words like dog since they occur in many variations such as dog. dog! dog? . This results in a sub-optimal allocation of limited vocabulary slots and model capacity.”
>
> Radford et al. [25]

Those changes gave to the model the ability to represent out-of-vocabulary words, without stretching the dictionary too much. The resulting dictionary consisted of slightly more than 50,000 tokens.

Architecture

OpenAI GPT-2 uses Transformer based architecture that was similar to the OpenAI GPT model. The core idea was based on stacking multiple decoder layers. There are four models proposed in the paper, which mainly differ in parameter size and amount of stacked decoder layers, ranging from 12 layers in the smallest model up to 48 layers in the biggest one.

Each model is designed to have a vector of 1024 token embeddings in the input, as well as a positional encoding vector of the same dimension. Each token embedding has

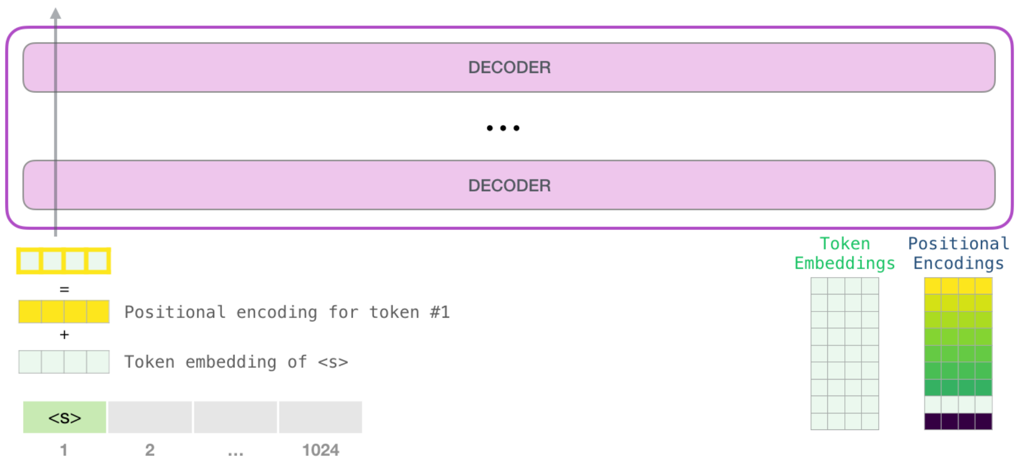


Figure 1.8. GPT-2 architecture: Positional encoding combined with token embedding is the input for the decoder layer. Alammar J. [1]

size d_{model}, which ranges from 768 up to 1600 depending on the size of the used model.

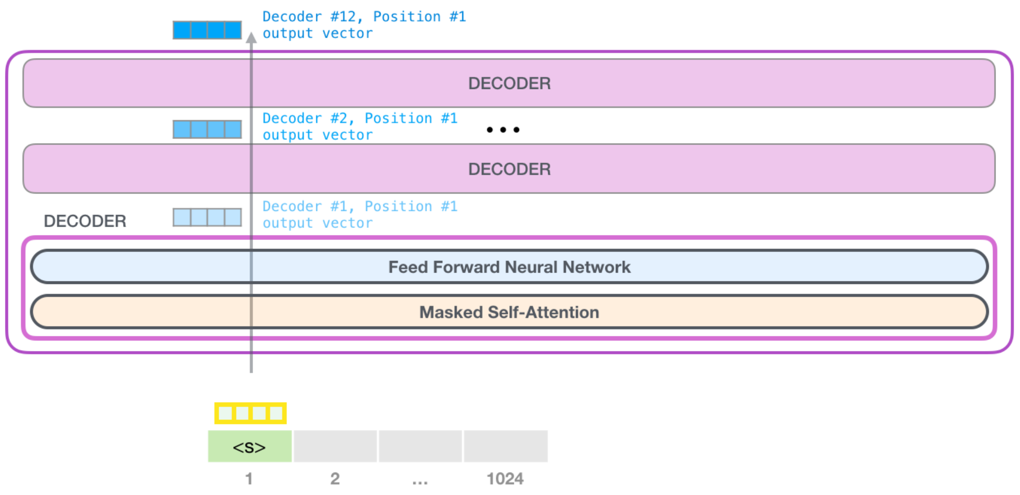

Decoder #12, Position #1 output vector
DECODER
Decoder #2, Position #1 output vector
...
DECODER
DECODER
Decoder #1, Position #1 output vector
Feed Forward Neural Network
Masked Self-Attention
<s>
1
2
...
1024

Figure 1.9. GPT-2 architecture: contents of a decoder layer - Feed Forward Neural Network on top of Multihead self-attention. Alammar J. [1]

Multi-head self-attention

Each decoder layer consists of a *multi-head masked self-attention* layer and *Feed Forward neural network* layer. The dimension of the neural network layer is the same as the embedding size.

Masked multi-head self-attention used for this model differs from the attention described in subsection 1.6.2. In GPT-2 masking operation is being performed for all the future positions, blocking the model from knowing the words that are further than the current position.

Examples and experiments

The model has been trained to predict the next word. This gave the model the ability to create synthetic text of great quality, continuing the text given by the user. As authors stated: “GPT-2 outperforms other language models trained on specific domains (like Wikipedia, news, or books) without needing to use these domain-specific training datasets”. An example of the capabilities of the model might be the prediction of the next words in the table 1.3 or in the figure 7.

User sentence	prediction	score
My penguins love to	eat	10,7%
	play	5.5%
	be	4.9%
	fly	2.6%
	go	2.0%
	[remaining tokens. . .]	74.3%

Table 1.3. Lemmatization transformation on a few examples

OpenAI GPT-2 model is a proof, that generalized language models can achieve the state of the art results in many tasks, without the need for supervised learning. Using BPE was a key element in creating GPT2. Thanks to it, this model is able to provide satisfactory results even when it encounters previously unseen words. A huge role in the success of this model was also the big text corpus WebText. It was not only filtered by people but also filtered out from all foreign posts. Despite this fact, authors were able to use GPT-2 for the task of machine translation. Scores achieved by the GPT-2 model suggest that given sufficient data and computing time, models can improve from unsupervised techniques.

Chapter 2

Related works

Many systems are trying to solve the GEC task. Such systems are put under the category of rule-based or neural network related. By using this method you can detect the most common mistakes in a language and fix them. The machine-learning-based techniques can recognize very complex grammar rules such as the conjugation rules, dependency rules, but also the sentence structure rules, which we find in the composition of words.

2.1. UEDINMS

UEDINMS is the state-of-the-art neural GEC system created by Grundkiewicz R. et al and described in "Neural Grammatical Error Correction Systems with Unsupervised Pre-training on Synthetic Data" [12]. This system achieved a very high score on BEA19 shared task — $69.47F_{0.5}$ in restricted track and $64.24F_{0.5}$ in the low-resource track. Moreover on CoNLL 2014 test set authors report state-of-the-art results.

2.1.1. General description

The creation of this system consisted of a few stages, in which the model was trained on different types of data. Overall there were more than 100 million sentences involved in producing the final system. UEDINMS is a Transformer [29] based system.

Attention heads embedded in the Transformer allowed to conduct unsupervised pre-training on a large text corpus. There were 100 million sentences used for pre-training which were taken from the English part of the WMT News Crawl corpus [4], and transformed with the use of a spell checker. Each sentence used for pre-training was divided into two sentences — one original and the second erroneous. On the table 2.1 there is an example of transformations applied on an original sentence that led to creating the erroneous one. Authors attempted synthetic recreation of errors made in real-life scenarios by applying few transformations similar to those used in the Damerou-Levenshtein metric.

First, by using a spellchecker they created confusion sets for each unique word in a corpus. From each sentence there were few words chosen at random, and either deleted, inserted a word after the chosen one, or substituted with a word from a confusion set. Words in confusion sets were chosen according to a probability distribution created as a weighted average of an edit distance between those two words and also a distance between their phonetic representation. Similar transformations were applied on a character level base.

The Transformer was pre-trained on those sentences and then fine-tuned on data with errors annotated by real people. Although other researchers showed it was possible to do the GEC system without the part of supervised fine-tuning, authors tried multiple strategies and found them to be beneficial.

Type	Output
Original input	But they have left their exam rooms and come out the streets to joining hands with the public and to fight for the country under the guidance of the monks .
+ Synthetic errors	But they have lift their exam rooms end ~~come~~ out the streets to joining lands with the public band to fight for country the unity the guidance of the monos .
+ Spelling errors	But they have lift their exm rooms end out the streets to joining lands with the public band to fight for counrty the unity the guidance of the monos .

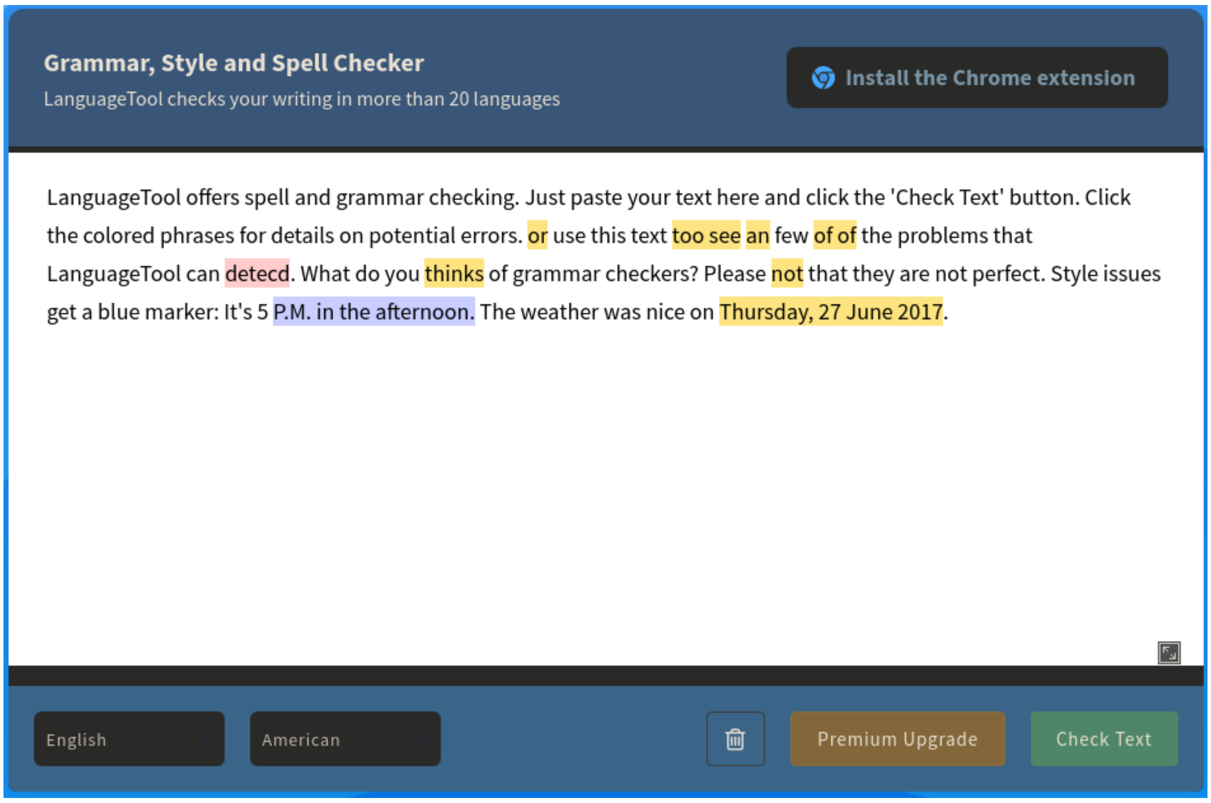
Table 2.1. An example of an artificially generated erroneous sentence. [12]

2.1.2. Architecture

UEDINMS's architecture was a modified Transformer. Instead of 8 attention heads there were 16 used. To handle out of vocabulary errors, subword tokens were embedded on a vector of size 1024. ReLU function is used between filters, and Dropout is used in a fine-tuning phase.

2.2. Languagetool

Languagetool [21] is a tool for GEC that can also be used for the GED task. It is an open-source rule-based system on a GNU GPL license. It exists in the form of an add-on to Firefox, Chrome, Word, Google Docs, as a desktop version, an API, or in a console. The multiplicity of available forms makes it easy to use and easy to automate.



Grammar, Style and Spell Checker

LanguageTool checks your writing in more than 20 languages

LanguageTool offers spell and grammar checking. Just paste your text here and click the 'Check Text' button. Click the colored phrases for details on potential errors. or use this text too see an few of of the problems that LanguageTool can detecd. What do you thinks of grammar checkers? Please not that they are not perfect. Style issues get a blue marker: It's 5 P.M. in the afternoon. The weather was nice on Thursday, 27 June 2017.

Figure 2.1. Example usage of an online LanguageTool at "languagetool.org"

2.2.1. General description

In figure 2.1 there is an example of a text with errors detected by LanguageTool. This tool is a rule-based, which means it wouldn't handle out of vocabulary entries. Here, a clear typo of "detecd" was detected. LanguageTool suggests "detect" as a correction.

Confusion sets are sets of word pairs, where each pair defines a word that can be easily misspelled and its correction. Although some of the errors can be detected by using the dictionary and a large file of confusion sets, not every real word error can be detected in this manner. Some errors, like the one in the sentence "Please <u>not</u> that they are not perfect" can only be detected using n-gram data.

2.2.2. n-gram data

As LanguageTool is easily extendable, it allows users to use n-gram data from any language, but unfortunately, easily available large n-gram data sets are only available for few languages. The biggest raw dataset is provided from the Google n-gram corpus. For example the 3gram data for the English language only takes about 1400GB of compressed space!

2.2.3. Docker API

To evaluate the GED abilities of LanguageTool, there was a need to test lots of sentences. The very first version of the LanguageTool has been written in 2003 by Daniel Naber in a Python as a part of his diploma thesis, but currently this project runs on Java and has been developed by many developers. While online tools are very good to quickly test the correctness of a short text, they are limited to the length of the text as well as a number of requests. Although it is possible to make web requests, Developer API calls are expensive, with a monthly fee of 150zł for 250 API calls/day.

For the reasons described above, the most reasonable thing to do was to set up a Docker instance of a LanguageTool, exposing the tool on one of the ports. An example query for the text "Peguins live on antarctica.":

```
url -X POST --header 'Content-Type: application/x-www-form-urlencoded'
--header 'Accept: application/json'
-d 'text=Peguins%20live%20on%20antarctica.&language=en-US&enabledOnly=false'
'https://languagetool.org/api/v2/check'
```

would get the response shown on the listing 2.1.

```
{
  "language": {
    "name": "English (US)",
    "detectedLanguage": {
      "name": "English (US)",
      "confidence": 0.891 }},
  "matches": [
{
  "message": "Possible spelling mistake found.",
  "replacements": [{"value": "Penguins"}],
  "offset": 0,
  "length": 7,
  },
  "sentence": "Peguins live on antarctica.",
  "rule": {
    "id": "MORFOLOGIK_RULE_EN_US",
    "issueType": "misspelling",
    "category": {
      "name": "Possible Typo" }},},
{ "message": "Possible spelling mistake found.",
  "replacements": [
    { "value": "Antarctica" },
    { "value": "Antarctic" },
    { "value": "antarctic" },
    { "value": "antarctic a" }
  ],
  "offset": 16,
  "length": 10,
  "sentence": "Peguins live on antarctica.",
  "rule": {
    "id": "MORFOLOGIK_RULE_EN_US",
    "issueType": "misspelling",
    "category": {
      "name": "Possible Typo"}},}]}
```

Listing 2.1. Important parts of JSON response from LanguageTool

LanguageTool not only informs the user about possible replacements, but it also gives pieces of information about the type of the error as well as rules that detected those mistakes. For the GED task it is sufficient to mark the wrong words, no matter what the replacements were.

Chapter 3

Experimental setup

The daunting problem of many young scientists in NLP research is related to the resources needed for computing models. The computation of many NLP models is both CPU and GPU exhaustive. For those reasons it was important to choose the right machine for the computations.

The author of this thesis decided to go with the portable Linux machine for the lighter tasks, and for the remote Arch Linux server to run all computationally expensive operations. The inquired server was accessible thanks to the courtesy of Dawid Jurkiewicz. It was equipped with "Intel^{R} Core^{TM} i7-5930K" CPU along with "GeForce GTX 1060 6GB" GPU.

The choice of the Linux operating system along with a large 6GB GPU was dictated by the sizes of the models. While a larger GPU with at least 8GB would be sufficient to fit all the models provided with UEDINMS and GPT2, the costs of such a machine prevented the author from using them for this thesis. The 6GB GPU card had enough size to work with 80% of the models — the remaining 20% of the models accounted for 80% of the computing time and were run on a much slower CPU.

3.1. System overview

3.1.1. Advantages of the Arch Linux operating system

The motivation behind the system of Arch Linux was that it is a lightweight and easily configurable distribution of Linux. It's a living system that evolves as time goes on. As such, it embraces the notion of rolling releases, as well as updates and minor revisions. This was necessary when working with the state-of-the-art GPT-2 system, which has been constantly improved upon, at the same time the author was working on a dissertation.

Linux has many advantages, among which are:

1. It has great scripting language — GNU Bash — which comes with many useful tools for text processing, command-line processing, and debugging. These include tools such as grep, sed, sort, count, find tar, and xargs. The tools are not only useful, but also come with great productivity-enhancing possibilities, so there is no need to constantly reinvent the wheel. It also has a good documentation.

2. Low-level development (fewer software dependencies) — that is, very little to no user configuration is needed in most cases. The majority of the required tools are already stored on the computer. Moreover, users can use Pacman package manager to install all the necessary libraries and tools with a single command. A programmer will find that most programs run out of the box, and when they don't — there is a large community of Arch Linux users that are willing to help.

3. The most important characteristic of Linux, which is unique to it, is its ease of automation for most of the tasks needed in NLP. Low memory footprint and low hardware requirements of Arch Linux provide important scalability of the project. A single CPU can perform well for word-level language processing tasks and can handle much larger datasets combined with a CUDA-based GPU.

4. Linux comes with direct support for ipython, git, and docker. It's important because ipython gives instant feedback on the tested programming code. Git ensures the ease of versioning and sharing the code across multiple computers. Docker on the other hand is necessary to ensure the project will work on any machine, regardless of the configuration.

Those tools make the infrastructure of neural language processing easy and nimble. Many of them are also accessible on Windows, but as mentioned before it would require a lot more configuration, and the high memory consumption of the Windows operating system will leave less of the memory for the NLP models.

3.1.2. System architecture

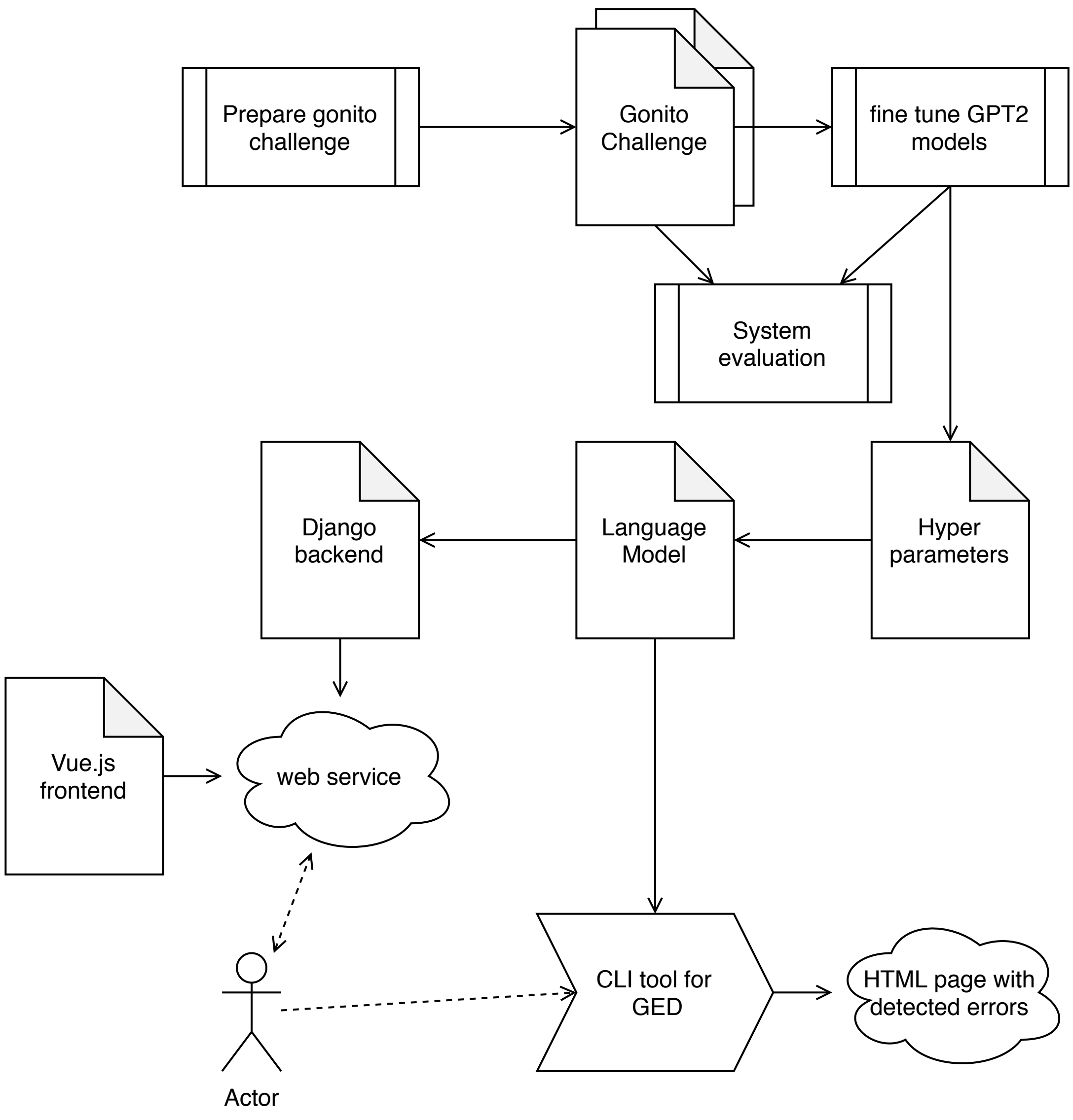


Figure 3.1. The general overview of the architecture used for this masters thesis.

In this section, the overall architecture of the system will be described.

Outline of the system

The main task of the system is to assist in writing English texts for non-native speakers. When designing such a system one has to consider the types of errors that would be detected and whether the task of correcting those errors relies on the system or on the user's end. If the system would be the one to correct the sentence, it is important to decide how far from the original sentence can it go to achieve acceptable sentences in an aspect of semantics, grammar, or punctuation.

Moreover for the solution to find its use in real life, it should be placed in an environment that is easy to access for the target group of users.

The general architecture

The architecture of this system is shown in figure 3.1. In this section, the general overview of this architecture will be described. The detailed description of those components will be shown in the following sections.

The first component of this system is related to the creation of a test-set. With the right data, a script prepares a challenge on the "Gonito.net". Gonito.net is a platform for evaluating machine learning tasks [10] [11], and will be described later in this chapter. This challenge is then used to fine-tune GPT-2 based language models. All the language models are evaluated and compared using the latest gonito challenge.

In the fine-tuning process, best hyperparameters are found. They serve as the key component of a GPT-2 based language model. There are two services that allow users to interact with the model. First, there is a web service, that has a Django backed and a Vue.js frontend. The backend is querying the language model. The second way to interact with this model is by the CLI tool, which will highlight the errors and create an HTML page displaying those errors.

3.2. Test set preparation

The foundation of all research is good data. There is a quote from Nick Harkaways book "The Gone-Away World" [13] stating:

Garbage in, garbage out. Or rather more felicitously: the tree of nonsense is watered with error, and from its branches swing the pumpkins of disaster.

This quote, repeated on many data science courses and in many books states an important truth — One cannot train or fine-tune a great model with a bad dataset. That's why there has been a focus on finding the right dataset. The data described in the next paragraph has been used to build plenty of models, including UEDINMS mentioned in section 2.1.

3.2.1. About the dataset

The Building Educational Applications (BEA) 2019 Shared Task on GEC, introduced by Bryant et. al [5] is a successor of the Conference on Natural Language Learning (CoNLL) 2014 shared task. BEA-2019 introduces a new annotated dataset, and aggregates few other datasets in one place.

Between all the datasets provided with BEA-2019 shared task, authors of this thesis decided to stick with two: "The Cambridge English Write & Improve (W&I) and LOCNESS" corpus, later referred to as W&I+LOCKNESS, and "The First Certificate in English" corpus, later referred to as FCE.

W&I+LOCKNESS corpus is composed of W&I corpus and LOCKNESS corpus. They differ in the source of data. W&I is a set of sentences harvested from the essays submitted to the Write & Improve online web platform [1]. To ensure the quality of the data, W&I corpus has been filtered from the essays that met any of the following three conditions:

1. The text contained fewer than 33 words,
2. More than 1.5% of all characters in the text were non-ASCII,
3. More than 60% of all non-empty lines were both shorter than 150 characters and did not end with punctuation.

Those manually chosen conditions ensured the text was cleaner. In the end, there were 3600 annotated submissions chosen. Each submission has been sent to a human annotator. On the other hand, the LOCKNESS corpus was designed to introduce native errors.

[1]Their website is accesible here: https://writeandimprove.com/

British and American undergraduates created 400 essays that were filtered, annotated, and added to the W&I corpus. The FCE corpus contained 1244 written answers to the exam's questions.

All the corpora were tokenized using SpaCy v1.9.0. All in all, there were 76311 sentences, divided into test set, development set, and train set with the proportions 10%, 10%, 80% respectively. They contained texts written by learners of levels A, B, C, and native (N). The sentences were stored in M2 format files. An example sentence in an M2 could be seen in a figure 3.2.

```
S This are a sentence .
A 1 2|||R:VERB:SVA|||is|||-REQUIRED-|||NONE|||0
A 3 3|||M:ADJ|||good|||-REQUIRED-|||NONE|||0
A 1 2|||R:VERB:SVA|||is|||-REQUIRED-|||NONE|||1
A -1 -1|||noop|||-NONE-|||REQUIRED|||-NONE-|||2
```

Figure 3.2. The example of M2 format [5]

3.2.2. Dataset preprocessing

The data preprocessing has been written as an automated bash script "prepare_challenge.sh". Its workflow is shown in figure 3.3.

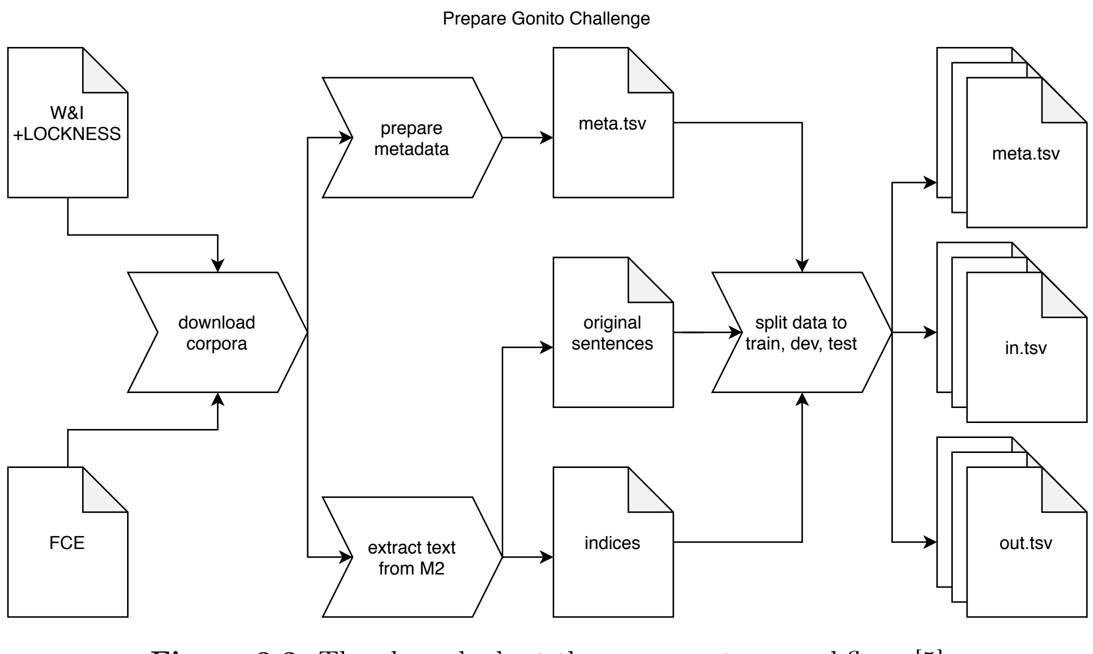


Figure 3.3. The closer look at the prepare stage workflow. [5]

The first step to prepare data is downloading the relevant corpora — W&I+LOCKNESS and FCE. This data is in the M2 form. Those two corpora are joined into one file for further processing. The next two scripts are performing separate operations. The first script denoted with "prepare metadata" text prepares meta information per sentence, where each line defines the proficiency level of the student writing that sentence. The second script, named "sent_from_m2.py" is responsible for extracting information from the M2 file. M2 format saves each correction in a separate line, so in order to get the corrected sentence, a script had to apply all the corrections on top of an erroneous sentence. While the script was designed to create files with original sentences, corrected sentences, and indices denoting where the errors were, for the final Gonito challenge only original sentences and their indices were needed.

Alignment

The script "sent_from_m2.py" played a crucial part in data preprocessing. It not only created sentence and indices, but it also imposed the alignment of sentences. Sometimes there may be few correct alignments, so special attention has been paid to choose the right algorithm. The differences, clearly shown by Felice et. al. in [8] are visible in a figure 3.4. After a few experiments author decided to stick with the Damerau-Levenshtein distance metric. The code that does alignment is visible on the listing 3.1.

(a)	This	wide	spread	propaganda	benefits	only	to	the	companys	.
	This	widespread	publicity	only	benefits	their	companies			.
(b)	This	wide	spread	propaganda	benefits	only	to	the	companys	.
	This	widespread		publicity	only	benefits		their	companies	.

Figure 3.4. Differences between (a) standard Levenshtein and (b) linguistically-enriched DamerauLevenshtein alignment. [8]

After choosing the right indices, the data was split into three test sets — one for evaluation, one for testing and one for development.

3.3. Evaluation system - Gonito.net

Gonito.net is a project written by Graliński et al [10] and further improved by Graliński et al [11]. It is a web-based platform for evaluation of machine learning tasks that is

```
def get_indices(orig_toks, cor_toks, policy = "levenshtein"):
orig = applySpacy(orig_toks, nlp)
cor = applySpacy(cor_toks, nlp)
if policy == "levenshtein":
    alignments = WagnerFischer(orig_toks, cor_toks, orig, cor,
                              substitution=AlignText.lev_substitution,
                              transposition=AlignText.lev_transposition )
else: #Damerou-Levenshtein
    alignments = WagnerFischer(orig_toks, cor_toks, orig, cor,
        substitution=AlignText.token_substitution)
alignment = next(alignments.alignments(True))
edits = AlignText.get_edits_split(AlignText.get_opcodes(alignment))
indices = []
for edit in edits:
    if edit[1] == edit[2] and edit[1] < len(orig_toks):
        indices.append(edit[1])
    else:
        indices.extend(list(range(edit[1], edit[2])))
return sorted(set(indices))
```

Listing 3.1. Alignment using Damerou-Levenshtein metric

widely used on the University of Adam Mickiewicz as well as for business-related projects. While it originated as a platform for NLP problems, all machine learning challenges can be organized with its help. Gonito.net was the main platform that allowed an organized evaluation of the project described in this master's thesis.

The challenge prepared for the evaluation of this project is available at Gonito.net platform [2]. The main advantage of the Gonito platform is the fact of it using Git. The first step of creating a challenge is making the repository that will host a challenge. All data files created in the "dataset preprocessing" step should end up here. Moreover a challenge should contain *README.md* file describing the challenge, the Directory structure along with additional information, and description of files. Another required file is *config.txt*. It defines the precision and the metrics used for the evaluation of the solutions. It may, for example, look like this (an actual config file of this challenge):

```
--metric Mean/MultiLabel-F0.5
--metric Mean/MultiLabel-F2
--metric MultiLabel-F0.5:P<2>N<F0.5>
--metric MultiLabel-F2:P<2>N<F2>
--metric Accuracy:s<^(\d+)(\s\d+)*$><\1>P<2>N<AccFstError>
--metric Accuracy:s<^(\d+)(\s\d+)*$><WRONG>P<2>N<AccAnyError>
```

[2]https://gonito.net/challenge-readme/grammatical-error-detection

```
--precision 4k
```

Last but not least a challenge may contain some additional files. This challenge contains utility script “detokenizer.py” which allows users to reverse SpaCy tokenized sentences used in the challenge.

After submitting the challenge, the system will automatically detect $test-A$ directory and hide *expected.tsv* files from future users. Git allows authors of the challenge to make revisions of it, correct mistakes, and submit improved versions of the same challenge without changing the address.

Users can submit solutions to a challenge in the form of a git commit. Each commit, along with its commit message and gonito.yaml file defines the submission to a challenge. They also provide useful information about submission, such as the model used for evaluation as well as the hyper-parameters. The example gonito.yaml file is visible below.

```
description: gpt2 oddballness
tags:
      - oddballness
      - left-to-right
      - gpt2-xl
params:
      - optimizedfor: AccAnyError
      - alpha: 1.15
      - threshold: 0.484375
```

3.4. System environment

NLP system that uses various tools and resources — from graphics card(if available) to multiple python packages, which work only with a specified version of python — might prove hard to reproduce. Fortunately there are measures which help to automate the process of configuring the system.

3.4.1. docker container

One of the ways to automate the configuration of a system is a docker container. Docker is an OS-level virtualization program that offers software in the form of containers. It allows us to bypass the problems associated with the installation of individual versions of the packages and libraries used in the described project. Thanks to this solution, the programmer can use the system in a container from the same image without manually doing the job. The images can be installed on any kind of machine. The environment and all of its configuration are run in the container.

The repository which has scripts for creating the container along with accompanying files. It contains a *Dockerfile* file that defines the system and a *run.sh* script, which builds the container and enables command-line access to it.

Chapter 4

Algorithm

In this chapter the reasoning behind choosing the Huggingface will be explained, as well as the reasoning behind a new measure called oddballness, which was invented by Filip Graliński in cooperation with the author of this thesis.

4.1. Core model - Huggingface GPT-2

4.1.1. Motivation

There is an increasing interest in NLP research. It is visible in the growing number of papers submitted to conferences. New models and algorithms are being developed all the time. GPT-2 is a state-of-the-art model in which he seemed to present great opportunities in the field of GEC. Huggingface is an organization providing multiple NLP architectures with an interface in TensorFlow 2.0 and PyTorch. With that in mind the author of this thesis chose GPT-2 and BERT models from Huggingface to explore the capabilities of GED.

4.1.2. Availability of the model

As GPT-2 came into the picture, it's language generating abilities were so good, that it was feared it would present a real threat to society in the form of fake websites, computer-generated spam, etc. That was the very reason why at first researchers didn't share the fully trained model. Because it was unclear if they would share it in the future, people tried to reproduce their trained model. The only problem were costs for such training, which could be estimated at around 50 000$

Huggingface was one of the first to provide a GPT-2 small model, as well as the bigger one. Although Huggingface was convenient, during the course of writing this thesis they changed the interface of the code. Those changes made it necessary for the author to rewrite perfectly fine bits of code to account for changes in function names and packages used. It meant an author had to be working on a bleeding-edge project, and there was no clue how many times the code would have to be adapted to the changes in the model.

4.2. Oddballness

This section is written based on the paper on Oddballness by Graliński et. al. that hasn't been published yet. Oddballness is a metric which measures how odd or unusual is a particular event. The assumption is, that not every event with a low probability is odd. A simple example would be meeting a penguin in the zoo in Europe. Although they are rarely kept in captivity, and few zoos have them — no one would be surprised to meet a penguin in the zoo. On the contrary, if you were to meet a penguin on the street in Europe it would be an unusual event.

4.2.1. Oddballness measure

The odballness measure is defined with help of the ReLU function described in subsection 1.4.4. Given a discrete probability distribution $D = \{p_1, p_2, p_3, \ldots\}$ the oddballness is defined as:

$$\xi_D(p_i) = \sum_j g(ReLU(p_j - p_i)), \tag{4.1}$$

where g is any function that is monotonic and continuous for which $g(0) = 0$ and $g(1) = 1$. In the experiments done as part of this thesis, the $g(x)$ function depends on the additional parameter α:

$$g(x) = x^\alpha \tag{4.2}$$

As discussed earlier, low probability not always is a premise for an odd event. The same could be said in terms of words — a word with a low probability is not necessarily odd (or incorrect) — it might just be rare. Hopefully the oddballness presents a more useful measure to decide whether a word is correct or not.

4.2.2. Axioms of oddballness

Equation 4.1 has been created to satisfy few common-sense axioms which are quoted from the paper mentioned at the beginning of this section:

(O0) $\xi_D(p_i) \in [0, 1]$ – let's assume our measure is from 0 to 1,

(O1) $\xi_D(0) = 1$ – if an impossible event happens, that's pretty oddball!

(O2) for any distribution $\xi_D(\max\{p_i\}) = 0$ the most likely event is not oddball at all

(O3) $p_i = p_j \to \xi_D(p_i) = \xi_D(p_j)$ – all we know is a distribution, hence two events of the same probability must have the same oddballness (within the same distribution),

(O4) $p_i < p_j \to \xi_D(p_i) \geq \xi_D(p_j)$, if some event is less likely than another event it cannot be less oddball,

(O5) (smoothness) for any distribution $D = \{p_1, p_2, p_3, \ldots\}$, the function $f(x) = \xi_{D_x}(x)$, where $D_x = \{x, p_2 \times \frac{1-x}{1-p_1}, \ldots, p_i \times \frac{1-x}{1-p_1}, \ldots\}$, is smooth – if we change the probabilities a little bit, the oddballness should not change much

4.2.3. Examples

The power of the oddballness measure could be shown in a few examples. First, let as assume that we have some words with a given distribution:

$$D_1 = \{p_1 = \frac{1}{1000}, p_2 = \frac{1}{1000}, \ldots p_{1000} = \frac{1}{1000}\},$$

then, even though all words are equally unlikely, the oddballness for any word is equal zero.

$$\xi_{D_1}(p_1) = 0.$$

On the contrary, given the distribution:

$$D_2 = \{p_1 = \frac{1}{1000}, p_2 = \frac{999}{1000}\},$$

then the oddballness would differ greatly, even though p_2 in D_2 has the same probability as any event from D_1.

$$\xi_{D_2}(p_1) = 0.998$$

$$\xi_{D_2}(p_2) = 0.$$

Assuming that some architecture has a very good language model, the oddballness should be able to detect errors in the text with great precision.

4.2.4. Oddballness in Grammatical Error Detection

As mentioned in the previous subsection oddballness measure can be used for the task of GED. For any sentence, we can detect errors by applying oddballness to each word, and filtering results that are higher than a given threshold t.

For the error detection model ED based on the oddballness to work, there are few requirements. First, there is a need for a good language model Lm that can generate a probability distribution D for any word w_i in the sentence. Then this error detection model needs to be fine-tuned for the best threshold t and the exponent α. The quality of the error detection model is still very much dependent on the quality of the language model.

$$w_i \in Correct \iff \xi_D(w_i) < t \tag{4.3}$$

In this master's thesis, the oddballness was computed with the help of the GPT-2 language model. It was fine-tuned to find the best values of t and α, and used as a model for grammatical error detection. GPT-2 model assigns probability distribution to subword units, so in order to get oddballness per word, the results were aggregated with a $max()$ function.

4.3. Bidirectional oddballness

The GPT-2 language model is powerful, but it comes at a cost. Not only are the computing costs high (chapter 3), but also the model was trained with the context on the right side being masked as mentioned in section 1.6.4. This meant that the model is only able to detect errors that are directly the cause of the previous words and is incapable of detecting complex errors that depend on the context of the following sentences.

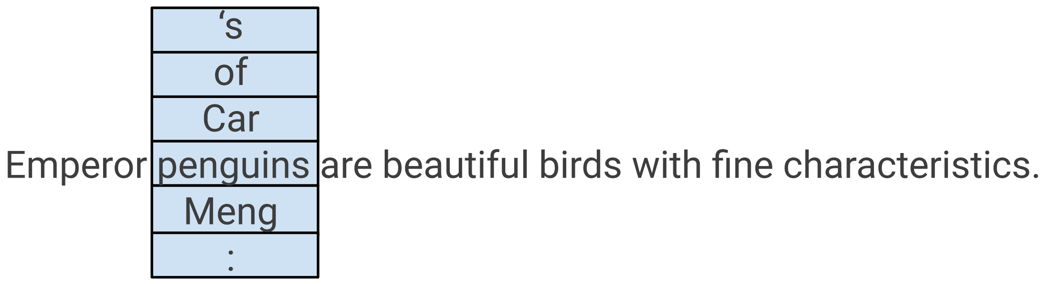

‘s
of
Car
Emperor penguins are beautiful birds with fine characteristics.
Meng
:

Figure 4.1. The example of a sentence that could be corrected with help of the bidirectional oddballness.

A good example of a sentence that could be corrected having known the context of the following words is “Emperor penguin are beautiful birds with fine characteristics”. It is illustrated on the figure 4.1. In the regular model the word “penguin” has a lower probability of appearing after “emperor” than the remaining options:

$$Options \in [\text{“}'s'', \text{“}of'', \text{“}Car'', \text{“}Meng'', \text{“ }:''].$$

In this scenario, even if the system would detect that there is an error in the word “penguin”, the other subwords might have substantially higher probability.

To accompany the lack of the right context, the bidirectional approach has been developed. Bidirectional oddballness with GPT-2 model is a way of computing oddballness measure that takes into account the whole context of the word. To compute the bidirectional oddballness, first, the probability of an original sentence needs to be calculated:

$$P(s) = \prod_{i=0}^{n} p(w_i)$$

where n is the number of words in a sentence s. Then there needs to be defined a probability distribution:

$$\forall w_i \in s \exists D_i \Big[D_i = \{P(s), \frac{p(c_{i,0}) \cdot \prod_{j=0}^{n} p(w_j)}{p(w_i)}, \ldots, \frac{p(c_{i,N}) \cdot \prod_{j=0}^{n} p(w_j)}{p(w_i)} \} \Big]$$

where N is the number of alternative words analized, and $c_{i,j}$ is the $j-th$ most probable alternative option (correction) for the word w_i. Each word w_i has bidirectional oddballness $\xi_{D_i}(P(s))$ computed by using P(s) as the probability of the word, and D_i as the probability distribution for the remaining N words.

Chapter 5

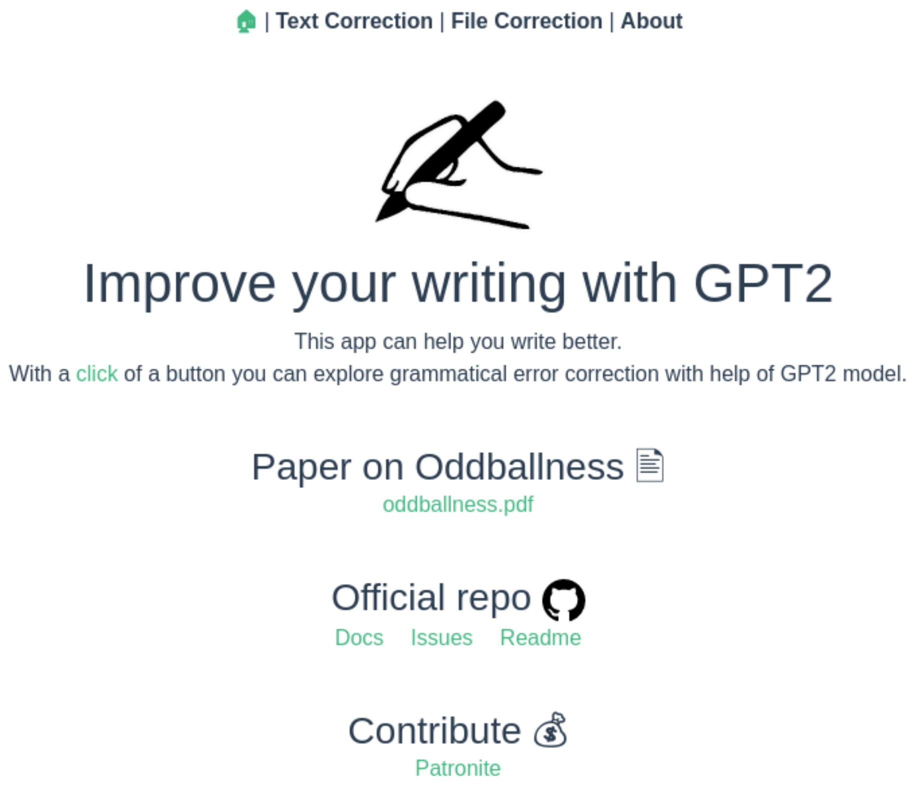
Evaluation - Master's project

In the chapter 1 GPT-2 was described. It was later extended with mentions of the Huggingface architectures in the chapter 4. In that chapter the oddballness was also introduced as a measure that can benefit from good language models like the GPT-2. This master's thesis aims to enhance the writing of English texts for non-native language speakers. The main measure in which it improves writing is GED.

In this chapter, first the elements of the language model will be described. In the subsection 5.1.1 the backend of this master's thesis is described. The subsection 5.1.3 is devoted to the visible part of the application — the frontend written in Vue framework. Last but not least, the subsection 5.1.2 describes the core engine that performs the detection of erroneous words.

The section 5.2 describes how the evaluation was automatized with the help of a docker container described in subsection 3.4.1.

The last section is crème de la crème of this master thesis. It is devoted to describing the evaluation results compared to the solutions described in chapter 2. They were computed on a challenge which was hosted on Gonito.net platform described in section 3.3



| Text Correction | File Correction | About

Improve your writing with GPT2

This app can help you write better.
With a click of a button you can explore grammatical error correction with help of GPT2 model.

Paper on Oddballness
oddballness.pdf

Official repo
Docs Issues Readme

Contribute
Patronite

Figure 5.1. “Home” tab of the correction tool.

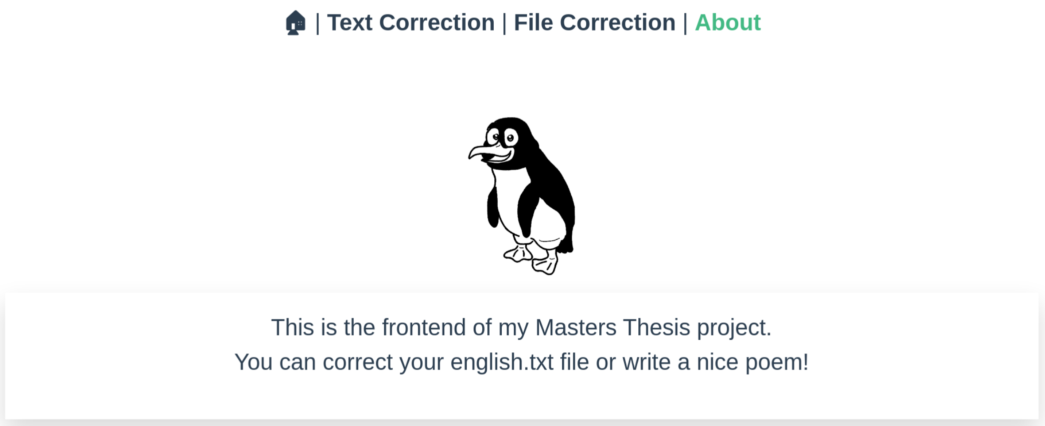
5.1. Elements of the Language Model/Masters project

5.1.1. Django backend

Masters thesis project uses Django 2.2 as a pure backend with the help of the Django REST framework. The backend offers one simple endpoint “/search/” which awaits for POST requests. It requires JSON data with two fields:

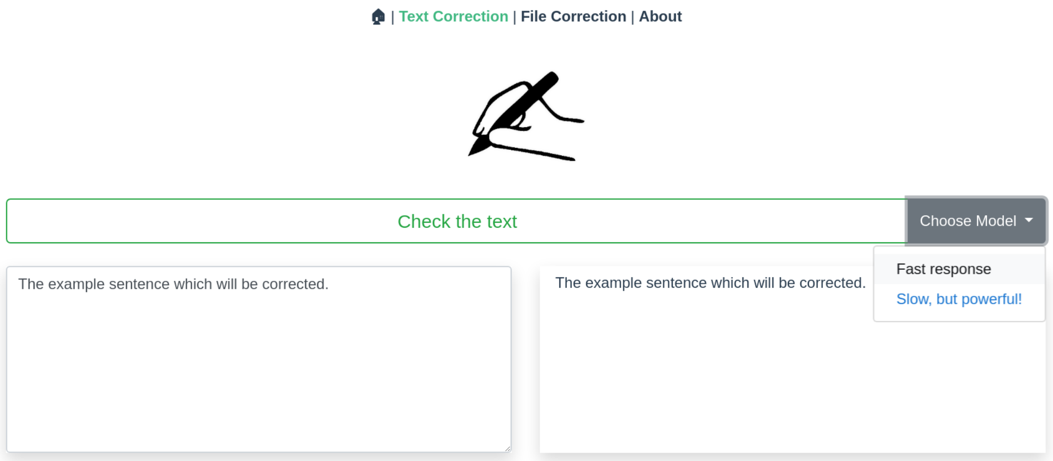
```
{
queryText: "The text that needs to be corrected",
modelType: "type of the model"
}
```

where `queryText` can be any text without constraints for length, and `modelType` is either “left-to-right” or “bidirectional”. The response contains an array of token objects, where



This is the frontend of my Masters Thesis project.
You can correct your english.txt file or write a nice poem!

Figure 5.2. “About” tab of the correction tool.



The example sentence which will be corrected.

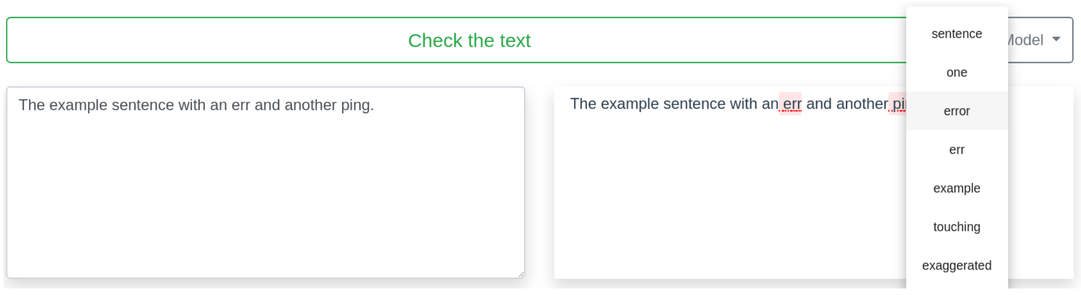
The example sentence which will be corrected.

Figure 5.3. “Text correction” tab of the correction tool.

one token object has the following structure:

```
{
name: "alice",
probability: 0.31,
oddballness: 0.03,
corrections: ["Alice", "has", "a", "cat"],
underline: false
}
```

where `name` is the BPE subword of the token, `probability` and `oddballness` are measures computed by the language model, `corrections` are an array of n most probable corrections and an `underline` is a boolean value that informs whether the oddballness was above the threshold.



The example sentence with an err and another ping.

The example sentence with an err and another pi

Figure 5.4. "Text correction" tab of the correction tool.

Although the backend is relatively simple as it just calls Language Model described in subsection 5.1.2 it is built with scalability and security in mind.

5.1.2. PyTorch engine

The core of the project has been written in Python 3.7. The engine features an abstract interface representing the general language model based on transformers architecture (subsection 1.6.2). It was adjusted for use with any model provided by the Huggingface and has been tested with the ones most important for the project - GPT-2 (subsection 1.6.4) and BERT (subsection 1.6.3).

This abstract class computes the oddballness and takes care of long chunks of text by splitting them to at-most 1024 characters. The children classes "Gpt2OddballnessEngine" and "BertOddballnessEngine" inherit from this abstract class and implement implementation-specific functions. The GPT-2 class offers regular oddballness as well as bidirectional oddballness. The BERT class is bidirectional by design and its regular oddballness doesn't need enhancing. The former one isn't included in the frontend due to low performance on evaluation, later described in section 5.3.

5.1.3. Vue frontend

The frontend is a web interface written in Vue 2.6.10. It was written to present the concepts of oddballness in GED. The home page is shown in figure 5.1. The whole design aims at simplicity and ease of use. The interface consists of four sections — namely "Home", "Text Correction", "File Correction", and "About".

The "About" section has a short description of the project as shown on the figure 5.2. The "File Correction" tab allows users to load a text file and correct it. After loading the file it enables the same functionality as the most important tab which is “Text Correction”. The “Text Correction” tab is shown on the figure 5.3. The field on the left is an editable *textarea* which stores the text. After clicking the big green “Check the text” button, the same content appears on the right with errors highlighted in red. The proprietary solution of clickable subwords shown in figure 5.4 allows users not only to detect errors but also to correct them easily.

Although the frontend leaves a lot of room for improvement, it is sufficient for the illustration purposes of concepts shown in this master’s thesis.

5.2. Evaluation automatization

During the evaluation of all the models required for this master’s thesis, there were 150 submissions. Because at the beginning it was hard to predict the final number of submissions, the first few of them were made manually. When it turned out that the cost of creating semi-automatic evaluations was too high, it was decided to fully automate this process. The architecture of the evaluation automatization is described on the figure 5.5 which is a zoomed-in fragment of “System evaluation” box from the figure 3.1.

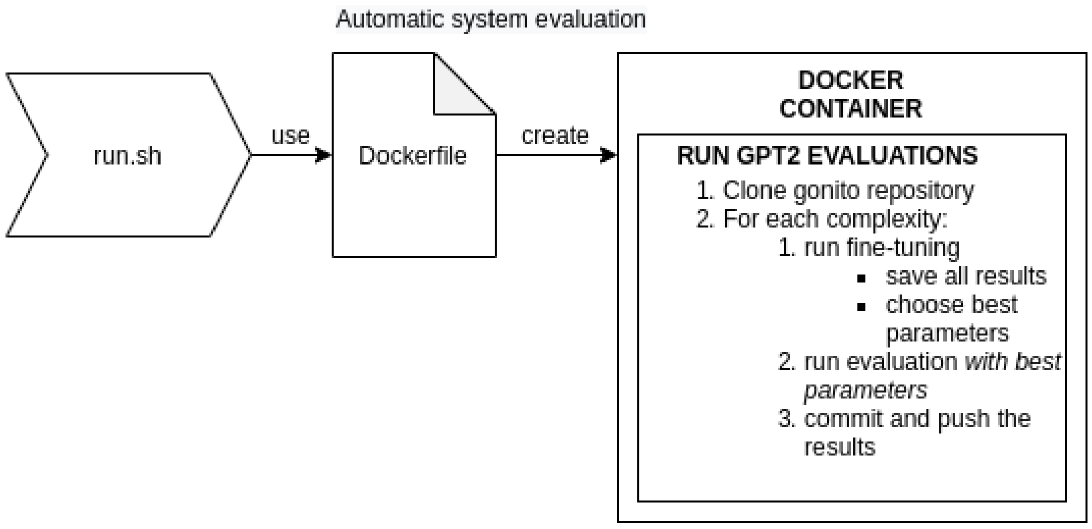


Figure 5.5. The general overview of the automatization of evaluation.

The docker container which provides the environment for the whole evaluation ensures the evaluations could potentially run on any machine, with minimal requirements of bash and docker being installed.

```
compute_model($\alpha$):
    for each sentence:
        normalize sentence
        sentence_data ← compute oddballness and probability
        return sentence_data
for each metric:
    T ← find best threshold
    I ← get indices of erroneous words
    S ← get score of metric for T
    return S, I, T
```

Listing 5.1. The pseudo-code of the inference for models based on huggingface architecture

5.2.1. Fine-tuning and inference scripts

"RUN GPT2 EVALUATIONS" script from figure 5.5 is fairly simple except for the steps 2.1 and 2.2. The fine-tuning script is a simple code written in bash, which in turn calls an inference script with different parameters as well as saves each result of the inference to a file. The evaluation script differs very little from the Fine-tuning. While the inference script varies from model to model, all of them inherit from the abstract inference class and follow a similar logical structure. The pseudo-code of the inference is shown on the Listing 5.1.

Each sentence is computed separately by the engine. Before sending the raw sentence to the engine, it is normalized. The preprocessing includes: cleaning out non-unicode characters with $unidecode()$ function, detokenization described in subsection 1.3.3, stripping out the surrounding spaces, and tokenizing again with simple pythonic $split()$. This ensures that the model will receive the text in the form it was trained with. For each parameter α, and for each subword unit the engine computes oddballness and probability values. The subword units often have different oddballness values among one word and are returned in a bunch for the whole sentence, so the sentence needs to be reconstructed. It turned out that the best way to aggregate different oddballness values form subword units is to use the $max()$ function.

Given the model computed oddballness values per word for every sentence in the data, it tries to optimize the threshold value. The threshold defines the cutoff for the correct words. It is also used to return the indices to erroneous words, as the output for the gonito.net task defined in section 3.3. For instant feedback, the score is also computed right away with the development test-set.

5.2.2. Evaluation scripts

The evaluation process is similar to the one described in subsection 5.2.1. The only difference is that now the algorithm has predefined values of α and *threshold*. It also has no insight into the expected values. The output from the model is then evaluated by the gonito.net platform.

5.3. Evaluation results

The gonito.net platform described in section 3.3 proved to be an unquestionable help in comparing and evaluating GED models. There were several models compared:

- UEDINMS (described in section 2.1)
- LanguageTool (described in section 2.2)
- oddballness with GPT-2 (described in subsection 1.6.4)

  GPT-2-small

  GPT-2-medium

  GPT-2-large

  GPT-2-XL

  GPT-2-XL with bidirectional oddballness (described in section 4.3)
- oddballness with BERT (described in subsection 1.6.3)
- probability with GPT-2

Both UEDINMS and LanguageTool models were taken with no modifications. The output from the UEDINMS is a proposed correction of the sentence. This means that there was a need to transform the output from words into the indices needed as the output in the gonito challenge. It was done with the same scripts which were used in dataset preprocessing (described in subsection 3.2.2). In the case of LanguageTool, the model returns JSON in the format similar to the one shown in figure 2.1. It was assumed, that any word with at least one correction suggestion classifies as the erroneous index in the final output.

All the models based on the GPT-2 shared a similar structure as was shown in subsection 5.2.1. The "probability with the GPT-2" model was based solely on the probability output from the model. Engine assigned the probability to each word based on the language model. The inference scripts then found the threshold which defined, what cut-off is best for the erroneous words. The remaining GPT-2 models were based on the oddballness. The main advantage of the oddballness is the fact that it allows us to take into an account probability distribution for the given word. It somewhat encodes new information in one simple number that is in the range between 0 and 1.

The author of this thesis also experimented by combining oddballness with the BERT model. Unfortunately, the results were on a similar level as a randomized baseline.

The results are shown on table 5.1

Model name	parameters			F-0.5		F-2		Error Accuracy	
	α	t	c	Mean	Total	Mean	Total	First	Any
UEDINMS				0.839	0.873	0.8057	0.758	0.816	0.892
Languagetool				0.398	0.26	0.372	0.143	0.377	0.599
GPT-2 XL oddballness	1.05	0.8125		0.422	0.288	0.386	0.1469	0.377	0.632
GPT-2 large oddballness	1.05	0.8125		0.414	0.276	0.380	0.135	0.373	0.611
GPT-2 large bidirectional	1	1	20	0.356	0.003	0.356	0.0008	0.356	0.357
GPT-2 large probability	1	0		0.119	0.133	0.273	0.371	0.063	0.644
	1	0.093		0.088	0.052	0.103	0.096	0.063	0.673

Table 5.1. Results of the experiments

It is prominent, that UEDINMS had the best results in all the available metrics. It was no surprise, as UEDINMS is a Transformer network purposefully trained for the task of GEC, and adjusted here for GED. Languagetool — the second model that results were compared to — scored (on average) lower than the optimized oddballness model. The probability with GPT-2 model version had results hardly distinguishable from random — the threshold was adjusted to 0, which implies that the model tried to output all the indices as erroneous. This behavior may imply that it is more profitable for the algorithm to mark everything as an error than to try at all.

Results also confirm the theory stated in chapter 4, that a good language model is essential for the oddballness to work. The bigger the language model, the better the results.

Unfortunately, some experiments were too expensive to run on the biggest GPT-2 XL model because — as mentioned in chapter 3 — the environment had a graphics card that wasn't big enough to store the model thus the computations could sometimes take 20 days or more for a single model.

Conclusions

The goal of this thesis was to explore the computer-assisted writing of English texts for non-native speakers. First the basic concepts of Neural Language Models were described. Basic concepts of text manipulation methods led through machine learning paradigms to complex models used currently in NLP. Next, related works were described in order to compare them with the proprietary solution described in this thesis. The following chapter described the architecture used for the evaluation of the thesis as well as running the webserver. Chapter 4 described the oddballness measure invented by the thesis supervisor and developed in the team with the author of this thesis. It was extended with the description of bidirectional oddballness. Last but not least masters thesis project was described.

The project development started with finding the right data for the gonito challenge. The first versions of the engine were used with a simple script that created an HTML with errors highlighted in the text. Then frontend and backend were quickly developed. With the basic components the evaluation allowed choosing the best parameters as well as comparing the model with UEDINMS and Languagetool.

It came out that the fully unsupervised model of GPT-2 with oddballness was sufficient to achieve better results than Languagetool. To no surprise UEDINMS proved to have better results than any other tested model — it featured a huge Transformers model that was explicitly trained for the task of GEC.

The improvements in writing English text are noticeable. The interface is intuitive and has as few buttons as possible. It could be used to enhance the writing experience. This tool has immense potential due to the dynamically developing field of NLP. The ideas for improvement include enhancing the translation module for the words with high oddballness. GPT-2 model could be easily used to give users a prediction for the next word as well as for filling the gaps in places where users forgot which word to use.

Appendices

APPENDIX A:

Before		After
a	$\rightarrow$	A
b	$\rightarrow$	B
c	$\rightarrow$	C
d	$\rightarrow$	D
e	$\rightarrow$	E
f	$\rightarrow$	F
g	$\rightarrow$	G
h	$\rightarrow$	H
i	$\rightarrow$	I
j	$\rightarrow$	J
k	$\rightarrow$	K
l	$\rightarrow$	L
m	$\rightarrow$	M
	...	
N	$\rightarrow$	N
O	$\rightarrow$	O
P	$\rightarrow$	P
Q	$\rightarrow$	Q
R	$\rightarrow$	R
S	$\rightarrow$	S
T	$\rightarrow$	T
U	$\rightarrow$	U
V	$\rightarrow$	V
W	$\rightarrow$	W
X	$\rightarrow$	X
Y	$\rightarrow$	Y
Z	$\rightarrow$	Z

Table 2. Lowercasing transformation on small and Capital letters of the alphabet

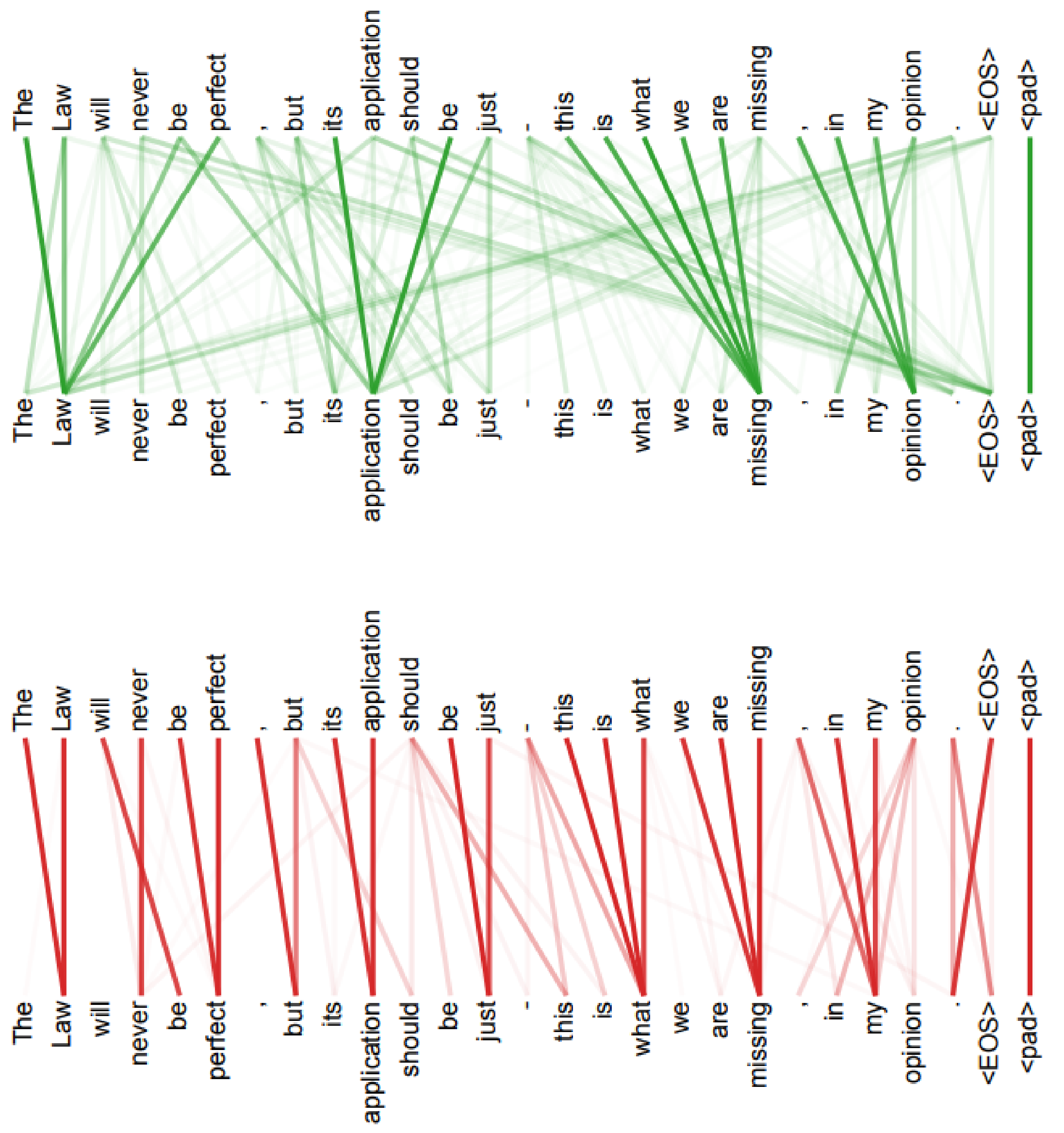
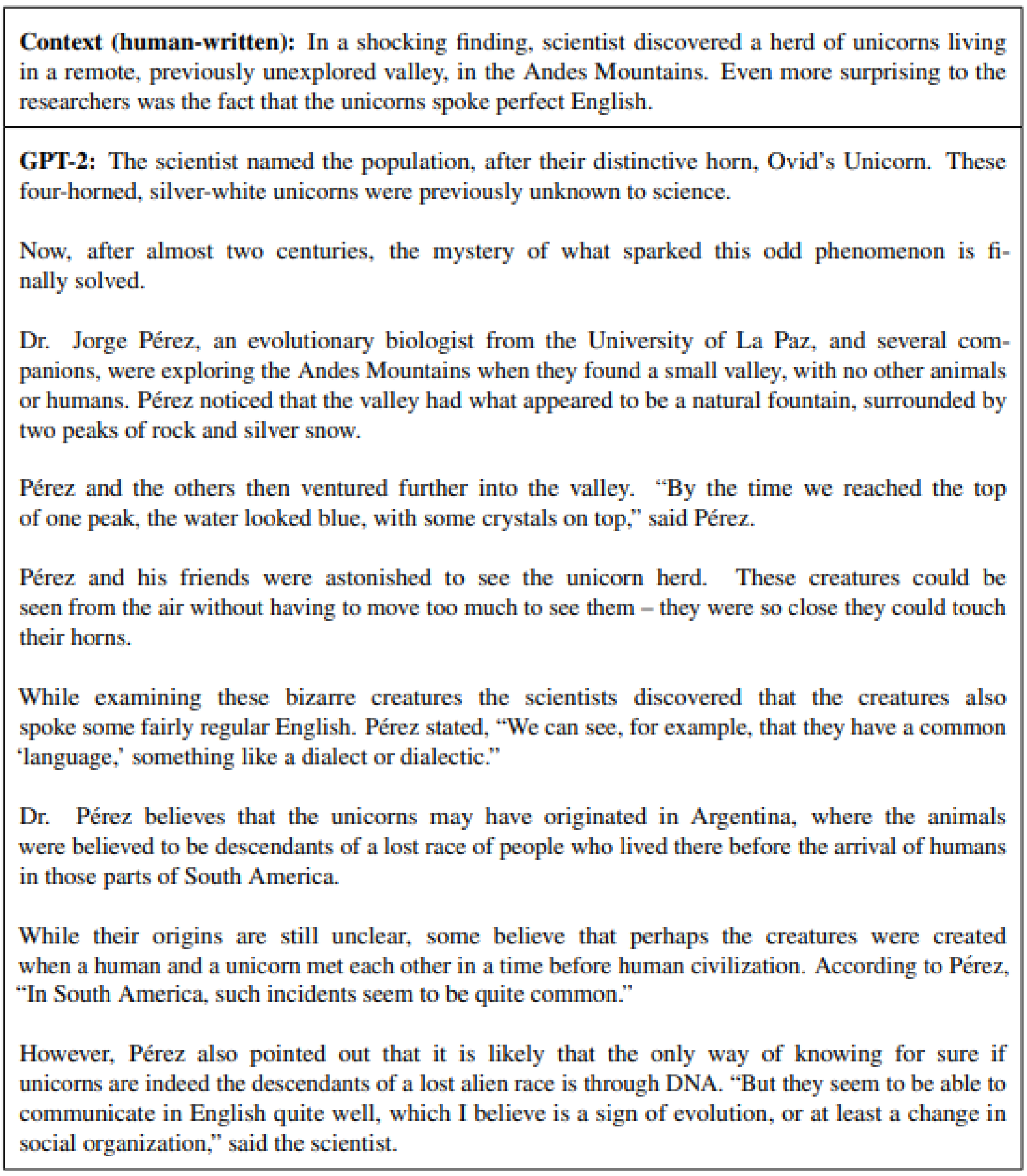


Figure 6. Attention from two different heads from the encoder self-attention layer. (Vasvani A. et al. in [29]).

```python
import spacy
import string

class Detokenizer:
    def _would_spaCy_join(self, tokens, index):
        """
        Check whether sum of lenghts of spaCy tokenized words
        is equal to length of joind and then spaCy tokenized words...

        In other words we say we should join only if the join is reversible.
        eg.:
            for the text ["The","man","."]
            we would join "man" with "."
            but wouldn't join "The" with "man."
        """
        left_part = tokens[index]
        right_part = tokens[index+1]

        left_toks = self.nlp(left_part)
        right_toks = self.nlp(right_part)
        join_toks = self.nlp(left_part + right_part)

        ## The sentence would change after join and tokenize
        ## (eg. "the red" -> "thered" -> "there d")
        if ([t.text for t in join_toks] \
                != ([t.text for t in left_toks] \
                + [t.text for t in right_toks])):
            return False

        elif (left_part == "\"" and
              (sum([t.count("\"") for t in tokens[:index+1]])%2 == 1) and
              right_part[0].isalnum()):
            return True
        elif (right_part == "\"" and
              (sum([t.count("\"") for t in tokens[:index+2]])%2 == 1)):
            return False
        elif (left_toks[-1].text in string.punctuation or
              left_toks[-1].text.count(".") == len(left_toks[-1].text)):
            return False
        elif (left_part[-1].isdigit() and
              right_part[0].isalpha()):
            return False
        length_before_join = len(left_toks) + len(right_toks)
        length_after_join = len(join_toks)
        return length_before_join == length_after_join
```

Listing 2. Code that detokenizes SpaCy tokenized list.



Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

Figure 7. An example of GPT-2 completion. Radford et al. [25]

Bibliography

[1] Alammar, J. The illustrated gpt-2 (visualizing transformer language models), Aug 2019.

[2] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *CoRR abs/1409.0473* (2014).

[3] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. A neural probabilistic language model. *J. Mach. Learn. Res. 3* (Mar. 2003), 1137–1155.

[4] Bojar, O., Federmann, C., Fishel, M., Graham, Y., Haddow, B., Koehn, P., and Monz, C. Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers* (Belgium, Brussels, Oct. 2018), Association for Computational Linguistics, pp. 272–303.

[5] Bryant, C., Felice, M., Andersen, Ø. E., and Briscoe, T. The BEA-2019 shared task on grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications* (Florence, Italy, Aug. 2019), Association for Computational Linguistics, pp. 52–75.

[6] Cover, T., and King, R. Convergent gambling estimate of entropy of english. *Information Theory, IEEE Transactions on 24* (08 1978), 413 – 421.

[7] Domingos, P. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World.* Basic Books, Inc., USA, 2018.

[8] Felice, M., Bryant, C., and Briscoe, T. Automatic extraction of learner errors in ESL sentences using linguistically enhanced alignments. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers* (Osaka, Japan, Dec. 2016), The COLING 2016 Organizing Committee, pp. 825–835.

[9] Gage, P. A new algorithm for data compression. *C Users J. 12*, 2 (Feb. 1994), 23–38.

[10] Graliński, F., Jaworski, R., Borchmann, Ł., and Wierzchoń, P. Gonito.net – open platform for research competition, cooperation and reproducibility. In *Proceedings of the 4REAL Workshop: Workshop on Research Results Reproducibility and*

Resources Citation in Science and Technology of Language, A. Branco, N. Calzolari, and K. Choukri, Eds. Graliński, Filip, 2016, pp. 13–20.

[11] Graliński, F., Wróblewska, A., Stanisławek, T., Grabowski, K., and Górecki, T. GEval: Tool for debugging NLP datasets and models. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP* (Florence, Italy, Aug. 2019), Association for Computational Linguistics, pp. 254–262.

[12] Grundkiewicz, R., Junczys-Dowmunt, M., and Heafield, K. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications* (Florence, Italy, Aug. 2019), Association for Computational Linguistics, pp. 252–263.

[13] Harkaway, N. *The Gone-Away World*. Vintage Contemporaries. Vintage Contemporaries, 2009.

[14] Hutchins, W. J., Dostert, L., and Garvin, P. The georgetown-i.b.m. experiment. In *In* (1955), John Wiley & Sons, pp. 124–135.

[15] Jurafsky, D., and Martin, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, second ed. Pearson Prentice Hall, 2009.

[16] Kamath, U., Liu, J., and Whitaker, J. *Deep Learning for NLP and Speech Recognition*. Springer International Publishing, 2019.

[17] Koehn, P. *Introduction*. Cambridge University Press, 2009, pp. 3–32,181–216.

[18] Lafferty, J., McCallum, A., and Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning* (2001), pp. 282–289.

[19] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady 10*, 8 (feb 1966), 707–710. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[20] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. Efficient estimation of word representations in vector space. *CoRR abs/1301.3781* (2013).

[21] Naber, D., Fakultät, T., and Bielefeld, U. A rule-based style and grammar checker, 2003.

[22] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics* (Philadelphia, Pennsylvania, USA, July 2002), Association for Computational Linguistics, pp. 311–318.

[23] Pierce, J. R., and Carroll, J. B. *Language and Machines: Computers in Translation and Linguistics.* National Academy of Sciences/National Research Council, USA, 1966.

[24] Radford, A., and Sutskever, I. Improving language understanding by generative pre-training. In *arxiv* (2018).

[25] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners.

[26] Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Berlin, Germany, Aug. 2016), Association for Computational Linguistics, pp. 1715–1725.

[27] Shannon, C. E. Prediction and entropy of printed english. *Bell System Technical Journal 30* (Jan. 1951), 50–64.

[28] Sidorov, G., Gupta, A., Tozer, M., Catala, D., Catena, A., and Fuentes, S. Rule-based system for automatic grammar correction using syntactic n-grams for English language learning (L2). In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task* (Sofia, Bulgaria, Aug. 2013), Association for Computational Linguistics, pp. 96–101.

[29] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS* (2017).

List of Figures

List of Tables

List of Code Listings

Index