# Automatic Text Correction for Chatbots

**Vasileios Palassopoulos**

Department of Informatics

Athens University of Economics and Business

This dissertation is submitted for the degree of

*M.Sc. in Data Science*

January 8, 2020

I would like to dedicate this thesis to my loving parents.

# Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Vasileios Palassopoulos
January 8, 2020

# Acknowledgements

I would like to express my gratitude to Professor Ion Androutsopoulos, my thesis supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of this thesis.

Also, I would like to thank Omilia Inc, and specifically Dr. Themos Stafylakis, Head of Machine Learning & Voice Biometrics, as well as Mr. George Mastrapas, for their advice and their help, both in the video conferences and in my visits to the offices of the Company.

# Abstract

The present thesis addresses an important, open, Machine Learning problem, namely the automatic correction of the involuntary errors, made by humans, when communicating by written messages with chatbots. First, the problem is formulated as a "noisy-channel model" problem, and all the needed algorithms are developed, employing both, n-gram and Transformer-based language models. Next, a complete software framework is developed for solving the problem by employing Machine Learning methods, using Python and C++ libraries, and partially modifying them, resulting in a 20-fold increase in the processing speed for the specific problem. Finally, the developed software framework is used for performing Machine Learning experiments, using the publicly available corpora of "WikEd" and "W&I". Although only a simple personal computer and limited use of cloud computing are used, and the publicly available corpora are not entirely appropriate for the machine training-tuning-testing procedures, certain interesting results are obtained, with respect to the relative efficiency of the various available methods for language processing. If, in the future, appropriate corpora become available and sufficient computer resources are used, it is expected that the developed software framework can provide acceptably efficient methods for the automatic text correction for chatbots.

# Contents

# Chapter 1

# Introduction

## 1.1 About Chatbots

The present thesis addresses an important problem for all businesses, which is in the general context of how to communicate, as efficiently as possible, with their customers. This communication capability is critically important for most businesses, because, among other advantages, it leads to repeat business and favorable referrals. On the contrary, inefficient communication of a business with its customers, will usually result in frustrated customers and negative word of mouth, among others. Furthermore, the development of efficient methods of communication for businesses with their customers may be generalizable to the public sector, with obvious benefits for the whole of society.

Because of their importance, the methods employed by businesses in order to communicate efficiently with their customers have a long history. In the older days, i.e. before the wide-spread use of the internet, these methods included sales representatives, customer service personnel and account representatives. However, since the wide-spread use of the internet, new methods of communication became available, which include:

- Email exchange,
- Text messaging (e.g. via mobile phones),
- Social networks participation,
- Carefully developed corporate websites, and
- Using Chat tools

The major advantages of the online communications is that they are fast and efficient, as far as the required manpower is concerned. However, with the exception of the Chat tools, they have major disadvantages, in that they are much less personal than the in-person communication, and in that they cannot handle complicated problems, which require human intervention.

Within this context, using Chat tools by businesses for communicating with their customers can be very advantageous, especially for small businesses, because:

- It can be done in real time (unlike the email exchange, and the social networks) and the customer can solve his or her problem immediately.

- It has many of the characteristics of the in-person communication, without requiring anyone to move from one place to another.

- It can handle efficiently quite complicated problems (and, certainly, questions outside the field of the "frequently asked questions").

The obvious disadvantage of Chat tools is that they must work in real time. This means that they may require a, greatly varying, number of personnel during the working hours (with unknown "rush hours", depending mostly on chance) and that they are not available during the non-working hours. It is for this reason that there is a really great need for automatic Chat tools. These automatic Chat tools, which can be used by businesses for the purpose of communicating with their customers, are known as chatbots. A more careful and exact definition of chatbots, for the purposes of the present thesis, is given in the next section.

Although it is not possible to predict the future accurately, it can be foreseen that the use of chatbots will greatly increase in the next few years, and almost every noteworthy business will need to employ at least one chatbot in order to communicate effectively with its customers. Furthermore, it is expected that, the availability itself of a chatbot at the business site, will serve as a reassurance mark to all customers, that the business is serious, both about its products and about its services, and that it can provide help and advice to its customers at any time. Also, hopefully, besides the businesses, one may envision that the public sector will also notice these advances in technology, and employ at least part of them, in order to reduce the bureaucracy and the current need to visit in person a number of public offices for the routine everyday questions and tasks of the citizen.

## 1.2 Description of the Problem

Since Chatbots in businesses are intended to replace employees working with Chat tools, they must be able to communicate with the customers in natural language. Furthermore, unlike the "task-oriented dialogue agents", such as Siri, Alexa, Cortana, etc., which are strictly concentrated on a very specific field of application and can answer only to questions, immediately related to this specific field, chatbots, as explained by Jurafsky & Martin [9], "must be capable of extended conversations, set up to mimic the unstructured conversations or 'chats', characteristic of human-human interaction", without necessarily addressing any specific practical task and/or specific business application.

It is these fundamental requirements, about the natural language and the unstructured conversations, which create the major problems for developing chatbots. Attempts to "Natural Language Processing" and "Computational Linguistics" have a long history. Unfortunately, the traditional methods of Artificial Intelligence (AI) have proved ineffective for this purpose and the progress in this direction was effectively stalled for a number of decades. It is only lately, with the great advances in data science and, in particular, in Machine Learning, that this purpose seems to be within reach.

More specifically, no efficient chatbots, capable of making unstructured conversations in natural language with humans, have been developed yet. It is for this reason that the present thesis does not address the general problem of developing a complete chatbot, but a rather more specific one. Namely, the present thesis addresses the problem of the unintentional errors that the customer may make in his/her messages to the chatbot, when trying to communicate with the business. The errors may be spelling, grammatical or even textual. And it is important that the chatbot, like a human, can recognize these errors and suggest or consider the appropriate corrections.

Although the subject of the present thesis may seem rather limited at first glance, it is, in fact, a very important and crucial first step in developing efficient and successful chatbots. This is not only due to the fact that, if the errors in the customer message remain unchecked or uncorrected by the chatbot, then the reply to the customer by the chatbot may be entirely wrong and/or misleading. The importance of the subject of the present thesis is in the fact, that it is the first, crucial and necessary, step, in the effort to develop complete and fully efficient chatbots. More specifically, the Machine Learning methods, employed in the present thesis, because of their generality, are expected to be extensible to the remaining steps for the development of a fully efficient and successful chatbot for any particular business.

## 1.3 Structure of the thesis

The main body of the thesis consists of three consecutive parts, each one in a different chapter.

- In the first chapter, the problem of the thesis is stated rigorously and is formulated as a "Noisy Channel" model in Machine Learning. Next, all the needed algorithms are presented and explained mathematically. Both, the n-gram language models and the Transformer models are employed and analyzed, introducing a number of hyper-parameters, which require appropriate tuning by the Machine Learning model.

- In the second chapter, a complete software framework is developed for solving the problem employing Machine Learning methods. The software framework employs libraries in both, the Python and the C++ languages, and partially modifies them, resulting in a 20-fold increase in processing speed in the particular problem. The final software framework is capable of performing all the Machine Learning tasks, i.e. training, tuning and testing, without any important restrictions on the data employed for these tasks.

- In the third chapter, it is first noted that there exist two important problems, with respect to the task of performing computer experiments with the software framework developed in the previous section. These problems are the non-availability of appropriate, publicly available, data for Machine Learning in this specific task, and the requirement of very significant computer resources for performing these computer experiments with sufficiently large corpora. Next, the two publicly available corpora, i.e. WikEd and W&I, are briefly described and employed in the computer experiments done. Employing these corpora, six (6) alternative language models are tested and compared. From these comparisons, a number of interesting and useful conclusions arise, with respect to the relative efficiency of the various alternative algorithms, and with respect to the requirements for the corpora employed and for the computer-resources needed.

The final chapter of the thesis covers the conclusions and the directions for future work on the subject.

# Chapter 2

# Analysis of the Subject as a Machine Learning Problem

## 2.1 Statement of the Problem

As defined in Wikipedia[1], “a chatbot is a piece of software, i.e. a computer program, that can conduct a conversation with a human, via auditory or textual methods. The main objective of the chatbot is to convincingly simulate how a human would behave as a conversational partner.”

From the above definition, it is clear that chatbots are fundamentally different from the “task-oriented dialogue agents”, such as Siri, Alexa, Cortana, etc., which have been developed by large technology companies lately and released for public use. Specifically, these “task-oriented dialogue agents” are strictly concentrated on a very specific field of application and can answer only to questions immediately related to this specific field. By contrast, “chatbots are systems designed for extended conversations, set up to mimic the unstructured conversations or ’chats’, characteristic of human-human interaction” [9], without necessarily addressing any specific practical task and/or specific business application.

As explained in detail in the introduction to the present thesis (Chapter 1), developing a fully efficient chatbot is a very important but, also, a very difficult problem and we are still a number of years away from achieving it. It is for this reason that the present thesis is limited to a more specific problem with chatbots which, nevertheless, constitutes the first step in the effort to develop a fully efficient chatbot, suitable for direct business applications.

More specifically, the problem in the present thesis is directly related to the involuntary, but also unavoidable, errors (either spelling errors or grammatical errors or contextual errors),

[1] https://en.wikipedia.org/wiki/Chatbot

which the human composer of a message may make, when trying to communicate with a chatbot by textual methods, i.e. by written messages. If these errors remain unchecked or uncorrected by the chatbot, the reply to the customer by the chatbot may be entirely wrong and/or misleading. Therefore, it is necessary to program the chatbot, so that it can identify and correct them automatically.

With this task in mind, **the problem addressed in the present thesis is the development of appropriate computer methods, which will enable the chatbot to identify any involuntary errors (i.e. spelling, grammatical or contextual) in the text of the messages, made by the composer of the message to the chatbot, and to propose appropriate corrections by the chatbot, so that the corrected text of the message is exactly the one intended by the composer of the message**. In what follows, this problem will be concisely denoted as "*automatic text correction for chatbots*".

As explained in more detail immediately below, this problem will be addressed in the present thesis employing Machine Learning methods.

## 2.2 The Noisy Channel Model

The starting point for addressing the problem of the automatic text correction for chatbots, within the framework of Machine Learning, is the "Noisy Channel" model [9]. The "Noisy Channel" model has been extensively employed in the past in communication networks and is currently employed, among other applications, in spell checkers, question answering, speech recognition, and machine translation.

Employing this model, consider an input (observed) word $o$ and denote with $c$ the corresponding word, candidate for correcting the word $o$. Then, according to Bayes rule, the following relation will always hold:

$$P(c|o) = \frac{P(o|c) \cdot P(c)}{P(o)} \tag{2.1}$$

where:

- $P(c|o)$ is the probability of the word $c$ to be the appropriately corrected word given the observed word $o$.
- $P(o|c)$ is the probability of receiving the word $o$, given that the appropriately corrected word is $c$. It will be denoted in the present thesis as "*conditional channel probability*".
- $P(c)$ is the probability for the word $c$ of appearing as input to the chatbot. It will be denoted in the present thesis as "*prior channel probability*".

- $P(o)$ is the probability for the word $o$ of appearing as input to the chatbot.

It is clear that the objective of any appropriate algorithm, for solving the problem under consideration, should be in the direction of maximizing $P(c|o)$, i.e. of maximizing the probability that the word $c$ is the appropriately corrected word, given the observed word $o$. Thus, provided a set $C \subseteq V$ of candidate words, where $V$ is our vocabulary, the best candidate word $\hat{c}$ for correction will result from the following formula:

$$\hat{c} = \arg\max_{c \in C} P(c|o) \overset{\text{Bayes' rule}}{=} \arg\max_{c \in C} \frac{P(o|c) \cdot P(c)}{P(o)} \overset{P(o)\text{ const.}}{=} \arg\max_{c \in C} P(o|c) \cdot P(c) \tag{2.2}$$

The above approach, which has been presented above for the case of just one word, can be readily generalized to the case of whole sentences. With the term sentence in the present thesis, we will just mean an ordered set of words, i.e. an observed sentence $o_1^n = \langle o_1, o_2, \ldots, o_n \rangle$, where $o_1, \ldots o_n$ are words. Then we can estimate a best candidate sentence $\hat{c}_1^k$ via the following formula:

$$\hat{c}_1^k = \arg\max_{c_1^k \in C_1^k} P\left(c_1^k | o_1^k\right) = \arg\max_{c_1^k \in C_1^k} P\left(o_1^k | c_1^k\right) \cdot P\left(c_1^k\right) \tag{2.3}$$

Employing the Equation 2.3 above in general, is a very difficult task. In order to simplify this task, we usually make an additional simplification, namely that $P\left(o_1^k | c_1^k\right) \approx \prod_{i=1}^{k} P(o_i|c_i)$. As it can be readily understood this simplification is a very strong one and has many limitations.

Finally, from practical applications, it may be useful to have different weights for the conditional channel probability and the prior channel probability. This is usually achieved by adding a hyper-parameter $\lambda$ to the Equation 2.3 above which takes the form:

$$\hat{c}_1^k = \arg\max_{c_1^k \in C_1^k} P\left(o_1^k | c_1^k\right) \cdot P\left(c_1^k\right)^{\lambda}$$

.

The resulting final equation is:

$$\hat{c}_1^k = \arg\max_{c_1^k \in C_1^k} \prod_{i=1}^{k} P(o_i|c_i) \cdot P\left(c_1^k\right)^{\lambda} \tag{2.4}$$

In order to apply the above equation in practical applications, the following 4 important questions arise:

- <u>Question 1</u>: How to find efficiently a set $C \subseteq V$ of candidate words for correction, from our vocabulary $V$, from which to choose the best candidate word. This question is directly related to the important concept of "Edit Distances" and to the "SymSpell" algorithm, which will be briefly presented in the following subsections (2.3 and 2.4) and employed in the present thesis.

- Question 2: How to estimate the conditional channel probability $P(o|c)$, i.e. the conditional probability of having received the word $o$, given that the appropriately corrected word is $c$. This question will also use the concept of "Edit Distances" and also require certain approximations, which will be described in the following subsection 2.5 and employed in the present thesis.

- Question 3: How to estimate the prior channel probability $P\left(c_k|\hat{c}_1^{k-1}\right)$, i.e. the probability, for the corrected word $c_k$, of been the intended word in the original message. This question is a deeper one and requires the development of appropriate "Language Models". From the relatively large number of available "Language Models", only the "n-gram Language Models" and some of the "Transformer-based Language Models" will be briefly presented in the following Section 2.7 and employed in the present thesis.

- Question 4: How to determine efficiently the best candidate sentence $\hat{c}_1^n$, i.e. the sentence with the maximum likelihood. This question will require the development of appropriate optimization search algorithms. For the present problem, the "Beam Search" algorithm and the "Viterbi" algorithm, will be briefly presented in the following Section 2.8 and employed in the present thesis.

## 2.3 Edit Distance

The concept of "edit distance" is relatively simple. Specifically, edit distance is a metric, which quantifies how dissimilar two strings are to one another.

The simplest way to quantify this metric is to count the minimum number of operations, which are required to transform one string into the other. Depending on the type of permitted operations, for transforming one string into the other, different metrics can be developed.

A commonly used metric for an edit distance is the "Levenshtein Distance". With this metric, only the following operations are permitted:

- Insertion of a single character (with a cost of 1),

- Deletion of a single character (with a cost of 1), and

- Substitution of a single character with another one (with a cost of 2).

In this thesis we employ a modification of the "Levenshtein Distance", called "Damerau–Levenshtein distance". This metric has additionally one operation:

- Transposition between two successive characters.

The "Damerau–Levenshtein distance" has the important properties of:

- Every edit operation has a positive cost of 1, and
- For every operation, there is an inverse operation with an equal cost of 1.

Because of these important properties, it satisfies the axioms of a metric, which gives us the benefit to employ the "SymSpell algorithm" for computing efficiently a set of candidate words, as we will discuss in the next subsection 2.4.

In general there are many more metrics of edit distances, that we could use to improve upon, taking into account factors such as:

- Permitting multiple-character operations, such as the Brill-Moore algorithm, [9]
- Employing different weights of cost depending on the relative distance of the characters on the keyboard, and
- Considering pronunciation mistakes, such as the aspell or the Soundex algorithm. [9]

However, employing the above additional metrics is beyond the scope of the present thesis.

## 2.4 Finding candidate words

An important problem we need to solve is how to find a set of candidate words $C$ from the entire vocabulary $V$. The naive approach would be to calculate the edit distance between the observed word and each word in the vocabulary and then sort the results and get the vocabulary words with the smaller distance to the observed word. However, when trying to apply this approach, it resulted in a very slow implementation. In order to solve this problem, it was necessary to use an additional algorithm, called SymSpell (Symmetric Delete Spelling Correction)[2], which utilizes the fact that the "Damerau–Levenshtein edit distance" meets the axioms of a metric, as already discussed on the previous subsection 2.3.

Specifically, the SymSpell algorithm utilizes the property of symmetry, i.e. that we can either perform edit operations on the observed word or on the vocabulary word or on both,

[2]**Source:** Wolf Garbe (2012). 1000x Faster Spelling Correction algorithm. Retrieved from: https://medium.com/@wolfgarbe/1000x-faster-spelling-correction-algorithm-2012-8701fcd87a5f

in order to determine their edit distance. It has an initialization step where it generates and stores all terms within close proximity (we need to pre-specify a maximum edit distance) for all vocabulary words. This can be done before the chatbot system goes online to chat with users in real-time. Afterwards, while the chatbot interacts with the user, it can generate all terms within close proximity for an observed word and then perform a check on whether:

- the observed word is a vocabulary word,
- the observed word is within the set of terms with close proximity to the vocabulary words,
- A term with close proximity to the observed word is a vocabulary word,
- the set of terms with close proximity to the observed word is within the set of terms with close proximity to the vocabulary words.

Furthermore, the SymSpell algorithm can be restricted to employ only delete operations when generating terms within close proximity, because adding a character on the observed word is equivalent to removing a character from the vocabulary word and vice versa. This important property makes the initialization step feasible while, additionally, the process running while the chatbot interacts with the user, is independent of the vocabulary size. Additionally, delete operations are language independent because the alphabet is not required to generate deletes as opposed to other operations.

For example, let's consider we have the vocabulary word "Hello". In the initialization step we will produce the term "Heo" (among other terms). Let's assume we received the observed word "Hemo". We will produce the term "Heo" (among other terms) with 1 delete. We can observe that the two produced terms match. Thus we can infer that the observed word "Hemo" has an edit distance of 2 from the vocabulary word "Hello".

For purposes of completeness, below are presented certain parts of the SymSpell algorithm as pseudo-code:

**Algorithm 2.1** Pseudo-code for generation of terms in close proximity

def gen_close_terms(term, max_edit_dist):

```
generated terms = {}
for ed in range(max_edit_dist):
  for generated_term in generate_deletes(term, ed):
    generated terms += {generated_term: (ed, term)}
return generated_terms
```

**Algorithm 2.2** Pseudo-code for initialization step

def init_symspell(vocab, max_vocab_edit_dist):

```
extended_set_vocab = {}
for term in vocab:
  extended_set_vocab += gen_close_terms(term, max_vocab_edit_dist)
return extended_set_vocab
```

**Algorithm 2.3** Pseudo-code for generation of candidate terms

def symspell_candidates(input_term, vocab, ext_set_vocab, max_edit_distance):

```
# max_edit_distance <= max_vocab_edit_dist
ext_set_input = gen_close_terms(input_term, max_edit_distance)
candidates = []
# input in dictionary
if input_term in vocab:
  candidates += 0, input_term
if input_term in extended_set_vocab:
  candidates += ext_set_vocab[input_term].ed,
                ext_set_vocab[input_term].origin_terms
elif ext_set_input in vocab:
  candidates += ext_set_input[for term in extended_set_vocab].ed,
                vocab[extended_set_input]
# for replaces and transpositions:
elif ext_set_input in ext_set_vocab:
  candidates += ext_set_input[for term in extended_set_vocab].ed,
                ext_set_vocab[].origin_terms
return candidates
```

## 2.5 Conditional Channel Probability

While, as we discussed in Section 2.3, we have a way for measuring how similar two words are, we haven't specified yet how to estimate the conditional channel probability $P(o|c)$, i.e. the conditional probability of having received the observed word $o$, given that the appropriately corrected word is $c$. There are two common ways to do it, both related to the edit distance.

### 2.5.1 Conditional Channel Probability Normalized and Inversely Proportional to Edit Distances

Given the observed word $o$ and a set of candidate words $C \subseteq V$ produced as explained in Section 2.4, we can calculate the "Damerau-Levenshtein distance" between $o$ and $c \in C$, which we will denote as $l_{o,c}$. We then estimate the conditional channel probability according to the following rules:

- For $c \in C$ and $c \neq o$, we set the probability inversely proportional to its edit distance from $o$.

- For $c \in C$ and $c = o$, which means $o \in C \subseteq V$ (i.e. $o$ is a vocabulary word), we set the probability to be $\alpha$ (a hyper-parameter probably in the range of $[0.9, 1)$) of being the correct word.

The probability for all candidate words will be 1 in total and, since for $c = o$ it is $\alpha$, then for all $c \neq o$ the probability will be $1-\alpha$ in total. Thus we define the conditional channel probability as:

$$\text{channel\_prob} = P(o|c) = \begin{cases} \alpha & c = o \\ \frac{1-\alpha}{\sum\limits_{i \in C} l_{o,i}^{-1}} \cdot l_{o,c}^{-1} & c \neq o \end{cases}$$

In practice we calculate:

$$\log P(o|c) = \begin{cases} \log \alpha & c = o \\ \log(1-\alpha) - \log\left(\sum\limits_{i \in C} l_{o,i}^{-1}\right) + \log l_{o,c}^{-1} & c \neq o \end{cases}$$

### 2.5.2 Conditional Channel Probability according to the Poisson Distribution

In this case, we use the Poisson distribution with parameter $\lambda$. $\lambda$ is a hyper-parameter (usually smaller than 0.01) and we employ the following formula:

$$\text{channel\_prob} = P(o|c) = P(l_{o,c}; \lambda) = e^{-\lambda} \frac{\lambda^{l_{o,c}}}{l_{o,c}}$$

## 2.6 n-gram Language Models

The most basic language models which are widely employed are the n-gram models [9]. Specifically, a n-gram is defined as a contiguous sequence of $n$ words. Depending on the value of $n$, one can have unigrams for $n = 1$, bigrams for $n = 2$, trigrams for $n = 3$ and so on. For the purposes of this thesis we choose trigrams. Figure 2.1[3] represents a simple example of unigrams, bigrams and trigrams.

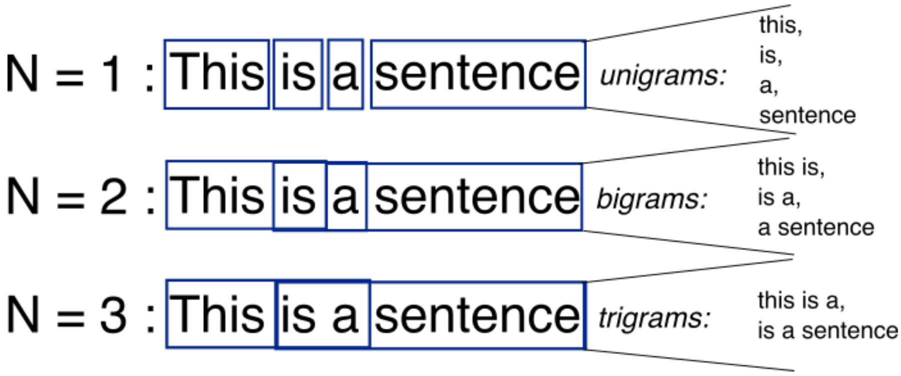



Figure 2.1: n-grams[3]

### 2.6.1 Markov Property

The great advantage of the n-gram models for natural language processing is the fact that one can employ the extended Markov property for them and greatly simplify the needed calculations. More specifically, for a sentence $w_1^k$, the probability $P\left(w_1^k\right) = P\left(w_1, \ldots, w_k\right)$ results from the following chain rule in general:

$$P\left(w_1^k\right) = P\left(w_1\right) \cdot P\left(w_2|w_1\right) \cdot P\left(w_3|w_1, w_2\right) \cdot \ldots \cdot P\left(w_k|w_1^{k-1}\right)$$

However, if the extended Markov property is assumed, then the probabilistic distribution of the n-th element of the n-gram will depend only on the previous (n-1) elements of the n-gram and the above formula will be greatly simplified.

For example, for the very simple case of a unigram, the above equation takes the following very simple form:

$$P\left(w_1^k\right) = P\left(w_1\right) \cdot P\left(w_2\right) \cdot P\left(w_3\right) \cdot \ldots \cdot P\left(w_k\right)$$

[3] **Source:** Antonio Maiolo (2015). Comparing n-gram models.
Retrieved from: https://web.archive.org/web/20180427050745/http://recognize-speech.com/language-model/n-gram-model/comparison

For the case of a bigram, the Markov property is equivalent to the assumption $P\left(w_k|w_1^{k-1}\right) \approx P(w_k|w_{k-1})$ and we can simplify the probability of $w_1^k$ occurring into:

$$P\left(w_1^k\right) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1,w_2) \cdot \ldots \cdot P(w_k|w_{k-1})$$

In the same way, for the case of the trigram, which we employ in the present thesis, we get the following formula:

$$P\left(w_1^k\right) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1,w_2) \cdot \ldots \cdot P\left(w_k|w_{k-2}^{k-1}\right)$$

### 2.6.2 Estimation of the Probability of a Word Occurring

In order to estimate the probabilities $P(w)$ for $w \in V$ occurring, we need to maximize $P(\text{Corpus})$. It can be rigorously proven, using the method of Maximum Likelihood Estimation, that this maximization requirement results in the following formulae:

- $P(w) = \frac{c(w)}{C}$ for the case of a unigram,
- $P(w_k|w_{k-2},w_{k-1}) = \frac{c(w_{k-2},w_{k-1},w_k)}{c(w_{k-2},w_{k-1})}$ for the case of a trigram which is used in the present thesis.

where $c(w_1,\ldots,w_n)$ is the count of the consecutive tokens $w_1,\ldots,w_n$.

The above formulae may also be deduced directly employing the frequentist approach to probability.

### 2.6.3 Shortcomings

The extended Markov property is a memory-less property, which is widely employed in various stochastic processes with great success. However, in natural language processing, this memory-less property is not always valid. More specifically, some of the problems, which are encountered with n-gram models under the extended Markov property, are the following:

- Natural languages incorporate many cases of unbounded dependencies among words (e.g. starting and final sentences in a paragraph) which, with an n-gram model, can only be modeled as noise,
- N-grams not encountered in the training data will be given a probability of 0, although they may be plausible, and

- Certain n-grams will contain special words (e.g. scientific terms or proper names), which will make them arbitrarily infrequent in the training data, although they are also quite plausible.

Because of all the above problems, one employs in the n-gram models an appropriate smoothing of the probabilities. There are a number of possibilities for selecting the appropriate smoothing, including the following, which are considered in the present thesis and briefly described immediately below:

- The Laplace smoothing,
- The Lidstone (additive) smoothing, and
- The Kneser-Ney Smoothing with interpolation

### 2.6.4 Laplace Smoothing

The Laplace smoothing consists of adding 1 count for each word, so that we cannot have a zero probability for any word. Applying Laplace smoothing, the above formulae take the following form:

- For a unigram: $P(w) = \frac{c(w)+1}{C+|V|}$
- For a trigram: $P(w_k|w_{k-2}, w_{k-1}) = \frac{c(w_{k-2}, w_{k-1}, w_k)+1}{c(w_{k-2}, w_{k-1})+|V|}$

In the above formulae, $V$ is the vocabulary (usually stripping out infrequent words) and $C$ is the count of all word occurrences in the training corpus.

### 2.6.5 Lidstone Smoothing

The Lidstone (additive) smoothing is different from Laplace smoothing in that, instead of adding 1, we add a hyper-parameter $\alpha$. Then, applying Lidstone smoothing, the above formulae take the following form:

- For a unigram: $P(w) = \frac{c(w)+\alpha}{C+\alpha \cdot |V|}$
- For a trigram: $P(w_k|w_{k-2}, w_{k-1}) = \frac{c(w_{k-2}, w_{k-1}, w_k)+\alpha}{c(w_{k-2}, w_{k-1})+\alpha \cdot |V|}$

### 2.6.6 Kneser-Ney Smoothing with Interpolation

The Kneser-Ney smoothing is in a different direction from the previously mentioned smoothings. Specifically, instead of adding a number, we "discount" the counts of an n-gram appearing in the corpus, with a fixed hyper-parameter $d$. By doing that, we remove some probability mass from the Maximum Likelihood Estimator to use for the n-grams that were not seen in the corpus. Then, applying Kneser-Ney smoothing, the above formulae take the following form:

- For a unigram: $P(w) = \frac{\max\{c(w)-d,0\}}{C}$
- For a bigram: $P(w_k|w_{k-1}) = \frac{\max\{c(w_{k-1},w_k)-d,0\}}{c(w_{k-1})}$
- For a trigram: $P(w_k|w_{k-2},w_{k-1}) = \frac{\max\{c(w_{k-2},w_{k-1},w_k)-d,0\}}{c(w_{k-2},w_{k-1})}$

The probability mass that remains from the "discounting" can be "distributed", via a function $\gamma$, to the unknown n-grams by estimating them, using lower order n-grams.

- For a bigram: $\gamma(w_{k-1}) = \frac{d\cdot|\{(w_{k-1},w'):c(w_{k-1},w')>0\}|}{\sum_{w'\in V} c(w_{k-1},w')}$
- For a trigram: $\gamma(w_{k-2},w_{k-1}) = \frac{d\cdot|\{(w_{k-2},w_{k-1},w'):c(w_{k-2},w_{k-1},w')>0\}|}{\sum_{w'\in V} c(w_{k-2},w_{k-1},w')}$

Combining both "discounting" and "distribution", the final formula, for Kneser-Ney smoothing with interpolation, takes the following form:

- For a bigram:

$$P(w_k|w_{k-1}) = \frac{\max\{c(w_{k-1},w_k)-d,0\}}{c(w_{k-1})} + \gamma(w_{k-1})\cdot P(w_k)$$

- For a trigram:

$$P(w_k|w_{k-2},w_{k-1}) = \frac{\max\{c(w_{k-2},w_{k-1},w_k)-d,0\}}{c(w_{k-2},w_{k-1})} + \gamma(w_{k-2},w_{k-1})\cdot P(w_k|w_{k-1})$$

## 2.7 The Transformer Model

In the last few years, a new deep learning model was presented and modified for use, among many other tasks, in the task of Grammatical Error Correction with relative success. This new model has been known as "The Transformer". The Transformer, as has been described in the original paper "Attention Is All You Need" [13], consists of four major components, namely:

- The input embeddings, which are the input to the encoding component,

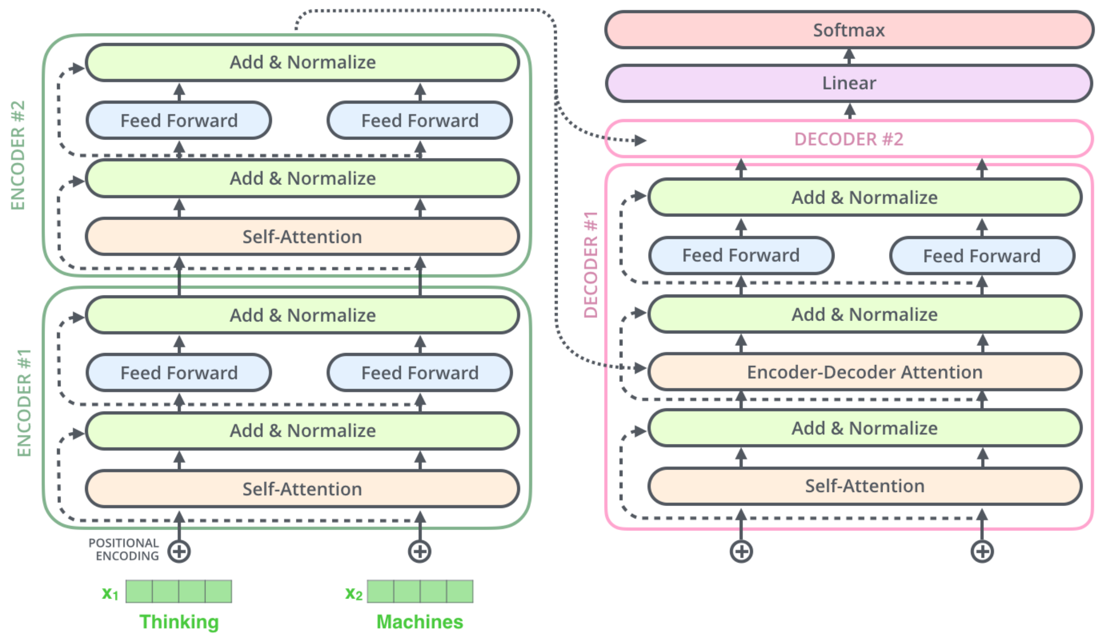



Figure 2.2: The Transformer[4]

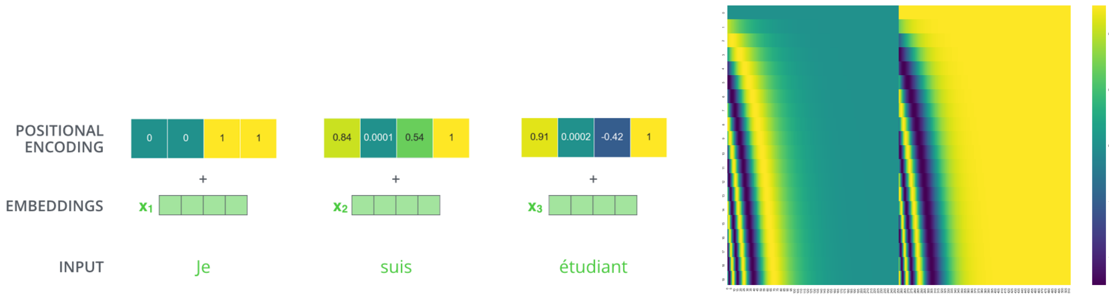
- The encoding component, which outputs a vector for the decoding component,
- The decoding component, which outputs a vector to the final layer, and
- The final linear and softmax layer, which outputs probabilities for each vocabulary word.

In what follows, a short description, along with a graphical depiction on Figures 2.2-2.5[4], of each of these components will be presented. In these descriptions, the concept of "token" will be used. With this concept, we mean, either a single word or an appropriate subword, each of which serves as an input to the Transformer model.

### 2.7.1 Input embeddings

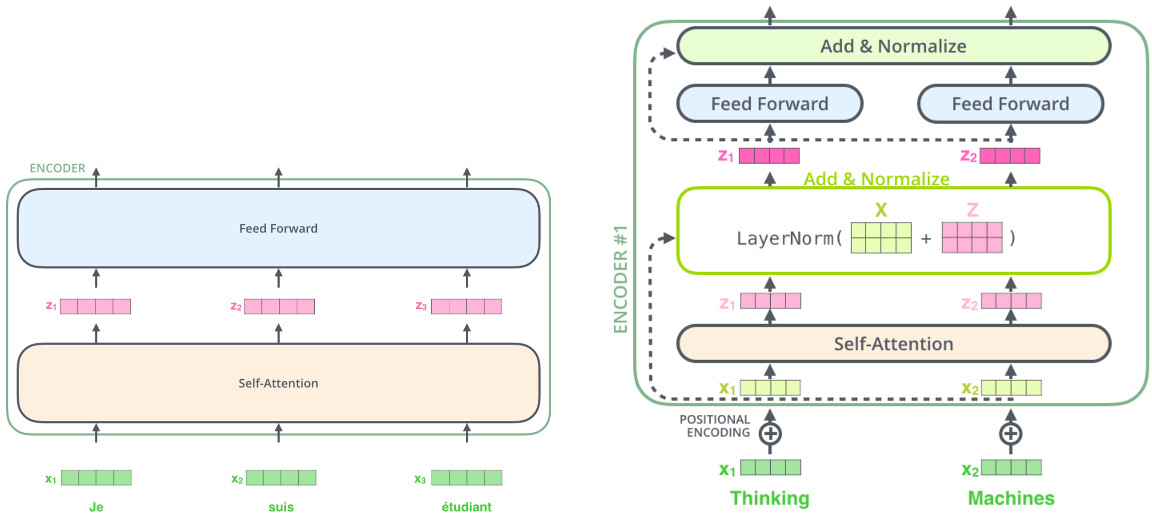
The input tokens are transformed into input embedding vectors. Next, these input embedding vectors are summed with positional encoding vectors. The end result is an embedding vector with a time signal. This embedding is the final input to the encoding component.

[4]**Source:** Alammar, Jay (2018). The Illustrated Transformer
Retrieved from: https://jalammar.github.io/illustrated-transformer/




(a) Adding a time signal to the Input Embedding (b) An example of a time signal's appearance

Figure 2.3: Creating the input embedding with the time signal[4]




(a) Encoding layer (b) The inner structure of an encoding layer

Figure 2.4: Encoding layer[4]

## 2.7.2 Encoding component

The encoding component is composed of a stack of encoding layers (in the original paper it's a stack of 6 encoding layers). Each encoding layer consists of a self-attention sub-layer and a feed forward sub-layer. Immediately below, a short description of them is provided.

**Self-attention** The self-attention sub-layer is used as a mechanism for the Neural Network to associate each token of a sentence to all the other tokens, in varying degrees. This is achieved via attention functions. An attention function is a mapping, of a query and of a set of key-value pairs, to an output.

Specifically, for each token, we start with a query, a key and a value vector created by weight matrices, which have resulted from the training of the model. For a fixed $i$ embedding (and a corresponding $x_i$ embedding vector), we have the query, key and value vectors $q_i = x_i \times W^Q$, $k_i = x_i \times W^K$, $v_i = x_i \times W^V$. Then, for each $j$ embedding, we produce a score $\text{score}_j = q_i \cdot k_j$, divide it by $\sqrt{\dim(k)}$ and apply the softmax function on them for each $j$. This operation results in a softmax score for each $j$ embedding. Finally, we calculate $z_i = \sum_j \text{softmax\_score}_j \cdot v_j$.

The above procedure, in matrix notation, has the following form:

- We calculate the query matrix $Q = X \times W^Q$, the key matrix $K = X \times W^K$ and value matrix $V = X \times W^V$
- We calculate the self-attention function via $Z = \text{softmax}\left(\frac{Q \times K^\top}{\sqrt{\dim(k)}}\right) \times V$.

**Multi-head attention** The above procedure of self-attention, instead of being executed only once, can be more successful if it's applied repeatedly. The number of repetitions depends on the particular application. Specifically, in the original paper, 8 separately produced self-attention calculation matrices $Z_0, \ldots, Z_7$, were repeatedly computed, then they were concatenated and the resulting matrix $Z$ was finally multiplied with a weight matrix $W^0$.

**Feed-Forward sub-layer** After self-attention there is the feed-forward sub-layer which consists of two linear transformations with a ReLU activation in between.

**Normalization sub-layer** On top of the self-attention sub-layer and the feed-forward sub-layer there is a normalization sub-layer.

### 2.7.3 Decoding component

The decoding component is composed of:

- a self-attention sub-layer,
- an encoder-decoder attention sub-layer,
- a feed-forward sub-layer

The self-attention and feed-forward sub-layers work exactly in the same way, as in the encoding component and, therefore, they are not repeated here. The only new component is the encoder-decoder attention sub-layer, which uses as input the vector produced by the decoder layer, as well as the output of the self-attention sub-layer of the encoding component, in order to focus on the relevant tokens of the input sequence.

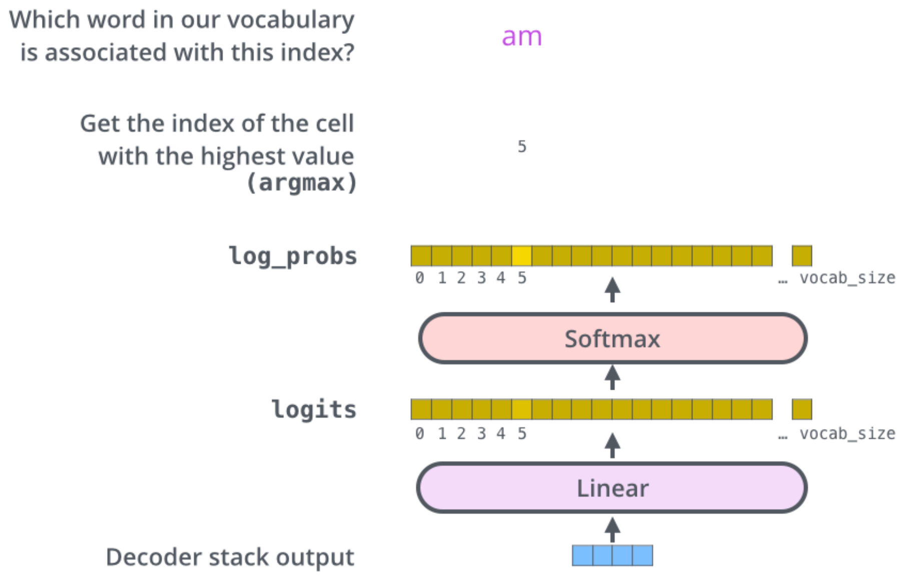



Figure 2.5: Final Layer[4]

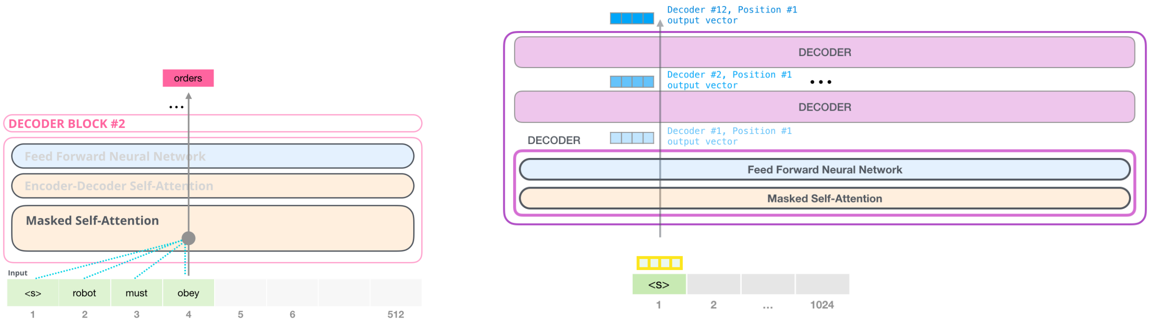
### 2.7.4 Final Linear and Softmax Layer

The linear layer is composed of a fully connected neural network, which outputs a vector of the same size, as the vocabulary size, where each vocabulary word is scored. Then, the softmax layer returns the probability of each vocabulary word. Finally, the word with the highest probability is selected as the output.

### 2.7.5 GPT-2 model

The GPT-2 model [10–12] is a pre-trained model, based on the Transformer architecture, with modifications. It is a "decoder-only Transformer" as it consists of only a decoding component, as depicted on Figure 2.6[5], as opposed to the original Transformer architecture. Additionally, the self-attention sub-layer takes into account only the previous tokens from the token under consideration, i.e. it masks all future tokens from the position of the token to be predicted. Thus, the input can be part of a sentence and the output can be the most probable token to follow from that sentence. We can add the "true" next token to the history (previous tokens). Thus, we can predict a new token on each iteration, and add it to the history, without having to re-calculate the query, key and value vectors for the previous tokens of the input in the self-attention layer. Because of this modification, we can utilize it as a language model in situations, where we have to make fast real time predictions, such as is the case of the dialogues

[5]**Source:** Alammar, Jay (2019). The Illustrated GPT-2
Retrieved from: https://jalammar.github.io/illustrated-gpt2/




(a) Transformer-decoder block (b) GPT-2 small

Figure 2.6: The GPT-2 Architecture[5]

by chatbots, which is the subject of the present thesis. For the purposes of this thesis we choose "GPT-2 Small" which consists of 12 decoder layers in the decoding component.

## 2.8 Finding candidate sentences

Up to now, we have defined how to calculate the prior channel probability from the language model, as well as the conditional channel probability. The next task is to find the best candidate sentences. This requires to actually calculate all possible candidate sentences $c_1^k \in C_1^k$. However, the set $C_1^k$ is too big for all practical purposes. Because of this fact, we will use these two algorithms:

- Beam Search, an algorithm which finds a local maximum, and
- Viterbi, an algorithm which finds a global maximum, but less efficiently

### 2.8.1 The Beam Search Algorithm

The Beam search algorithm is a modification of the breadth-first search. The idea of the algorithm is relatively simple. Specifically, at each step of the iterations for the maximization of $P\left(c_1^k|o_1^k\right)$, instead of taking into account all the possibilities, we take only into account a predefined limited number of possibilities, called "beam width". In this way, the procedure of maximization, instead of exploding exponentially, is kept constant in width. A general depiction of the algorithm is provided on Figure 2.7.

Immediately below is our implementation of the Beam Search algorithm in pseudo-code.

It is noted that we use the SymSpell algorithm in order to generate all the candidate words that have, at most, "max_edit_distance" edit distance from the word we examine on each iteration.

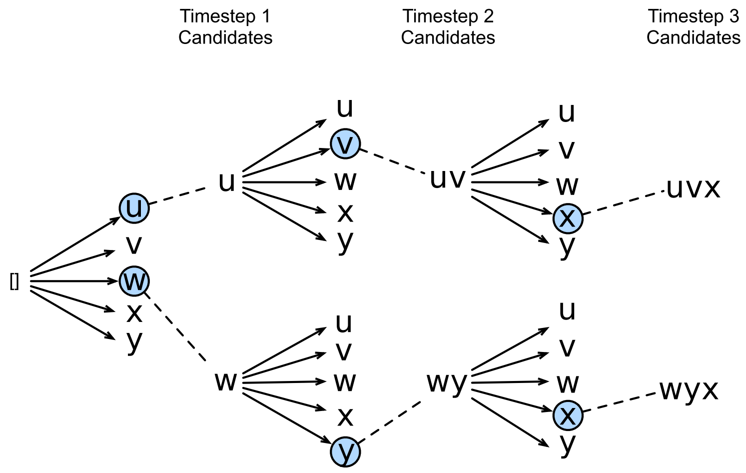



Figure 2.7: Beam search with beam width 2

**Algorithm 2.4** Our implementation of the Beam Search algorithm in pseudo-code

def beam_search(input_sentence, beam_width=10, max_edit_distance=3):

```
best_sentences = [input_sentence]
for word in input_sentence:
  candidate_words = symspell_candidates(word, max_edit_distance)
  channel_probabilities = channel_prob(candidate_words, word)
  best_sentences += prior_prob(candidate_sentences(candidate_words))
  best_sentences = sort(best_sentences, top=beam_width)
```

### 2.8.2 The Viterbi Algorithm

As defined in Wikipedia[6], "the Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of states, called the Viterbi path, that results in a sequence of observed events". Below, we will briefly describe the Viterbi algorithm, as it will be used for the purposes of the present thesis, specifically for the n-gram language models.

The input of the algorithm consists of:

- A n-gram language model, such as the ones we discussed in Section 2.6.
- A conditional channel probability, such as the ones we discussed in Section 2.5.
- A sequence of $T$ observations $O = o_1, o_2, \ldots o_T$, which are a sequence of $T$ successive words from the observed sentence.

[6] https://en.wikipedia.org/wiki/Viterbi_algorithm

From the n-gram language model, the Viterbi algorithm employs the following three mathematical constructs:

- An observation space $V$, where $V$ is the vocabulary of the n-gram language model.
- A state space $S = \{s_1, s_2, \ldots, s_k\}$, where $s_1, s_2, \ldots, s_k$ are the states of the history of the sentence. More specifically, each state $s_i$ consists of the ordered set of the (n–1)-long previous words. In particular:
  - For the case of bigrams ($n = 2$), a state consists of a single word and, thus, all the possible states are $|S| = |V|$.
  - For the case of trigrams ($n = 3$), a state consists of an ordered pair of two words and, thus, all the possible states are $|S| = |V|^2$.
- A transition matrix $A$ of size $|S| \times |S|$, such that $A_{ji}$ stores the transition probability of transitioning from state $s_j$ to state $s_i$. In particular:
  - For the case of bigrams: $P(v|u)$, obtained from the bigram language model, is the transition probability of transitioning from state $s_j = u$ to state $s_i = v$.
  - For the case of trigrams: $P(w|u,v)$, obtained from the trigram language model, is the transition probability of transitioning from state $s_j = (u,v)$ to state $s_i = (v,w)$.

Also, from the conditional channel probability, the Viterbi algorithm employs an additional mathematical construct, namely the emission matrix $B$ of size $|S| \times |O|$, such that $B_{io_t}$ stores the probability of observing word $o_t$ from state $s_i$. In particular:

- For the case of bigrams: $P(o_t|v)$, obtained from the conditional channel probability, is the emission probability of observing word $o_t$ from state $s_i = v$.
- For the case of trigrams: $P(o_t|w)$, obtained from the conditional channel probability, is the emission probability of observing word $o_t$ from state $s_i = (v,w)$.

Having defined the above mathematical constructs, it is now possible to describe accurately the Viterbi algorithm. In this respect it is noted that the above defined matrices $A$ and $B$ need to be computed employing only lazy evaluation. More specifically, the Viterbi algorithm consists of the following procedure:

1. We initialize a probability matrix (or lattice): $\text{viterbi} : |O| \times |S| \longrightarrow [0,1]$. More specifically:
   (a) For the bigram: $\text{viterbi} : T \times V \longrightarrow [0,1]$

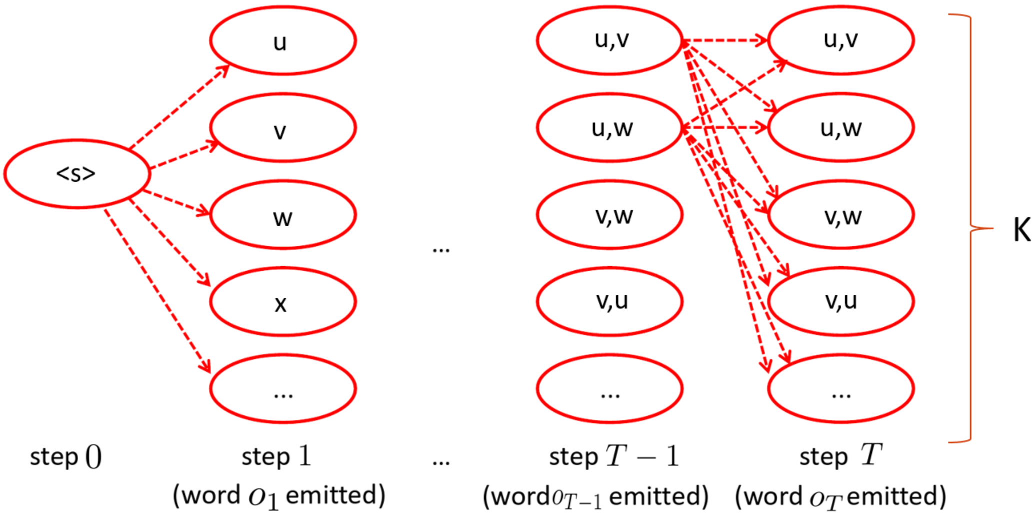



Figure 2.8: Lattice of possible transitions

(b) For the trigram: viterbi : $T \times V \times V \longrightarrow [0,1]$

2. We iterate over the sequence of $T$ observations $O = o_1, o_2, \dots o_T$, selecting successively one observed word $o_t$ ($t = 1, 2, \dots, T$) for each iteration.

3. In each one of the above iterations, we conduct an additional nested iteration over the state space $S = \{s_1, s_2, \dots, s_k\}$, selecting successively one state $s_i$ ($i = 1, 2, \dots, k$) for each iteration.

4. Within each nested iteration, we perform the following two steps.

   (a) First, we update the Viterbi matrix via the rule:

$$\text{viterbi}\,[o_t, s_i] \leftarrow \max_{s_j \in S} \text{viterbi}\,\left[o_{t-1}, s_j\right] \cdot A_{ji} \cdot B_{io_t}$$

   More specifically:

   i. For the bigram ($s_i = v$):

$$\text{viterbi}\,[o_t, v] \leftarrow \max_{u \in S} \text{viterbi}\,[o_{t-1}, u] \cdot P\,(v|u) \cdot P\,(o_t|v)$$

   ii. For the trigram ($s_i = (v, w)$):

$$\text{viterbi}\,[o_t, (v, w)] \leftarrow \max_{(u,v) \in S} \text{viterbi}\,[o_{t-1}, (u, v)] \cdot P\,(w|u, v) \cdot P\,(o_t|w)$$

(b) Next, for each new insertion in the Viterbi matrix, we store, in a separate matrix, a pointer to the position of the cell, on which the calculation was based (i.e. the cell that was chosen by the max operator).

The output of the Viterbi algorithm consists of the final form of the Viterbi matrix and the accompanying matrix, where we stored the best previous path(s).

An important consideration for the Viterbi algorithm is its algorithmic complexity. It can be derived that this complexity is $O\left(|O| \cdot |S|^2\right)$. More specifically:

- For bigrams the complexity is $O\left(T \cdot |V|^2\right)$.
- For trigrams the complexity is $O\left(T \cdot |V|^4\right)$.

In this respect, it is noted that the above algorithmic complexity is very high for the case of the trigrams, which we employ in the present thesis. A possible way to solve this problem may be by pruning the state space. It is also noted that the Viterbi algorithm cannot be applied for the case of the GPT-2 language model, as it is only applicable to n-gram language models, because it is only for these language models that the Markov assumption holds.

# Chapter 3

# Computational Implementation of the Machine Learning Problem

## 3.1 Selection of the appropriate Machine Learning models

Based on the previous Chapter 2, where a detailed analysis of the subject of the present thesis as a Machine Learning problem has been presented, two alternative general directions arise for the selection of the appropriate language models that will employ the "Noisy Channel" model. The first choice is in the direction of the n-gram language models and the second choice is in the direction of the Transformer models.

With respect to the first general direction, i.e. the n-gram language models, one must decide on the following 2 options:

- Option 1: Choose the value of n, i.e. choose for the n-gram language model among a 2-gram, 3-gram, 4-gram and so on. In this respect, it is noted that, most practical applications extend up to 5-grams. However, as the order of the n-gram increases, the complexity and the additional requirements for the size of the training corpus increase much faster. Taking this fact into account, and also the available resources for the present thesis, we have decided to utilize only trigrams.

- Option 2: Choose the appropriate smoothing. Based on the presentation of the various alternative methods for smoothing, we choose for the present thesis the Lidstone smoothing and the Kneser-Ney smoothing with interpolation.

With the above considerations in mind, the first 2 language models, which will be developed and tested in the present thesis, are the following:

- "trigram-L": trigram with Lidstone smoothing as a baseline.

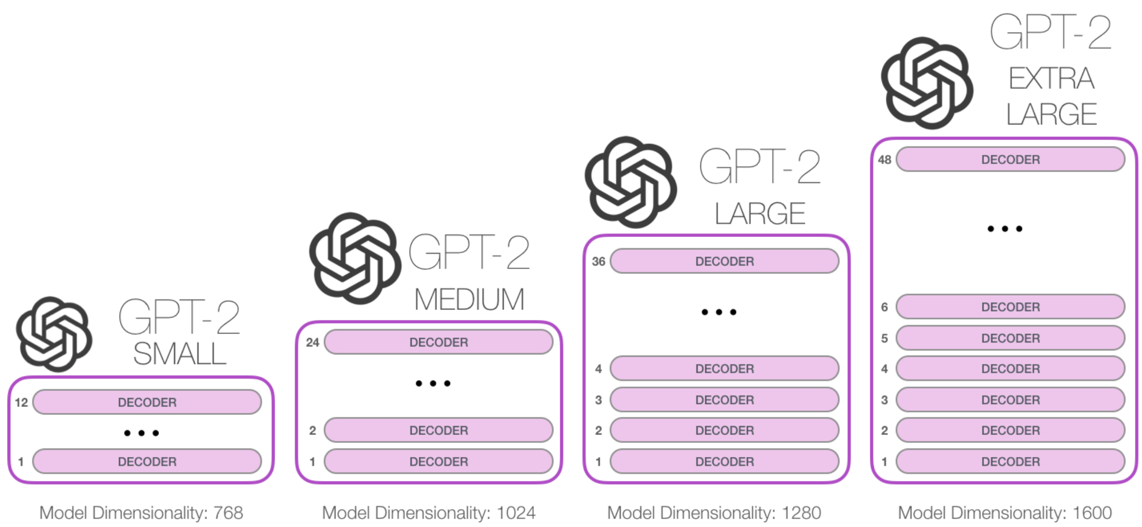



Figure 3.1: GPT-2 model sizes[1]

- "trigram-KNI": trigram with Kneser-Ney smoothing with interpolation.

With respect to the second general direction, i.e. the Transformer models, it is clear that training a Transformer model would require massive computational resources, as well as a large volume of data, much exceeding the available resources that are available for the present thesis. Fortunately, within this general direction, the GPT-2 model is available, which is a pre-trained Transformer decoder model and does not require any resources for training (Subsection 2.7.5). With the GPT-2 model, we have the option of selecting the size of the model, from the smallest one with just 12 layers and 124 million parameters, to the largest one with 48 layers and 1.5 billion parameters (as graphically depicted in Figure 3.1[1]). With the consideration of limited hardware availability in mind, the next model, which will be developed and tested in the present thesis, is the following:

- "GPT-2 Small": 12 layer decoder-only Transformer with 124 million parameters.

Finally, for calculating the conditional channel probabilities, as defined in Section 2.5 we choose the following 2 methods:

- Conditional channel probability 1: Normalized and inversely proportional to edit distances ("ED" for short).

- Conditional channel probability 2: According to the Poisson distribution ("Poisson" for short).

[1] **Source:** Alammar, Jay (2019). The Illustrated GPT-2
Retrieved from: https://jalammar.github.io/illustrated-gpt2/

Both of these methods are employed with all the language models. Therefore there are altogether six (6) noisy channel models used in the present thesis.

These noisy channel models are compared by testing them with a specific error corpus, employing the Word Error Rate (WER) metric, as defined in Wikipedia[2]:

$$\text{WER} = \frac{S+D+I}{N} = \frac{S+D+I}{S+D+C}$$

where:

- $S$ is the number of substitutions,
- $D$ is the number of deletions,
- $I$ is the number of insertions,
- $C$ is the number of correct words,
- $N$ is the number of words in the target sentence.

## 3.2 Requirements for the Data

The main tasks performed by our implementation are training and tuning the n-gram language models and tuning and testing all the noisy channel models. This requires:

- A “training corpus” for training and tuning the n-gram language models, and
- An “error corpus” for tuning and testing all the noisy channel models.

The “training corpus” should be a relatively large (of the order of many millions) set of correct sentences, in order for the n-grams to be trained efficiently. The “error corpus” should be composed of pairs of sentences, one original and one corrected for grammatical errors. For practical reasons, due to the complexity of tuning and testing a noisy channel model, the “error corpus” will be of the order of thousands of pairs of sentences.

[2]https://en.wikipedia.org/wiki/Word_error_rate

## 3.3 Programming Libraries Employed

### 3.3.1 NLTK

The Natural Language Toolkit (abbreviated as NLTK[3]) is a common open source Python library, used for various tasks related to Natural Language Processing. We employed this library for sentence and word tokenization from text sources, as well as for the creation and training of the n-gram language models, which were used in the present thesis. Modifications were made to the library to improve the performance of calculating sentence probabilities from the n-gram language models, as described in Section 3.4 below.

### 3.3.2 PyTorch and CUDA

PyTorch[4] is an open source computational framework, written in Python and C++, which is being used for Machine Learning and, particularly, for Deep Learning, especially in the field of Natural Language Processing. PyTorch can also utilize the parallelism provided by a GPU (Graphics Processing Unit) via the CUDA library, as provided by Nvidia Corporation for their GPUs.

### 3.3.3 Transformers

Transformers[5] [14] is an open source Python library, which provides some of the most recent Neural Network architectures (such as BERT, GPT-2, XLNet etc.) via TensorFlow or PyTorch back-ends, as well as access to pre-trained Neural Network models. It's been used in the present thesis for the GPT-2 model with the PyTorch back-end.

### 3.3.4 hyperopt

Hyper-parameter tuning is important for identifying the optimal hyper-parameters for each model being developed. A simple method, such as "grid search", where we are exhaustively trying all different hyper-parameters in a development set, would suffice to find the optimal hyper-parameters. The problem with this approach is that running exhaustively all different combinations is time-consuming and requires a lot of computational resources.

It is for this reason that we use, instead, a Bayesian hyper-parameter optimization method, called "Tree Parzen Estimator" [1], and implemented by the open source Python library hy-

[3] https://www.nltk.org/
[4] https://pytorch.org/
[5] https://huggingface.co/transformers/

peropt[6]. This method makes use of the results of the cost function from previous iterations of hyper-parameter tuning, and, employing these previous results, the method estimates the next best possible candidate values for the hyper-parameters, thus minimizing the number of iterations required.

### 3.3.5 SymSpellPy

The SymSpellPy[7] open source Python library implements the SymSpell algorithm, which is used in the present thesis for the efficient generation of candidate sentences, in the beam search algorithm as well as in the Viterbi algorithm, as described in more detail in Section 2.4.

### 3.3.6 jiwer

We employ the jiwer[8] open source Python library for calculating the Word Error Rate (WER) between two sentences.

### 3.3.7 Other useful libraries

We employ the well-known, for data science, libraries NumPy[9], pandas[10], scikit-learn[11], scipy[12], as well as the generic Python library tqdm[13]. We also employ m2-correct[14] for pre-processing of the error corpus data provided in the M2 format.

## 3.4 Necessary Modifications on NLTK

The goal of "text correction for chatbots" requires for the task of text correction to be performed in real-time, i.e. while the chatbot engages with the user. Unfortunately the implementation of assigning probability to a sentence based on a trained n-gram language model was computationally inefficient for our purposes.

It is for this reason that the classes `Lidstone`, `KneserNey`, `KneserNeyInterpolated`, `WittenBell` and `WittenBellInterpolated` from the NLTK library have been modified ac-

[6] https://hyperopt.github.io/hyperopt/
[7] https://github.com/mammothb/symspellpy
[8] https://github.com/jitsi/asr-wer/
[9] https://numpy.org/
[10] https://pandas.pydata.org/
[11] https://scikit-learn.org/
[12] https://www.scipy.org/
[13] https://tqdm.github.io/
[14] https://github.com/samueljamesbell/m2-correct

cordingly. And, in fact, after the modification, much better performance (of the order of 20 times) was achieved, when calculating the prior channel probability of a sentence.

It is outside the scope of the present thesis to list in detail all these modifications. However, all the modifications are available within the `src/utilities.py` file. We intend to contribute the modifications to the NLTK project.

## 3.5 Main Algorithms Developed and Tested

We developed a general class, called `NoisyChannelModel`, implementing the noisy channel model. This class consists of the following parts:

- Implementation of the Beam search algorithm via the `beam_search` method
- Implementation of the Viterbi algorithm for n-grams via the `viterbi` method
- Channel probability generation via the channel_probabilities method which also includes:
  - Poisson channel model via the `poisson_channel_model` method
  - Normalized and inversely proportional to edit distances channel model via the `inv_prop_channel_model` method
- Generation of new sentences via the `generate_suggestion_sentences` method
- Multiple other utility methods for initialization, history handling etc.

We would like to note that the SymSpellPy library was utilized for generating candidate words for the Beam Search and Viterbi algorithms. This choice was crucial as generation of candidate words is a computationally difficult problem.

The implementations are included in the `src/utilities.py` file.

Class `ngramTokenizer` was implemented in order to tokenize sentences properly. It is used to input sentences into the noisy channel models with ngram language models, as well as for the suggested sentences, that the noisy channel model outputs.

For the noisy channel models with the GPT-2 language model we utilize GPT-2's tokenizer which implements Byte Pair Encoding. [11]

## 3.6 Preprocessing of the Data

The preprocessing part is run before all the other parts in order to prepare the data in a suitable form for computer processing. It consists of the following four Python files:

```
src/preprocessing/gen_error_corpus.py
src/preprocessing/wiked.py
src/preprocessing/datasets_split.py
src/preprocessing/preprocess_sentences.py
```

The first three files are dependent on the specific datasets we choose to use, namely WikEd and W&I (which we will discuss further in Section 4.4). The rest of the program is independent of the data, and can be re-used with different data sources.

More specifically, we utilize a "learning corpus" (WikEd in our case) for training and tuning the n-gram language models and an "error corpus" (W&I in our case) for tuning and testing the noisy channel models. In particular:

- The `gen_error_corpus.py` file transforms the W&I error corpus from the M2 format into a csv file that can be easily loaded into a pandas DataFrame in Python.
- The `wiked.py` file transforms the WikEd corpus into a csv file for the same reason.
- The `datasets_split.py` file generates the different data sets from the WikEd and the W&I error corpora that will be used in later steps. These are specifically:
  - The "train n-grams set" composed of 10% from the "learning corpus" (2858851 sentences from WikEd) named `train.csv`
  - The "tune n-gram smoothing hyperparameter set" composed of 2573 sentences from the "learning corpus" named `perpl.csv`
  - The "tune noisy channel model hyperparameter set" composed of 50 sentences from the "error corpus" named `noisy.csv`
  - The "test noisy channel model set" composed of 2500 sentences from the "error corpus" named `test.csv`
- The `preprocess_sentences.py` transforms all the sentences in a suitable format to be used for the later steps. As part of the pre-processing step, we decided to perform limited word normalization and, specifically, case folding, i.e. turning every upper case letter to lower case.

For purposes of better understanding the whole process, it is noted that we use the same variable names both for the DataFrames and for the file names, e.g. `perpl.csv` is loaded into "perpl" DataFrame etc..

## 3.7 Training and tuning of trigram language models

The next part is training the trigram language models and tuning their hyper-parameters It consists of 3 files:

```
src/n-grams/tokenize_train.py
src/n-grams/gen_counter_vocab.py
src/n-grams/ngram_perpl.ipynb
```

In particular:

- The file `tokenize_train.py` prepares the "train n-grams set" into tokens which are consumed by the `gen_counter_vocab.py` file.
- The `gen_counter_vocab.py` file produces the counts of unigrams, bigrams and trigrams encountered in the "train n-grams set" as well as the vocabulary of unique words (cutting off words which appear less than 10 times).
- The `ngram_perpl.ipynb` jupyter notebook uses the "tune n-gram smoothing hyperparameter set" for tuning the hyper-parameters of the trigrams. Specifically these are:
  - The $\alpha$ hyper-parameter for the trigram with Lidstone (add-a) smoothing
  - The $d$ discount hyper-parameter for the trigram with Kneser-Ney smoothing with interpolation

## 3.8 Tuning the Resulting Noisy Channel Models

After we have trained the trigram language models, we continue by tuning the hyper-parameter $\lambda$ of the Noisy channel models. This is implemented in the files:

```
src/noisy/ngrams_noisy.ipynb
src/noisy/gpt2_noisy.ipynb
```

In particular, for the trigrams we tune the hyper-parameter $\lambda$ (as appears in Equation 2.4) for:

- Conditional channel probability according to the Poisson distribution and the trigram Lidstone smoothing for the language model
- Conditional channel probability normalized and inversely proportional to edit distances and Lidstone add-a smoothing for the language model
- Conditional channel probability according to the Poisson distribution and the trigram with Kneser-Ney smoothing with interpolation for the language model
- Conditional channel probability normalized and inversely proportional to edit distances and the trigram with Kneser-Ney smoothing with interpolation for the language model

Finally, for the GPT-2, we only tune for the conditional channel probability according to the Poisson distribution because, as explained in more detail in Sections 4.7 and 4.8, we noticed from the numerical results that performance was considerably worse when the conditional channel probablility was normalized and inversely proportional to edit distances.

## 3.9 Testing the Resulting Noisy Channel Models

After tuning the hyper-parameter $\lambda$ of the noisy channel models we use the “test noisy channel model set” to test the following cases:

- Conditional channel probability according to the Poisson distribution and the trigram Lidstone smoothing for the language model.
- Conditional channel probability according to the Poisson distribution and the trigram with Kneser-Ney smoothing with interpolation for the language model.
- Conditional channel probability according to the Poisson distribution and the GPT-2 for the language model.

The implementation is in the files:

```
src/testing/ngrams_test.ipynb
src/testing/gpt2_test.ipynb
src/testing/test_results.ipynb
```

The last file implements the calculation of the Word Error Rate (WER), for each of the best sentence suggestions, made by the models (from 1st up to 10th), as well as the WER if we were to “magically” choose the best sentence suggestion from all 10.

# Chapter 4

# Experiments and Results

## 4.1 Overview

Having developed an efficient software framework, as it is described in detail in the previous Chapter 3, it may seem an easy task to run all the needed computer experiments with this software framework, and establish the necessary conclusions for the purposes of the present thesis. Unfortunately, this is not true in the present case, because of two important problems, which arise and must be solved, before fully conclusive experiments can be run with the developed software framework. These two problems are:

- The problem of finding the appropriate data for training, tuning and testing the Machine Learning algorithms, and
- The problem of finding the necessary computer resources for running the above algorithms.

Both of these problems are briefly described immediately below and addressed, as best as possible, with the resources available to the author of the present thesis, as described in the remaining part of the present chapter.

## 4.2 The Problem of Finding the Appropriate Data

The appropriate data for training, tuning and testing a Machine Learning algorithm for the automatic text correction for chatbots, which is the subject of the present thesis, are authentic data, from real dialogues between humans and chatbots, in which all the grammatical errors by humans have been appropriately tagged and corrected. Furthermore, “since chatbots are systems designed for extended conversations, set up to mimic the unstructured conversations

| | |
|---|---|
| observed | some uniforms are like the singapore & apos; s school uniform. |
| corrected | some uniforms are like singapore & apos; s school uniform. |
| observed | total 4 points |
| corrected | total 1 points |
| observed | the seven who decreed the fates |
| corrected | the seven who decreed the fates. |
| observed | he was placed in training with edwin parr. |
| corrected | lord clifden was placed in training with edwin parr. |

Table 4.1: A sample from the WikEd error corpus

or 'chats' characteristic of human-human interaction" [9], the appropriate data must also be of sufficiently general scope and not restricted to a specific task domain.

Unfortunately, such data, in the required volumes for training Machine Learning algorithms, are not available in the public domain for use in the present thesis. On the contrary, the available data in the public domain are fundamentally different from the above requirement. Samples of these, publicly available, data are shown in Tables 4.1 and 4.2, taken from the corpora of WikEd and W&I, which will be employed for the purposes of the present thesis, as it will be described in more detail in the next subsections.

As it is clearly indicated in these figures, and can be readily deduced from a more extensive and deep investigation of all the data, which are available in the public domain, all publicly available data have one or more of the following disadvantages, with respect to the purpose of the present thesis:

- They consist mostly of sets of independent pairs of erred and corrected sentences, with no apparent relation between them in their context, i.e. they are not dialogues. Therefore, models such as GPT-2, which also consider previous sentences, become less effective.

- A disproportionally large percentage of the corrections (more than 50%) are not grammatical errors but errors of a different type, e.g. errors in concepts, errors in proper names, errors in dates, errors in numbers, errors in reasoning, etc.

- A relatively large percentage of sentences contain many corrections (e.g. corrections in more than 3 words), which is quite unusual in real human-chatbot dialogues.

Addressing all the above problems, for the purpose of selecting the appropriate data for training a Machine Learning algorithm for the automatic text correction for chatbots, is beyond the scope of the present thesis. Therefore, the data, which will be used in the present thesis

| | |
|---|---|
| observed | i would like to play a world cup game and i also want to win champions league. |
| corrected | i would like to play a world cup game and i also want to win the champions league. |
| observed | i believe that i can be a good helper for two reasons. |
| corrected | i believe that i could be a good helper for two reasons. |
| observed | it was just a dream, very real like. |
| corrected | it was just a dream, but very real. |
| observed | besides, public transport will reduce traffice jam. |
| corrected | besides, public transport will reduce traffic jams. |

Table 4.2: A sample from the W&I error corpus

and will be described below, represent just the best possible compromise from the publicly available data, without any filtering or any other operation on them.

## 4.3 The Problem of the Needed Computer Resources

Machine Learning algorithms, by their nature, are strongly repetitive and need to operate on relatively very large data sets (e.g. the WikEd corpus consists of approximately 28 million sentences). Therefore, running deep learning algorithms, requires very significant computer resources. A clear indication of the computational difficulty of the relevant problems in Machine Learning is the fact that, a very important consideration in this direction, has been the use of parallel processing. Specifically, by using parallel processing, it is possible to train deep learning models with a very large number of data within a reasonable time limit.

It is beyond the scope of the present thesis to fully address the above problem with the needed computer resources. Nevertheless, in an effort to have, as good results, as possible, parallel processing will be used in the present thesis in a limited sense, by employing cloud computing, for all the computing tasks, which are above the capabilities of an ordinary personal computer.

## 4.4 Data Used in the Experiments

### 4.4.1 The WikEd Error Corpus

The WikEd error corpus [6] is a freely available error corpus (under the CC BY-SA 3.0 license), produced in 2014, that consists of 28,588,505 sentences. Each sentence is available both as the original erred sentence and as the corresponding corrected sentence.

The corpus was produced from the entire English Wikipedia revision history from its beginning up to 2014. Every Wikipedia page was iterated over adjacent revisions. A partial filtering was employed in order to avoid incorporating some of the corrections which are obviously unrelated to grammatical errors (such as vandalism, formatting etc.) via a set of hand-written regular expression rules. Afterwards, markup was removed, sentence tokenization was applied, and the pairs of edited sentences were identified via the Longest Common Subsequence algorithm. Edits that added or deleted entire paragraphs were excluded. Finally, the sentences were kept in the final corpus if they met certain requirements:

- The sentence length was between 2-120 tokens
- The length difference was less than 5 tokens
- The relative token-based edit distance with respect to the shorter sentence was smaller than 0.3

Unfortunately, as the original author noted himself [7] , WikEd is a quite noisy corpus, each sentence is completely unrelated to the next one, most of the errors are not grammatical and the corpus includes sentences with a relatively large number of errors. However, because it is freely available under the same license with Wikipedia, it will be used in the present thesis for the purpose of training the n-gram language models. We took a 10% sample (2.86 million sentences) of the corrected sentences from the WikEd corpus for training the n-grams as well as 2500 of the corrected sentences for tuning them.

### 4.4.2 The W&I Error Corpus

Because of the above limitations of the WikEd error corpus, we decided to use additionally the Cambridge English Write & Improve (W&I) error corpus for the tuning and testing of the noisy channel models.

The Cambridge English Write & Improve (W&I) error corpus [2] was produced from essays written by non-native English speakers and is provided in a standardized format by the "Building Educational Applications 2019 Shared Task". Although the provided sentences are composed from essays, they are not clearly separated into essays.

The noisy channel models we developed process each sentence separately, without taking into account any context from previous sentences. We do not expect this to have any considerable impact on the noisy channel models that use an n-gram as a language model but it could be less beneficial for the case of using the GPT-2 as a language model.

## 4.5 Hardware Resource Employed for the Experiments

For the purposes of most of the experiments done, we employed an ordinary personal computer (PC), with computing capabilities comparable to the usual laptop computers. Specifically, we used a 4-core Intel i5 processor with 32GB of available RAM. The only exception has been the GPT-2 model, for which we used an Nvidia Tesla K80 GPU via cloud computing.

## 4.6 General Direction of the Experiments

The general direction of the experiments done, consists of the following three steps:

1. We start with an observed sentence, the erred one, from the error corpus.

2. We employ a noisy channel model, as described in detail in the next section 4.7, having this sentence as an input. The model produces a number of candidate sentences, ordered successively in terms of decreasing probability of been the correct one. The number of the resulting candidate sentences can be arbitrarily predefined and is usually of the order of 10.

3. Finally, we compare the candidate sentences with the correct one from the error corpus, employing the Word Error Rate (WER) metric, as defined in Section 3.1, and summarize the results.

As a simple example, just for purposes of understanding the general direction of the experiments, one corrected sentence, together with the observed one and the suggestions made by a noisy channel model, are shown in Table 4.3.

## 4.7 Short Description of the Experiments Done

In the first series of tests, we compared the Viterbi algorithm to the Beam search algorithm. From all these tests, it resulted conclusively that the Viterbi algorithm was much more computationally inefficient, as compared to the Beam search algorithm, with differences in the computer-time needed, of the order of 20 times. This can be also theoretically understood by taking into account the computational complexity of both algorithms, as they have been presented in Section 2.8. We attempted bypassing this problem by pruning the search space but the Viterbi algorithm's performance was still unsatisfactory. It is for this reason that it was decided to use only the Beam search algorithm for all the following experiments.

| | |
|---|---|
| observed | besides, public transport will reduce traffice jam. |
| corrected | besides, public transport will reduce traffic jams. |
| 1st | besides, public transport will reduce traffic jams. |
| 2nd | besides, public transport will reduce traffic jam. |
| 3rd | besides, public transport will reduce traffice jam. |
| 4th | beside, public transport will reduce traffice jam. |
| 5th | brides, public transport will reduce traffice jam. |
| 6th | desires, public transport will reduce traffice jam. |
| 7th | resides, public transport will reduce traffice jam. |
| 8th | asides, public transport will reduce traffice jam. |
| 9th | sides, public transport will reduce traffice jam. |
| 10th | b-sides, public transport will reduce traffice jam. |

Table 4.3: An example as corrected by "trigramKNI"

Next, we performed a first series of experiments, employing only the WikEd corpus, which are briefly described immediately below, together with the computer time needed:

1. Training of the trigram language models, employing the WikEd corpus with 3 million sentences. The computer time needed was approximately 8 PC hours. The output of the training was the counting of the unigrams, the bigrams and the trigrams in the corpus, as well as the creation of the vocabulary for the whole corpus, as described in Section 3.7.

2. Tuning of the trigram language models, employing the WikEd corpus with 2500 sentences and 100 evaluation trials. The computer time needed was approximately 8 PC hours. The output of the tuning was the optimized value of the hyper-parameters $\alpha$ and $d$ of the n-grams, as described in Section 3.7.

3. Tuning of the noisy channel models for n-grams as language models, employing the WikEd error corpus with 50 sentences and 20 evaluation trials, and employing the relevant previously optimized hyper-parameters. The computer time needed was approximately 12 PC hours. The output of the tuning was the optimized value of the hyper-parameter $\lambda$, considered for all the 4 possible alternative cases of noisy channel models with n-grams, as described in detail in Section 3.8. From the numerical results of these alternative cases, it was concluded that the conditional channel probability, according to the Poisson distribution, was always performing much better than the conditional channel probability, normalized and inversely proportional to edit distances. Based on this fact, in the next series of experiments, we employed only the conditional channel probability according to the Poisson distribution.

4. Tuning of the noisy channel models for the GPT-2 as language model, employing the WikEd error corpus with 50 sentences and 20 evaluation trials. The computer time needed was approximately 4 cloud-computing hours. The output of the tuning was the optimized value of the hyper-parameter $\lambda$. Because of the previous conclusion, in this experiment, we employed only the conditional channel probability according to the Poisson distribution.

5. Testing of the noisy channel models for n-grams as language models, employing the WikEd error corpus with 2500 sentences, and employing all the relevant previously optimized hyper-parameters. The computer time needed was approximately 8 PC hours. The output of the testing was the 10 best candidate sentences for each erred sentence in the testing corpus. Employing the above 10 best candidate sentences, we additionally calculated the Word Error Rate (WER) of each one of them, as compared to the corrected one.

6. Testing of the noisy channel models for the GPT-2 as language model, employing the WikEd error corpus with 2500 sentences, and employing the relevant previously optimized hyper-parameters. The computer time needed was approximately 8 cloud-computing hours. Again the output of the testing was the 10 best candidate sentences for each erred sentence in the testing corpus. And again, employing the above 10 best candidate sentences, we additionally calculated the Word Error Rate (WER) of each one of them, as compared to the corrected one.

However, the numerical results obtained for the Word Error Rate (WER), from the above first series of experiments, were quite noisy and no definite conclusion could be drawn from them. We understood that this is due to the unsuitability of the WikEd error corpus, which was employed for tuning and testing, in this series of experiments, without any filtering. And, because we didn't want to introduce any bias in the data by filtering the WikEd error corpus, we, instead, used an alternative error corpus for tuning and testing, namely the W&I error corpus.

We then performed a second series of experiments, exactly the same as the above experiments from numbers 3 to 6, by using the W&I error corpus instead of the WikEd error corpus. The results by using this new error corpus were more satisfactory and are presented immediately below.

## 4.8 Numerical Results with the W&I Corpus for Tuning and Testing

### 4.8.1 Tuning the Hyper-parameter $\lambda$ of the Noisy Channel Models

Immediately below, in Table 4.4, the results for the optimal values of the hyper-parameter $\lambda$ are shown, after tuning the noisy channel models, as described in steps 3 and 4 of the previous section. It is noted that, as it was also explained in the previous section, we show only the results with the W&I error corpus.

| | $\lambda$ | WER |
|---|---|---|
| Lidstone + ED | 0.376 | 0.690 |
| Lidstone + Poisson | 0.310 | 0.328 |
| KNI + ED | 0.381 | 0.684 |
| KNI + Poisson | 0.205 | 0.264 |
| GPT-2 | 1.131 | |

Table 4.4: Results from tuning the $\lambda$ hyper-parameter

### 4.8.2 Comparing the alternative Noisy Channel Models

All Noisy Channel Models compared are utilizing the Beam search algorithm and the conditional channel probability according to the Poisson distribution. The models compared differ only in the language models employed. Specifically the language models employed are the "trigram-L", "trigram-KNI" and "GPT-2 Small", as described in sections 3.1 and 3.9.

In order to compare the above noisy channel models, we determined the 10 best candidate sentences, i.e. the sentences with the resulting highest probability, for each model, employing 2500 sentences of the W&I error corpus, as described in steps 5 and 6 of the previous section. Next, we grouped the candidate sentences in terms of their suggestion order, i.e. we grouped together the 1st candidate sentences (highest probability), then the 2nd candidate sentences (next highest probability), and so on, until the 10th candidate sentences. For each one sentence in these groups, we calculated the Word Error Rate (WER), by comparing the candidate sentences with the correct ones in the W&I error corpus. Finally, we computed the resulting mean of the Word Error Rate (WER) in each group.

The results of this comparison are shown immediately below in Table 4.5.

In the same table, there is an additional line, denoted as "best choice", which results by forming a new group, composed of the candidate sentences, which have the lowest Word Error

| | Language Model used on the Noisy Channel Model | | |
|---|---|---|---|
| candidate sentence | trigram-L | trigram-KNI | GPT-2 Small |
| 1st | 0.337 | 0.265 | 0.267 |
| 2nd | 0.341 | 0.302 | 0.298 |
| 3rd | 0.340 | 0.306 | 0.314 |
| 4th | 0.335 | 0.308 | 0.322 |
| 5th | 0.332 | 0.307 | 0.325 |
| 6th | 0.328 | 0.308 | 0.330 |
| 7th | 0.325 | 0.307 | 0.335 |
| 8th | 0.323 | 0.305 | 0.340 |
| 9th | 0.318 | 0.304 | 0.346 |
| 10th | 0.311 | 0.296 | 0.354 |
| best choice | 0.235 | 0.222 | 0.240 |

Table 4.5: Word Error Rate per candidate sentence for each Noisy Channel Model

Rate (WER) among all 10 suggestions, that are taken into account. For this new group, the mean Word Error Rate (WER) for all 2500 sentences is shown separately for each model.

From the above table, it results, that the candidate sentence with the highest probability (that is the 1st candidate sentence), is not always the sentence with the smallest Word Error Rate (WER) as compared to the correct one. It is for this reason that an additional comparison was deemed necessary. Specifically, for each one of the 2500 sentences employed, the order of the “best choice”, i.e. the order in which the sentence with the smallest Word Error Rate (WER) is suggested by the Noisy Channel Model (1st, 2nd, etc, 10th), was determined. And Table 4.6 below summarizes the results.

| | Language Model used on the Noisy Channel Model | | |
|---|---|---|---|
| order of best choice | trigram-L | trigram-KNI | GPT-2 Small |
| 1st | 481 | 1332 | 1372 |
| 2nd | 357 | 418 | 680 |
| 3rd | 301 | 284 | 401 |
| 4th | 312 | 251 | 306 |
| 5th | 300 | 247 | 253 |
| 6th | 307 | 215 | 237 |
| 7th | 291 | 222 | 223 |
| 8th | 309 | 231 | 198 |
| 9th | 310 | 230 | 198 |
| 10th | 396 | 355 | 207 |

Table 4.6: Number of best choices per candidate sentence for each Noisy Channel Model

For example, from the above table and for the “trigram-L” language model, it results that in 481 sentences the best choice was the 1st suggestion, in 357 sentences the best choice was the 2nd suggestion and so on and in 396 sentences the the best choice was the 10th suggestion. It is noted that the sum of the numbers for each model is more than 2500, because there are cases where different suggestions resulted in the same Word Error Rate (WER).

## 4.9 Conclusions

The first conclusion from the above experiments is that the Viterbi algorithm is computationally inefficient for trigrams. It could be employed efficiently for bigrams but generally bigrams do not give good results as language models.

The second conclusion from the above experiments is that the conditional channel probability, according to the Poisson distribution, performed much better than the one, which is normalized and inversely proportional to edit distances. Furthermore, while their hyper-parameter wasn't tuned, it was comparable for both methods.

The third conclusion from the above experiments is that selecting the appropriate corpus, especially for tuning and testing the noisy channel models, is very important. In fact the WikEd corpus, if used alone without filtering, leads to very noisy results.

The fourth conclusion from the above experiments is that, if no further improvements are made to the models, the "trigram-KNI" and the "GPT-2 Small" models are much better than the "trigram-L" model, because:

- They have much lower average Word Error Rate (WER) for the the 1st and the 2nd candidate sentences (Table 4.5), and
- Their 1st candidate sentence is much more frequently the "best choice" (Table 4.6).

The fifth conclusion from the above experiments is that, if a further improvement is employed in the models, so that the "best choice" can be somehow identified from the candidate sentences, the "trigram-KNI" model results with the least Word Error Rate (WER) (Table 4.5, last line).

## 4.10 Directions for Further Needed Experiments

The directions for further experiments on the subject are quite clear from the above discussion of the results.

- Filtering of the publicly available data: As already stated above, the publicly available data suffer from a number of problems, including the fact that they consist of sentences unrelated to each other, that they contain a disproportionally large percentage of non-grammatical errors and that many sentences contain an unusually large number of errors. Clearly, with an appropriate filtering, the publicly available data may become much more useful for the purposes of the Machine Learning of chatbots. However, this filtering requires extensive work and must be done with great care, so that it does not result in data, which are (statistically) biased in a certain direction. As a simple example, some immediate steps we could take for the W&I error corpus, would be to manually annotate the beginning and the end of the essays, as well as partially filter out errors that are obviously unrelated to chatbot dialogues.

- Use of proprietary data: As already stated above, the appropriate data for training and testing of chatbots are authentic data, from real dialogues between humans and chatbots, in which all the grammatical errors by humans have been appropriately tagged and corrected. Although such data are not publicly available, many businesses may already have such data from their operations. Furthermore, once an automatic chatbot is employed by a business, all the dialogue data from its previous chats, appropriately corrected, may be used for the training of the specific chatbot, thus continuously increasing its correction capability.

- Using more computer resources: As already noted above, the computer resources required for the Machine Learning of chatbots are quite extensive. It is clear that, in order to achieve relatively conclusive results, all the above experiments must be run on computers with much more resources than a simple personal computer. Furthermore, it may be possible to design a computer specifically for the purpose of the Machine Learning of chatbots, e.g. by appropriately specifying and connecting additional or more powerful GPUs (Graphics Processing Units) to it for longer periods of time.

- Using parallel processing: Parallel processing is not possible with all the models developed in the present thesis. However, whenever it is possible and the necessary computer resources are available, it is clearly a very efficient technique and should definitely be employed. Specifically, the training of the n-grams could be done in parallel, incorporating a larger corpus and utilizing more processors. Additionally, parts of the Beam search algorithm can also be parallelized (e.g. up to the size of the beam width).

- Comparing the WikEd corpus with other, larger and more general, corpora for the purpose of training the language models: Employing pre-trained n-gram language models (trained by larger and more general corpora), as well as training the GPT-2 language model with the WikEd corpus (instead of using the pre-trained GPT-2), we can examine the efficiency of the WikEd corpus as a training corpus for language models, employed within the context of the noisy channel model.

# Chapter 5

# Conclusions and Future Work

## 5.1 Conclusions

The first important conclusion of the present thesis is that the “Noisy Channel” model, as it was presented and analyzed mathematically, is an appropriate Machine Learning model, for addressing the problem in the present thesis, namely the automatic text correction for chatbots. Furthermore, by employing Python and C++ libraries, and partially modifying them, it became possible to develop a computer program, capable of performing efficiently the necessary training-tuning-testing procedures in Machine Learning, for the purpose of efficiently solving the problem.

Based on the software framework developed, computer experiments were performed, using the publicly available corpora of “WikEd” and “W&I”, and employing a simple personal computer and limited cloud computing for the required parallel processing. Six alternative noisy channel models were compared, by selecting:

- Between the trigram language model with Lidstone smoothing, the trigram language model with Kneser-Ney smoothing with interpolation, and the GPT-2 language model,
- Also, for the conditional channel probability, between the normalized and inversely proportional to edit distances, and the one according to the Poisson distribution,
- And, finally, for selecting the best candidate sentences, between the Beam search and the Viterbi algorithms.

From these experiments, it resulted that:

- The Beam search algorithm is much more efficient than the Viterbi algorithm.

- The conditional channel probability according to the Poisson distribution is much more effective than the one according to normalized and inversely proportional to edit distances.

- That selecting the appropriate error corpus is very important, especially for tuning and testing.

- The trigram language model with Kneser-Ney smoothing with interpolation as well as the GPT-2 appear to be the most efficient language models for the noisy channel model if the metric of the Word Error Rate is employed.

The above conclusions of the performed experiments can, at best, be considered preliminary, because of two fundamental reasons:

- Because the data employed in the comparisons and, also, all the publicly available data, are not appropriate, without careful filtering, for the Machine Learning tasks, needed for the specific problem addressed in the present thesis. As it was noted in the main body of the thesis, the appropriate data for these tasks are authentic data, from real dialogues between humans and chatbots, in which all the grammatical errors made by humans have been appropriately tagged and corrected. Unfortunately, such data are not available presently in the public domain.

- Because the needed Machine Learning tasks are very computer intensive and require significant computer resources, well outside the capabilities of a simple personal computer and limited cloud computing, employed for the purposes of the present thesis.

In order to address the above two practical problems and reach final conclusions about the possible, alternative, noisy channel models possible, it is needed to perform further numerical experiments, employing the software framework developed in the present thesis, with the following additional resources, as explained in more detail in Section 4.10:

- Application of appropriate preliminary filtering to the, publicly available, data (i.e. to the “WikEd” and “W&I” corpora), so that they resemble more closely the real dialogues between chatbots and humans.

- Use of proprietary data, instead of only the “WikEd” and “W&I” corpora, with authentic dialogues, between chatbots and humans, in which all the grammatical errors have been appropriately tagged and corrected in advance.

- Use of more computer resources, with significant Graphic Processing Unit (GPU) capabilities, for longer periods of time.

- Use of parallel processing capabilities in the developed models where this is possible.
- Comparing the WikEd corpus with other, larger and more general, corpora for the purpose of training the language models, within the context of the noisy channel model.

## 5.2 Directions for Further Work

The software framework developed in the present thesis can form the basis for adding new capabilities to the language models employed for the present problem. Some of these additional capabilities, which have also been suggested in some of the more recent (August 2019) research papers on the subject, are:

- Performing additional right-to-left parsings: [2] All the usual models, including the models in the present thesis, use the left-to-right direction for parsing, since this is the direction of writing in most languages (with notable exceptions). Adding the reverse direction for parsing, in addition to the normal one, for the purpose of training the system, has been reported to result in a significant increase in the performance of the system, without requiring additional data sets.
- Using additionally a Part-of-Speech (PoS) tagger: [2] Part-of-speech tagging is a well-studied task in natural language processing. By means of this tagging, each word in a sentence is assigned to one specific part of speech (such as a noun, a verb, a pronoun, a preposition, an adverb, a conjunction, a participle or an article). It is anticipated that, if this tagging is added to the models of this thesis, it can result in a significant increase in the performance of the system.
- Employing additionally specific confusion sets: [4, 15] A confusion set is a small set of words that are likely to be confused with one another. Confusion sets can be utilized for proposing new candidate words for correction. It is anticipated that, if a sufficient number of confusion sets is developed in advance and employed, in conjunction with the particular methods of the present thesis, it can also result in a significant increase in the performance of the system.
- Generating additional artificial errors, and/or additional artificial parallel sentences, and/or synthesizing the noisy sentences: [4, 15] In view of the afore-mentioned problem in Section 4.2, with the scarcity of appropriate data for the training, tuning and testing of the Machine Learning models, one possibility is to generate, more or less artificially, such a large data set from a much smaller one. It is clear that the important question in this

respect is how the artificial data must be generated, so that the resulting final set is representative, as much as possible, of the real-world situation. This is still an open question, which, if solved satisfactorily, will greatly enhance the range of applications of Machine Learning models for chatbots.

- Pipelining with adapted Neural Machine Translation (NMT) based systems: [2, 3, 5, 7] In view of the recent state-of-the-art results achieved by the latest Hybrid SMT-NMT systems, adapted for Grammatical Error Correction, we could utilize NMT-based systems, such as a Transformer encoder-decoder model, for post-editing of the output of our software framework or even for re-scoring it, thus greatly improving the final result.

# References


[1] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages 115–123. JMLR.org, 2013. URL http://www.jmlr.org/proceedings/papers/v28/bergstra13.pdf.

[2] Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. The BEA-2019 Shared Task on Grammatical Error Correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 52–75, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4406. URL https://www.aclweb.org/anthology/W19-4406.

[3] Yo Joong Choe, Jiyeon Ham, Kyubyong Park, and Yeoil Yoon. A Neural Grammatical Error Correction System Built On Better Pre-training and Sequential Transfer Learning. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 213–227, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4423. URL https://www.aclweb.org/anthology/W19-4423.

[4] Mariano Felice and Zheng Yuan. Generating artificial errors for grammatical error correction. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 116–126, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. doi: 10.3115/v1/E14-3013. URL https://www.aclweb.org/anthology/E14-3013.

[5] Simon Flachs, Ophélie Lacroix, and Anders Søgaard. Noisy Channel for Low Resource Grammatical Error Correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 191–196, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4420. URL https://www.aclweb.org/anthology/W19-4420.

[6] Roman Grundkiewicz and Marcin Junczys-Dowmunt. The WikEd Error Corpus: A Corpus of Corrective Wikipedia Edits and its Application to Grammatical Error Correction. In Adam Przepiórkowski and Maciej Ogrodniczuk, editors, *Advances in Natural Language Processing – Lecture Notes in Computer Science*, volume 8686, pages 478–490. Springer, 2014. URL http://emjotde.github.io/publications/pdf/mjd.poltal2014.draft.pdf.

[7] Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. 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*, pages 252–263, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4427. URL https://www.aclweb.org/anthology/W19-4427.

[8] Marcin Junczys-Dowmunt, Roman Grundkiewicz, Shubha Guha, and Kenneth Heafield. Approaching Neural Grammatical Error Correction as a Low-Resource Machine Translation Task. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 595–606, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1055. URL https://www.aclweb.org/anthology/N18-1055.

[9] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (3rd Edition draft)*. October 2019. URL https://web.stanford.edu/~jurafsky/slp3/.

[10] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. June 2018. URL https://openai.com/blog/language-unsupervised/.

[11] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. February 2019. URL https://openai.com/blog/better-language-models/.

[12] Irene Solaiman, Miles Brundage, Jack Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, Alec Radford, Gretchen Krueger, Jong Wook Kim, Sarah Kreps, Miles McCain, Alex Newhouse, Jason Blazakis, Kris McGuffie, and Jasmine Wang. Release Strategies and the Social Impacts of Language Models. *arXiv:1908.09203 [cs]*, November 2019. URL http://arxiv.org/abs/1908.09203. arXiv: 1908.09203.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V.

Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[14] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv:1910.03771 [cs]*, October 2019. URL http://arxiv.org/abs/1910.03771. arXiv: 1910.03771.

[15] Ziang Xie, Guillaume Genthial, Stanley Xie, Andrew Ng, and Dan Jurafsky. Noising and Denoising Natural Language: Diverse Backtranslation for Grammar Correction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 619–628, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1057. URL https://www.aclweb.org/anthology/N18-1057.

# Appendix A

# Implementation of the software framework

The source code of the software framework that was developed for the purposes of the present thesis is released under the MIT license and is available on github[1]. Detailed description of the file structure of the source code can be found on chapter 3.

[1] https://github.com/palasso/text-correction-chatbot