8-15-2023

OCR Post-processing Using Large Language Models

Mahdi Hajiali
University of Nevada, Las Vegas

OCR Post-processing Using

Large Language Models

By

Mahdi Hajiali

Doctor of Philosophy – Computer Science
University of Nevada, Las Vegas
2023

A dissertation submitted in partial fulfillment
of the requirements for the

Doctor of Philosophy – Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
August 2023

UNLV | GRADUATE COLLEGE

Dissertation Approval

The Graduate College
The University of Nevada, Las Vegas

May 24, 2023

This dissertation prepared by

Mahdi Hajiali

entitled

OCR Post-processing Using Large Language Models

is approved in partial fulfillment of the requirements for the degree of

Doctor of Philosophy – Computer Science
Department of Computer Science

Kazem Taghva, Ph.D.
Examination Committee Chair

Laxmi Gewali, Ph.D.
Examination Committee Member

Wolfgang Bein, Ph.D.
Examination Committee Member

Ashok Singh, Ph.D.
Graduate College Faculty Representative

Alyssa Crittenden, Ph.D.
Vice Provost for Graduate Education & Dean of the Graduate College

Abstract

Optical Character Recognition (OCR) technology transforms textual visuals into an electronically readable, non-graphical format of the text. This allows the editing and other text manipulation of the content by language technology software such as machine translation, text comprehension, query-answering systems, and search engines. While Optical Character Recognition (OCR) systems continually progress towards greater precision, several complications persist when dealing with low-resolution source images or those with multicolored backgrounds. Consequently, the text derived from OCR necessitates additional refinement to optimize accuracy, beneficial for various subsequent applications. It is recognized that the character accuracy of OCR generated text may influence certain natural language processing tasks, including Information Retrieval, Named-Entity Recognition, and Sentiment Analysis.

Post-processing techniques for Optical Character Recognition (OCR) consist of three fundamental stages of identifying incorrect words, producing a list of potential corrections, and selecting the accurate word from the list to replace the erroneous word. In this work, we are using large language models and word embeddings to detect recognition errors caused by the OCR software. In addition, we use the generative capabilities of these language models to suggest correction candidates to possibly fix the errors. Our work also includes the development of tools that can be used to further improve the OCR post processing technologies.

Acknowledgements

Throughout the pursuit of my doctoral degree, many individuals have played a crucial role in helping me achieve this significant milestone. First, my amazing wife, who has always been there for me, cheering me on and supporting me. Her constant encouragement and understanding have been invaluable in helping me persevere through the tough times.

I also want to thank my parents for their love and guidance. Their dedication and sacrifices have played a significant role in shaping the person I am today.

A special thank you goes to my PhD adviser, Dr. Kazem Taghva, for his exceptional guidance, expertise, and patience throughout this journey. His constant support and dedication to helping me grow as a researcher have truly made a big difference in my academic journey.

Finally, my gratitude extends to my committee members for their valuable insights and feedback on my work.

The collective support of all these individuals has been instrumental in shaping my success, and I could not have accomplished this without them.

Mahdi Hajiali

University of Nevada, Las Vegas

August 2023

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

OCR (Optical Character Recognition) software transforms textual visuals into an electronically readable, non-graphical format of the text. (e.g. ASCII). More broadly, OCR accepts an image obtained from a scanner as input and performs a sequence of operations to extract text content. These operations include pre-processing of the image to remove noise, zoning to separate text from non-text components, and segmentation to isolate individual characters. The development of OCR software has a long history and is a partly summarized in [2]. The OCR software are mainly commercial. The only open source software is Tesseract developed by Ray Smith [3] at Hewlett-Packerd (HP) and donated to Information Science Research Institute at the University of Nevada, Las Vegas (UNLV). Later on, after some initial clean up of the code, it was taken over by Google for further development.

The text output of OCR allows the editing and other text manipulation of the content by language technology software such as machine translation, text comprehension, query-answering systems, and search engines. Despite the ongoing advancements made to augment the accuracy of OCR systems, there are still many challenges that persist, particularly when the source images have poor resolution or backgrounds with intense color schemes [4]. Therefore, the output from OCR necessitates further post-processing in order to enhance the results and increase precision.

Moreover, ensuring the quality of OCRed texts holds significant importance in achieving optimal performance across various Natural Language Processing (NLP) tasks [5]. These tasks include Information Retrieval [6], Named-entity recognition (NER), Sentiment Analysis, Topic Modelling, etc.

The steps involved in post-processing Optical Character Recognition (OCR) primarily encompass three tasks of identifying the errors, creating a list potential alternatives (candidates), and rectify-

ing the errors. There are other tasks associated with post processing such as identification of meta data, sentences, paragraphs, and reading order (two column vs one column articles). This work does not address the latter component of the post processing.

The remainder of this dissertation is structured as follows: In Chapter 2, the necessary preliminary concepts and background information are presented to provide the reader with a clear understanding of the research context. Chapter 3 conducts an extensive literature review to identify and analyze existing research studies on the topic, highlighting the gaps and limitations in the current knowledge. Chapter 4 describes the methodology employed in this research and Chapter 5 presents the results of the research study. Finally, Chapter 6 provides a summary of the research findings and draws conclusions based on the research outcomes, while also identifying areas for future research.

Chapter 2

Preliminaries

2.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a sub-field of Machine Learning (ML) which is used to extract insights from linguistic data. NLP is concerned with developing algorithms and models that can understand and process human language in all its complexity, including its syntax and semantics. NLP employs a range of techniques from various disciplines, including linguistics, statistics, machine learning, and computer science, to create methods that can perform tasks such as Machine Translation, Text Summarization, Text Generation, Sentiment Analysis, Information Retrieval, Topic Modeling, Question Answering, and OCR Post-Processing.

NLP faces various challenges when it comes to comprehending human language. For example, words can mean differently in different contexts, and there are so many languages and dialects worldwide that need to be accounted for. In addition, it's not just about understanding individual words but also how they fit together in a sentence or conversation to convey meaning. Sometimes, people use sarcasm or irony, which can be difficult for machines to understand. All of these challenges require researchers to come up with better ways for machines to understand and interpret human language.

There are two main categories of methods to solve NLP tasks: rule-based approaches and machine learning-based approaches. In the following, we will discuss the differences between these two approaches in more detail.

2.2 Rule-based approaches (Top-Down)

The fundamental idea behind Natural Language Processing (NLP) is to enable computers to understand human language. NLP, prior to late 1980s, predominantly employed rules-based systems that utilized linguists' handcrafted rules to dictate how computers analyzed language. These rules were responsible for guiding computers on how to analyze and interpret language. Rule-based methods are considered top-down because they start with a predefined set of rules or instructions that govern how a computer should analyze language. However, this approach had its shortcomings because developing rules for each language or each task required significant time and effort. Also the rules were often inflexible in capturing the intricate nuances of language. In other words, the rules were often too rigid to convey the subtleties of language.

In 1957, Noam Chomsky's introduction of syntactic structures brought about a true revolution in the field of NLP [7]. According to Chomsky's theory, language is not just a product of learned behaviors, but rather an inherent capability of the human mind that is biologically hardwired. He proposes the existence of a universal grammar that underpins all human languages, and this "universal grammar" can be uncovered and formalized through the study of language structure.

While Chomsky's theories have made a significant contribution to the field of linguistics and rule-based approaches, they have not been without criticism, especially concerning language acquisition and learning. Some linguists have raised concerns that his theories overemphasize innate knowledge and do not adequately address the role of experience and learning in language acquisition.

Chomsky's view was challenged by Hockett, who argued that language is not as well-defined, stable, or formal as previously thought. Hockett discovered several limitations with Chomsky's approach, particularly the belief that language is a precise, unchanging, and formal system that can only exist in an idealized setting [8].

NLP underwent a significant transformation in the late 1980s when machine learning (ML) algorithms were introduced for language processing. This change was a result of advancements in computational power, as well as a shift away from Chomsky's theories of linguistics, which discouraged the use of corpus linguistics, the base of the Machine Learning approach.

2.3 Machine Learning-based approaches (Bottom-Up)

With the rise of machine learning and corpus-based approaches to NLP, rule-based methods have been largely replaced by more data-driven and flexible techniques. Machine learning methods are

considered bottom-up because they start with data and use algorithms to learn patterns and relationships within that data. These methods allow the computer to learn from the data and identify patterns that may not have been explicitly programmed by humans.
Traditional machine learning methods and neural network-based methods have emerged as two of the main approaches for processing and analyzing text data using ML techniques.
In the following sections, we will discuss the strengths and weaknesses of these two Machine Learning-based approaches for NLP. Specifically, we will examine how traditional ML methods and neural network-based methods differ in their ability to handle complex language structures, their efficiency in training and inference, and their suitability for different NLP tasks.

2.3.1 Traditional Machine Learning Methods

These methods for NLP have been used for decades to solve various NLP tasks. These methods utilize statistical models and algorithms to learn from input features and their relationships to output labels. To accomplish this, domain experts must perform extensive feature engineering by manually selecting and engineering relevant features from the raw input data. The quality of the features directly impacts the performance of the model, and feature engineering can be a time-consuming and challenging task. Additionally, traditional ML methods struggle to handle complex language structures, such as sarcasm and irony due to their reliance on hand-crafted features. These methods are generally more efficient in training and inference than neural networks. This efficiency is due to the lower complexity of the models, which results in faster training times and lower memory requirements. Traditional ML methods are particularly useful in scenarios where there is a limited amount of training data.
An example of a Traditional machine learning approach is to use a Naive Bayes classifier or a Support vector machines (SVM) classifier that is trained on a bag-of-words representation of text to categorize sentiment into positive, negative, or neutral categories [9].

2.3.2 Neural Networks-based methods

The primary difference between traditional ML methods and neural network-based methods is in their approach to feature engineering and their ability to handle intricate language patterns. Neural network-based methods have the ability to learn the representation of the input text directly from raw data. This feature eliminates the need for extensive feature engineering, making the training

process more efficient. Neural networks can handle complex language structures, making them more suitable for tasks that require understanding of the deeper meaning of text, such as sentiment analysis and language translation.

In the 2010s, deep learning became popular in natural language processing, leading to the widespread use of deep neural networks. These networks are neural networks that have more than three layers including input and output. In other words, they have more than one hidden layer [10]. These methods have demonstrated the ability to achieve superior results in a variety of NLP tasks. This shift towards deep learning was prompted by several factors including advancements in deep learning algorithms, the availability of larger datasets, and more powerful computing resources.

Deep neural network-based models for NLP include Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), Long Short-Term Memory Networks (LSTMs), and Transformer-based models like BERT and GPT.

An example of using an Artificial Neural Network-based method is to use a Multi-layer Perceptron (MLP) classifier to categorize documents [11].

2.4 Text Representation

Data is the most important part of any data science project. Unlike images, which can be represented as matrices of pixels, text data consists of sequences of symbols such as letters, words, and sentences that convey meaning and context. Moreover, language is highly nuanced and context-dependent, and the same word or phrase can have different meanings in different contexts. The primary objective of natural language processing (NLP) is to construct a representation of the text that adds structure to unstructured natural language. This is achieved by transforming raw text data into a numerical or mathematical format that can be easily processed by machine learning models.

To effectively represent text data and capture its meaning and context, NLP researchers have devised a range of techniques. In the following, I will discuss the evolution of text representation methods in NLP.

2.4.1 One-Hot Encoding

The earliest approach to text representation in NLP involved the use of one-hot encoding. This method assigns a unique integer index to each word in the vocabulary and creates a vector of zeros

with a length equivalent to the vocabulary size. The vector is then populated with a value of one at the index corresponding to the word in the text.

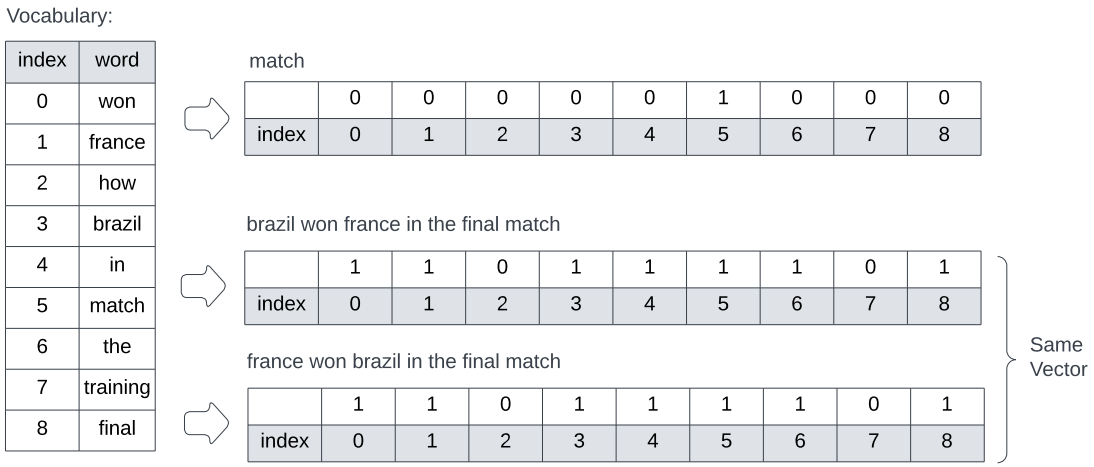

Figure 2.1: An example of One-Hot Encoding

While this technique is easy to implement, it has several limitations such as:

1. Loss of semantic meaning: One-hot encoding treats each word as a separate entity and fails to capture the semantic meaning and relationships between words. This can lead to poor performance on tasks that require contextual understanding. For example in Figure 2.1, both sentences are represented by the same vector, even though they have different meanings.

2. Sparsity: One-hot encoding creates a sparse feature space, with most values being zero, which can lead to inefficient computations and slower training times.

3. High dimensionality: One-hot encoding creates a feature space with a high number of dimensions, making it computationally expensive and difficult to analyze.

4. Inability to handle out-of-vocabulary words and lack of generalization: One-hot encoding requires a fixed vocabulary size and cannot handle new words that are not in the vocabulary. This can result in a loss of information.

2.4.2 Bag-of-Words (BoW)

A subsequent development in text representation was the bag of words (BoW) technique. In this method, the text is treated as a bag of words, and the order in which they appear is not considered.

Each word is viewed as a separate feature, and its occurrence in the text is counted and stored in a vector. The vector represents the text, and its size corresponds to the size of the vocabulary. An example of this method is shown in Figure 2.2.

While the BoW model is a simple and effective text representation technique, it has several limitations that can affect its usefulness in certain NLP applications. This method is more expressive than one-hot encoding and can capture the frequency of words in the text. Nevertheless, there are some potential drawbacks to using BoW text representation:

1. Loss of Sequence Information: BoW representation does not take into account the order of words in a sentence or document. It treats each word as an independent unit and ignores the context in which the word appears. This can result in a loss of important semantic information that is conveyed by the order and structure of words in a sentence.

2. Vocabulary Size: To utilize the Bag-of-Words (BoW) representation, it is necessary to generate a vocabulary containing all distinct words present in the corpus. This can result in a large and sparse feature space, which can increase computational costs and lead to overfitting in the model.

3. Ignoring Rare Words: BoW representation tends to ignore rare words, which can be important for certain tasks. For example, in sentiment analysis, rare words can often carry a strong emotional meaning and may be critical for accurate classification.

4. Lack of Semantic Understanding: BoW representation does not capture the semantic relationships between words. It treats each word as a separate entity and does not take into account the meaning and relationships between them. This can limit the model's ability to understand the underlying meaning of text data.

Document	the	ball	world	france	and	brazil	was	last	match	vs	intense	between
the match between france and brazil was the last match	2	0	0	1	1	1	1	1	2	0	0	1
the france vs brazil match was intense intense intense	1	0	0	1	0	1	1	0	1	1	3	0

Figure 2.2: An example of BoW text representation

2.4.3 TF-IDF-based method

To address the drawbacks of BoW, the TF-IDF (Term Frequency multiplied by Inverse Document Frequrncy) technique was introduced. In this method, the frequency of words in the text is weighed based on their frequency across all the documents in the corpus. The TF-IDF score is determined by multiplying the term frequency with the inverse document frequency. The term frequency represents the frequency of a word in the text, while the inverse document frequency indicates the rarity of the word across all the documents. This representation is effective in capturing the importance of words in the text and reducing the impact of common words such as "the" and "and." TF-IDF is frequently used in information retrieval tasks.

TF-IDF value of term x within document y is calculated as follows:

$$TF = \frac{\text{Number of times term x appears in document y}}{\text{Number of terms in document y}}$$

$$IDF = \log_e \frac{\text{Number of documents present in the corpus (N)}}{\text{Number of documents where term x has appeared}}$$

$$TF - IDF = TF(x, y) \times IDF(x) = tf_{x,y} \times \log_e \frac{N}{df_x}$$

Let's assume that we have two documents, namely:

Document 1: the match between france and brazil was the last match

Document 2: the france vs brazil match was intense intense intense

We first need to calculate the term frequency (TF) for each document, which is the number of times each word appears in the document:

TF("the", Document 1) = 2/9

TF("match", Document 1) = 2/9

TF("between", Document 1) = 1/9

TF("france", Document 1) = 1/9

TF("and", Document 1) = 1/9

TF("brazil", Document 1) = 1/9

TF("was", Document 1) = 1/9

TF("last", Document 1) = 1/9

TF("vs", Document 1) = 0/9

TF("the", Document 2) = 1/8

TF("match", Document 2) = 1/8

TF("france", Document 2) = 1/8

TF("vs", Document 2) = 1/8

TF("brazil", Document 2) = 1/8

TF("was", Document 2) = 1/8

TF("intense", Document 2) = 3/8

Next, we can calculate the inverse document frequency (IDF) for each word, which is a measure of how important the word is in the corpus:

IDF("the") = log(2/2) = 0

IDF("match") = log(2/2) = 0

IDF("between") = log(2/1) = 0.693

IDF("france") = log(2/2) = 0

IDF("and") = log(2/1) = 0.693

IDF("brazil") = log(2/2) = 0

IDF("was") = log(2/2) = 0

IDF("last") = log(2/1) = 0.693

IDF("vs") = log(2/1) = 0.693

IDF("intense") = log(2/1) = 0.693

Note that the IDF values for "the", "match", "france", and "brazil" are 0, since they appear in both documents.

Finally, we can calculate the TF-IDF values for each word in each document, which is simply the product of the TF and IDF values:

TF-IDF("the", Document 1) = (2/9) * 0 = 0

TF-IDF("match", Document 1) = (2/9) * 0 = 0

TF-IDF("between", Document 1) = (1/9) * 0.693 = 0.077

TF-IDF("france", Document 1) = (1/9) * 0 = 0

TF-IDF("and", Document 1) = (1/9) * 0.693 = 0.077

TF-IDF("brazil", Document 1) = (1/9) * 0 = 0

TF-IDF("was", Document 1) = (2/9) * 0 = 0

TF-IDF("last", Document 1) = (1/9) * 0.693 = 0.077

TF-IDF("vs", Document 1) = (0/9) * 0.693 = 0

TF-IDF("the", Document 2) = (1/8) * 0 = 0

TF-IDF("match", Document 2) = (1/8) * 0 = 0

TF-IDF("france", Document 2) = (1/8) * 0 = 0

TF-IDF("vs", Document 2) = (1/8) * 0.693 = 0.087

TF-IDF("brazil", Document 2) = (1/8) * 0 = 0

TF-IDF("was", Document 2) = (3/8) * 0 = 0

TF-IDF("intense", Document 2) = (3/8) * 0.693 = 0.260

The vector representation of Document 1 and Document 2 is shown in Figure 2.3.

	Document	the	match	between	france	and	brazil	was	last	vs	intense
1	the match between france and brazil was the last match	0	0	0.077	0	0.077	0	0	0.077	0	0
2	the france vs brazil match was intense intense intense	0	0	0	0	0	0	0	0	0.087	0.26

Figure 2.3: An example of TF-IDF-based text representation

It should be noted that TF-IDF typically uses smoothing to avoid 0 values in the actual implementation. This is represented in the formula:

$$IDF(x) = \log_e \left(\frac{N}{1 + df_x} \right)$$

The advantage of TF-IDF is that it can effectively capture the most important words in a document or corpus, while ignoring common and less informative words. However, there are some limitations to this method:

1. Lack of semantic understanding: One major disadvantage is that it does not capture the semantic meaning of words, so two words with different meanings but similar frequencies may have similar TF-IDF scores. Figure 2.3 displays a few examples of such words.

2. Documents of different lengths can affect the TF-IDF score of terms, which may impact the results of information retrieval.

3. Another significant limitation of TF-IDF is that it cannot handle the Out-of-Vocabulary (OOV) problem effectively. OOV refers to the problem of encountering words in the test set that were not seen in the training set. Since TF-IDF relies on the frequency of terms in the corpus, it cannot assign a weight to new words that are not present in the corpus.

2.4.4 N-Gram

Another method of text representation is N-gram, which views the text as a sequence of contiguous words or characters of a specified length (n) [12]. For example, a bigram representation of the sentence "Lionel Messi is a soccer superstar" would be ["Lionel Messi", "Messi is", "is a", "a soccer", "soccer superstar"]. N-grams can capture the context of the words and are employed in several NLP tasks such as speech recognition, machine translation, and Optical Character Recognition (OCR) [13].
There are several cons associated with using n-gram text representation in NLP:

1. Limited context: N-gram models only capture short-distance context, which means they cannot capture long-range dependencies in text data. This can lead to limitations in capturing the meaning and semantics of text.

2. Large feature space: N-gram models generate a large number of features, which can lead to the curse of dimensionality. This can make it challenging to process and analyze the data.

3. Sparse data: N-gram models can be extremely sparse, meaning that many of the features may have zero values, making it difficult to extract meaningful information from the data.

4. Limited generalization: N-gram models are trained on specific training data, and they may not be able to generalize well to new, unseen data. This can lead to limited performance on tasks that require generalization.

As for the advantage of n-gram, capturing short-distance context can be useful in certain applications where it is possible that local context may be more relevant than global context, such as in some text classification tasks. One example of a text classification task where n-gram models can be useful is sentiment analysis. In such cases, the local context of the text, i.e., the individual words or short phrases used, may be more informative than the global context or longer-range dependencies. For example, a 2-gram model can capture pairs of words that often co-occur in positive or negative reviews, such as "great service" or "terrible experience".

2.5 Word Embedding

The neural probabilistic language model proposed by Bengio et al. [14] has been influential in the development of language models and neural networks for natural language processing. By leveraging the power of neural networks to learn the representation of words, the model can capture more complex relationships between words, and better understand the meaning behind sentences.

The last few years have brought about a paradigm shift in NLP's text representation through the emergence of models like Word2Vec, GloVe, and FastText. These models utilize neural networks to learn the semantic associations between words and represent them as continuous vector spaces, generating word embeddings that capture the semantic similarity between words. As a result, word embeddings have become a widely adopted technique in various NLP tasks, thanks to their ability to accurately capture the meaning of textual data.

Word embedding is the process of translating words into a vector of real numbers within a multi-dimensional space. In essence, word embedding has the ability to position words with similar meanings close to each other within that space.

The resulting word embeddings grasps the syntactic and semantic relationships between words in the corpus, without requiring any explicit labeling of the data.

In the following discussion, we will focus on the three prominent methods of text representation - Word2Vec, GloVe, and FastText and we will analyze their respective advantages and limitations for word embedding generation.

2.5.1 Word2Vec Embedding

Word2Vec [15] is a neural network-based method that learns word embeddings by predicting the context in which words occur. Specifically, Word2Vec uses a training corpus to build a model that maximizes the probability of predicting a word given its surrounding words (CBOW) or maximizing the probability of predicting surrounding words given a center word (skip-gram).

1- Continuous Bag of Words (CBOW)

1. The first step in CBOW is to create a training dataset by generating input-output pairs of words. For example, given the sentence "the match between france and brazil was the last match", the input-output pairs might be as shown in Table 2.1.

   Table 2.1: An example for the input-output pairs of the CBOW method in Word2Vec

Input	Output
"the", "match", "france", "and"	"between"
"match", "between", "and", "brazil"	"France"
"between", "france", "brazil", "was"	"and"
"france", "and", "was", "the"	"brazil"
"and", "brazil", "the", "last"	"was"
"brazil", "was", "last", "match"	"the"

   Each input-output pair consists of a list of context words and the target word. The context words are the words within the specified window size (2 words on either side) of the target word.

2. Creating One-Hot Encoding: The context words are represented as a list of one-hot vectors, where each vector represents a word in the context words and has a value of 1 in the position corresponding to the index of the word in the vocabulary and 0s elsewhere.

3. Next, the input-output pairs are fed into the neural network. The input to the neural network is the sum of the one-hot vectors for the input words, which produces a single vector representation of the context words.

   The neural network is trained to maximize the probability of predicting the correct target word given the input words. The output of the neural network is a probability distribution over the vocabulary for the target word.

After the neural network has been trained, the weights of the hidden layer can be employed as word vector representations. These representations are dense and contain non-zero values for multiple dimensions, unlike one-hot vectors which only have one non-zero value.
Each word in the vocabulary is depicted by a fixed-length dense vector that captures its semantic significance according to its association with other words in the corpus.
These vector representations can be utilized in different tasks related to natural language processing, including named entity recognition, language translation, and sentiment analysis. Figure 2.4 illustrates the CBOW method in Word2Vec.

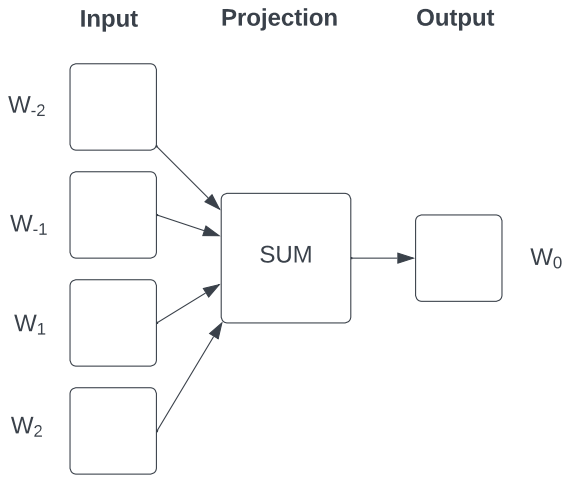

Figure 2.4: CBOW Method in Word2Vec

2- Skip-Gram

The skip-gram method works by training a neural network to predict the context words (i.e., the words that appear within a certain window of the target word) given the target word. Here are the steps involved in the skip-gram method:

1. Create word pairs: For each token in the sentence, we need to create pairs of words within a certain window size. The pairs we would create for the sentence "the match between france and brazil was the last match" using a window size of 2 are shown in Figure 2.5. The word colored in red is the source word and the words colored in blue are its neighboring words (context words).

2. Train the model: Once we have created all the word pairs, we can train the skip-gram model by optimizing a loss function that measures how well the model predicts the context words

Text	Training Samples
the match between france and brazil was the last match	(the, match) (the, between)
the match between france and brazil was the last match	(match, the) (match, between) (match, france)
the match between france and brazil was the last match	(between, the) (between, match) (between, france) (between, and)
the match between france and brazil was the last match	(france, match) (france, between) (france, and) (france, brazil)
the match between france and brazil was the last match	(and, between) (and, france) (and, brazil) (and, was)
the match between france and brazil was the last match	(brazil, france) (brazil, and) (brazil, was) (brazil, the)
the match between france and brazil was the last match	(was, and) (was, brazil) (was, the) (was, last)
the match between france and brazil was the last match	(the, brazil) (the, was) (the, last) (the, match)
the match between france and brazil was the last match	(last, was) (last, the), (last, match)
the match between france and brazil was the last match	(match, the) (match, last)

Figure 2.5: Word Pairs in Skip-Gram

(i.e., the words that appear within the window) given the target word (i.e., the word at the center of the window). In other words, this method tries to maximize the probability of observing the context words given the target word. In this process, it updates the weights of a neural network to capture the patterns in the input data, i.e., the co-occurrence patterns of words. The goal is to learn a set of word embeddings that capture the semantic relationships between words in the corpus.

3. Generate word embeddings: After training the model, we can extract the word embeddings for each token in the sentence. These embeddings are vectors in a high-dimensional space that represent the meaning of each word based on its co-occurrence with other words in the corpus.

The weights are typically stored in a lookup table, also known as an embedding matrix. Each row of the lookup table represents the learned vector representation of a word. The standard Word2Vec pre-trained vectors have 300 dimensions, and each word is represented by a vector of 300 values. These values are learned by a neural network with a hidden layer of 300 neurons. The input vector will have a dimension of $1 \times n$, where n represents the number of words in the vocabulary. The hidden layer will have a dimension of $n \times E$, with E being the size of the word embedding (300 in this case), which is a hyper-parameter. After processing, the hidden layer will output a dimension of $1 \times E$, which will then be passed into a softmax layer. Finally, the output layer will have a dimension of $1 \times n$, with each value in the vector representing the probability score of the target word at that particular position. Figure 2.6 shows the Skip-Gram neural netowork.

It is worth noting that relationships between words can be captured through mathematical oper-

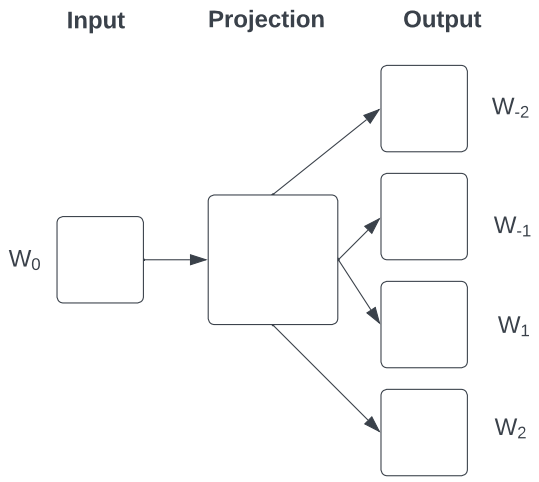

Figure 2.6: Skip-Gram Method in Word2Vec

ations between their embedding vectors. For example, If we subtract the vector for "Klose" from the vector for "Germany" and add the vector for "Iniesta", we get a vector that is close to the vector for "Spain" (e.g., "Germany" - "Klose" + "Iniesta" = "Spain")[1]. In light of these vector relationships, Andres Iniesta is a retired Spanish footballer and Miroslav Klose is a retired German footballer, both of whom are considered legends in their respective countries for their contributions to the soccer.

This operation is often referred to as the "analogical reasoning" task, where we try to find a word that is related to a target word in the same way that another word is related to a reference word. Word2Vec has some limitations and drawbacks:

1. Inability to handle out-of-vocabulary words: word2vec relies on a pre-defined vocabulary, so it cannot generate embeddings for words that are not in its vocabulary. This can be a problem for certain domains or languages with many rare or specialized terms.

2. Conflation of Words with Multiple Meanings: words with multiple meanings may be conflated into a single representation, which can lead to confusion and inaccuracies.

2.5.2 GloVe Embedding

GloVe (Global Vectors) [17] is another popular word embedding technique that is similar to Word2Vec, but it uses a different approach to generate the word embeddings. The main idea

[1]The example utilized the 'word2vec-google-news-300' model from gensim, which was trained on a subset of the Google News dataset. This model encompasses around 3 million words and phrases [16].

behind GloVe is to combine the co-occurrence matrix of words with a matrix factorization technique to generate the word embeddings.

The co-occurrence matrix contains information about how often each word co-occurs with every other word in a given corpus. This matrix can be quite large and sparse, making it difficult to directly use as a basis for generating word embeddings. To address this issue, GloVe applies matrix factorization to decompose the co-occurrence matrix into two smaller matrices, U and V, where each row in U represents the embedding for a given word and each row in V represents the context embeddings (i.e., the words that co-occur with the given word).

The goal of the factorization is to find the optimal values for U and V such that their dot product approximates the co-occurrence matrix:

$$X = U \cdot V^T$$

where X is the co-occurrence matrix, U is a matrix of word embeddings, and V is a matrix of context embeddings.

To achieve this, GloVe defines an objective function that measures the difference between the dot product of U and V and the co-occurrence matrix.

$$J = \sum_{i,j} f(X_{ij}) \left(u_i^T v_j + b_i + b_j - \log(X_{ij}) \right)^2$$

where:

- u_i and v_j are the row vectors corresponding to the i-th and j-th words in the embedding matrices U and V, respectively.
- X_{ij} is the co-occurrence count between the i-th and j-th words in the co-occurrence matrix.
- $f(X_{ij})$ is a weighting function that down-weights the contribution of very frequent co-occurrences to the objective function, in order to mitigate the impact of noise in the data.
- b_i and b_j are bias terms associated with the i-th and j-th words, respectively, which are included to account for differences in the frequency of the words across the corpus.
- $\log(X_{ij})$ is the logarithm of the co-occurrence count, which is included in the objective function to reflect the observation that the strength of the relationship between two words grows

linearly with the logarithm of their co-occurrence count.

This objective function is optimized using a form of gradient descent to iteratively update the values of U and V until convergence.
By decomposing the co-occurrence matrix in this way, GloVe is able to generate dense and meaningful word embeddings that capture the syntactic and semantic relationships between words.
GloVe considers not only the nearby proximity of words within a text window, like Word2Vec, but also incorporates the collective occurrence patterns of words across an entire corpus This helps GloVe capture global semantic relationships between words and can result in better word embeddings. Nevertheless, There are some potential drawbacks to this method:

1. Out-of-Vocabulary (OOV) Words: Like Word2Vec, GloVe can struggle with out-of-vocabulary words, or words that do not appear in the training data. This can be problematic when working with domain-specific jargon or rare words.

2. Issue about capturing subword information: GloVe embeddings do not consider subword information, such as prefixes and suffixes, which can be useful in capturing morphological [2] variations and rare words. Therefore, it may struggle to represent words with such variations in their spellings. To address this limitation, methods like FastText have been developed that incorporate subword information into the embedding learning process.

3. Training of GloVe embeddings can be computationally expensive and time-consuming, especially when using large corpora or high-dimensional embeddings. GloVe can be slower to train than FastText, as it relies on matrix factorization techniques that can be computationally expensive.

2.5.3 FastText Embedding

Word embedding techniques, including Word2Vec and GloVe, integrate semantic and similarity information into their embeddings. However, they face challenges when it comes to handling "Out Of Vocabulary" (OOV) words that were not present in their training data.
Contrarily, FastText [18] doesn't perceive a word as a single, indivisible unit. Instead, it considers it as a collection of its character combinations. Essentially, FastText, an extension of the skipgram word2vec model [15], calculates embeddings for both character and word n-grams. FastText, by

[2]The field of morphology delves into the arrangement and modification of word components such as stems, root words, prefixes, and suffixes, aiming to generate varied meanings.

constructing word vectors from subword vectors, effectively utilizes morphological details and generates word embeddings even for words that were not encountered during the training process.

FastText represents each word, such as "soccer," as a collection of character n-grams. For instance, when considering the word "soccer" and setting the value of n as 3, the corresponding 3-grams would be: <so, soc, occ, cce, cer, er>, and <soccer>[3].

Let's imagine we live in a 2-dimensional embedding space for simplicity and that we've already trained our FastText model. The model assigns the following vector to each of these n-grams:

<so: (0.2, 0.3)

soc: (0.1, 0.2)

occ: (0.3, 0.1)

cce: (0.2, 0.3)

cer: (0.15, 0.25)

er>: (0.25, 0.15)

<soccer>: (0.18, 0.22)

In FastText, the vector for the word "soccer" is calculated by summing the vectors corresponding to its n-grams:

X coordinate: $0.2 + 0.1 + 0.3 + 0.2 + 0.15 + 0.25 + 0.18 = 1.4$

Y coordinate: $0.3 + 0.2 + 0.1 + 0.3 + 0.25 + 0.15 + 0.22 = 1.52$

So, the final vector representation for "soccer" in our simplified, 2-dimensional embedding space is (1.4, 1.52). This vector captures both the word as a whole and its substructures, providing a richer representation than traditional word2vec embeddings.

Training

The goal of predicting the context given a word is essentially a classification problem. Given a word, we'd like to assign probabilities to all other words in our vocabulary such that words that frequently appear in the context of the given word have high probabilities.

The dot product of two vectors is a measure of how similar they are. If the dot product is high, the vectors are pointing in roughly the same direction. Instead of trying to predict the actual probabilities of all words in the vocabulary, the authors try to maximize the dot product (or 'score') of the vectors of words that appear in the same context ($s(w_t, w_c)$), and minimize the dot product of the vectors of words that do not ($s(w_t, n)$).

[3]Special characters < and > are added to distinguish prefixes and suffixes from other character sequences.

Let's define L_w as the collection of n-grams found within w. This model assigns a vector V to all n-grams present in L_w. Assuming the dataset is expressed as a succession of words $w_1, ..., w_k$, the goal is to maximize the likelihood function:

$$\sum_{t=1}^{k}[\sum_{c \in C_p} \ell(score(w_p, w_c)) + \sum_{n \in I_{p,c}} \ell(-score(w_p, n))]$$

Here, C signifies the selected context position, W_c represents the context word, W_p is the word at located at index p, and $N_{p,c}$ constitutes a group of negative instances chosen from the words in vocabulary [4], ℓ is the log loss, and the score is expressed as:

$$score(w, c) = \sum_{g \in L_w} V \cdot c.$$

This objective function is then optimized using a method known as Stochastic Gradient Descent (SGD).

The original formulation of Word2Vec involved predicting the target word given its context (or vice versa) for every word in the vocabulary. This was computationally expensive because it involved performing a softmax operation over the entire vocabulary, which could be tens or hundreds of thousands of words. To make this computation feasible, the authors introduced a technique known as negative sampling. Instead of trying to distinguish the target word from every other word in the vocabulary, negative sampling aims to distinguish the target word from just a few randomly sampled 'negative' words.

In this dissertation, we will use FastText to generate correction candidates.

Word2Vec, GloVe, and FastText Limitations

Word embeddings such as Word2Vec, GloVe, and FastText grasp the syntactic and semantic connections among words within the textual dataset on which they were trained. However, BERT has a significant advantage over fastText, Word2Vec, and GloVe when it comes to context. This is because BERT is a contextualized language model, implying that it produces context-dependent word embeddings. In other words, the meaning of a word in a particular sentence can be different from the meaning of the same word in another sentence.

The traditional word embeddings represent each word as a fixed-length vector that does not change

[4] $score(w_p, n)$ is the score for a pair of the t-th word and a negative sample. The model wants to give these pairs a low score, so it's multiplied by -1 in the loss function.

based on the context in which it appears. In Earlier models like BERT (Bidirectional Encoder Representations from Transformers), the weights of the neural network are stored and used to generate the contextualized word embeddings at runtime, rather than simply looking up precomputed embeddings from a table. Once BERT is trained, it can be used to generate embeddings for words and sentences by feeding them into the network and extracting the hidden representations from one of the intermediate layers. This allows BERT to generate more accurate representations of words based on the context in which they appear in the input text.
To illustrate this, consider the following two sentences:

1. The goalkeeper made an incredible save to keep the ball out of the net.

2. I need to save up some money to buy tickets for the next soccer game.

In the first sentence, "save" refers to a goalkeeper stopping a shot from going into the goal, while in the second sentence, "save" refers to setting aside money for a future purchase. Traditional word embeddings may not be able to distinguish between the two different meanings of "save" in these sentences, but with context-aware language models like BERT, the embeddings would be different because they take into account the specific context in which the word appears. The attention mechanism helps BERT to focus on the specific words in the sentence that are most relevant to the meaning of the word "save" in that context. This allows BERT to generate more accurate and nuanced representations of words based on the specific context in which they appear in the input text. BERT will be discussed in more detail in Section 2.8.

2.6 Neural Language Modeling

A language model is referred to as a model that provides probabilities to a sequence of tokens. In simpler terms, it can estimate the likelihood of tokens appearing in a sequence, considering the tokens already observed in that sequence.
Language models and word embeddings are related in the sense that they are both used to represent language in natural language processing tasks. A language model has the capacity to be trained in order to anticipate the probability distribution of the subsequent word within a sequence, given the preceding words. By utilizing word embeddings as inputs, the model can generate these predictions. Neural language models have been developed over time, with the adoption of neural network models in natural language processing (NLP) starting in 2013 and 2014. The most widely used neural network types in NLP include recurrent neural networks (RNNs) [19], convolutional neural networks

(CNNs) , and recursive neural networks [20].

Recurrent neural networks (RNNs) were first extensively employed in the field of natural language processing because they are particularly well-suited to model sequential data, which is a characteristic feature of language. One of the key features of RNNs that makes them well-suited to NLP is their ability to handle variable-length sequences. This is important in language processing, where sentences can be of different lengths and have complex structures. RNNs can process each element of a sequence one by one and use the information from previous elements to inform their current output. However, one of the major issues with traditional RNNs is that they suffer from the vanishing gradient problem, where the gradients used for updating the model weights become very small as they propagate backward through time, making it difficult for the model to learn long-term dependencies.

Long short-term memory (LSTMs) [21] were developed to address this problem by introducing a "memory cell" that allows the network to selectively remember or forget information over time. This enables LSTMs to learn long-term dependencies more effectively than traditional RNNs.

Convolutional Neural Networks (CNNs) were originally developed for computer vision tasks, but they have also been applied successfully to natural language processing (NLP) tasks in recent years [22]. Unlike RNNs and LSTMs, which process input sequences sequentially, CNNs can process input sequences in parallel, which can lead to faster training and inference times.

While RNNs and CNNs both treat language as a sequence, Recursive Neural Networks process sentences as trees instead. Recursive Neural Networks can recursively combine information from a variable number of inputs to form a hierarchical structure, making them well-suited for tasks that involve parsing and understanding the syntactic structure of sentences. Recursive neural networks provide a way to model the hierarchical structure of language, making them particularly effective for tasks that involve modeling compositionality in language. Compositionality in language refers to the idea that the meaning of a sentence can be derived from the meanings of its individual parts and the way they are combined. In other words, the meaning of the sentence is not simply the sum of its parts, but rather is determined by the way in which those parts are combined.

In 2014, Sequence to sequence learning method was proposed [23]. The technique involves using a neural network to map one sequence (e.g., a sentence in natural language) to another sequence (e.g., a translated version of the sentence). This technique involves two neural networks - an encoder and a decoder. The encoder takes in the input sequence and creates a compressed vector representation of it by processing each element (a unit within the input sequence). This vector is then passed

on to the decoder network which predicts the output sequence element by element based on the encoder state. At each step of this prediction process, the previously predicted symbol is used as an input to the decoder.

2.7 The Attention Mechanism, The Transformer Architecture, And The Large Language Models (LLMs)

One of the main limitations of sequence-to-sequence learning is the necessity to represent the entire input sequence as a single fixed-length vector. This approach can result in the absence of valuable contextual information and can limit the ability of the model to accurately generate output sequences that are dependent on the entire input sequence.

The attention mechanism [24] is a fundamental building block of the Transformer architecture [1], which utilizes this technique to overcome this limitation of sequence-to-sequence learning [23].

Instead of encoding the entire source sequence into a fixed-length vector, the attention mechanism in the transformer architecture calculates a weighted sum of the source sequence components based on their relevance to the current target element being generated.

The transformer architecture [1], is a specific type of neural network architecture that is designed to improve the efficiency and effectiveness of the attention mechanism [24]. Transformer architecture is the underlying framework for Large Language Models (LLMs)[5], the language models that are trained on vast amounts of textual data.

In the Transformer architecture, the neural network is composed of two main components: an encoder and a decoder. The model architecture is shown in Figure 2.7. The encoder processes the input sequence of data, while the decoder generates the output sequence.

The encoder is made up of multiple layers of self-attention and feed-forward neural networks. Each layer takes as input the output from the previous layer and passes it through a set of operations to generate a new output. The self-attention mechanism in each layer allows the encoder to learn contextual relationships between words in the input sequence, enabling it to capture the meaning of the text more accurately. The final output of the encoder is a representation of the input sequence that can be fed into the decoder.

The decoder is also consists of multiple layers of self-attention and feed-forward neural networks. However, it also includes an additional cross-attention mechanism that allows it to focus on specific

[5]By definition, large language models (LLMs) typically refer to Transformer-based language models with hundreds of billions or more parameters, trained on massive text data [25].

parts of the encoded input sequence when generating the output.

During training, the decoder is given access to both the input sequence and the output sequence, and it learns to generate the correct output sequence by attending to the relevant parts of the input sequence.

During inference, the decoder is fed the encoded input sequence and generates the output sequence one token at a time.

The encoder and decoder components work together to form a powerful neural network architecture that can be used for a wide range of NLP tasks. For example, in machine translation, the encoder processes the input sequence of text in the source language, and the decoder generates the output sequence of text in the target language. In text generation, the encoder can be used to generate a representation of a given prompt, and the decoder can be used to generate a coherent response based on that prompt.

2.8 BERT

Google's BERT [26] is a language model that has been pre-trained on BooksCorpus data[6] as well as Wikipedia text, comprising approximately 3.3 billion words. BERT is a versatile model that can be effectively utilized for multiple NLP tasks, including Sentiment Analysis, Name Entity Recognition (NER), and Question Answering.

While the Transformer is a more general sequence-to-sequence model that can be applied to various NLP tasks, BERT is a pre-trained model that is fine-tuned for specific downstream tasks, such as sentiment analysis, question-answering, and text classification.

Furthermore, BERT undergoes training on vast quantities of textual data through unsupervised learning methodologies, whereas the Transformer can be trained on smaller annotated datasets tailored to specific supervised learning objectives. Consequently, BERT is frequently employed as a robust feature extractor for Natural Language Processing (NLP) tasks, while the Transformer serves as a versatile NLP model suitable for diverse task domains.

Furthermore, BERT does not have an explicit decoder. This means that it is primarily used for tasks that involve generating a single output based on the input sequence, rather than generating an entire output sequence. In other words, BERT is more suitable for tasks such as sentiment analysis or question answering, where the model needs to produce a single output based on the input text.

[6]https://yknzhu.wixsite.com/mbweb

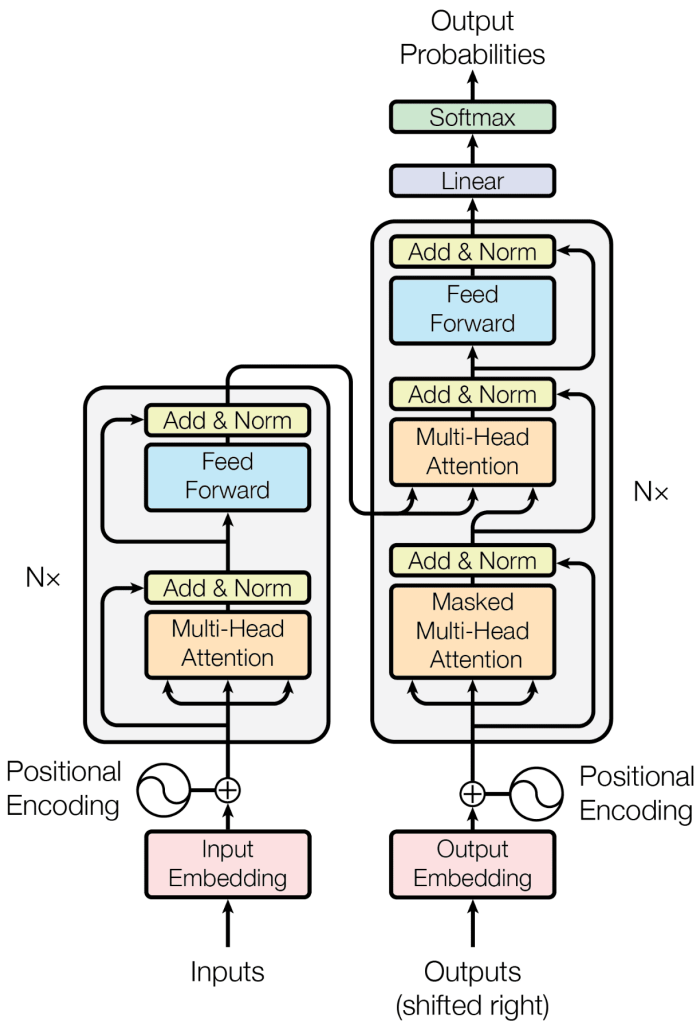

Figure 2.7: The Transformer model architecture. Adapted from [1].

2.8.1 BERT Architecture

BERT utilizes the Transformer architecture [1] and attention mechanism that establishes contextual associations among tokens.

The transformer architecture comprises two key components: an encoder responsible for processing the input text and a decoder that generates predictions for various NLP tasks supported by this architecture [27].

BERT, on the other hand, is a specific variant of the transformer model that exclusively employs the encoder mechanism. This unique characteristic gives rise to its name, Bidirectional "Encoder" Representations from Transformers (BERT). BERT combines both the forward and backward language models, resulting in a "Bidirectional" Encoder Representation from Transformers (BERT).

The architectural diagram of the BERT model is depicted in Figure 2.8.

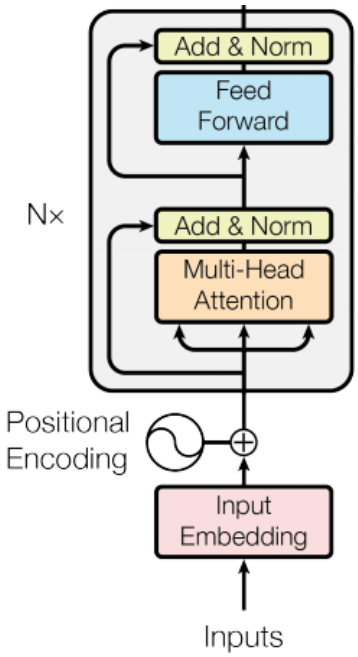

Figure 2.8: BERT model architecture. Adapted from [1].

Positional Encoding

Unlike traditional recurrent neural networks (RNNs) which process the input sequentially, BERT processes the entire input sentence in parallel, allowing it to capture dependencies between words in both directions. However, the transformer-based model does not have any notion of the position of each word in the input sequence.

While an index value could be used to represent position, it becomes impractical for long sequences where the magnitude of the indices grows large. Normalizing the index to a range between 0 and 1 could also cause issues with variable length sequences.

To incorporate positional information, Transformer-based models and BERT adds a fixed-length vector called the "positional encoding" to each word embedding.

The positional encoding vector, which is designed to have unique values for each position in the sequence, allows the transformer to distinguish between different positions. This vector is calculated based on a mathematical formula that generates unique values for each position in the sequence,

such that similar positions have similar vectors:

$$PE_{(k,2i)} = \sin\left(\frac{p}{n^{2i/d}}\right)$$

$$PE_{(k,2i+1)} = \cos\left(\frac{p}{n^{2i/d}}\right)$$

This formula shows that even positions are assigned to sine values, while odd positions are assigned to cosine function. In these equations:
p represents the position of the word in the sequence,
i represents the index used for mapping to column indices in the output matrix,
d represents the total number of embedding dimensions,
and n represents a user-defined scalar, set to 10,000 by the authors of [1].
The vectors are then added to the token embeddings to create the final input representation for the model.

Attention Mechanism

In the self-attention mechanism, each word in a given sequence is compared to every other word in the sequence, including itself. The word embedding of each token would be reweighted to include the relevance of the words to its own meaning in the sentence. This method takes in n word embeddings without context and returns n word embeddings with contextual information.

Dot Product Similarity

The self-attention module in transformer models uses dot product similarity to determine how similar a word's embedding is to the embeddings of other words in the input sequence, including itself. By comparing each word to every other word in the sequence, the module generates alignment scores that reflect the relevance of each word to the others. The higher the alignment score, the more attention a word will have to pay to the other word in the pair. If the semantic meaning of two words is similar, then the alignment score will be high, and the two words will pay a high amount of attention to each other. This process enables the self-attention module to incorporate contextual relevance into word embeddings, allowing transformer models to generate more accurate

and context-aware representations of the input sequence.

The reweighing process in self-attention involves multiplying the weights with the original word embeddings and adding them together to obtain the final contextualized embedding for each word in the sentence. This process is done for every word in the sentence.

Weights are introduced in the dot product similarity comparison, and the final reweighing process. Introducing weights at these locations ensures that the dimensions of the vectors multiplied remain the same. The original embedding vectors are multiplied with the corresponding weight matrices (key, query, and value) in each step of the self-attention process.

Suppose we have a sentence consisting of four words, and we wish to compute the final re-weighted word embedding vector for the 3rd word:

$$Score_{31} = (word_1 \times KeyMatrix) \times (word_3 \times QueryMatrix)$$

$$Score_{32} = (word_2 \times KeyMatrix) \times (word3 \times QueryMatrix)$$

$$Score_{33} = (word_3 \times KeyMatrix) \times (word3 \times QueryMatrix)$$

$$Score_{34} = (word_4 \times KeyMatrix) \times (word3 \times QueryMatrix)$$

$$Score_{31} -> SoftMax -> Weight_{31}$$

$$Score_{32} -> SoftMax -> Weight_{32}$$

$$Score_{33} -> SoftMax -> Weight_{33}$$

$$Score_{34} -> SoftMax -> Weight_{34}$$

$$WeightedWord1 = (word_1 \times ValueMatrix) \times Weight_{31}$$

$$WeightedWord_2 = (word_2 \times ValueMatrix) \times Weight_{32}$$

$$WeightedWord_3 = (word_3 \times ValueMatrix) \times Weight_{33}$$

$$WeightedWord_4 = (word_4 \times ValueMatrix) \times Weight_{34}$$

$$ContextualizedWord_3 = WeightedWord_1 + WeightedWord_2 + WeightedWord_3 + WeightedWord_4$$

To prevent the dot-product from becoming too large and causing numerical instability during training, the attention scores are scaled by the inverse square root of the dimensionality of the key vectors. This scaling factor is often referred to as the "scale" factor and is applied to the dot-product of the query and key vectors before the softmax activation function is applied to obtain the final attention weights.
The formula for the scaled dot-product attention can be written as:

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

In the transformer encoder architecture, there is also a mask block that is used to prevent the model from attending to padded positions in variable-length input sequences. It is a binary mask applied to attention scores. By setting the attention weights to 0 for all padded positions, the model can focus on the relevant words in the input sequence and ignore the padded tokens. This mechanism allows the transformer encoder architecture to handle variable-length inputs efficiently and effectively, which is particularly important in natural language processing tasks where input sequences can vary greatly in length.

Multi-Head Attention

Multi-Headed Attention is a technique used in natural language processing when a single self-attention block is insufficient to capture the contextual relevance of a word with multiple other words, especially in larger input texts. This technique involves using multiple self-attention blocks in parallel, each producing an embedding for a word. These embeddings are concatenated and passed through a dense layer (a type of layer in neural networks that performs a linear operation on the input and applies an activation function), resulting in improved contextual understanding of words.
After the self-attention layer, the next steps in the Transformer Encoder are:

- Add & Norm: The output of the attention will be added to the input embeddings of the same layer and will be normalized to get a new representation of the input sequence. This step helps in capturing the relevant context from the input sequence.

- Feedforward Neural Network: The normalized output obtained from the preceding step is passed through a straightforward feedforward neural network. The feedforward layer represents a fully connected feedforward neural network that operates individually on each position.

- Add & Norm: Finally, the output from the Feedforward Neural Network is added to the output of the previous step and normalized to get the final output representation of the sequence for that particular layer.

These steps are then repeated for each layer in the Transformer Encoder, with the output from the previous layer serving as the input to the next layer. The final output of the Transformer Encoder is a sequence of contextualized embeddings that capture the meaning of each word in the input sequence in the context of the entire sequence. The architecture of the Transformer Encoder, as described above, is shown in Figure 2.9.

It is worth mentioning that BERT's base version consists of 12 Transformer layers, and the large version has 24 Transformer layers. In the base version of BERT, each layer has 12 attention heads, while in the large version, each layer consists of 16 attention heads.

The transformer layers are connected in a sequential manner, with the output of each layer serving as the input to the next layer. The input to the first transformer layer is the tokenized input text sequence, which is first passed through an embedding layer that maps each token to a high-dimensional vector representation. These token embeddings are then transformed by the first transformer layer using multi-headed self-attention and feedforward neural network sub-layers. The result from the initial transformer layer is subsequently utilized as input for the subsequent transformer layer, and this cycle is iterated until the ultimate transformer layer is reached. The outcome from the concluding transformer layer is then employed as input for a specialized output layer dedicated to the specific task needed.

2.8.2 BERT Pre-training

During pre-training, BERT undergoes training on two specific NLP tasks: Masked Language Modeling (MLM) and Next Sentence Prediction. These two pre-training tasks are known as unsupervised learning, as BERT is not given any specific task-related labels or annotations during the pre-training process.

In the Masked Language Modeling (MLM) task, approximately 15% of the words within each sequence are randomly substituted with a [MASK] token. The model is then tasked with predicting the masked words by considering the entire context of the sequence. Through this pre-training process, BERT acquires contextual representations of words in a sentence. By successfully predicting the masked words, BERT develops an understanding of the connections between words and their surrounding context.

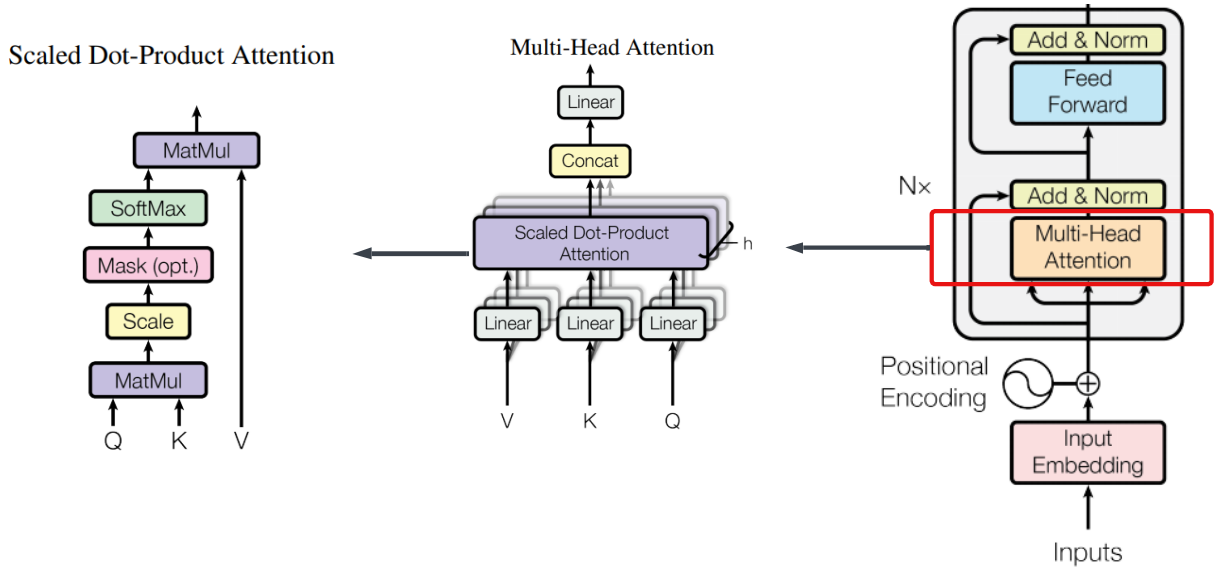

Figure 2.9: Multi-Head architecture. Adapted from [1].

In the Next Sentence Prediction task, BERT is given pairs of sentences and it has to predict whether the second sentence is a continuation of the first or not. This helps BERT understand the relationships between sentences and the overall context of a text.

2.8.3 BERT Tokenizer

In traditional tokenization, words are split into tokens based on spaces or other delimiters, such as commas or periods. For example, the sentence "zidane scored in the final" would be tokenized into 'zidane', 'scored', 'in', 'the', 'final'. However, BERT's tokenization (WordPiece) is a subword tokenization method, where words are broken down into smaller subwords, or tokens. For example, the same sentence will be broken down into 'z', '##ida', '##ne', 'scored', 'in', 'the', 'final'. This allows BERT to handle out-of-vocabulary words, as well as improve its performance on tasks that require understanding of the context of individual subwords.

Note that in BERT's WordPiece tokenization, some of the subword tokens have a "##" prefix attached to them. This is because these subwords are not complete words on their own, but rather are parts of a larger word. The "##" prefix indicates that the subword is a continuation of the previous subword, and should be combined with it to form the complete word.

2.8.4 Converting to base forms

Stemming and lemmatization are employed to reduce words to their base forms, thereby reducing the number of unique words and facilitating subsequent analysis. Stemming is the process of reducing a word to its root form, while lemmatization involves reducing a word to its base form based on the context. For example, the word "running" might be stemmed to "run", while the word "is" might be lemmatized to "be". In other words, while stemming relies on certain rules to obtain the stem of a word, understanding of language and linguistics is required to obtain the lemma of the word.

BERT does not perform any kind of stemming or lemmatization and instead uses the subword tokens generated by its WordPiece tokenizer directly.

2.8.5 BERT Embedding

In BERT, each word is represented by a fixed vector, but the actual vector values can change depending on the context in which the word appears. BERT uses contextual word embeddings, which means that the embedding for a word is determined by the words that appear around it in a particular sentence.

To create these contextual word embeddings, BERT uses a process called pretraining, which involves training a deep neural network on large amounts of text data in an unsupervised manner. During pretraining, BERT learns to predict missing words in a sentence (a task known as masked language modeling) and to distinguish between sentences that occur in a sequence (a task known as next sentence prediction). This training process allows BERT to capture context-dependent representations of words that can be fine-tuned for a wide range of downstream NLP tasks, such as sentiment analysis, question answering, and language translation.

When a new text is fed into a BERT model, each word is first tokenized into a sequence of subwords, which are then passed through the model's network of transformer layers to compute the contextual embeddings. Each layer of the transformer network refines the representations of the words in the input sequence, incorporating information from both the preceding and following words to capture the context of each word. The output of the final layer is a sequence of contextual embeddings, where each embedding captures the meaning of the corresponding word in the context of the input text.

2.9 GPT

GPT (Generative Pre-trained Transformer) [28] is another natural language processing model that is based on the Transformer architecture. In fact, GPT is an extension of the Transformer model that is also pre-trained on a large amount of textual data and is fine-tuned for specific nlp-related tasks.

The main difference between GPT and BERT is that GPT is a decoder-only language model, while BERT is an encoder-only model.

GPT is designed for tasks that involve generating entire sequences of text, such as language translation or text summarization. This is because GPT includes an explicit decoder that is able to generate output sequences based on the input text. OpenAI developed and released newer versions of GPT in 2019, 2020, and 2023, namely GPT-2 [29], GPT-3 [30], and GPT-4 [31] respectively. It is worth noting that ChatGPT is an instance of GPT-4 fine-tuned for the task of conversational dialogue.

It is important to highlight that GPT-4, the most advanced and state-of-the-art Generative AI developed by OpenAI, maintains a proprietary and closed-source status. As a result, information like the underlying code, model architecture, data, and model weights remain inaccessible. This restricted access hinders researchers and developers from replicating the outcomes.

However, despite the proprietary nature of GPT-4, OpenAI has revealed some of the methodologies they use for improving its performance, such as Instruction Tuning (IT) and Reinforcement Learning from Human Feedback (RLHF) [32].

Instruction Tuning (IT): This is an approach that makes the model more responsive to instructions. In the instruction tuning approach, an initial model is fine-tuned on a dataset where every instance is paired with an instruction. It's essentially asking the model to generate a certain type of response. For example, the instruction could be to summarize a text or to translate a sentence into a different language. This method can help make the model more helpful, as it becomes more adept at following the user's instructions.

Reinforcement Learning from Human Feedback (RLHF): This is a process of refining or fine-tuning the GPT-4 model by incorporating human feedback. RLHF involves creating a feedback loop with humans, who review and rank different responses or outputs generated by the GPT-4 model. These rankings are based on the quality of the output, including factors like accuracy, relevance, and clarity. Once the rankings are done, the next step is to fine-tune the model. Fine-tuning is a term used

in machine learning to describe a process where an already trained model (like GPT-4, which was initially trained on a large dataset) is further trained, but on a smaller, more specific dataset. In this case, the specific dataset consists of the model's own outputs and their corresponding rankings by humans. Through fine-tuning, the model learns to generate responses similar to those that were highly ranked by humans. The aim here is to make GPT-4 more effective by helping it understand and generate the kind of responses that humans find most useful. The fine-tuning process doesn't end with one round. It goes through several iterations involving reward modeling. In these iterations, GPT-4 is fine-tuned again to improve its ability to predict how new responses would be ranked by humans. If it makes incorrect predictions, it adjusts its parameters to improve future predictions. The last part of the process involves fine-tuning GPT-4 using a technique called Proximal Policy Optimization [33] which is an algorithm used in reinforcement learning to make the learning process more efficient.

Ultimately, these methods together allow the development of models that are:

- Helpful: By understanding and following instructions better, the models can provide more valuable and relevant responses, thus being more helpful to the user.
- Honest: With the training being grounded in data generated or ranked by humans, the model has a lower likelihood of generating misleading or incorrect information, enhancing its honesty.
- Harmless: The models are trained to avoid harmful or offensive outputs, and to produce outputs that are considered safe by human feedback.

Chapter 3

Literature review

3.1 OCR definition

Optical Character Recognition (OCR) is a tool that enables the conversion of text images into machine-readable text data. When a document, such as a form, legal documents, or a book is scanned, it is saved as an image file, which cannot be edited or searched using text editors. However, OCR technology can transform these images into editable text documents.

3.2 The Significance of OCR

In today's fast-paced business world, managing large volumes of paperwork is no longer a feasible option. Not only it is time-consuming and resource-intensive, but it also requires substantial storage space. Digitizing documents is one of the steps towards paperless document management, but simply scanning documents into images is not enough. This process presents its own set of challenges, as manual intervention is often necessary and can be slow and laborious. To address this issue, OCR technology has emerged as a valuable tool for businesses. OCR technology enables the conversion of text images into machine-readable text data, facilitating the utilization of such data by various software applications. OCR technology has proven to be an effective solution for businesses looking to digitize their documents. The OCR text output allows businesses to extract important information from their documents quickly and accurately. This information can then be used to streamline operations and automate processes, reducing the need for manual intervention and increasing overall efficiency. Furthermore, OCR technology also helps businesses to perform analytics by making it easier to search for specific keywords and phrases within large volumes of documents. By doing so, businesses can gain valuable insights into their operations and make

informed decisions based on the data they collect. Overall, OCR technology has emerged as a vital resource for organizations aiming to enhance their document handling and boost efficiency.

3.3 What advantages does OCR offer?

OCR software is used in a variety of industries, such as healthcare, finance, government, and education, to convert paper-based documents into digital format.

One of the major benefits of OCR technology is searchable text. Once the text is digitized, it can be easily searched using keyword queries, allowing businesses to locate specific information within documents without the need for manual search. For example, a law firm can use OCR software to digitize their client contracts, allowing them to easily search for specific clauses within the contracts.

Another benefit of OCR technology is operational efficiency. OCR software can automate document workflows, which can help reduce the time and cost associated with manual data entry. For example, instead of manually typing in data from paper-based forms, OCR software can extract the data automatically, saving time and reducing errors. This is particularly useful in industries such as healthcare, where large amounts of patient data must be entered accurately and quickly.

OCR technology is also an important component of artificial intelligence solutions. For example, OCR can be used in conjunction with computer vision to read and understand text in images. This can be useful for tasks such as recognizing street signs in self-driving cars or identifying products in advertising images.

In summary, OCR technology provides businesses with a range of benefits, including improved operational efficiency, searchable text, and the ability to leverage artificial intelligence solutions. By digitizing paper-based documents, businesses can make better use of their data, reduce costs, and improve their overall performance.

3.4 What are the applications of OCR?

OCR is a technology that has transformed the way various industries handle their paperwork. In the followings, we will mention the common OCR use cases in various industries.

- Education: Education is one of the biggest adopters of OCR technology. Educational institutions use OCR technology to process student records, transcripts, and other documents. OCR helps to streamline the process of entering data from these documents, reducing the amount of time it takes to process them and improving accuracy. OCR also makes it easier to

search for specific information within these documents, which is especially useful for academic research and data analysis.

- Banking: One of the industries that has embraced OCR technology the most is the banking industry. OCR technology has revolutionized the way banks handle their paperwork, from processing loan applications to verifying financial transactions. By using OCR technology, banks can quickly and accurately process a large volume of paperwork, reducing the time and cost required to complete these tasks.

  One of the key advantages of OCR technology in the banking industry is its ability to improve fraud prevention and enhance transaction security. OCR technology can detect fraudulent documents by verifying the authenticity of signatures, dates, and other information on loan documents and deposit checks. This verification process greatly reduces the risk of fraudulent activity, ensuring that transactions are secure and trustworthy.

  OCR technology has also helped to improve the efficiency and accuracy of data entry in the banking industry. With OCR technology, bank employees can easily scan documents and extract data automatically, reducing the need for manual data entry. This not only saves time but also reduces the likelihood of errors, ensuring that customer data is accurate and up-to-date.

  Moreover, OCR technology has also enabled banks to provide better customer service. With faster processing times and increased accuracy, banks can quickly approve loan applications and process financial transactions, improving customer satisfaction. Additionally, OCR technology has made it easier for customers to submit their paperwork, whether it is through online channels or mobile applications. This has reduced the need for customers to physically visit a bank, improving convenience and accessibility.

- Healthcare: The healthcare sector stands out as one of the industries that relies heavily on information, and OCR technology has been instrumental in streamlining workflows, reducing manual labor, and improving accuracy and efficiency.

  The healthcare industry extensively utilizes OCR technology to process and analyze various patient records, such as hospital records, tests, treatments, and insurance payments. The technology has revolutionized the way hospitals manage their paperwork, allowing for faster and more accurate processing of medical records. With OCR technology, hospital staff can easily scan documents and extract data automatically, reducing the need for manual data

entry. This not only saves time but also reduces the likelihood of errors, ensuring that patient data is accurate and up-to-date.

OCR technology has also enabled healthcare providers to keep track of patient records more efficiently. By automating the process of entering data from medical records, OCR technology has reduced the time required to maintain patient records, enabling healthcare providers to focus more on patient care. This has also led to a reduction in errors, improving the overall quality of healthcare services.

Furthermore, OCR technology has made it easier for healthcare providers to manage medical billing. By using OCR to extract data from medical bills, healthcare providers can quickly and accurately process payments, reducing the time and cost required to manage medical billing. This has led to improved efficiency and accuracy, as well as reduced costs for healthcare providers

- Legal: In the legal industry, OCR is used to process and extract data from legal documents such as contracts, court documents, and case files. OCR helps to automate the process of entering data from these documents, reducing the time and cost required to manage them.

  One of the key advantages of OCR technology in the legal industry is its ability to improve accuracy. Legal documents are often lengthy and complex, and manually processing them can lead to errors. With OCR technology, legal departments can quickly and accurately extract data from these documents, reducing the likelihood of errors.

  OCR technology also makes it easier to search for specific information within legal documents, which is especially useful for e-discovery and litigation support. By using OCR, legal departments can quickly and easily search for relevant information within large volumes of legal documents, saving time and increasing efficiency. This has improved the speed and accuracy of legal research and discovery, making it easier for legal departments to prepare for trials and other legal proceedings.

  OCR technology has also made it easier for law firms to manage their contracts. With OCR, law firms can quickly and accurately extract data from contracts, reducing the amount of manual work required to manage them. This has improved contract management, making it easier to track contract renewals, deadlines, and other important details.

One of the earliest large application of the OCR is the implementation of the Licensing Support

Network (LSN) by Nuclear Regulatory Commission (NRC) of the United States [1]. The LSN converted over one hundred million pages of documents to its equivalent ASCII data set. The challenges associated with the construction of the LSN and the use of OCR'ed text for search and retrieval were discussed in [34] and [35].

To sum up, OCR technology has revolutionized the way numerous industries manage their paperwork, streamlining workflow and enhancing efficiency while minimizing manual labor. As technology continues to progress, it is expected that OCR will become even more widespread in other industries, leading to further improvements in workflow and efficiency.

3.5 Steps Involved in OCR

Optical Character Recognition software uses a set of steps to convert scanned documents into text that can be edited and searched on a computer [2]. In the followings, we will explore the different steps involved in the OCR process and delve into each of these steps in detail:

3.5.1 Image Capture

The first step in OCR is image capture, which involves capturing the document using a scanner or a camera. The quality of the image captured is crucial as it can affect the accuracy of the OCR engine. The scanner reads the document and converts it into a binary format, which means that each pixel is either black or white. The Optical Character Recognition (OCR) software examines the digitized image and distinguishes the brighter regions as the background while identifying the darker areas as the textual content. This operation is commonly referred to as the binarization process.

3.5.2 Pre-processing

During this phase, the software employs various cleaning techniques to enhance the quality of the scanned image. One of the primary techniques used is deskewing, which adjusts the scanned document's alignment to fix any alignment issues that occurred during scanning. This helps to ensure that the OCR software accurately identifies and recognizes the text.

Another common technique used during preprocessing is despeckling. This method involves the removal of digital image spots or the smoothing of text image edges to eliminate any unwanted

[1] https://www.nrc.gov/reading-rm/lsn/index.html

noise or distortion that could hinder the OCR software's recognition process.

Additionally, the software tidies up any boxes and lines present within the image that may disrupt the OCR process. Boxes and lines can appear in scanned documents in the form of tables, charts, graphs, and borders, to name a few examples. While these elements may enhance the readability of the document, they can also interfere with the OCR process. OCR software can detect these boxes and lines and remove them or redraw them to ensure that the text is extracted accurately. For instance, when an OCR engine encounters an image with a table, it may not recognize the text in the table cells as part of the document's text, and instead recognize it as an image. By removing the table borders and lines, OCR software ensures that the OCR engine can recognize the text accurately. Similarly, OCR software may also identify and remove lines and boxes that are not part of the text, such as header and footer lines, or page borders. This ensures that the OCR software can focus on recognizing the text and improving the accuracy of the OCR results.

Furthermore, some OCR software incorporates script recognition to facilitate multi-language OCR technology, enabling the software to identify and recognize text in different languages.

Script recognition is a technology that allows the software to identify and recognize different scripts or writing systems, such as Latin, Arabic, Cyrillic, Chinese, or Japanese. This feature helps the OCR software to accurately convert scanned documents or images containing different languages and fonts into editable and searchable text. By employing these advanced preprocessing techniques, OCR software can effectively convert scanned documents or images into editable and searchable text with a high degree of accuracy.

3.5.3 Text detection

When it comes to recognizing text, OCR software uses different algorithms to recognize text from an image. A glyph is a graphical representation of a character, symbol, or letter in a particular font or typeface. Pattern matching is one of the text detection methods that works by comparing a glyph with a previously stored glyph that has a similar font. The OCR application extracts an individual graphical symbol from the image and proceeds to compare it with the stored symbol. In the event of a resemblance between the two symbols, the OCR software acknowledges the symbol and transforms it into text that can be edited. For pattern recognition to work, the stored glyph must have a similar font and scale to the input glyph. This means that it may not work well with handwritten text or with text that has been typed in a different font or size. Therefore, pattern matching is best suited for recognizing printed text in a specific font.

In contrast, feature extraction involves analyzing or decomposing the glyphs into distinctive elements, such as line patterns, enclosed shapes, directional attributes, and points of intersection. These extracted features are then utilized to find the most similar or compatible counterparts among the stored glyphs. Feature extraction proves particularly effective in recognizing text presented in various fonts or sizes, as well as handwritten text. Essentially, feature extraction enables the OCR software to recognize text based on its inherent characteristics, rather than relying solely on the font and size properties. However, this approach requires a more advanced algorithm and entails higher computational complexity compared to pattern matching.

OCR software typically uses a combination of pattern matching and feature extraction algorithms to achieve the best possible text recognition results. By combining the strengths of both methods, OCR software can recognize a wide variety of text in different fonts and sizes, as well as handwritten text.

3.5.4 Post-processing

OCR post-processing is the process of refining and improving the results of OCR software after the initial recognition process. While OCR software is designed to recognize text accurately, there are still many factors that can lead to errors in the recognized text. These errors can be caused by various factors such as low image quality, distorted characters, and varying font types and sizes.

Detecting and correcting errors is indeed a critical part of the pre-processing stage in OCR. In fact, the accuracy of an OCR system depends heavily on how effectively it can detect and correct errors in the input image.

Therefore, in this dissertation, we primarily focus on the techniques and algorithms that can be used to detect and correct errors in OCR systems. Overall, the goal of this dissertation is to provide insights into the state-of-the-art techniques for error detection and correction in OCR, and to help improve the accuracy and reliability of OCR systems for various applications.

3.6 Evolution of OCR Post-Processing Methods

Over the years, OCR systems have progressed significantly, with post-processing methods serving a crucial function in improving the accuracy of OCR results [36]. In the paragraphs below, we will delve deeper into the various post-processing methods that have played a crucial role in the evolution of OCR systems.

3.6.1 Word unigram based methods

A word unigram language model is a statistical model that assigns probabilities to individual words in a language, based on their frequency of occurrence in a corpus of text. These approaches typically involve comparing the OCR output against a dictionary of valid words, and correcting any words that are not recognized by the dictionary. These methods compare the OCR output with words in the lexicon and use various distance measures, like Damerau-Levenshtein edit distance or ngram distance, to find similar words that could be the correct ones.

Schulz et al [37] introduced enhancements to the Levenshtein edit distance algorithm. These enhancements involve restricting the sets of merges, splits, and substitutions as the operations for editing. This modification enables the algorithm to efficiently select small candidate sets from the dictionary while maintaining a high recall rate. In a study conducted by Mihov et al.[38], Levenshtein automata, which are deterministic finite state machines, are utilized to propose potential correction candidates. Furrer et al.[39] present a method that leverages comparable character substitutions and searches for shared 3-grams to identify suitable correction candidates. The research by Taghva and Agarwal [40] explores an alternative approach using Google's indexed data instead of constructing a traditional dictionary. OCRed tokens are utilized as search queries for the Google search engine, which suggests potential replacement candidates for any detected errors in the query. Cappelatti et al. [41] utilize the PyEnchant spell checker to propose corrections, while a modified version of the Needleman-Wunsch algorithm is applied to rank these suggestions.

3.6.2 Error correction techniques

Error correction models take into account the probabilities of certain character transformations, insertions, and deletions that might occur during the OCR process. The aim is to correct errors and improve the overall quality of the OCRed text.

These methods often utilize the Levenshtein distance as a measure to determine the number of edits (insertion, deletion, transposition) or changes needed to correct an error. Additionally, these models also associate probabilities with single and multiple-character transformations, insertions, and deletions to determine the most likely correction for an error.

OCRSpell [42] is a technique developed by Taghva and Stofsky, which employs several edit operations on characters to enable interactive post-processing. This method first identifies the largest possible word boundary and then checks each tokenized word for accuracy using a lexicon-lookup.

For erroneous tokens, replaceable candidates are generated based on a confusion matrix and its updated variant, which is enriched by user-generated corrections. To score suggested candidates, OCRSpell employs a Bayesian function that considers collocation frequencies and character ngrams and the method provides users with a ranked list of suggested replacements in the end.

3.6.3 Statistical language models

Statistical language modeling refers to the process of approximating the likelihood distribution of word sequences. These models are created by analyzing the frequency of word occurrences and they mainly use word n-grams.

In the study conducted by Cacho et al.[13], OCRSpell[42] was employed to identify errors generated by OCR. Once identified, the neighboring words of the erroneous word are utilized to find matching n-grams in Google Web-1T [43]. These words serve as initial correction candidates, which are further refined by considering the Levenshtein edit distance [44] and the occurrence rate of the 3-gram in Google Web 1T. This context-based approach provides the advantage of selecting the most likely candidate, similar to how a proofreader would choose the most appropriate word for error correction based on the surrounding words or context.

3.6.4 Machine learning based methods

OCR post-processing using machine learning-based techniques involves training models to learn from different features to enhance candidate selection. These models generate potential candidates, extract their features, and rank them to improve the accuracy and robustness of the OCR process.

In a study by Mei et al. [45] conducted in 2016, they presented a method capable of rectifying 61.5% of OCR errors. Their approach involves utilizing the Damerau-Levenshtein distance measure to generate a list of candidates. The method creates several features, such as string similarity, language popularity, and lexicon existence, to evaluate the candidates and rank them based on their likelihood of being the correct correction for the OCR error. Moreover, they frame the problem as a regression task. By utilizing the scores of potential features, they make estimations on the likelihood that each candidate serves as a correction for the erroneous word. This confidence score is then used to rank candidates for each error, with the method selecting the candidate with the highest confidence score as the correction for the error.

In [46], authors proposed a method that incorporates OCRSpell [42] for detecting errors and gen-

erating their replacements, followed by SVM classification to select the best alternative using five features generated for each candidate (unigram frequency, confusion weight, levenshtein edit distance, and bigram frequencies of predecessor and successor).

3.6.5 Neural network-based language models

Neural network-based language models are another type of approach used to correct errors in OCR-generated text. Neural network-based language models are capable of learning to connect each word with a set of continuous-valued features, referred to as word embeddings (discussed in section 2.5). They are usually trained as probabilistic classifiers that forecast the likelihood of the next word, given the context.

In a 2016 study [47], a method was proposed for addressing OCR error detection and correction specifically in French clinical foetopathology reports, employing character-level LSTM. The researchers pointed out the inadequacy of traditional OCR error correction systems when dealing with medical text corpora. This is due to constraints such as the obligation to maintain patient confidentiality, inconsistency in the quality of OCR output, and the lack of domain-specific external resources. The uniqueness of the proposed method is that it relies solely on a sample of clean, domain-specific text, requiring no additional external information.

3.6.6 Transformer-based language models

These language models such as BERT (discussed in section 2.8) have shown great success in different tasks related to natural language processing such as machine translation, sentiment analysis, and text summarization. In OCR post-processing, transformer-based models can be used to detect and correct errors generated by the OCR system.

In the context of BERT (Bidirectional Encoder Representations from Transformers) utilization, Nguyen et al. [48] conducted a study focusing on error detection and correction using various models and techniques. For error detection, they employed the BERT model and adapted a Named Entity Recognition (NER) model for this purpose. Instead of NER tagging, they assigned labels of 1 (invalid token) or 0 (valid token) to the tokens. Simplifying the model architecture, they introduced a fully connected layer on top of the hidden states output, replacing the more complex combination of convolutional and fully connected layers found in state-of-the-art methods. Additionally, they incorporated popular pre-trained word embeddings like as Glove and Fasttext, leveraging them instead of randomly initialized embeddings. These word embeddings were used to embed the sub-

tokens, which were then fed into BERT's token classification model for labeling the sub-tokens as valid or invalid.

In terms of error correction, Nguyen et al. employed a character-level Neural Machine Translation (NMT) model built on the OpenNMT toolkit. This NMT model was utilized to translate OCRed text into its corrected version, providing an automated approach for error correction.

Chapter 4

Methodology

In this chapter, we will present our methodology for addressing the challenge of detecting and correcting OCR errors. Our methodology consists of three main components: detecting OCR errors, generating correction candidates for identified errors, and selecting the best candidate for each error. In the following sections, we will elaborate on the use of BERT and FastText to detect OCR errors and generate correction candidates, as well as leveraging Levenshtein distance to choose the most suitable candidate for error correction. This approach aims to improve the accuracy and readability of OCR-generated text.

4.1 Detecting the OCR Errors

PROBLEM DEFINITION

Identifying errors in Optical Character Recognition (OCR) can be perceived as a procedure of categorizing OCR-produced tokens as either flawed or exact words. The essential function of spotting errors within OCR is often overlooked, as it's not frequently examined independently from the automatic rectification of errors.

APPROACH

Identification of errors in Optical Character Recognition (OCR) can be accomplished using the BERT language model (referenced in 2.8). To identify these mistakes, we require a metric to scrutinize the words in a phrase and ascertain their correctness. In this portion, we present a straightforward approach employing Masked-Language Modeling with BERT, enabling us to evaluate the significance of words within a phrase.

Quantifying the meaningfulness of a word in a sentence:

In our proposed method, we define S_i, which is a score to measure how removing the i^{th} word would help in making the sentence more meaningful. BERT can provide us with a score vector (L_i) of size V (Vocabulary Size) for each of the words we mask in a sentence in position i. We compute the probability of the word in position j of the sentence by dividing the score of the word in L_i vector into sum of the scores of all the words in vocabulary in position i. Consequently, to calculate S_i we need to compute the probability of all the words in all the positions, pick the word which has the highest score in L_i vector, and divide it by sum of all the words in L_i. Finally, we calculate the logarithm of the sum of probabilities to obtain S_i.
For instance, suppose we have the following sentence:

This is the best movvie I have ever seen.

And we want to calculate how removing the word **"movvie"** in position 4 would help in making the sentence more meaningful. For this purpose, we need to mask that word and feed it to the BERT language model:

This is the best [MASK] I have ever seen.

In order to calculate S_4 (the index starts with 0), we have to compute the probability the word may appear in position j, for all the words in the sentence. For example, to calculate the probability of the word "best" for position 3, we have:

$$P_3['best'] = \frac{L_3[best]}{\sum_{w \in V} L_3[w]}$$

Also for the masked word we would have:

$$P_4 = \frac{Max[L_4]}{\sum_{w \in V} L_4[w]}$$

And ultimately we can calculate S_4:

$$S_4 = \log(P_0[This]) + \log(P_1[is]) + ... + \log(P_4) + ... + \log(P_8[seen])$$

Finding the wrong word:

In order to identify whether a word is an OCR error, we need to measure how removing the word would affect the degree of meaningfulness of the sentence. Toward this end, we replace all the words with the [MASK] token one by one, and calculate S. If the S score of a word compared to other words in the sentence is significantly higher, the probability that the word is an OCR error and is contextually inconsistent with others is higher as well. Figure 4.1 illustrates this process. After calculating the scores, we would have a list of numbers representing scores associated with words. The next step would be finding outliers, which are the words with the highest scores that are significantly larger than the others. To achieve this, we utilized the Isolation Forest algorithm [49] for outlier detection in a list of numbers. The technique employed by this approach involves segregating data points by randomly choosing a dividing threshold within the range of values in the dataset. The count of these divisions needed to isolate a specific sample corresponds to the length of the path from the starting point to the endpoint in the tree structure. The Isolation Forest algorithm tends to generate reduced traversal depths for anomalies. Thus, when a forest of random trees collectively yields lower traversal depths for specific samples, those samples are highly likely to be outliers. Figure 4.1 provides a detailed overview of our approach.

4.2 Generating the Candidates for OCR Errors

PROBLEM DEFINITION

In this section, our primary emphasis is on the generation of correction candidates for OCR errors. We introduce an approach that leverages pre-trained BERT Masked Language Modeling and FastText subword embedding to generate these correction candidates.

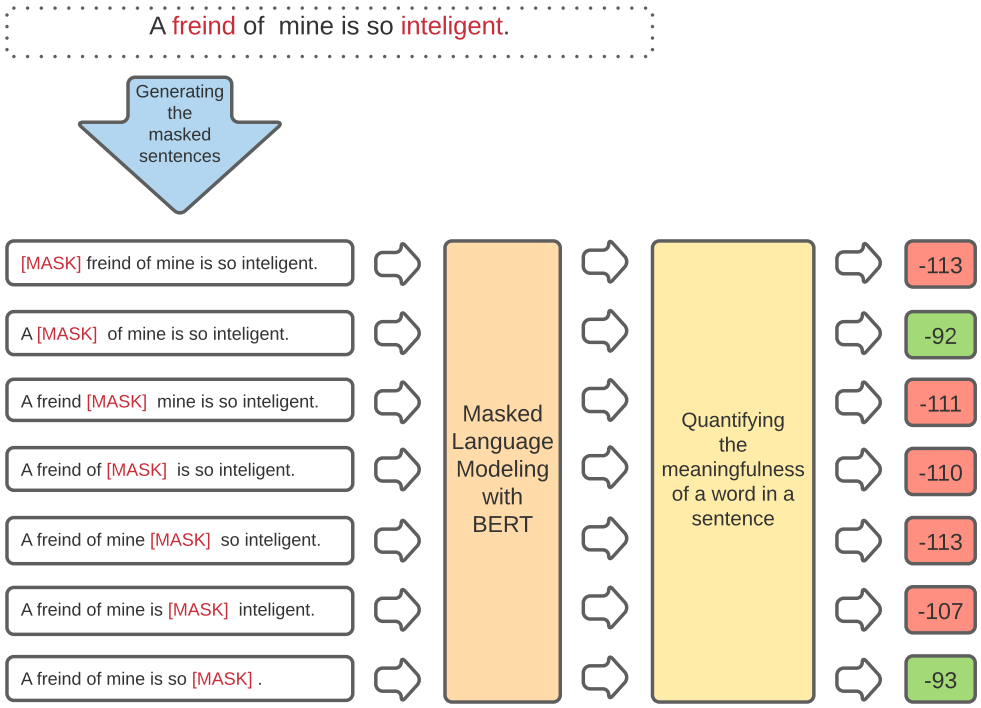

Figure 4.1: Detecting OCR errors using Masked Language Modeling in BERT and computing the score of sentences

APPROACH

In this section, our suggested methodology, as previously published in [50], predominantly comprises two key components. In the first part, we employ BERT (discussed in 2.8) language model to produce potential candidates for correction . During the subsequent part, FastText [18] is trained on our dataset and employ it to generate an additional collection of suitable candidates for OCR errors. Ultimately, we merge the outcomes from both components to obtain the final list of candidates.

It is important to mention that training BERT on a small data is not rational, as BERT typically requires large amounts of data to effectively learn contextual representations.

It is worth noting that he BERT model has already undergone pre-training using a vast amount of textual data. It has been trained on the BooksCorpus dataset, which consists of 800 million words, as well as the Wikipedia dataset, which contains 2,500 million words. In total, the BERT model has been exposed to approximately 3,300 million words during its pre-training phase.

However, FastText is better suited for training on smaller datasets, making it a more practical

choice in such situations. Given that we planned to use a dataset consisting of text generated by OCR software applied to a book on birds, we aim to train FastText on several books with a similar context, which is Ornithology. We chose this dataset in order to compare our result with other already published works in OCR post processing techniques.
By utilizing FastText's word embeddings, we are able to take advantage of the context and relationships between words in the books related to birds it was trained on.
On the other hand, when using a pre-trained BERT model, we obtain word embeddings based on the surrounding words in the input text.
As a result, the combination of the pre-trained BERT model, which helps in understanding the context of the input text, and the FastText model, which is adapted to the domain-specific language patterns in bird-related books, provides a comprehensive solution for correcting OCR errors.

BERT:

As previously mentioned in the preliminaries section, it is important to note that BERT undergoes pre-training on a specific task called Masked Language Modeling (MLM). Within our suggested approach, we utilize this capability to generate potential candidates for OCR errors. To illustrate, consider the scenario where the following sentence is the output generated by an OCR engine from a given image:

The nest of the Golden Oriole is nsually, though not invariably, suspended between the forking branches of an oak, frequently at a considerable height from the ground, and at the end of a somewhat slender bough.

The word 'nsually' represents an OCR-generated error, where its correct form should be usually'.
To input the sentence into BERT, erroneous work will be replaced with the [MASK] token:
Based on the provided sentence, here is the compilation of candidates that can be generated by the BERT Language Model for the token we masked:

'usually', 'casually', 'annually', 'unusually', 'equally', 'visually', 'normally', 'ideally', ...

The nest of the Golden Oriole is [MASK], though not invariably, suspended between the forking branches of an oak, frequently at a considerable height from the ground, and at the end of a somewhat slender bough.

We utilize this collection of candidates as a list of potential corrections, where we aim to locate the correct word. As observed, the correct word, 'usually', is the first suggestion in the list provided by BERT.

FastText:

Morphological structure refers to the way in which words are composed of smaller units called morphemes. For example, the word "unhappiness" can be broken down into three morphemes: "un-" (a prefix that means "not"), "happy" (a free morpheme meaning "feeling or expressing joy"), and "-ness" (a suffix that turns an adjective into a noun indicating a state or quality). FastText creates vector representations for words by considering sub-word units. This allows FastText to capture the morphological structure of words, which can be useful for handling OCR errors. To generate potential corrections for OCR errors, we utilized the FastText embedding technique.
The first step is to train the model. The dataset we are working with is the text output obtained by applying OCR software on an image of a book, which focuses on the topic of birds. Section 5.1 contains additional information regarding the dataset. To train the FastText model effectively and capture the domain-specific jargon, we utilized text files from several books with a similar context, specifically focusing on the subject of birds. The books used for training include [51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66]. The dataset comprised of 16 books, approximately 1.5 million tokens, and about 75,000 sentences.
Subsequently, we employed cosine similarity between word embedding vectors to detect words that exhibited similarity to the error. This enabled us to generate a comprehensive list of potential correction candidates. We can calculate the cosine similarity of two vectors such as $a = \{a_1, a_2, \ldots, a_n\}$

and $v = \{b_1, b_2, \ldots, b_n\}$ using these formula:

$$similarity(a, b) = \frac{a \cdot b}{|a||b|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\left(\sum_{i=1}^{n} a_i^2\right)\left(\sum_{i=1}^{n} b_i^2\right)}}$$

Consider a sentence structured as follows:

> Dr. Gadow observes in his Catalogue that "The Laniida:, as described in this volume, form neither a group complete in itself, nor are the lines of distinction always drawn closely enough."

The word 'Laniida:' is an error in this sentence. Below is a list of suggestions for the misspelled word 'Laniida:' that FastText could generate:

> 'laniidae', 'land', 'laridae', 'laid', 'planina', 'laid', 'lanai', 'linota', 'inside', 'liliesa', 'dinia', ...

The correct word is "laniidae" which is latin and the scientific name assigned to a particular family of birds.

4.3 Choosing the Best Candidate

PROBLEM DEFINITION

After detecting OCR errors and creating a list of candidates for each error, it is crucial to find the best candidate among them and replace the error with the most accurate match, ultimately correcting the OCR error.

APPROACH

The Levenshtein distance is a measure that indicates how different two strings are. The higher the number, the more different the two strings are. In other words, it is a metric to determine the dissimilarity between two strings.
For instance, the Levenshtein distance between “manhattan” and “nnanbattan” is 3 since at least 3 edits (insertion, deletion or substitution) are required to change one into the other:

nnanbattan → nanbattan (Deletion of ”n”)
nanbattan → manbattan (substitution of “m” for “n”)
manbattan → manhattan (substitution of “h” for “b”).

By using the Levenshtein distance, we can determine the best candidate by selecting the one with the smallest distance to the OCR error.

Chapter 5

Results

In this chapter of the dissertation, we focus on the results of our research on OCR Post-Processing using different evaluation metrics. First, we introduce the dataset used for our investigation, which consists of OCR text and the ground truth text, and describe its main features. Next, we discuss the evaluation process, which is split into two key parts: Detection Evaluation and Correction Evaluation. Detection Evaluation assesses the effectiveness of our proposed method in identifying OCR errors present in the dataset, while Correction Evaluation examines the accuracy of generating candidates for OCR errors and correcting the errors. By exploring these different aspects, we aim to provide a clear understanding of the performance of our OCR error post-processing approach.

5.1 Dataset

In our research, we employed the MiBio dataset[1] [67], which comprises aligned OCR tokens produced by the Tesseract[2] OCR engine, along with the corresponding ground truth recognition texts from the book titled "Birds of Great Britain and Ireland, Vol. 2" [68].

The MiBio dataset is designed to serve as a benchmark for evaluating the performance of OCR post-processing models. It is derived from an English biodiversity book and is generated using the Tesseract 3.0.2 OCR engine. With 2,907 OCR-generated erroneous words, the dataset provides researchers with corrected OCR texts, or ground truth, to compare their models against. It should be noted that by OCR errors in this dataset, we mean word errors as opposed to character errors. It is known that an erroneous word contains 2.5 character errors on the average. The characteristics of this dataset are displayed in table 5.1.

[1] https://github.com/jie-mei/MiBio-OCR-dataset
[2] https://github.com/tesseract-ocr/tesseract

Table 5.1: Features of the MiBio dataset

Feature	**Description**
Dataset Name	MiBio
Purpose	Benchmark dataset for OCR post-processing evaluation
Source	English biodiversity book
OCR Engine	Tesseract 3.0.2 (github.com/tesseract-ocr/tesseract)
Errors	2,907 OCR-generated errors
Ground Truth	Corrected OCR texts
File Formats	TXT amd TSV
Access	GitHub (github.com/ie-mei/MiBio-OCR-dataset)

In the following sections, we will present the outcomes of our approach in two distinct phases of OCR post-processing. First, we will showcase the results for OCR error detection, and subsequently, we will demonstrate the effectiveness of our method in correcting these errors.

5.2 Detection Evaluation

This section is dedicated to assessing the effectiveness of our method in identifying OCR errors within the dataset we introduced earlier. The evaluation results are presented in the form of confusion matrices.
OCR error detection can be thought of as a binary classification problem, where you classify each character as either "correct" or "incorrect". In this context, we can use the following terms:

True Positives (TP): The number of correctly detected errors
False Positives (FP): The number of incorrectly detected errors
True Negatives (TN): The number of correctly detected correct tokens
False Negatives (FN): The number of incorrectly detected correct tokens

These terms are the fundamental components used in calculating the evaluation metrics Precision, Recall, and F1-score.

Precision

Precision measures the proportion of true positive predictions (correctly identified errors) out of all positive predictions (both true positives and false positives). A higher precision means fewer false

positives.

$$Precision = \frac{TP}{TP + FP}$$

Recall

Recall measures the proportion of true positive predictions (correctly identified errors) out of all actual positive instances (both true positives and false negatives). A higher recall means fewer false negatives.

$$Recall = \frac{TP}{TP + FN}$$

F1-score

The F1-score, a metric that combines precision and recall through the harmonic mean, offers a balanced evaluation of both measures. This metric proves particularly valuable in scenarios involving imbalanced datasets or when false positives and false negatives hold equal significance.

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

It is important to highlight that in OCR post-processing, error detection is generally evaluated based on its **recall**-oriented performance. This approach places a strong emphasis on identifying all possible errors within the OCRed text, ensuring that the accuracy and reliability of the converted digital content is maximized.

5.2.1 BERT

We employed the methodology outlined in section 4.1 to detect ocr error. Table 5.3 presents the outcomes obtained from the application of our proposed method.

Table 5.2: Confusion Matrix for BERT model

	Actual	
Predicted	True (Error)	False (Correct)
True (Error)	TP:2466	FP:1374
False (Correct)	FN:391	TN:97472

$$Precision = \frac{TP}{TP + FP} = 64.21\%$$

$$Recall = \frac{TP}{TP + FN} = 86.31\%$$

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 73.64\%$$

5.2.2 GPT-4

This section will cover the outcomes of feeding our dataset into a state-of-the-art language model, GPT-4, to assess its ability to detect OCR errors using its API [3]. The goal is to detect error tokens using GPT-4 to determine the accuracy of the detections.

In this approach, an essential aspect is the prompt engineering, which ensures that the GPT-4 API receives clear instructions to provide the desired output. The prompt is designed to guide the model in detecting the errors. By crafting a prompt that addresses potential pitfalls and provides explicit guidance on the desired output, the model is more likely to detect OCR errors in the dataset. The prompt we provided to the model is as follows:

> You will be provided with a sentence delimited by triple @ signs. Please find OCR errors in this sentence among all tokens given (delimited by triple ! signs and tokens separated by @ sign) and return the list of OCR errors as a python list:
> @@@{sentence}@@@
> !!!{token1@token2@...}!!!
> Attention: Only provide the list of errors as a python list, without any accompanying explanations or additional context. If the sentence has no errors return an empty python list like [].

The results of our GPT-4 evaluation are presented in Table 5.3.

[3] https://openai.com/product/gpt-4

Table 5.3: Confusion Matrix for GPT-4 model

Predicted	Actual	
	True (Error)	False (Correct)
True (Error)	TP:2266	FP:309
False (Correct)	FN:591	TN:98537

$$Precision = \frac{TP}{TP + FP} = 88\%$$

$$Recall = \frac{TP}{TP + FN} = 79.31\%$$

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 83.43\%$$

5.3 Correction Evaluation

In this section, we will present the results of our two distinct OCR correction approaches. The first approach combines the candidates generated by BERT and FastText models, whereas the second method uses the GPT-4 model.

5.3.1 BERT and FastText

In this section, we applied the techniques discussed in sections 4.2 and 4.3, which involve the pre-trained BERT model and the trained FastText model, to correct OCR errors in our dataset. The effectiveness of this combined approach can be observed through the accuracy metric presented in Table 5.4. It should be highlighted that in the baseline paper [45], an accuracy of 61.5% was

Table 5.4: BERT- FastText Model Accuracy

Model	**BERT-FastText**
Task	OCR Error Correction
Accuracy	68.56%

achieved in correcting the errors within the same dataset. The improvement in accuracy of our

approach is statistically significant.

5.3.2 GPT-4

This approach aims to correct Optical Character Recognition (OCR) errors in a given dataset using the GPT-4 API. The goal is to generate corrected tokens using GPT-4 and compare them with the ground truth values to determine the accuracy of the corrections. The process begins by reading the dataset and extracting sentences containing errors. The GPT-4 API is then called iteratively with instructions to provide the corrected word. The GPT-4 corrected token is then compared to the ground truth and if they match, the correction is considered successful.
Similar to what we discussed in section 5.2.2, the prompt we provided to the model is as follows:

Please correct the specified error in this sentence and provide only the corrected term in lowercase: {Sentence} The error to correct is {Error Token}. Attention: Provide only the corrected word, without any accompanying explanations or additional context.

Table 5.5: GPT-4 Model Accuracy

Model	**GPT-4**
Task	OCR Error Correction
Accuracy	75.33%

During each iteration of the API call, we substitute the {Sentence} and {Error Token} placeholders with corresponding values from the dataset. The results of the GPT-4 model for OCR error correction is presented in Table 5.5. This table shows the model's performance, demonstrating an accuracy of 75.33% in correcting OCR-generated errors. This notable accuracy highlights how well the GPT-4 works in fixing OCR errors and demonstrates its ability to improve text post-processing tasks.

Chapter 6

Conclusion and Future Work

In this study, we presented a comprehensive methodology for detecting and correcting OCR errors using BERT, FastText, and GPT-4 models. Our approach involved three main components: detecting OCR errors, generating correction candidates for identified errors, and selecting the best candidate for each error. By leveraging the power of BERT in detecting OCR errors and generating correction candidates, as well as employing FastText to capture the morphological structure of words, our method demonstrated considerable success in improving the accuracy of OCR error correction. Furthermore, we utilized the Levenshtein distance to choose the most suitable candidate for error correction, effectively enhancing the overall performance of our OCR error post-processing approach.

Future work may focus on further refining the methodology by incorporating Named Entity Recognition (NER) techniques for aiding OCR error detection and correction. Named entities, such as names of people, organizations, locations, and dates, play a critical role in understanding the context of the text. By integrating NER into the existing framework, the model can be enhanced to recognize these entities, improving the overall efficiency of OCR error correction.

NER could be employed to identify named entities and help determine the likelihood of the detected errors. By considering the context and the entities involved, NER could provide additional information for the model to make more accurate corrections. For example, if the model detects a potential OCR error within a person's name, it could use NER to verify whether the detected sequence is, in fact, a name, and then suggest appropriate corrections based on common names or the context of the document.

Moreover, integrating NER into the OCR error correction pipeline can also help refine the candidate generation and selection process. By leveraging NER to identify possible entity types associated

with detected errors, the model can narrow down the list of correction candidates to those that are more contextually relevant. This can lead to a more efficient and accurate correction process, as the model will be more likely to select a suitable replacement from a smaller pool of candidates.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[2] H. Bunke and P. Wang, *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997. [Online]. Available: https://books.google.com/books?id=yn6DN5hAPywC

[3] R. Smith, "An overview of the tesseract ocr engine," in *Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR)*, 2007, pp. 629–633.

[4] K. Taghva, J. Borsack, A. Condit, and S. Erva, "The effects of noisy data on text retrieval," *Journal of the American Society for Information Science*, vol. 45, no. 1, pp. 50–58, 1994.

[5] D. van Strien, K. Beelen, M. C. Ardanuy, K. Hosseini, B. McGillivray, and G. Colavizza, "Assessing the impact of ocr quality on downstream nlp tasks." in *ICAART (1)*, 2020, pp. 484–496.

[6] K. Taghva, J. Borsack, and A. Condit, "Information retrieval and ocr," in *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997, pp. 755–777.

[7] R. B. Lees, "Syntactic structures." JSTOR, 1957.

[8] C. F. Hockett, "Language, mathematics and linguistics, current trends in linguistics." Mouton, The Hague, 1966, p. 155–304.

[9] Q. Ye, Z. Zhang, and R. Law, "Sentiment classification of online reviews to travel destinations by supervised machine learning approaches," *Expert systems with applications*, vol. 36, no. 3, pp. 6527–6535, 2009.

[10] O. Delalleau and Y. Bengio, "Shallow vs. deep sum-product networks," *Advances in neural information processing systems*, vol. 24, 2011.

[11] R. F. Correa and T. B. Ludermir, "Web documents categorization using neural networks," in *Neural Information Processing: 11th International Conference, ICONIP 2004, Calcutta, India, November 22-25, 2004. Proceedings 11*. Springer, 2004, pp. 758–762.

[12] A. Esmaeilzadeh, J. F. Cacho, K. Taghva, M. Kambar, and M. Hajiali, "Building wikipedia n-grams with apache spark," in *Intelligent Computing*. Springer, 2022.

[13] J. R. F. Cacho, K. Taghva, and D. Alvarez, "Using the google web 1t 5-gram corpus for ocr error correction," in *16th International Conference on Information Technology-New Generations (ITNG 2019)*. Springer, 2019, pp. 505–511.

[14] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Advances in neural information processing systems*, vol. 13, 2000.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[16] R. Řehůřek, "Word2vec model," *Gensim*, 2022. [Online]. Available: https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html

[17] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[18] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[19] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[20] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] Y. Chen, "Convolutional neural network for sentence classification," Master's thesis, University of Waterloo, 2015.

[23] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[24] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[25] M. Shanahan, "Talking about large language models," *arXiv preprint arXiv:2212.03551*, 2022.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[27] T. Wolf, J. Chaumond, L. Debut, V. Sanh, C. Delangue, A. Moi, P. Cistac, M. Funtowicz, J. Davison, S. Shleifer *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 38–45.

[28] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[29] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[30] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[31] OpenAI, "GPT-4 technical report," Mar. 2023.

[32] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving, "Fine-tuning language models from human preferences," *arXiv preprint arXiv:1909.08593*, 2019.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[34] K. Taghva, J. Borsack, and A. Condit, "Evaluation of model-based retrieval effectiveness with ocr text," *ACM Trans. Inf. Syst.*, vol. 14, no. 1, p. 64–93, jan 1996. [Online]. Available: https://doi.org/10.1145/214174.214180

[35] K. Taghva, J. Borsack, S. Lumos, and A. Condit, "A comparison of automatic and manual zoning an information retrieval prospective," *International Journal on Document Analysis and Recognition*, vol. 6, no. 4, pp. 230–235, 2003.

[36] T. T. H. Nguyen, A. Jatowt, M. Coustaty, and A. Doucet, "Survey of post-ocr processing approaches," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–37, 2021.

[37] K. Schulz, S. Mihov, and P. Mitankin, "Fast selection of small and precise candidate sets from dictionaries for text correction tasks," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 1. IEEE, 2007, pp. 471–475.

[38] S. Mihov, S. Koeva, C. Ringlstetter, K. U. Schulz, and C. Strohmaier, "Precise and efficient text correction using levenshtein automata, dynamic web dictionaries and optimized correction models," in *Proceedings of Workshop on International Proofing Tools and Language Technologies*, 2004.

[39] L. Furrer and M. Volk, "Reducing ocr errors in gothic-script documents," 2011.

[40] K. Taghva and S. Agarwal, "Utilizing web data in identification and correction of ocr errors," in *Document Recognition and Retrieval XXI*, vol. 9021. International Society for Optics and Photonics, 2014, p. 902109.

[41] E. Cappelatti, R. D. O. Heidrich, R. Oliveira, C. Monticelli, R. Rodrigues, R. Goulart, and E. Velho, "Post-correction of ocr errors using pyenchant spelling suggestions selected through a modified needleman–wunsch algorithm," in *International Conference on Human-Computer Interaction*. Springer, 2018, pp. 3–10.

[42] K. Taghva and E. Stofsky, "Ocrspell: an interactive spelling correction system for ocr errors in text," *International Journal on Document Analysis and Recognition*, vol. 3, no. 3, pp. 125–137, 2001.

[43] S. Evert, "Google web 1t 5-grams made easy (but not for the computer)," in *Proceedings of the NAACL HLT 2010 Sixth Web as Corpus Workshop*. Association for Computational Linguistics, 2010, pp. 32–40.

[44] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[45] J. Mei, A. Islam, Y. Wu, A. Moh'd, and E. E. Milios, "Statistical learning for ocr text correction," *arXiv preprint arXiv:1611.06950*, 2016.

[46] J. R. Fonseca Cacho and K. Taghva, "Ocr post processing using support vector machines," in *Science and Information Conference*. Springer, 2020, pp. 694–713.

[47] E. D'hondt, C. Grouin, and B. Grau, "Low-resource ocr error detection and correction in french clinical texts," in *Proceedings of the Seventh International Workshop on Health Text Mining and Information Analysis*, 2016, pp. 61–68.

[48] T. T. H. Nguyen, A. Jatowt, N.-V. Nguyen, M. Coustaty, and A. Doucet, "Neural machine translation with bert for post-ocr error detection and correction," in *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*, 2020, pp. 333–336.

[49] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 eighth ieee international conference on data mining*. IEEE, 2008, pp. 413–422.

[50] M. Hajiali, J. R. Fonseca Cacho, and K. Taghva, "Generating correction candidates for ocr errors using bert language model and fasttext subword embeddings," in *Intelligent Computing*. Springer, 2022, pp. 1045–1053.

[51] A. R. Wallace, *Island Life: Or the Phenomena and Causes of Insular Faunas and Floras*, 2010. [Online]. Available: https://www.gutenberg.org/files/32021/32021.txt

[52] H. E. Walter and A. H. Walter, *Wild Birds in City Parks: Being hints on identifying 145 birds, prepared primarily for the spring migration in Lincoln Park, Chicago*, 2010. [Online]. Available: https://www.gutenberg.org/cache/epub/33125/pg33125.txt

[53] P. L. Sclater and W. H. Hudson, *Argentine Ornithology, Volume I (of 2): A descriptive catalogue of the birds of the Argentine Republic*, 2012. [Online]. Available: https://www.gutenberg.org/files/38956/38956-0.txt

[54] J. A. Leach, *An Australian Bird Book: A Pocket Book for Field Use*, 2010. [Online]. Available: https://www.gutenberg.org/cache/epub/34781/pg34781.txt

[55] Various, *Birds and Nature, Vol. 10 No. 1 [June 1901]*, W. K. Higley, Ed., 2015. [Online]. Available: https://www.gutenberg.org/files/49969/49969-0.txt

[56] W. B. Stallcup, *Myology and Serology of the Avian Family Fringillidae: A Taxonomic Study*, 2010. [Online]. Available: https://www.gutenberg.org/cache/epub/33914/pg33914.txt

[57] M. A. Jenkinson, *Thoracic and Coracoid Arteries In Two Families of Birds, Columbidae and Hirundinidae*, 2010. [Online]. Available: https://www.gutenberg.org/cache/epub/33558/pg33558.txt

[58] W. W. Dunmire, *Birds of the National Parks in Hawaii*, 2019. [Online]. Available: https://www.gutenberg.org/files/59398/59398-0.txt

[59] W. Swainson, *Zoological Illustrations, or Original Figures and Descriptions. Volume I, Second Series*, 2013. [Online]. Available: https://www.gutenberg.org/cache/epub/44056/pg44056.txt

[60] J. L. Bonhote, *Birds of Britain*, 2018. [Online]. Available: https://www.gutenberg.org/files/56397/56397-0.txt

[61] H. E. Howard, *Territory in Bird Life*, 2010. [Online]. Available: https://www.gutenberg.org/files/31987/31987.txt

[62] W. W. Fowler, *A Year with the Birds: Third Edition, Enlarged*, 2015. [Online]. Available: https://www.gutenberg.org/files/48677/48677-0.txt

[63] J. M. Bechstein, *The Natural History of Cage Birds: Their Management, Habits, Food, Diseases, Treatment, Breeding, and the Methods of Catching Them*, 2012. [Online]. Available: https://www.gutenberg.org/cache/epub/40055/pg40055.txt

[64] Various, *The Auk: A Quarterly Journal of Ornithology, Vol. XXXVI APRIL, 1919 No. 2*, W. Stone, Ed., 2019. [Online]. Available: https://www.gutenberg.org/files/59190/59190-0.txt

[65] C. Whymper, *Egyptian Birds: For the most part seen in the Nile Valley.* Project Gutenberg, 2014. [Online]. Available: https://www.gutenberg.org/cache/epub/46825/pg46825.txt

[66] A. C. Bent, *Life Histories of North American Shore Birds, Part 1 (of 2).* Project Gutenberg, 2014. [Online]. Available: https://www.gutenberg.org/cache/epub/47028/pg47028.txt

[67] J. Mei, A. Islam, A. Moh'd, Y. Wu, and E. E. Milios, "Mibio: A dataset for ocr post-processing evaluation," *Data in brief*, vol. 21, pp. 251–255, 2018.

[68] A. G. Butler, F. W. Frohawk, and H. Gronvold, *Birds of Great Britain and Ireland.* Brumby & Clarke, 1907, vol. 2. [Online]. Available: https://www.biodiversitylibrary.org/item/35947#page/13/mode/1up

Curriculum Vitae

Graduate College

University of Nevada, Las Vegas

Mahdi Hajiali

Contact Information:

Email: Mahdihajiali1990@gmail.com

Degrees:

Doctor of Philosophy in Computer Science 2023

University of Nevada Las Vegas

Dissertation Title: OCR Post-processing Using Large Language Models

Dissertation Examination Committee:

Chairperson, Dr. Kazem Taghva, Ph.D.

Committee Member, Dr. Wolfgang Bein, Ph.D.

Committee Member, Dr. Laxmi Gewali, Ph.D.

Graduate College Representative, Dr. Ashok Singh, Ph.D.