UNIVERSITY OF PISA

DEPARTMENT OF COMPUTER SCIENCE

Master Degree in Data Science & Business Informatics

Neural Machine Translation Training Strategies for Effective Named Entity Processing in Low-Resource Domain

Supervisor:

Prof: Alessandro Bondielli

Candidate:

Davide Innocenti

ACADEMIC YEAR 2022/2023

Contents

List of Figures

List of Abbreviations

AI Artificial Intelligence

ANN Artificial Neural Network

ASR Automatic Speech Recognition

BDLSTM Bi-Directional Long Short-Term Moemory

BOW Bag of Words

BPTT Backpropagation Through Time

CBOW Continuous Bag Of Words

DNT Do Not Tranlsate

ELMO Enough, Lets Move On

FFNN Feed Foward Neural Network

GPU Grapic Processor Unit

HMM Hidden Markov Model

IT Information Tecnology

LLM Large Language Model

LRL Resource-limited Language

LSTM Long Short-Term Memory

MLP Recursive Neural Network

MTL Multi-Task Learning

MT Machine Translation

NER Named-Entity-Recognition

NLM Neural Language Model

NLP Natural Language Processing

NMT Neural Machine Translation

OOV Out-of-Vocabulary

PCA Principal Component Analysis

POS Part of Speech

RBMT Rule Based Machine Translation

ReLU Rectified Linear Unit

Seq2Seq Sequence-to-Sequence

SMT Statistical Machine Translation

SOTA State-Of-The-Art

SOV Subject-Object-Verb

SVD Singular Value Decomposition

SVO Subject-Verb-Object

TF-IDF Term Frequency - Inverse Document Frequency

TL Transfer Learning

VOS Verb-Object-Subject

A tutti coloro che hanno creduto in me,
alla pazienza dei miei genitori,
alla memoria di mia nonna,
alla mia compagna a me sempre vicina,
a tutti gli amici con i quali ho passato momenti fantastici,
un forte ringraziamento.

Abstract

Natural Language Processing (NLP) and Machine Translation (MT) stand at the forefront of technological innovation, bridging the gap between academia and industry. They have transformed the paradigm of human-machine and inter-human communication. The growth and influence of these sectors in academic and commercial spheres are undeniable. This research addresses a specific requirement: crafting an MT system tailored for low-resource languages. Central to this is devising an end-to-end methodology for detecting and managing terms that shouldn't be translated, namely "Do Not Translate" terms (DNTs). Leveraging Named Entity Recognition, the objective is to streamline entity manipulation within the Neural Translation domain. The study collects data from online sources to build Machine Translation models capable to capture specific pre-made tags for DNTs, while the aim is for these models to properly process selected entities during the inference phases, yielding coherent translations. State-of-the-art tools have been exploited to achieve the goal, and new training strategies have been presented to develop these models, reporting results by means of generic Machine Learning evaluative metrics and those specific for MT quality assestment. The findings can serve as guidance for future applications in low-resource domains of NLP and MT for both companies and academics when dealing with entities manipulation.

Chapter 1

Introduction

Neural Language Models (NLMs) based on deep learning architectures have brought about a major change in the domain of Natural Language Processing (NLP) and consequently to the world of research and development, solving taks such as Machine Translation, Speech Recognition and so on. By NLMs we mean all those models developed on top of the classical Neural Networks structures, on which almost all state-of-the-art models are usually based nowadays.

These models capture, and consequently generate, latent representations of the text, creating embeddings that contain a range of linguistic and semantic information that, combined with the model's ability to understand certain statistical features of the training data, including word distribution, actually enable it to make probabilistic predictions in the generation of a word, given other previous words.

This thesis comes to life in the domain of Neural Machine Translation (NMT), in this case as an advancement in machine translation solutions. As described above, it deals with the exploitation of particularly advanced deep learning architectures in order to respond to the need of translate texts, documents and corpora of any nature or size in real time without any intervention on the part of a human translator. Nowadays, the strong use of neural networks is due to the progressive improvement of computer tools such as the Grapic Processor Unit (GPU), which achieve tens of times higher performance in relation to the computational power existing at the dawn of machine translation and let us exploit strong models for our purposes.

The project stems from a collaboration with Pervoice, a company belonging to the wider Almawave and Almaviva group, which operates in the Automatic Speech Recognition (ASR) and Automatic Translation fields, creating efficient transcriptions and speech-synthesis technologies for conversational agents in the first case, and accurate translation models in the second, and which interfaces with customers in the Public Administration, Health and other private sectors.

1.1 Motivation

The work presented here responds to a specific need put forward by an institution from Italian Ministry of Intern which cannot be named here for privacy reasons, whose IT department requires a system to translate legal-domain document for foreigners from a varirty of countries including non-European ones, and involves the creation of a process for identifying and managing terms that must not be translated (do not translate terms, DNTs). DNTs include a wide class of terms ranging from proper names to alphanumeric codes, but also any part of the text which we intend to leave intact during translation from one language to another. It might be the case one requires a system not to translate specific parts fo the text, usually names and places which may be erroneously not recognized as such and then translated with some unwanted outcome. This phenomena is already quite common when translating from and to English, which is the richest language in terms of data resources availability, and it is surely amplified for low-resources languages for which the creation of a functional Machine Translation system is even harder to achieve. The customer's current systems dinamically generates a variety of legal documents, always different one from another and containing blank sections to be filled by both sides; these slots refer to personal data, including names, dates, locations, but also free textual informations. These sections represents textual entities that must not be translated, to provide pairs of documents with original filled text. Our final goal is to investigate a novel approach to treat such DNTs and then create different translation engines which, for each given language, these are able to output given tags to let us map the original entities back on the translated text.

1.2 Work Overview

The current literature does not really offer any particular strategy to address this specific problem. Instead, it has mostly focused on handling Machine Translation systems that unintentionally were unable to translate certain words due to structural and lexical deficiencies [70] by realising solutions restictedly related to the correct function of the architectures themself such as Byte-Pair-Encoding [23], WordPiece [77] or other techniques that obviate the problem of dealing with Out-Of-vocabulary (OOV) words, without directly investigating the problem of not translating given words in a specific text, leaving us with several paths to follow.

For this work we are going to use a specific DNT tag to replace entities inside our datasets; these are based on a prefix "${DNT0}", a single token in our Machine Tranlsation framework (to be discussed later), and a suffix "x" where x stands for a variable number used for enumeration purpose, such that, once the translation is output, we are able to reconnect in post-processing the given enumerated tags to the original entities. Thus, for each kind of document of intereset for the customer we are given a HTML file containing a specific tag indicating entities not to be translated and to be treated accordingly. A pre-processing step will so transform the given entities coming from a source-language file to those ${DNT0}$x$ tags, subsequently translated in the target language, then mapped back to the original text in a final post-processing step. Pre and post-processing steps are out of scope on this thesis

and will be not treated here, while the creation of a model is the main goal of the project this thesis is founded on.

This project is mainly based on the creation of a data augmentation pipeline for training data, followed by experimentations on training and fine-tuning engines for translation purposes. By collecting parallel corpora as input data for our models, we are going to replace a priori certain entities via Named-Entity-Recognition (NER) and by means of entity alignment between the pairs of languages we are going to replace these aligments with enumerated DNTs tokens. Afterwards, such corpora are supplied to the model in the training phase allowing the system to generate untranslated tokens during the inference phase, to be eventually reconnected to the starting entities.

Due to the nature of the documents provided by the customer, the experimentation will therefore be carried out not only in the generic linguistic domain but also in the legal domain with particular emphasis on resource-limited language pairs (LRLs). By this definition, we mean languages for which there is a lack of useful training attributes such as supervised data, number of native speakers or experts, etc., making the development of suitable solutions for technologies that intend to exploit such languages inaccessible. The number of spoken languages of this kind is incredibly high; suffice it to consider that almost all research studies focus on only 20 of the more than 7,000 languages that exist today and that, moreover, do not include a different percentage of dialects that would increase this number [50].

During the internship, the language pairs used to train our models given by the company to investigate the problem and develop a solution have been Italian-Turkish and Italian-Urdu, with a purpose of scalability to other languages.

1.3 Thesis Outline

This thesis focuses on developing an end-to-end pipeline to handle specific entities for the customer to assist the IT department in translating documents in low-resource languages domain, to be futherly applied for other research. The paper depicts the whole process, starting from data collection to model creation and experimentation. It is divided in 5 chapters summarized as follows:

- **Chapter 2:** A deep background overview is provided to the reader, starting from oldest methods for NLP and Machine Translation starting from the last century, until modern and SOTA models daily applied for aforementioned tasks nowadays.
- **Chapter 3:** A comprehensive overview of approaches, tools and methodologies to answer the business request is given to the reader, reporting the whole data manipulation process, the creation of the Machine Translation model as well as its development.
- **Chapter 4:** Here, the implementation results are given and explained reporting all the scores obtained by means of different domain-specific metrics for both

DNTs recognition and translation quality.

- **Chapter 5:** Finally, a last chapter presents the conclusions drawn from this work, summarizing the findings of this case-study and discussing potential improvements to be applied for better performances.

Chapter 2

Background

In this chapter, theoretical aspects are introduced to provide a suitable context for the experiments on which this thesis is based. An excursus is then presented, ranging from an introduction to Natural Language Processing (NLP), Machine Translation (MT), and the related challenges arising from its interaction with scientific approaches useful for structuring an empirical basis, and then we discuss the very first models proposed in the field of Machine Translation, historically quite distant to us, up to the State-Of-The-Art (SOTA) models currently in circulation, such Transformers and models suitable for processing sequences (textual in our case) with particular emphasis on applications in the domain of Machine Translation and the metrics currently circulating for evaluating the output of these architectures.

2.1 NLP Overview: Challenges and Goals

"Natural Language Processing is the field of designing methods and algorithms that take as input or produce as output unstructured, natural language data" [28]. Natural Language Processing (NLP) is a tool that enables machines to understand human language and is adopted to provide solution to text-related tasks. It is a sub-field of Artificial Intelligence (AI) and it combines the power of linguistics and computer science to study the rules and structures of language, intelligent systems (based on machine learning and NLP algorithms) that can understand text, analyze and process speech, and extract meaning from it.

2.1.1 Natural Language Processing

While computational linguistics is more focused on aspects of language, natural language processing emphasizes its use of machine learning and deep learning techniques to complete tasks, like language translation or question answering and it works by taking unstructured data and converting it into a structured data format to gain value on. The fundamental goal of Natural Language Processing (NLP) is to bridge the gap between human language and computational understanding. Achieving this goal requires addressing several significant challenges inherent to language itself. Computational Natural language processing has its roots in the 1950s when Alan Turing proposed the well-known Turing Test [90] in which there's an exten-

sive analysis on the assumption that "*machines were capable of behavior which must be accounted as intelligent*" [25]. Although at the time that was articulated as a problem separate from artificial intelligence, the proposed test includes a task that involves the automated interpretation and generation of natural language. Current systems instead rely on transforming texts to a computer-readable format to feed those with input data and generate an output related to a precise use case. Using better computation capabilities in the last decades, NLP has gained a lot of popularity due to the wide range of applications directly connected to this field. Just to mention a few:

- **Speech recognition:** This task refers to the process of transforming a given sound clip of a person or people speaking to a textual representation, determining the discrete characters composing the clip itself. This process is currently one of the most challenging due to several factors such as the same languages being spoken by different people with eventually different accents and thus the system must process a wide variety of input data.
- **Part-of-speech tagging:** Given a sentence, determine for it the part of speech (POS) for each word. For instance, the English words, *book*, *set*, and *out* can simultaneously refer to a noun or a verb. This is a quite challenging sub-task as well, mostly concerning low-resource languages [61]
- **Named Entity Recognition:** The task of determining which elements in the texts map to proper names, such as people or places, and what the type of each such name is (e.g. person, location, organization).
- **Sentiment Analysis:** Extract subjective information to determine the polarity of a given text about specific subjects, mostly exploited to identify trends of public opinion, analyze social media, or for marketing purposes.

A more exhaustive dive into NLP tasks can be found in [40]. Human language, as a complex and multifaceted medium of communication, presents a range of intricate challenges when it comes to processing and understanding it through machines, thus NLP approaches must face this complexity to produce valuable systems, setting this up as its main goal.

Human Language complexity

As much as human beings have innate abilities that enable them to produce, process, and analyze text, whether written or spoken, it remains a challenge beyond their reach to extract patterns and highlight intrinsic rules of language modeling itself. Human language, in all its existing declinations, is incredibly vast and articulate, incorporating within it numerous layers of information that go beyond the simple communicative atom composed of the word. It encodes articulate syntactic structures, intricate semantic features, and context-dependent pragmatics. Each of these elements contributes to the richness and depth of language itself, allowing human beings to channel abstract ideas, emotions, and experiences into the instrument of communication. However, all these elements introduce a certain level of complexity

that poses a challenge to machines, which are still struggling to decode and emulate language in depth, and thus do not yet provide a high level of understanding of it.

These elements of complexity are manifold and can be summarised as (i) Ambiguity, (ii) Context-Dependency (iii) Cultural Nuances (iiii) Dynamic Nature (iiiii) Lack of Explicit Rules:

- **Ambiguity:** Language is rife with ambiguity, where each word or phrase can have different meanings depending on the context in which it occurs. The basic element of every language is a character; the concatenation of them creates words, which denote a particular object, event, idea, etc. Both characters and words are discrete symbols: words evoke in us a certain mental representation, but they are also distinct symbols, whose meaning is external to them. There is no inherent information about the words that can be inferred from the symbols themselves, or from the individual letters they are made of. Resolving this ambiguity requires understanding the broader context and disambiguating based on surrounding words or phrases.

- **Context-Dependency:** Letters make up words, and words make up sentences. The meaning of a word given in a sentence is not always the same; it can vary with the context surrounding it. Machine translation systems must be able to take the context into account, which can be especially challenging in longer texts or conversations.

- **Dynamic Nature:** Language tends to evolve, changing its characteristics and incorporating new words, expressions, and meanings. Furthermore, language use is profoundly influenced by cultural norms, idiomatic expressions, and social contexts, and machines must navigate these cultural nuances to avoid misinterpretation or insensitivity.

- **Lack of Explicit Rules:** Last but not least, unlike programming languages with strict syntax and semantics, human language often lacks rigid rules. Understanding the subtle variations in how language is used requires models that can generalize from data.

In several contexts, such as Machine Translation one, these challenges take on a particularly pronounced relevance because translations do not have to take into account individual words but constantly keep an eye on the surrounding context to capture the intended meaning, cultural connotations, and subtle contextual cues of which corpora and language resources are made of.

2.1.2 Machine Translation

Machine Translation is concerned with studying how to develop statistical and computational techniques to translate from one language to another and has been considered to be one of the most challenging tasks in the field of natural language processing (NLP). The MT concept was first proposed by Warren Weaver in 1947 [93], just one year after the development of the first computer, the electronic digital integrator, and the calculator. Since then, MT has been regarded as one of the most difficult tasks in the field of natural language processing.

In terms of methodology, Machine Translation approaches mainly fall into two categories: rule-based methods and statistical methods. From the time the MT idea was first proposed until the 1980s, rule-based methods prevailed. Rule-based machine translation (RBMT) uses bilingual dictionaries and handwritten rules to translate source language text into target-language text. However, writing the rules manually takes a lot of work. Furthermore, the rules are difficult to maintain and difficult to transfer from one domain to another and from one language to another. Therefore, it is difficult for rule-based systems to scale for open-domain translation and multilingual translation. Statistical approaches (STM) outclassed the former due to the utilization of mathematical and statistical tools to provide more accurate and reliable solutions. Finally Machine Learning, following a growth trend in computational power, was a breakthrough for setting a new threshold of performance. While the utilization of Machine Translation may appear obvious, its main goal is to develop stronger solutions regarding divergences among different languages and to overcome bad-structured and low-resource inputs.

Languages Divergences

As stated in Chapter 1, there are a variety of languages currently populating the world and challening MT capabilties. Some aspects of these seem to be universal, meaning that despite cultural, geographical, and social aspects rules hold for the vast majority of them. Every language for example, seem to have words or symbols referring to people, or to common aspect of human life such as eating, drinking and social relationships. Aside rules, other structures seem to be respected such as nouns or verbs as way to interact with others. Yet, there are a lot of obvious differences that Machine Translation has to deal with.

- **Word Order Typology:** Languages use to diverge in the basic sentence word order composition. German, French, English, and Mandarin, for example, are all Subject-Verb-Object (SVO), meaning that there is the structure which forces a verb to be encapsulated between a subject and an object. Eastern languages, such as Japanese or Hindi, use instead the Subject-Object-Verb (SOV) paradigm instead, while Irish and Arabic uses the VSO one.

- **Lexical Divergences:** For what concernes the translation of individual words, we must take into account that the appropriate word translation may depend upon the context. A clear example for this can be the English word *bank* can appear in Italian as the bank company *banca* or such as the river bank

riva. Or again, there might be a doubling given a single word like *brother* which in Japanese have distinct translations given the subject for *older brother* and *younger brother*. In all these cases a MT system is required to somehow disambiguate the different uses of the words.

- **Referential capacity:** Lastly, every language may vary among a common referential system along the typological dimension, indeed omitting referential subjects. English language pretends to explicity refer to a subject when this referent is given in the discours, while other languages are defined as $pro-drop$, meaning that they are prone to omit references. This phenomena tend to amplify among existing languages, leading to different leves of referential density.

Translating has never been a simple task due to the existance of these kind of problems that need to be overcome, thus this filed is continuously expanding towards model capable of better understaning all dependencies, contexts and ambiguities. The subsequent paragraphs of this chapter delve into how various techniques rised to answer the need of dealing the aforementioned issues from an empirichal point of view and then aknowleging how machine learning and neural networks have been harnessed to tackle these challenges and advance the field of machine translation.

2.2 Approaches to Natural Language Processing

The journey to automate language processing has seen a significant evolution from initial rule-based approaches to the more data-driven methods that form the foundation of modern Natural Language Processing (NLP) techniques. This progression has been driven by the growing realization that capturing the complexity and intricacy of human language demands approaches that can adapt and learn from vast amounts of linguistic data.

2.2.1 Rule-Based Approaches

In the early days of Natural Language Processing and Machine Translation, rule-based approaches (RBMT) were predominantly used. These models use handcrafted rules and expert knowledge to perform tasks such as part-of-speech tagging, named entity recognition, and sentiment analysis. Rule-based models are interpretable and can provide detailed insights into the decision-making process. However, they often lack the ability to capture complex language patterns and struggle with handling ambiguity and variations, thus they heavily depend on the availability of given rules and may not perform good enough in particular scenarios in which there are few or none well-defined rules.

One of the main examples of rule-based approaches in the field of machine translation was surely the system called Systran [67], made by one of the oldest Machine Translation focused company. Born in 1968 it was a system in which translation was generated solely using a combination of pre-defined linguistic rules and transformational grammar to map source sentences to their target equivalents. It was such a

breakthrough in that field that it was also used for the Apollo-Soyuz project (1973) and by the Europian Commission (1975) [89]. Currently is still available available and able to translate more than 20 languages, evolved in more robust model exploiting Statistical Machine Translation (SMT) and Neural Machine Translation (NMT), approacces which will be covered in next paragraphs. Although such an approach represents a great development in NLP in general, it was constrained by the reliance on pre-defined rules and a limited capacity to handle the complexities of natural language. Rule-based approaches incur several complications that limit their effectiveness in capturing the characteristics of a language:

- **Scalability:** making intricate rules for every possible linguistic variation is a task beyond reach, making rule-based systems unwieldy and difficult to maintain.
- **Ambiguity Handling:** Polysemy (several meanings for a single word), as well as Homonymy (several words with different meanings with the same spelling/pronunciation) have posed challenges to rule-based systems to accurately interpret context.
- **Generalization:** Language exhibits substantial variation and creative expression, necessitating models that can generalize beyond rigid rule boundaries.

Having highlighted the shortcomings and limitations of rule-based models, the researchers themselves find in data-driven approaches the way forward to create models suitable for understanding language correctly, allowing machines to learn directly from data.

This move towards statistical and computational approaches marked a pivotal turning point in MT. Machine learning algorithms began to take hold in increasingly substantial roles, allowing systems to learn from large amounts of text and extrapolate patterns otherwise not recognisable by hand.

2.2.2 Statistical Approaches

Given the constant evolution of technology, computer resources and the availability of data in text format, there has been a gradual shift from rule-based approaches to more modetological ones that rely instead on statistics and machine learning. This paradigm shift in dealing with human language has made it possible to extract knowledge directly from linguistic data rather than through manually predefined rules.

An example of this transaction is probabilistic models, also known as Language Models. A language model is a probability over a sentence, so given a sequence of words $X = \{x_1, ..., x_n\}$ a language model gives us the probability of a such sequence of words to appear together in that order, $p(X) = p(x_1, ..., x_n)$.. In concrete terms, the probability distribution is learnt by the model over a large text corpus and the purpose of this model is to learn and then generate a probability distribution that generalises enough to be applicable even over unseen texts, dealing with different syntatic and semantic structures. Among the most popular Language Models is the

rudimentary *N-Gram* Model [10]. An N-gram model (where N means the number of words) is such that the text can be considered as a Markovian Process with the simple assumption that only the most recent $n-1$ words, belonging to a fixed-size window, are relevant for predicting the next word.

$$p\left(\{x_1,\ldots,x_n\}\right) = \prod_{l=1}^{n} p\left(x_i \mid x_1^{i-1}\right) \approx \prod_{l=1}^{n} p\left(x_i \mid x_{i-N+1}^{i-1}\right) \tag{2.1}$$

The probabilities of the N-grams $p\left(x_i \mid x_{i-N+1}^{i-1}\right)$ are given by the relative frequencies (let $f(s)$ the frequencie of a string s in the text corpora) $p\left(x_i \mid x_{i-N+1}^{i-1}\right) = \frac{f\left(x_{i-N+1}^{i}\right)}{f\left(x_{i-N+1}^{i-1}\right)}$.

One MT-specific case of SMT is represented by IBM in the early 1990s [11], Model 1 and later Model 2 (an adaptation of the eraliest one) are systems capable of translating from one language to another relying on algorithms to identify alignment probabilities between the various pairs of sentences in the source and target language, as well as between the various words from which they were composed. They assigned at every pair of strings (e, f) a value $Pr(f|e)$ as the probability that a translator, given string e, would produce f as it's translation. Assumed that, they were seeking the most likely translation given how correct a candidate translation is and how well it fits in the context, transforming this stament into a probabilistic approach exploiting Bayes Theorem to find the right order of words:

$$Pr(e|f) = \frac{Pr(e)Pr(f|e)}{Pr(f)} \tag{2.2}$$

Although it represent a significant avancement to apply SMT to Machine Translation these model had few limitations, regarding a lack of contextual information, inadeguate handling of word ambiguity and mostly the fact that they were restricted to compute only One-to-One aligments, meaning that each word in the source sentence was aligned to a single word in the target sentence.

Other approacces to overcome these problems includes the exploiting of Hidden Markov Models (HMM), which is a mathematical technique representing a substantial and helpful collection of stochastic processes. HMM was first formulated by Baum and Petrie [5] with its first and most important application being in automatic speech recognition [68]. Markov models have extremely high-order mathematical structures and, with proper application, they have important practical uses in several areas [64]. The HMM are useful for those tasks which involves not directly observable events, e.g., words that have been observed in a sentence or a whole text, as well as invisible events, e.g., part-of-speech tags. HMMs introduced the concept of alignment states, allowing words in the source sentence to be aligned with multiple words in the target sentence, and considering not only the current word alignment but also the previous alignment states, capturing sequential dependencies.

Models of this kind make it possible to overcome many of the limitations discussed above: generalisation, ambiguity management and adaptability are handled more robustly than previous rule-based approaches.

However, although revolutionary in the NLP sphere, they had substantial limitations. It quickly becomes apparent, as by definition, that the whole text is not counted in the probabilities of generating the next word, consequently systems such as this one had difficulty in carrying out dependencies among long sentences, just as HMMs are not always able to model deeper linguistic complexities.
The subsequent evolution towards the use of Neural Networks has further overcome these limitations, allowing models to capture complex and contextual relationships in linguistic data by exploiting the availability of complex computational models introduced due to a strong technological advancement.

2.2.3 Machine Learning and Neural Approacces

The advent of neural networks and deep learning have marked an epochal turning point in terms of language processing, if not Computer Science in general. These architectures have induced models capable of learning complex language representations easily and with a good level of depth, effectively overcoming the limitations of previous rule-based and purely statistical-based models. Machine learning is that subfield of Artificial Intelligence that focus on developing algorithms and models that allow computers to extract and learn patterns from data and consequently produce predictions or decisions without being explicitly programmed beforehand [54]. Machine Learning utilises a wide range of techniques that allow the models themselves to improve their performance through experience (e.g. adaptation) towards the available data, adjusting their parameters accordingly. It can be divided into two macro categories:

- **Supervised learning:** In a supervised learning approach, a system is trained using training sample features coupled with a label and thus its outcome is observed. In order to model future events, algorithms of this type use labelled examples to learn patterns and make accurate predictions on new and unseen data.
- **Unsupervised Learning:** This is a kind of learning that does not consider human support as the model captures information without being given any hints, unlike the previous approach. The algorithm is trained with a set of unlabelled, unclassified, or uncategorised data and it is required to generalise on it independently from the data, trying to extract possibly latent patterns from it [18].

Supervised techniques are mostly used in NLP. Supervised Learning is an extremely varied and commonly used topic in computer science and is not discussed in detail here. For further and more in-depth studies, please refer to more demanding readings such as [42].

2.2.4 Perceptron

An artificial Neural Network (ANN) is a computational model that is inspired by the structure of neural networks in the human brain. The brain consists of a large number of simple computing devices (neurons) that communicate with each other in a complicated communication network that allows the brain to perform high-level calculations. Artificial Neural Networks are formal cognitive constructs that are based on this computational paradigm. Neural networks were first proposed in the middle of the 20th century [73], where the author asks himself in what form information is stored, processed and reimagined proposing a system named Perceptron, which is currently the first milestone of Deep Neural Network and Transformers used in NLP. To make a comparison, is a computational unit that has scalar inputs and outputs. Each input is linked to an otput via weights, which when summed and processed by an activation function generate an output, as in 2.1.

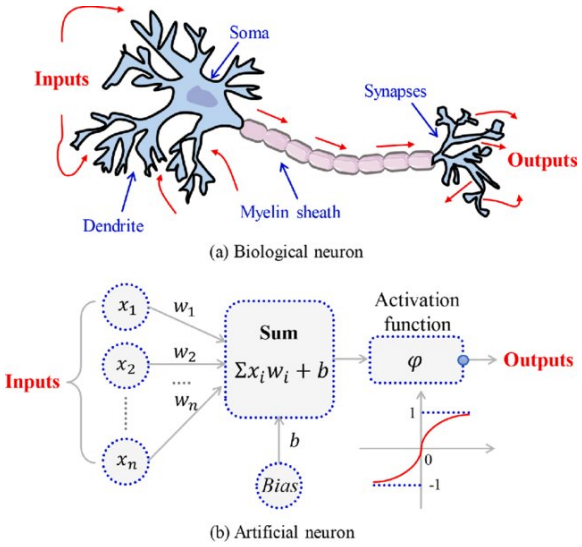


Figure 2.1: Comparison between biological neuron and artificial neuron [43]

Mathematically:

- **Inputs:** Let $x_1, x_2..., x_n$ be the input features. Each of them is then associated with a a weight $w_1, w_2..., w_n$
- **Weighted Sum**: The weighted sum of the inputs and weights is calculated as: $z = w_1x_1 + w_2x_2 + ... + w_nx_n$
- **Activation Function:** The previously calulated weighted sum z is the passed through an activation function $f(z)$ or φ

$$o = \varphi(\sum_i w_i x_i + b) \tag{2.3}$$

Activation Functions vary from the Identity one in which the output remains the same to a huge variety of non linear ones: example of those are Sigmoid, Tanh, RectifiedLinearUnit (RELU), which is just a sample of the may available ones; [47] goes in deep in this topic presenting and explaining all the commonly used ones.

2.2.5 Multi Layer Perceptron

While the perceptron with a single output neuron suffer from capability in generalizing non linear functions, Multilayer Perceptron (MLP), by means of using non-linear Activation Functions and more output units inside Hidden Layers is considered a universal approximator, which means it can model any function [39].

We refer to T as the number of layers V in the network (excluding V_0), or the depth of the network. The size of the network is $|V|$. The width of the network is $max_t\ |V_t|$. An illustration of a layered feedforward neural network of depth 2, size 10, and width 5, is given in the following 2.2.

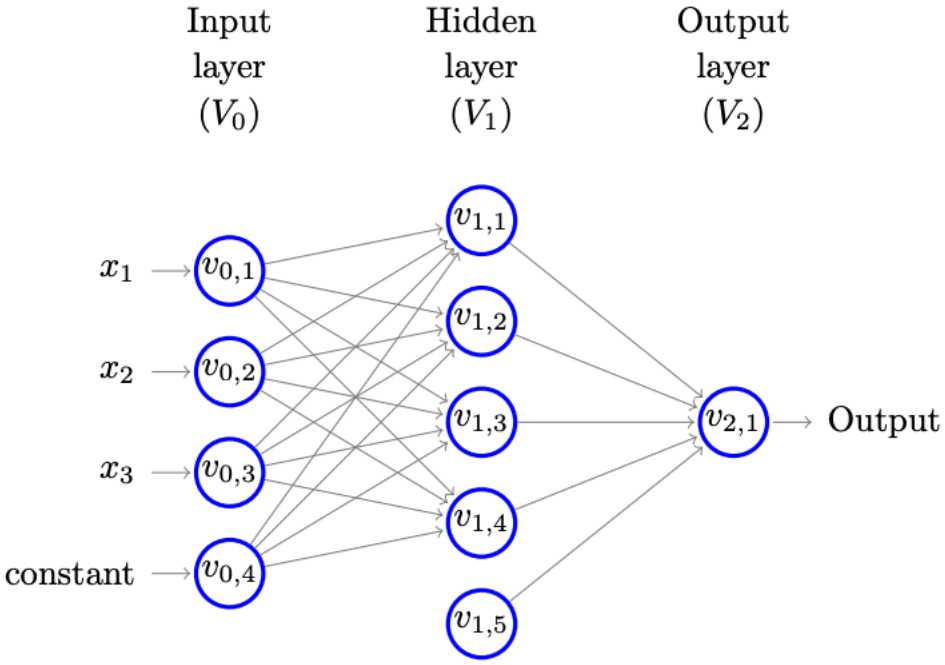


Figure 2.2: Multilayer Neural Network structure [81]

In this case loops are not allowed since a neural network in which loops are present is called recursive neural network (RNN), a special kind of structure which suits sequential parsing which we will explore later in this chapter due to it's importance for Machine Translation. This kind of architecture instead is indeed namely Feed Forward. FFNN process data propagating the information from the input layer to the output layer because, just like in the Perceptron, inputs are combined with the initial weights in a weighted sum and subjected to the activation function. The difference lives in the fact that each leayer is feeding the next one with the results of their computation, a latent representation made from the combination of different non-linear activation functions to process the data. FFNN are usually trained by by means of gradient-based approaces such as Back Propagation [76]. Back Propagation is the learning mechanism which many machine learning models are based on, which allows the latters to iteratively adjust the target (in the case of the MLP it adjusts the weights in the network) with the ultimate goal of minimizing a cost function, with the only constraint for this cost funtion to be differentiable. There are a lot of

Loss Functions already available in literature. The most common comprehend a set of functions used on a broad variety of tasks such as:

- **Mean Absolute Error** MAE finds the average of the absolute differences between the target and the predicted outputs, commonly used also for Regression purposes:

$$\mathcal{L}_{\mathcal{MAE}} = \frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i)| \tag{2.4}$$

Where:

N : Number of samples
i : Index of the sample (ranging from 1 to N)
y_i : Ground truth value for the i-th sample
$f(x_i)$: Model's prediction for the i-th sample

- **Binary Cross-Entropy:** This is the loss function used in binary classification models where the model takes in an input and has to classify it into one of two pre-set categories.

$$\mathcal{L}_{\mathcal{BCE}} = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log\left(p(x_i)\right) + (1 - y_i) \cdot \log\left(1 - p(x_i)\right) \tag{2.5}$$

Where:

N : Number of samples
i : Index of the sample (ranging from 1 to N)
y_i : Ground truth label for the i-th sample (0 or 1)
x_i : Model's prediction (probability) for the i-th sample
$p(x_i)$: Probability predicted by the model for the i-th sample

- **Categorical Cross-Entropy Loss:** like the aforementioned CE loss, Categorical CE comes handy for classes being more than only two.

$$\mathcal{L}_{\mathcal{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \cdot \log\left(f(x_{ij})\right) \tag{2.6}$$

Where:

N : Number of samples
M : Number of classes
i : Index of the sample (ranging from 1 to N)
j : Index of the class (ranging from 1 to M)
y_{ij} : Ground truth label for the i-th sample and j-th class
x_{ij} : Logit or raw score for the i-th sample and j-th class
$f(x_{ij})$: Softmax function applied to the logit for the i-th sample and j-th class

Lot more on this can be found in [92] where custom Loss Functions are depicted in a more exhaustive way.

Let's now take for example the MAE Loss function 2.4. Once error E has been calculated, to propagate it back and adjust relatives values of weights in the network straight to the starting point the Gradient Descend algorithm [74] is used and repeated iteratively:

$$\Delta_w(t) = -\varepsilon \frac{dE}{dw_{(t)}} + \alpha \Delta_{w(t-1)} \tag{2.7}$$

where as $\Delta_w(t)$ we denote the Gradient Current Iteration, E the derivative of the error, $w_{(t)}$ the Weight vector, α is the Learning Rate (ratio at which the model learns) and by $\Delta_{w(t-1)}$ we mean the Gradient at previous Iteration. This process keeps going untill a *convergence threshold* is reached compared with previous iterations, or in other words, a converged status of the Neural Network is reachead, meaning that the error is minimum and the net is now able to estimate correct outputs given input data, by means of correct learnt weights.

This whole process is particularly relevant due to its presence as the base learning mechanism of every Deep Learning architecture, regardless of the size of the latter. Various different approaches have been proposed in the last decade [32, 35], but it still remains the most incisive one.

2.2.6 Transitioning from Text to Numbers

Neural networks are currently the most commonly exploited tools in NLP and Machine Translation fields. These models take as input a vector of X to be fed to the various layers of the network to produce an output. The need then arises to transform the textual format to numeric in order to allow the various levels of computation we have seen in the previous section. This transformation involves capturing the semantic and syntactic nuances of language while retaining the information essential for the learning process. This process is called *feature extraction*. Various techniques have been developed over the years providing robust solutions for this task.

The earliest techniques on extrapolating informations from text involved a particularly basic and unarticulated approach, in which the focus was given to the words themself rather than the latters in their context of reference, thus, the main source of information fell on the characteristics of the word itself; Aspects such as: the letters that compose it, their capitalisation (e.g. 'c' or 'C'), the presence or absence of punctuation symbols (apostrophes, hypen, prefixes, suffixes and so on), or one can consider the order within the reference document by promoting a first timid approach to contextualisation, perhaps by highlighting how many times the word was present within a corpus or a single sentence. Bag-of-Words (BoW) [33] is based on the latter. The idea is very simple: each document in our corpus is represented by counting how many times each word appears in it. This creates what is referred to as a *document-matrix* $W = N * J$ where, document-wise, N refers to the phrase and J to the tokens of the vocabulary where each element W_{ij} it's the number of times that token it self appears into the corpora. Limitations of such a model are

clear:

- **Lack of Context:** Bag-of-words models do not preserve the order of appearance of features in a document, which can remove important information in some cases. For example, is this a good day and this is a good day would be considered equivalent if context is not taken into account while analyzing the text data.
- **Poor Quality Model:** Including all features from a document in a bag-of-words model can increase the model size, resulting in sparsity and numerical instabilities. Careful preprocessing of the document text is often required to build a useful bag-of-words model.

This approach has been futherly explored by introducing the so called Term Frequency - Inverse Document Frequency (TF-IDF) [83]:

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right) \tag{2.8}$$

which assigns weights to words based on their importance in a specific document relative to a larger corpus. Ineeded the first term returns the frequency of the word in a give text, while the second term is used to calculate the weight of rare words across all documents in the corpus. Again, this method suffers from severes limitations such as it makes no use of semantic similarities between words and assumes that the counts of different words provide independent evidence of similarity.

2.2.7 Word Embeddings

Word Embeddings obviate the need to obtain a data structure suitable for processing text formats, which is light in size and actually manages to capture some form of relationship between words. Normally Word representations, as explained in the previous paragraph, tended to be particularly high-dimensional and very sparse, the idea behind Word Embeddings is instead to represent words in a much more compact vector that nevertheless retains the characteristics, as well as the semantics and contextual relationships of each word. They are continuously learned vector representations of discrete variables mostly used in neural networks. In this low-dimensional space, algorithms that generate word embeddings vectors (and thus words) with a strong semantic relationship will be represented close together in the space itself. Different embeddings exist:

- **Count-Based Embeddings:** which promote the idea that words that appear in similar contexts also share a common meaning. These are based on a matric that takes into account how many times a word appears in the context of every other word in the corpus. Embedding is then generated by applying dimensionality-reduction techniques such as Principal Component Analysis (PCA) [52] or Singular Value Decomposition (SVD) [45], returning a dense low-dimensionality vector.

- **Prediction-based embeddings:** Later introduced in [8], these make use of neural networks, from which they take the name *neural embeddings*. Obtained by training a neural network on a large corpus whose final weights will represent the embedding vectors for each word, all with the same dimensionality.

The rise in popularity in the use of vector representations of words, as well as the numerous taks applicable to NLP, gave rise to several models, the most famous of which are *Word2Vec* [53] and *Glove* [62]. These pre-trained word embeddings have revolutionised the world of Natural Language Processing by improving the performance of many models, saving significant time and especially computational resources by being the result of other previously trained models.

Word2Vec

Strongly cited within the academic world, this model has made word embeddings very popular because of their easy accessibility. It originated through two different implementations, *continuous-bag-of-words* (CBOW) and *skipgram*.

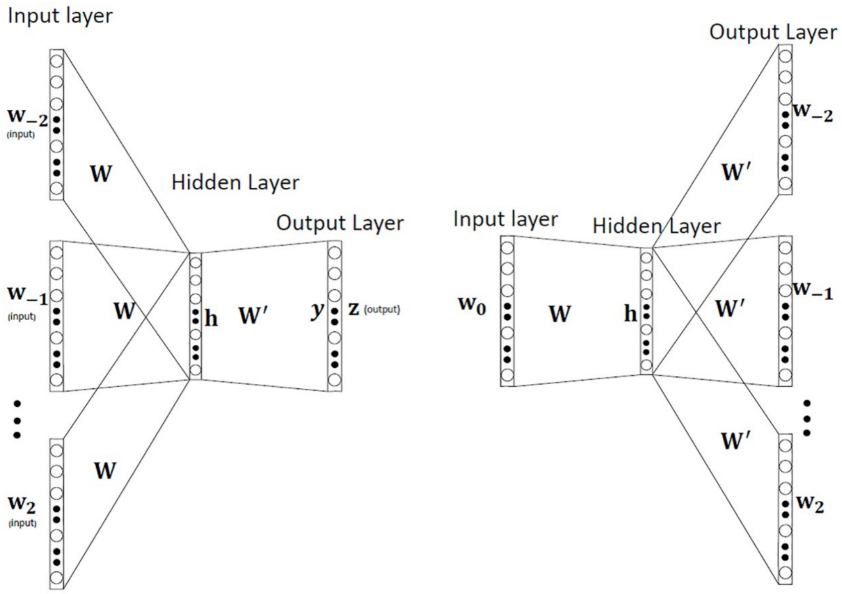


Figure 2.3: (a) Continuous-Bag-of-Words and (b) Skipgram models [72]

CBOW model operates is that given a set of input words the model tries to predict a target word based on the input words context. Each word is given as input as a sparse vector where all positions are 0 apart from the index (based on the vocabulary) in which that word is present, thus represented as 1, multiplied then by the weight matrix as to produce the embedding for each word. By means of the process we have seen in 2.2.5 the Neural Network is trained with the goal of minimizing the difference between the predicted words and the true words. By looking at 2.3, *CBOW*(a) wants to predict the word W_i given a window context of a fixed size

$$p\left(x_i \mid x_{j(|j-i|\leq l, j\neq i)}\right) = \text{Softmax}\left(W\left(\sum_{|j-i|\leq l, j\neq i} M x_j\right)\right), \tag{2.9}$$

where l is the window size, M (projection layer) and W (output layer) are the trainable parameters, V the vocabulary and m the embedding dimension. On the contrary *Skipgram*(b) model predict the words in the context given a word W_i in the center

$$p\left(x_j \mid x_i\right) = \text{Softmax}\left(W_j M x_j\right) \forall j : |j - i| \leq j, j \neq j, \tag{2.10}$$

where W_j is the output matrix for the j-th context word. The embedding of a word x_i is given by its product with the learned projection matrix M. Despite being very effective for word latent representation, those model suffer from not being able to capture different meaning of single words in a real huaman-scenario, in which every word might embed various nuances depending on the context. For this reason new architectures (presentend in the following paragraphs) have been proposed to fix this problem.

2.2.8 Recurrent Neural Networks

Now that in 2.2.3 we have presented Machine Learning, Deep Learning and related architectures, while in 2.2.7 it is explained how static Neural Networks are able to create embeddings specifically for NLP purposes, it is clear that conventional FFNNs are a great tool for general-purpose taks but lack any linguistic adaptation. Recurrent Neural Networks, or RNNs, belong to a set of Neural Networks particularly suitable for sequential data and thus text. By processing vectors representing words the idea behind RNN in NLP is to exploit some kind of memory of the previous word belonging to a sentece or a text in order to have a latent understanding of the context of each of them. The structure is designed to capture the sequential paradigm of natural language by means of a hidden state which is updated at each time step and then passed to the next one, actually enabling some sort of memory of the previous words. A first approach come directly from 1980s with Hopfield [38] and its network not specifically designed to handle sequential data but still introducing a layer with binary states representing some primitive memory. Afterwards, other models were presented introducing the concept of backpropagation through time (BPTT) [75] to train RNNs and enabling this architecture to be capable to handle textual data. A RNN processes inputs $x_i, \ldots, x_j$ by updating the hidden state at each time step based on current given input combined with the previous hidden state where the is used as the memory structure for the algorithm.

The output is calculated at each step by:

$$\begin{aligned} H_t &= f\left(V \cdot x_{t-1} + W \cdot H_{t-1} + b_h\right) \\ y_t &= g\left(U \cdot H_t + b_y\right) \end{aligned} \tag{2.11}$$

Where we have that :

- x_t is the input at time step t.
- y_t is the output at time step t
- H_t is the hidden state at time step t.

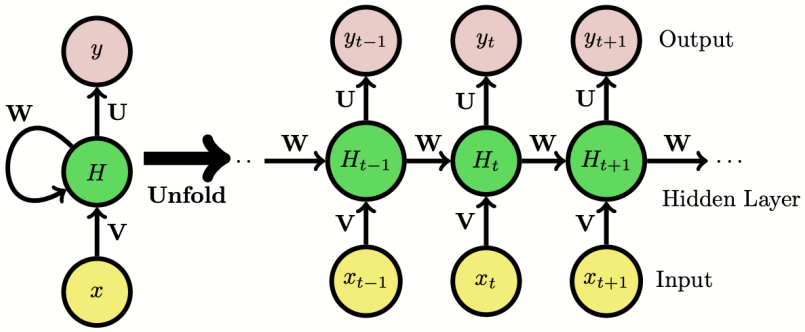


Figure 2.4: Recurrent Neural Network Structure [94]

- V is the weight matrix for the input in the recurrent layer.
- W is the weight matrix for the hidden units in the recurrent layer.
- U is the weight matrix for the hidden units in the output.
- b_h, b_y are bias vectors.
- f and g are activation functions.

The goal is to minimize a loss function over time $\sum_{t=1}^{T} L\left(x_t, y_t, \boldsymbol{\theta}\right)$, where T represents the number of time steps. Commonly a Cross-Entropy Loss function (2.2.5) is used and fed to BPTT for evaluation at each time step. Following the prediction of a distribution across the following output words, a token ti is selected, and its related embedding vector is provided as the input to the following phase.

Limitations

While RNNs provide a good mechanism to handle sequential data, learning long range dependendcies in text is quite a challenge for this architecture. Since the Equation 2.11 is computed multiple times through time, it let the gradinet to become either very small or very big, leading to phenomena called *Vanishing Gradient* and *Exploding Gradient*. This problem appears when the gradient at each time-step is the product of several factors that are less than 1, causing the gradient to gradually become very small over time, eventually reaching values next to 0, or, contrarely, going way above single-digit value without any cap, messing things up. While the latter is actually fixable by introducing some given treshold such that the gradient can't go higher than it, the vanishing gradient problem remains something to take into account, not letting RNNs to be good for longer senquences. Thus, suppose we want to propagate some error:

$$\frac{\delta E}{\delta W} = \frac{\delta E}{\delta y_t} \cdot \frac{\delta E}{\delta y_{t-1}} \cdot \ldots \cdot \frac{\delta E}{\delta y_2} \cdot \frac{\delta E}{\delta y_1} \tag{2.12}$$

To compute it at any given time step t the formula will appear like in Equation 2.12, where if any gradient will get close to 0, it will propagate the behaviour to the others, leading to a multiplication which approaches 0 value. This problem is actually solved by Long Short-Term Memory (LSTM) in the next section.

2.2.9 Long short-term memory Networks

Long short-term memory Networks (LSTM) represent a more complex RNNs which includes specific memory cells capable of resolving the problem of vanishing/exploding gradients, being definitely better for process longer dependencies. Proposed by Hochreiter and Schmidhuber [36], this architetcure introduces a memory cell structure that allows the model to chose which information to keep or to "forget". This cells store informations throughout time steps to module the output. The ability to have control on information's flow is a big advantage to handle sequential data, thus improving performances on several NLP task. It has since then evolved into a lot of different variants of LSTM, that builds upon the first model, for example with additional gates or different activation functions.

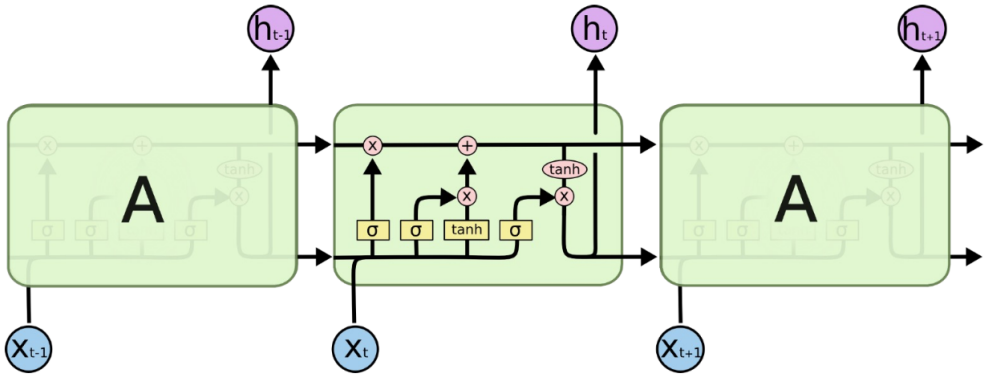


Figure 2.5: LSTM architecture

LSTMs have 4 Neural Networks in each cell interacting with each other. The architecture itself is base upon the so called gates: input gates, forget gate and output gate. As the name suggests, the forget gate filters memory by deciding how much information should be forgotten. The input gate purpose is to decide how much of the memory should be updated based on the input. Finally, the output gate is responsible to decide how much should be sent to the next cell.

Let C_{t-1}, H_{t-1} and X_t be the previous cell's cell state and hidden state, the first thing to do is to calculate the forget gate F_t at time stamp t. This layer is defined as:

$$F_t = \sigma\left(W_f \cdot X_t + U_f \cdot H_{t-1} + b_f\right) \tag{2.13}$$

where W_f, U_f and b_f are the layers weight and bias, while σ is the sigmoid function which returns a positive value between 0 and 1 to squiggle the result in this range to be more manageable. Equation 2.13 indicates how much to be forgotten, where 0 would mean to forget and 1 not to. The input gate, useful to channel how much of the input gate should be updated, is calculated as follows:

$$I_t = \sigma\left(W_i \cdot X_t + U_i \cdot H_{t-1} + b_i\right) \tag{2.14}$$

To update the cell state, a candidate state $C\tilde{}_t$ is made to capture what information should be added to the current cell state:

$$\tilde{C}_t = \tanh\left(W_c \cdot X_t + U_c \cdot H_{t-1} + b_c\right) \tag{2.15}$$

where *tanh* is the activation function which squiggles the results between -1 and 1, where -1 means the information should be forgotten and while 1 means to keep it. The current cell state is calculated by:

$$C_t = F_t \circ C_{t-1} + I_t \circ \tilde{C}_t \tag{2.16}$$

After that the the O_t gate is calculated with a sigmoid layer again:

$$O_t = \sigma\left(W_t X_t + U_t \cdot H_{t-1} + b_o\right) \tag{2.17}$$

And finally the hidden state H_t is computed:

$$H_t = O_t \circ \tanh\left(C_t\right) \tag{2.18}$$

leading to an output H_t being the combination of the current input and the update memory at time step t. By introducing the structure containings memory cells, the LSTM improves RNN's architecture with respect to sequence parsing and context learning, being widely utilized in past years for many tasks due to their versability. Note that the computaion can be made more accurate and concise by adding more layers of LSTMs (LSTM-stack) on the same base structure or even a Bi-Directional Long Short-Term Moemory approach (BD-LSTM), leading to a more powerful tool albe to exploit computational power to produce better results.

ELMO (Enough, Let's Move On) [63] is a two-layer BiLSTM sequence encoder which indeed captures the semantics of the context based on the BiLSTM accumulated information. The trained ELMO model has been used mainly to extract contextualized embbeddings (used to learn sequence-level semantics by considering the sequence of all words in the documents) which perform really well on a variety of tasks. However, more sofisticated architectures have been developed lately which higely surclass outperform LSTM by means of efficency, versatility, scalability and so on, namely Transformers.

2.2.10 Transformers

Firstly proposed in 2017 by Vaswani et al. [91], being one of the most cited paper in the Artifical Intelligence domain, Trasfomer architectures represent a real breakthrough in the Natural Language Processing field. Nearly all NLP tasks are now covered by a Transformer model, hence given a taks the first step is to grab one of the many available pre-trained Large Language Models (LLMs)[15, 13, 7, 49] based on a Transformer architecture and use it as it is or fine-tune for a specific domain-task The Trasformer is a Deep Neural Network model that relies on a *encoder-decoder* architecture, relying exclusively on an attention mechanism. Attention mechanism [91] was originally proposed as enhancement for encoder-decoder RNNs applied to Sequence-to-Sequence (Seq2Seq) applications, where the entire input was compressed by the Encoder into a single fixed-lenght vector to be fed into

the Decoder. The basic idea of Bahdanau's mechanism was, instead of compressing the input, to revisit and update the input sequence at every step, enabling the decoder to to selectively focus on certain parts of the input sequence at a particular decoding step, leading the Decoder to *attend* to different part of the sequence at each step.

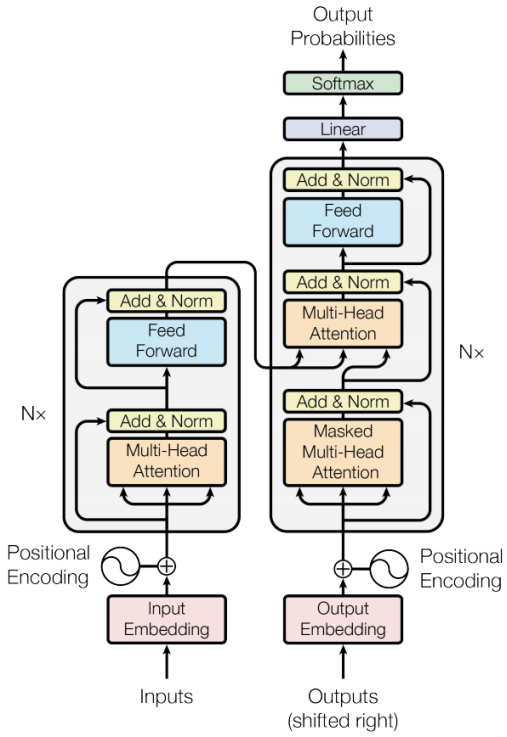


Figure 2.6: Transformer architecture from [91]

Starting from 2.6, the architecure can be futherly decomposed in:

- **Encoder:** A stack of 6 identical layers where each of them is made of one multi-headed self-attention layer and one position-wise layer, each of them has have 512 dimensions.
- **Decoder:** Still made by a stack of 6 layers, composed similarly to the encoders ones but with a third sub-layer, which performs multi-head attention over the output of the encoder stack.

Positional Encoding

Since the model does not really contain any part of convolution nor recurrences, a specific trick is used in order to inject some information regarding position, both relative and absolute, of the tokens in the sequence. It is called *Positional Encoding* which is a layer put right above the input embeddings at the bottom of the encoder and decoder stacks and this layer has the exact same dimension d_{model} of the embeddings, and it describes the location or position of an entity in a sequence so that each position is assigned a unique representation. Although there are few variants of

this mechanism, one of which in [26], base transformers use sine and cosine functions of different frequencies:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right) \end{aligned} \tag{2.19}$$

where *pos* is the potion and i the dimenstion, so that every dimension of the positional encoding corresponds to a sinusoid. By varying *pos* on the x-axis of the function, it will land up with different position values on the y-axis. Therefore, words with different positions will have different position embeddings values. This leads to the evidence that since *sin* curve obviously repeat intervals, differente positions may have the same embedding values despite being into two very different positions. This is where the i comes into place: by varying it in the above equation we can get a bunch of curves with varying frequencies, and this lets this layer to always produce diffrent values from these functions, regardless of the sequence's length (2.7).

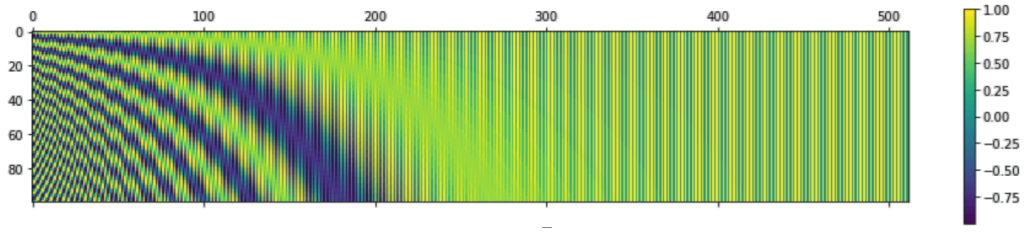


Figure 2.7: The positional encoding matrix for n=10k, d=512, sequence length=100 [91]

Attention Mechanism

The attention model was introduced to address tehe bottleneck problem found by using fixed-length econding vector where the decoder would have not completely-free access to the information provided by the input. This is indeed a very problematic aspect to address for long/complex sequences where the dimensionlaity of the longer representation would be forced to be collapsed as to the shorter one. Mapping a query to a set of key-value pairs can yield an attention function that produces a vector output. All components involved (query, keys, values, and output) are represented in vector form. Output is generated through a weighted summation of values, determined by how compatible the query is with each key. One type of attention mechanism utilized in Transformers is *Scaled Dot-Product Attention* (2.8), requiring input of queries, keys and values with dimensions d_k and d_v, respectively. Softmax function decides the weights of values by scaling the dot products between keys and queries with $\sqrt{d_k}$. For simultaneous attention function, Q matrix holds various queries. Keys and values are together in matrices K and V.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.20}$$

In practice, transformers perfrom several attention computations parallelym computing *Multi-Head Attention*, which allows the architecture to attend to informations from different sub-spaces at different positions.

$$\text{MultiHead}(Q, K, V) = \text{Concat}\,(\ \text{head}_1, \ldots,\ \text{head}_h)\, W^O \tag{2.21}$$

$$\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{2.22}$$

Where the projections are parameters matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_q}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

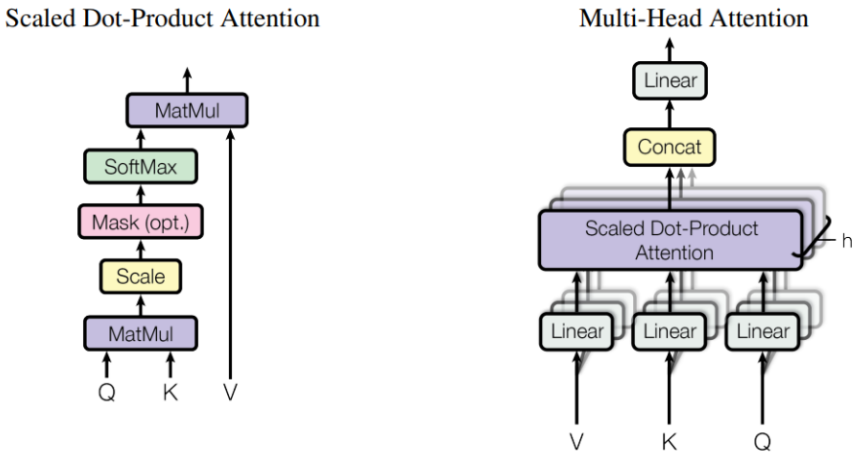


Figure 2.8: (a) Scaled Dot-Product attention and (b) Multi-Head attention [91]

Various learned linear projections are used to project queries, keys, and values h times to d_k, d_k, and d_v dimensions, respectively. This approach is more beneficial rather than performing a single attention function with d_{model}-dimensional keys, values, and queries. The attention function is simultaneously carried out on each of these projected versions of queries, keys, and values in order to generate d_v-dimensional output values. These output values are then consolidated and projected once more for the final values, as shown in 2.8. Due to multi-head attention, the model can attend to data from several representation subspaces in different areas concurrently providing overall better performances in a wide ranges of tasks.

To conclude, the Transformer architecure revolutionized the domain of natural language processing, particularly for Machine Translation. The self-attention mechanism enables the model to capture the most intricate contextual relationships, enabling to better handle long-range dependencies in texts. One of the most important contribution is that eliminating recurrencies found in other architetcures, this kind of model cuts down complexity and need for resources by accellerating training and being also able to parallelize the job. The architecture success brought numerous variats in all the text-related field and even more, including SOTA translation models. As we move foward, the Transformer's impact keeps shaping the landscape of mordern language processing and AI-driven applications.

2.3 Neural Machine Translation

Neural approces in Machine Translation takes advantage of the architectures presented in the previous paragraphs. Compared to previous methods (RBMT, SMT), Neural machine translation rely solely on Neural Network architectures, enabling a End-To-End Leaning apporach since NMT maps directly source to target language without the need of any pipeline regarding word alignment or any other statistical step, simplifying the translation process by also reducing error propagation.

2.3.1 MT exploiting Encoder-Decoder

Nowadays Machine translation systems rely on *encoder-decoder* or *sequence-to-sequence* models. Given a *source* sentence, the MT architecture will translate it into *target* output. Neural Machine translation, differently than RBMT and SMT uses a Supervised Machine Learning approach (2.2.3), which means that as training input it is given a large set of parallel sentences (e.g. English $\rightarrow$ *"this is just a sentence"*, Spanish $\rightarrow$ *"esta es solo una frase"*) fed into the system which is supposed to learn and map source sentences into trarget ones. The system is then trained to maximize the probability of the sequence of token ins target language $y_1, ..., y_m$ given the input token sequence from soruce language $x_1, ..., x_n$:

$$P\left(y_1, \ldots, y_m \mid x_1, \ldots, x_n\right) \tag{2.23}$$

By means of such structure involving more neural network's layers, the encoder is able to take in input those words $x = [x_1, \ldots, x_n]$ and produce an output representation $H^{enc} = [h_1, \ldots, h_t]$; usually by means of various stacked encoder blocks. Afterwords, the decoder will attend to the encoder representations generating target-language words at each time step, contitioned by previously generated words to generate next ones. To do this there a re a bunch of techniques the model can exploit, e.g. Sampling [37], Beam Seach [22] or Greedy Decoding [30]. In the latter case, we generate the most probable token $\hat{y}_t$:

$$\hat{y}_t = \text{argmax}_{\text{w}\in\text{V}}\, P\left(w \mid x, y_1 \ldots y_{t-1}\right) \tag{2.24}$$

where w refers to tokens and V to the vocabulary. The main difference with general-purpose transformers it that the decoder block contains an addition *cross-attention* layer, also known as *encoder-decoder attention* or *source-attention* which has the same structure of the multi-head attention (2.2.10) but in this case while Q matrix commes from the previous layer of the deconding block, K and V matrices come from the output of the encoder, thus allowing the decoder to attend to each of the source language words as projected into the entire encoder final output representations. Finally, by means of an autoregressive behaviour, the model predicts the next token y_t using the Cross-Entropy Loss.

2.3.2 Multilingual Models

The models presented so far only apply for bilingual translations, considering only one source and one target language. Multilingual machine translation models are instead designed to translate between multiple languages using a single shared model and are recently gaining popularity due to a few advantages they bring to the current scene. These models leverage the similarities and transferable knowledge across languages to improve efficiency and translation quality. In this prospective, we train these kind of models by feeding them with parallel sentences in many different pairs of source-target languages, with the additional parameter used to inform the model which language to translate from and to. This is done by adding a special token l_s given to the encoder to specify the source language we are translating from and again, another one to decoder l_t to the decoder to specify the target language.

$$\mathbf{h} = \text{encoder}\,(x, l_s) \tag{2.25}$$

$$y_{i+1} = \text{decoder}\,(\mathbf{h}, l_t, y_1, \ldots, y_i)) \quad \forall i \in [1, \ldots, m] \tag{2.26}$$

One of the most important advantage of such models is that they can improve the translation for those low-resourced languages by extrapolating hints and leveraging knowledge from high-resource languages. Futhermore, various learning scenarios are enhanced:

- **Zero-Shot Learning:** Translation without prior training, known as zero-shot translation [97], is achieved by training the model on various language pairs. This enables the model to perform translation for a language pair that was not part of its training. Multilingual models greatly benefit from this technique as it helps them recognize similarities between languages. While translating between unseen language pairs, the model uses its knowledge of language structures and relationships that it learned from the training set.

- **Few-Shot Learning:** The transfer of knowledge from multilingual models allows for excellent performance in few-shot learning [6], where only a meager amount of training data is available for a specific language pair. Even a few examples can make an impact. Limited resources are not an issue because the target language benefits from the acquired expertise in other languages. This way, translation quality is improved.

- **Transfer Learning:** The shared representations in multilingual models can be fine-tuned for specific language pairs or tasks. This transfer learning process allows the model to specialize its knowledge while still benefiting from the pretraining on multiple languages.

- **Improved Resource Allocation:** Training a single model is obviously more resource-friendly compared to train different models for any language pair. This allows for a more efficient allocation of resources and still good translation quality across multiple languages.

2.3.3 Related Works

While there's not exactly a comprehensive list of SOTA models for Machine Translation due to the continuous adaptation done on top of existing Transformer architectures, the NMT revolution was clear due to the burst in the number of related scientific publications during the last decade, as well as the attention received from the media, often related to tangible improvements in the quality of online MT system. Although much remains to be done, the quality of the most common languages (English, German, French, and other European languages) has reached unprecedented levels of accuracy when it comes to translation. Both academic and commercial environments quickly adopted NMT systems providing a robust answer for professional and consumer needs providing easily accessible instruments. Around the end of 2016, online MT offered by big companies Bing, DeepL or Google [19], or open-source systems such as OpenMT [44], was powered by deeper and deeper Neural attention-based architectures. In 2016 Google officially proposed its own NMT system, enhancing the already existing statistic-based one, by claiming to *"bridge the gap between Human and Machine Translation"* for EN/FR and EN/DE [96] and both output accuracy and computation speed were addressed and indeed improved by a revisited attention mechanism and low-precision arithmetic adoption during inference computations. Microsoft NMT team released their EN/ZH model by claiming the achievement of Human Parity translations [34]. As already stated, since the release of Transformers models, research rapidly moved towards faster and more robust architectures since the focus shifted from accuracy-seeking to faster training (due to always more complex structures). 2018 Conference on Machine Translation (WMT18) [9] stood out Facebook NMT model [57] achieving first ranking position on EN/DE translation. The system was known to be trained nearly 5x faster on a single 8-GPU machine due to a reduced numeric precision approach throughout the whole training computation instead of only inference like in Google's. While bi-lingual translators got a huge boost in performance in recent years, thanks to the continuous research on this topic, Multilingual models (2.3.2) arose to be the nowaday's standard by embedding multiple pair-language models into single ones. These models rely on transferable knowledge among pre-trained languages, bringing attention to low-resource languages which increasingly become the real bottleneck in MT research moving the general interest towards this topic leading to a broad variety of approaches such as Multi-Task Learning (MTL) [98], Transfer learning (TL) [99], Linguistic knowledge leverage [58], pre-trained lexicon embedding exploitation [51] and a vast related literature [69]. Transformer architectures are currently considered to be the best way to improve SOTA performance, and, as a result, a democratization trend is registered in the Machine Translation field (and NLP in general) through developments of cummunities and hubs such as of HuggingFaces Transformer hub [95], which is an open-source platform that allows users to upload and download standard or customized models using simple high-level APIs, and also the implementation of a variety of specifically-made frameworks for training, evaluating and inferencing NMT models like the ones we are going to use in the following chapters, namely ModernMT [27] and Nvidia NeMo [85].

2.3.4 Current open challenges in NMT

So far we have discussed on what Transformers and previous architectures are madeof, by also giving a brief overview on how they work. However, the impact of this architectures and the wide adoptation of these by researches arise numerous issues regarding language handling and data gathering. We must indeed take into account that there are many large parallel corpora containing translation between English and other languages, but the vast majority of word languages do not really have this availability. This takes us to consider the following topics to be directly involved in the qualitative development of Machine Translation systems.

Data Augmentation

This technique is a statistical approarch to deal with insufficient training data which could lead to a poorly performant trained model, by adding syntetic data made by current original data. One of the most common approach is namely *Backtranslation* [80]. It relies on the fact that although parallel corpora may be limited in size with respect to low-resources language, monolingual corpora are usually very easy to find and surely more robust in terms of size, and we can use the latter to augment smaller corpora. The idea is is to improve source-to-target Machine Translation, given a small parallel text (bitext) in source-target languages (where target is supposedly a low-resource language) and some mono-lingual data in the target language. This is done by firstly training a MT system in a reverse manner: target $\rightarrow$ source, then use it to translate monolingual target data to the source language. Having now new syntetic data made by the aforementioned system, we can now train a classic source $\rightarrow$ target model.

Do Not Translations

Alike the previous topic, a similar discourse can be made with respect to the output manipulation, specifically on handling given parts of the sentences or special tokens not to be translated by the system. Despite of the given language, these tokens can refer to names, dates, locations, or other parts of speech (POS) relevant for any particular use-case, for example the business needs which this thesis is based on, where we are creating a system capable of identifying some text entities during the translation phase. We think that there are mainly two ways to settle this problem: either to re-eengineer the Transfomer architecture and make it able to directly interface with the tokens during the training and inference step (e.g. by managing the input vocabulary or model's structure somehow) or refactor the input data in order to let the model identify special tokens and understand they must not be translated. Unfortunately, as stated in Chapter 1, literature lacks researches aiming to provide solutions to this topic. We therefore had to create a novel pipeline for parallel-corpora handling and data augmentation specifically implemented for this task that can be hopefully taken as hint for other works.

Evaluation Metrics Reliability

Common ML systems rely on a strong base of evaluative measures such as Precision, Recall, F-score, etc. When it comes to Machine Translation there are a few of them mainly used, i.e. BLEU-score [59], METEOR-score [4] or TER [82]. While these metrics have been valuable tools for evaluating machine translation models, they do come with certain strong limitations. Translations indeed often involve complex linguistic constructs, idiomatic expressions, and nuances that cannot be adequately captured by simple n-gram matching on which these metrics are based on, leading to underestimation of translation quality. Shorter translations can be favored due to their brevity penalty, leading to the risk of models producing overly concise translations to inflate scores, without fully conveying the original meaning. Furthermore, translations that accurately paraphrase the original content providing a valid alternative expression, might be marked as incorrect by these metrics. This brings to the acknowledgment that new metrics for MT are needed to capture the true quality of a translation model.

Social and Ethical Issues

Finally, many problems regarding low-resource languages go beyong techincal aspects, reaching social and ethical spheres. One substantial issue is that for those low-resource languages belonging to also low-incoming countiers, native speaker are not really being involved in dealing with quality of the provided content or to validate it. This paper [55] regarding african language makes evident how in many multilingual datasets very few sentences were acceptable being either missing parts, repeated, or not generally quaitatively valuable, suggesting that native speakers were not truly involved in generating robust data. This behaviour does not refer only to Africa's countries but it has been seen to be replicated in many other middle-east and asian countires as well. Moreover, Machine Translation may rise some ethical issues with respect to the capability of the system itself to correctly refer to specific subjects, pronouns in general or genders due to particular gender-neutral language coupled with any multiple-gender one. These systems infact often default to male gender and can't properly assest pronouns between original source and translated sentence, eventually leading to miss-leading stereotypes [66]. Last but not least, its often the case that the output generated tends to carry over and amplify social biases learnt during the pre-training phases and developed as a result of the low-quality input data. These stereotypes and biases cover the whole social stratum penalizing minority and marginalized groups and being responsible for various issues regarding politics, ethicality, justice, but mostly ethnicity [2], religion [14], gender like in [86] which ultimately underlines the fact that non-binary genders are not even being considered as crucial aspect of a debiasing process. Although various techniques have been implemented that generate texts with a less obvious bias a posteriori, working on the input data and current MT architectures is still an open challenge that could improve the results already obtained.

Summary

This chapter provides a comprehensive overview of the background inherent to the work to which this thesis refers. Initially, in 2.1 Natural Language Processing and Machine Translation are formally presented and defined, as well as any challenges and golas concerning them are highlighted. Subsequently, in 2.2 a comprehensive overview of past and modern SOTA approaches is provided for a comprehensive view of the field. Finally, in 2.3, the chapter delves into Neural Machine Translation, explaining its fundamentals, presenting related work and highlighting current open challenges for this specific field.

Chapter 3

Methodology

This work has been developed both on local machines, for Python and bash scripting, and on a remote Amazon EC2[1] machine, mounting a NVIDIA Tesla T4 for CUDA operations. In this Chapter we are going to provide the metodologies, tools and techniques used to develop the project. An overview of the system is proposed in Figure 3.1. Starting from an overview of the collected data, will be then explained how the latter has been pre-processed and how the Named Entity replacement has been done by means of few tools, namely DeepPavlov[2]for Multilingual Named Entity Recognition and FastAlign[3] to retrieve alignments for the parallel corpora. Finally, we will show the training and fine-tuning process using ModernMT[4].

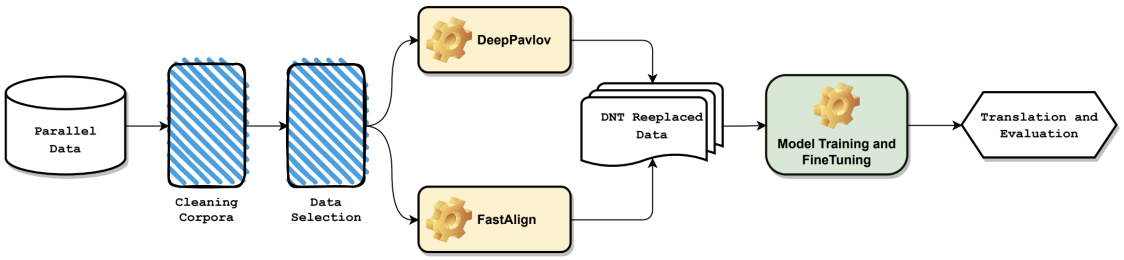


Figure 3.1: Machine Translation complete pipeline

3.1 Corpora Collection

Data collection is a process of primary importance for Machine Translation models. Quantitatively abundant and qualitatively valuable data is necessary in order to adequately create translation-accurate systems. LRLs constitute a major obstacle in both research and commercial applications related to machine translation.

There is no single, universal metric to define a parallel corpus as *high*, *low*, or *extremely − low* resources. Authors in [99], took the initiative to define a treshold below which a bilingual corpus could be considered as *low − resource*, setting this

[1] https://aws.amazon.com/it/ec2
[2] https://deeppavlov.ai/
[3] https://github.com/clab/fast_align/tree/master
[4] https://www.modernmt.com/

treshold at 1M parallel sentences. Subsequent works such as [46] or [48] contribute to defining a minimum threshold, lowering it to 0.5M and 0.1M sentences per corpora respectively. Despite these works, we can argue that there are no absolute values and the quantification of the level of sentences within a corpus remains only a convention in research. It should also be emphasised that this threshold relates specifically to parallel corpora. Therefore, even if a given language has very large single-language data but lacks representation in parallel corpora it will be defined as low-resource in this context.

The languages on which we base our work are Turkish and Urdu. In their work, [41] define five classes to which they assign various non-European languages (including those of interest to us) according to number of languages, number of speakers, and percentage of total languages for each language class. Class 1 languages have the greatest accessibilty. Examples are English, Spanish, German, few of the most spoken languages in the world. Class 0 instead includes languages with low corpora availability such as Dahalo, Warlpiri, Popoloca, with few or no representation in any resource. Turkish and Urdu are considered class 4 and class 3 respectively, indicating, although to a lesser extent compared to the first classes, the presence of different resources for the collection of data for these languages. For Class 1 languages it is easy to collect over few houndreds of million parallel-sentences, while for those more lower-class languages (such as Urdu) this amount could drop below a single million units. For our specific case we are going to exploit a single source, which happens to be one, if not the, best archive for multilingual parallel corpora.

3.1.1 Opus Corpora

Open Parallel Corpus (OPUS)[5] [88] is a growing collection of freely accessible translated text and thus parallel corpora collected from the web. There are over 3,800 language pairs with sentence-aligned data comprising a total of over 40 billion tokens in 2.7 billion parallel units (aligned sentences and sentence fragments). The collection comprehends the vast majority of parallel corpora currently available for LRLs on the internet. It is usually the case to expand the research of corpora to other sources when dealing to extremely-low-resource languages since opus it self may not be not enough. Luckily, both Turkish and Urdu are not really to be considered so low in resources, which means that OPUS is the best choice for our use-case. Although OPUS is only a tool to find data, inside it is possible to find the most famous corpora currently available on the web used for Machine Translation, provided both in *translation memory exchange* (TMX) and *Moses* formats. The first provide monolingual separated versions of the latter, which is a format specifically made for parallel corpora. Data has been then collected locally by means of OPUS API[6] by a hand-made script. Many of the available corpora are shared among languages, meaning that there are multilingual versions of the same collection and so that for both Turkish and Urdu there it is a set of common corpora coming from the same collection to be used to train our models.

For a more extensive overview, check Appendix A where precise versions, links

[5]More on `https://opus.nlpl.eu`

[6]`https://github.com/Helsinki-NLP/OPUS-API`

and paper references are reported.

CORPUS	DOCs	SENs	IT TOKENS	TR TOKENS
MultiCCAligned	1	6.7M	816.8M	207.9M
OpenSubtitles	36134	27.4M	211.4M	165.0M
WikiMatrix	1	0.7M	298.2M	56.6M
CCMatrix	1	11.1M	184.9M	147.0M
TED2020	3239	0.3M	6.4M	5.0M
QED	2806	0.4M	5.2M	4.1M
NeuLab-TedTalks	1876	0.2M	2.9M	3.6M
LinguaToolsWiki	1	1.1M	2.8M	3.0M
XLEnt	1	0.9M	2.4M	2.5M
GNOME	1220	0.5M	2.3M	2.3M
Tanzil	9	88.2k	1.4M	1.4M
KDE4	1160	0.2M	1.1M	0.8M
bible-uedin	1	30.1k	0.8M	0.6M
EUbookshop	55	16.4k	0.7M	0.5M
Mozilla-I10n	1	36.3k	0.5M	0.3M
PHP	2853	29.1k	0.2M	0.1M
Tatoeba	7	15.7k	0.1M	0.2M
GlobalVoices	70	2.2k	53.1k	40.4k
wikimedia	1	1.4k	34.0k	42.8k
ELRC-wiki_health	1	12.3k	0.5k	8.8k
KDEdoc	5	54	0.5k	0.3k
Total	**49443**	**49.7M**	1.5G	600.9M

Table 3.1: Turkish Language Corpus Statistics

CORPUS	DOCs	SENs	IT TOKENS	UR TOKENS
MultiCCAligned	1	0.6M	816.8M	24.4M
Tanzil	8	49.9k	1.3M	1.7M
XLEnt	1	0.2M	0.4M	0.5M
Mozilla-I10n	1	17.7k	0.5M	0.2M
QED	275	29.2k	0.3M	0.3M
TED2020	177	14.9k	0.3M	0.3M
OpenSubtitles	17	23.3k	0.2M	0.2M
GNOME	62	12.3k	0.3M	53.2k
NeuLab-TedTalks	101	6.0k	0.1M	0.1M
GlobalVoices	62	1.6k	40.3k	46.9k
Total	**705**	**0.9M**	**820.2M**	**27.8M**

Table 3.2: Urdu Language Corpus Statistics

Tables 3.1 and 3.2 report the collection of corpora currently available for the languages used in this work. The latter are ordered in descending order by the

number of available tokens for the source language (IT). Moreover, the number of documents contained in the collection, the number of target tokens (TR) aswell as the number of parallel and aligned sentences considering both languages is also reported.

Corpora inside OPUS have different domains, ranging from very generic to highly specialistic. For example, *MultiCCAligned* is one of the most generic. It has been collected from multiple sources or locations on the Internet. Other corpora such as *ELRC-wiki_health* are centred on medicine. *Gnome* corpus include collections of texts that deal with code-related issues and are therefore very specific. The latter, although highly specific, are nevertheless useful in the case of LRLs in order to make up the numbers to create at least high-performance models. With reference to the definition of low-resource language in 3.1, looking at the tables 3.1 and 3.2 and more precisely at the number of sentences per parallel-corpora, we see that Turkish, although it does not come close to the numbers of more common languages such as English, Spanish or French, does not suffer from a too marked lack of data, by reaching around 50M units of parallel sentences. The opposite is true for Urdu, which, on the other hand, falls under LRL definition, failing to reach even 1M parallel sentences. For this reason we collected more data through proprietary software that scans YouTube subtitles, reaching almost 5M sentences.

3.2 Data Preprocessing

Corpora preprocessing is one of the most important steps to create valuable inputs for our models by filtering out noisy and redundant parallel senteces and cleaning the remaining ones. This task is usally tackled manually before training the model, unless a high level frameworks such as ModernMT is used. In these cases, the framework tipically performs several cleaning operations on the given corpus. In our case, we chose to process our data following the process represented in Figure 3.2.

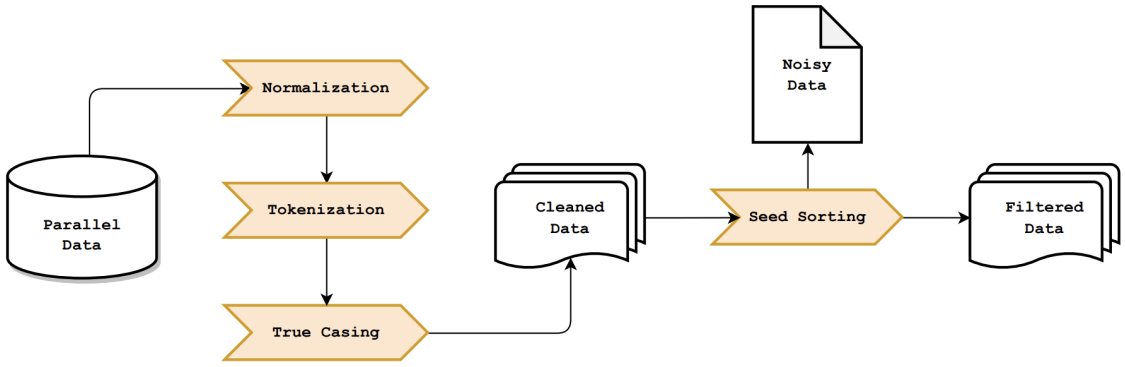


Figure 3.2: Data preprocessing pipeline

3.2.1 Data Cleaning

Being numerical or textual data, it must be processed and cleaned in order to provide to our models the best possible version of the data we have collected. We therefore apply the following cleaning steps:

- **Normalization:** Punctuation, especially things like quotes, can be written in different ways among different languages.
- **Tokenization:** this is a very common mechanism applied in NLP to separate different words and thus letting any uptop model to perfrom easier operations on tokenized text, where a token is a word belonging to the sentence.
- **True Casing:** This approach aims to statistically understand for each word if it is usually seen as cased or uncased and replace it with the correct version. For example, consider any word which is not a name or any relevant entity. In western languages we use to upper-case the first letter of the sentence and so that word, thus we make the latter lower-case since it will be mostly seen as lower-cased in the sentence.

3.2.2 Seed Sorting

Further, we sort and filter our corpora with respect to a seed. The purpose of this step is to sort the corpora according on how clean, or better similar, the sentences are with respect to a given domain. In our specific case we are going to apply the following approach only to Turkish language, which doesn't really suffer as much as Urdu of a lack of resources. For Urdu we chose to keep all the available sentences due to their lower numerosity. Moreover, it must be considered that the vast majority of sentences available for Urdu are mostly restricted to religious domains due to the fact that Koran is the source of which there are the most translations from professionals. For seed creation, it is advisable to select the "cleanest" corpora, ensuring that the domain is varied enough, and it needs to be between 100k and 10M words long; these are the main reasons for not seed filtering on Urdu.

Since we want to use this procedure to remove noisy and poorly-structured sentences from our corpora, we create a seed which is eventually able to clean lexically, semantically and sintactically the available corpora and which belongs to a neutral domain, since for a staring point we want to create the most general Machine Translation model possible, to evanually fine tune it with other data. Seed corpora are small, high-quality collections of parallel texts used to bootstrap machine translation systems. They may serve as the initial training data for machine translation models. These corpora typically consist of professionally translated sentences or documents, and they are used to kickstart the training process for machine translation models. The best case scenario would be the one in which we would have this seed available from official sources for both source and target languages. This is not the case since we are working with less resourced languages. Thus, we took a sample from our availabe corpora, and considered it our seed. The key idea is that having a seed which is mostly clean and mostly of neutral domain allows us to score all the sentences in our corpora by comparing them with the seed itself, ending up

with a score for each sentence according to how much it is clean and it belongs to neutral domain. We therefore take the assumption that the data downloaded from OPUS is mostly clean and mostly of neutral domain. Of course this is a sub-optimal solution. However, when it comes to low-resource language due to lack of official and curated corpora to work with it may be the only viable option.

The scoring is performed by training a Language Model on the seed, and then evaluating each sentence of each corpus by means of Perplexity for the Language Model. Higher scores will mean that the sentences are well explained by the model, whereas lower scores will point out that sentences are not really well explained.

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P\left(w_i \mid w_{i-1}\right)}} \tag{3.1}$$

Perplexity is a statistical measure of how confidently a Language Model predicts a text sample, where N is the number of words, w the single words, and W the model we are using. In other words this metric quantifies how surprised the model is when it sees new data answering to the question "how much can the Language Model can explain specific sentences?". The lower the perplexity, the better the model predicts the text, additionaly minimixing perpelxity is the same as maximizing probability.

In particular situations (mostly when a seed is not pre-available) seed creation process involves the exploitation of multiple corpora to extract it from, and so does for our case.

The seed is usually of the order of houndred of thousand or very few millions of words and bilingual, just like the raw corpora. It is then given to a script which will train a Language Model on the seed itself and returns us a score for every sentences on the overall corpora regarding how likely is each sentence given the language model. Since we are considering two languages in this score, both source and target, the resulting score will be an average of the two.

Finally, the corpora is sorted in ascending order with respect to the perplexity scores, meaning that the best sentences for each corpus will appear at the beginning of the newly sorted corpus while the worst at the end, allowing us to proceed to an effective filtering based on this result.

3.2.3 Data Selection

Lastly, we are going to perform Data Selection by indeed selecting increasing amount of the aforementioned scored-corpora starting from those sentences scored as the cleanest and the most neutral, then getting increasing portions of the latters and finally evaluate how well a Language Model trained of this portion of corpora is able to explain another corpora (so called development set, or devTest) which comes from an offical source for Machine Translation systems evaluation. For this problem we are going to exploit Facebook Low Resource MT benchmark (FLoRes[7]) [56]. It is a dataset developed for the specific purpose of Multilingual Machine Translation. It covers a large amount of languages. In our case we are going to use the *flores200* [29, 31] version which includes both Turkish and Urdu, and it contains a mostly neutral

[7]https://ai.meta.com/blog

and clean corpora manually translated from professionals in all those languages to effectively use it as a benchmark for development and testing purpose. The development part of of FLoRes will be indeed used for Data Selection, while Test part will be eventually exploited to evaluate or final models. Once the devTest is extracted for the available languages, we compute the perplexity on each portion of corpora with respect to the devTest; this can be done for both languages, but for time-complexity times, this has been done only for the source language. The portions are fixed in the range $R = \{5, 10, 15, \ldots, 95\}$, where each portion is evaluated like stated above, and this process is operated on each available corpus so that we can keep, for each of them, the portion needed to have the best Perplexity-Quantity tradeoff, eventually leading to our final training parallel-corpora.

3.3 DNTs replacement

In this section it is going to be presented the various steps and tools adopted for DNTs creation and replacement in the training data.

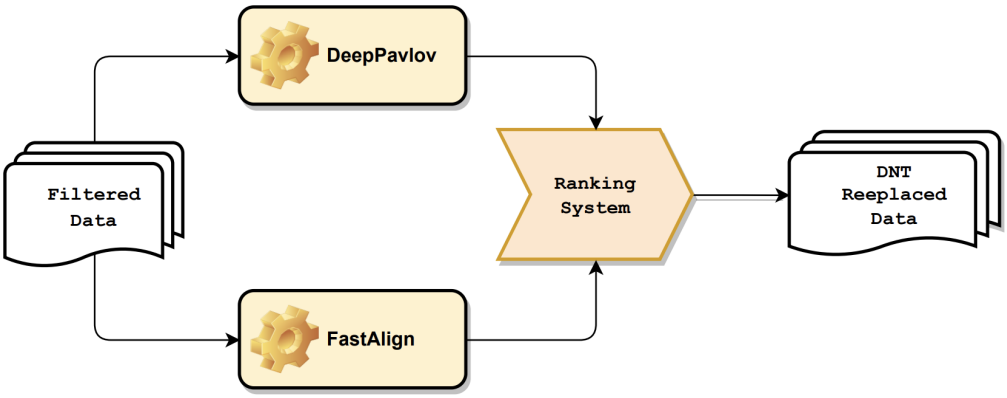


Figure 3.3: DNTs replacement and model pipeline

As shown in Figure 3.3, starting from our filtered training data, the methodology workflow now involves a process of DNTs-tags augmentation in the training corpora in order to make those tags relevant for the Machine Translation engines during inference, allowing us to have our tags returned in the translated text and sequently map related entities back in a post-processin step. The main idea here is to extract specific Named Entities by means of a NER system and align pieces of texts (words) between source and target corpora, leading to a map $source \rightarrow target$ of the found entities, to finally tag them as DNTs on both sides and train the model accordingly. For this task we exploited few SOTA pre-trained system which allow us to both extract those entities and align parallel-corpora pieces of text. Here the expression "*pieces of text*" is used due to the fact that our system will eventually provide alignments among languages that don't respect the same syntactic rules. Instead, different languages, most likely in low-resource languages domain, means different alphabet, words and sentences arrangement, so that a single word in source language doesnt really refers to a single word in the target one, as well as longer-

entities are not necessarily the same length in between languages, and this must be taken in consideration for better results.

3.3.1 Named Entity Recognition (NER)

Named Entity Recognition (see Sec. 2.1.1), is the task of assigning a tag (from a predefined set of tags) to each token in a given sequence. In other words, NER consists of identifying named entities in the text and classifying them into types (e.g. person name, organization, location etc). In this work NER has been developed by exploiting DeepPavlov ecosystem. DeepPavlov[8] is an open-source library tailored for development of conversational agents. The library itself contains a wide range of SOTA solutions for NLP tasks, ranging from low-level tasks such as Tokenization to complex ones e.g. Context Answering and NER. DeepPavlov exploits **BIO** tag system, which is tipically used to label spans of text through token-level tagging. It adopts 3 tags: B for the beginning of the entity, I for the inside of the entity, and O for non-entity tokens. The second part of the tag stands for the entity type. Table 3.3 better explains what BIO tagging performs:

Elon	Musk	owns	SpaceX	company	made	in	March	2002
B-PERSON	I-PERSON	O	B-ORG	O	O	O	B-DATE	I-DATE

Table 3.3: Tokens and Labels BIO Tagging

Here we can see three extracted named entities: Elon Musk (which is a person name), SpaceX (which is a name of an organization) and March 2002 (which is a date).

The reason why DeepPavlov has been chosen over other pre-trained systems is that is one of the best performing for multilingual NER tasks, as shown in Table 3.4.

Model	F_1-score
DeepPavlov	$\mathbf{87.07} \pm 0.21$
Strubell at al. [84]	86.84 ± 0.19
Spacy[9]	85.85
Chiu and Nichols [12]	86.19 ± 0.25
Durrett and Klein [16]	84.04

Table 3.4: SOTA NER Systems, shown in Anh et al. [3]

The NER Model architecture is indeed based on a BERT architecture specifically adapted for NER task, and it is trained on OntoNotes 5.0[10]. It is a large corpus comprising various genres of text, designed for multilinguality, thus covering both Turkish and Urdu. DeepPavlov offers a variety of BIO tags, and since the provided tasks mostly refers to names, dates, and places the set of tags shown in Table 3.5 have been chosen.

[8] https://github.com/deeppavlov/DeepPavlov

[10] https://paperswithcode.com/dataset/ontonotes-5-0

BIO Tag	Description
{B,I}-PERSON	People including fictional
{B,I}-LOCATION	Non-GPE locations, mountain ranges, bodies of water
{B,I}-NORP	Nationalities or religious or political groups
{B,I}-GPE	Countries, cities, states
{B,I}-ORGANIZATION	Companies, agencies, institutions, etc.
{B,I}-FACILITY	Buildings, airports, highways, bridges, etc.

Table 3.5: Chosen BIO Tags

We refer to the DeepPavlov documentation page[11] for other available tags and procedures to run the computation on GPU, which is a crucial aspect considering that, altough we are not training a system from scratch, we are still dealing with millions of parallel sentences to make inference on and we are exploiting a transformer, thus a GPU is needed for computational reasons.

3.3.2 Fast Align

In order to perform word-level aligment, we exploited FastAlign.[12] FastAlign is an effective reparametrization of IBM Model 2, already presented in [11] which eventually outperforms Model 4. in [17], authors show how well alignments work in downstream translation systems on a variety of language pairs. FastAlign takes in input parallel sentences and produces back alignments word by word. Therefore, the parallel input sentences must be tokenized and merged togheter in a single corpus where each line is a source language sentence and and it's target language translation, separated by a triple pipe symbol (|||). Since we can only pass tokenized data to the model, we are going to exploit DeepPavlov generated tokens for our purpose, since original text mantaining natural syntax would lead the word aligner not to work properly. An example 2-sentence Italian-Turkish parallel corpus is:

congratulazioni professor Battiston . ||| tebrik Ederim sayn Battlestar .
hotel vicino alla Truong Tien Bridge ||| Truong Tien Bridge yaknndaki oteller

Table 3.6: Caption

FastAlign will then generate asymmetric alignments (i.e. by treating either the left or the right sentences in the corpus as the source language, eventually generating slightly differnt aligments between the two modality). Therefore, it produces output in the format *i-j* where each *i-j* pair will indicate that the i^{th} word (zero-indexed) of the left language (which by convention is the *source* language) is aligned to the j^{th} word of the right sentences, assuming that as the target language. For the above example, the output is presented in Table 3.7.

FastAlign produces a zero-index enumerated alignment exploiting two kind of approaches: one considering the left sentence as source ($Forward$ alignment) and another by considering the right one instead ($Reverse$), therefore slightly different

[11] `https://docs.deeppavlov.ai/en/master/features/models/NER.html`
[12] `https://github.com/clab/fast_align/tree/master`

SRC	congratulazioni$^{(0)}$ professor$^{(1)}$ Battiston$^{(2)}$.$^{(3)}$
TRG	tebrik$^{(0)}$ ederim$^{(1)}$ sayn$^{(2)}$ Battlestar$^{(3)}$.$^{(4)}$
Forward al.	(0, 0), (0, 1), (2, 2), (2, 3), (3, 4)
Reverse al.	(0, 1), (1, 2), (2, 3), (3, 4)
Union al.	(0, 0), (0, 1), (1, 2), (2, 2), (2, 3), (3, 4)

SRC	hotel$^{(0)}$ vicino$^{(1)}$ al$^{(2)}$ Truong$^{(3)}$ Tien$^{(4)}$ Bridge$^{(5)}$
TRG	Truong$^{(0)}$ Tien$^{(1)}$ Bridge$^{(2)}$ yaknndaki$^{(3)}$ oteller$^{(4)}$
Forward al.	(0, 0), (1, 1), (3, 2), (4, 3), (5, 4)
Reverse al.	(0, 0), (1, 1), (2, 1), (3, 0), (4, 3), (5, 2))
Union al.	(0, 0), (1, 1), (2, 1), (3, 0), (3, 2), (4, 3), (5, 2), (5, 4)

Table 3.7: Source (SRC) and Target (TRG) examples with different alignments

alignments are generated, where *Union* alignments come handy representing a concatenation of the two. The sentences in 3.7 are extracted from the It-Tr corpus, thus they are a random sample of input data. Translations are not granted to be perfect since these are not made by any professional and that source-target sentences may not be equally long (4 vs 3 tokens in this example), thus, in this work we mostly take into account the *Union* aligments in order to have a more complete overview of the aligiments and reduce errors. Moreover, FastAlign model produces word alignments not really considering entities as a whole word, we therefore collapse multiple-words entities into a single one (to be de-collapsed back later) for a better matching. Finally we iterate trought alignments to extract the best match in case of overlaps, filtering unwanted overlaps and producing and extracting the best matches. This procedure will be further explained in the next Sections.

3.3.3 DNTs replacement

By collecting all the requirements, a specifically-made script has been realized to replace DNTs with given tags. The goal is to replace selected words with the tag "*${DNT0}X*" where *X* stands for an randomly chosen number on an adaptive range based on sentences length, not consecutive from a sentence-level point of view. The reason of chosing such a replacement hides behind the framework chosen to create the final Machine Translation model. In fact, it considers the above tag as a unique non-sub-tokenizable token, thus managing the related translation easily. Also, we chose to make the the DNTs enumeration not consecutive due to several tries leading to better results with randomly generated enumeration. We remind that DeepPavlov Named Entity Recognition model is effectively a transformer based architecture, meaning that even if the predictions are accurate enough, it may be prone to error, mostly for those non-rich language, while FastAligns relies on heuristics and may return overlapping or non-precise alignments, mostly for those very sintactically-different pair of languages in which entities don't necessarily respect the consequentiality of the sentence. Therefore, we leveraged a few ideas on top of both DeepPavlov and FastAligns results in order to provide the best result possible

by considering possible missleading inputs.

Ranking system

While parsing alignments, we often encountered cases in which we have several alignments to refer to, while not really being able to identify which one, or ones, are the best to pick. Consequently, we create a rank system to grade the alignmenta found based on sorting all the possible matches exploiting a specific string-distance metric. For each source entitiy, to be considered more robust since beloning to a high-resource language, we calculate this distance among the latter and the elegible alignments, and additionall to the target language entities in general. To do so, we exploited Levenshtein distance, which is a metric widely used in information theory, linguistics, and computer science, to measure the distance and thus the difference among different words. The Levenshtein distance, coined after the Soviet mathematician Vladimir Levenshtein in 1965, measures the minimum number of individual character-level operations (such as insertions, deletions, or substitutions) needed to transform one word into another. While it is commonly referred to as edit distance, this term can also encompass a broader range of distance metrics, collectively known as edit distance. The Levenshtein distance between two strings a, b (of length $|a|$ and $|b|$ respectively) is given by *lev(a, b)*, like show in :

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) & \text{if } a[0] = b[0], \\ 1 + \min \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{otherwise} \end{cases} \tag{3.2}$$

where the tail of some string x is a string of all but the first character of x, and $x[n]$ is the nth character of the string x, counting from 0[13].

This kind of approach is usually defined and utilized for comparisons of words belonging to the same language due to obvious limitations when it comes to measure distances among different sintactically and grammatically-ruled languages which are also made of various alphabeths. We take into account our target languages Turkish and Urdu: while the first makes use of a Latin alphabet[14], Urdu is an Indo-European language of the Indo-Aryan language group[15] which defacto makes impossibile to compare letters and thus words/entities starting from our source language Italian.

To overcome this limitation we exploit the python Unidecode library[16]. It takes Unicode data and aims to represent it in ASCII characters, by transliterating the latter. The quality of resulted ASCII representation vary depending on the alphabeth: western origin and Latin languages appear to be the best ones to transliterate while the further the given text is from Latin alphabet, the worse the transliteration

[13] `https://en.wikipedia.org/wiki/Levenshtein_distance`

[14] Turkish uses an adapted version of the Latin alphabet that includes six additional letters: *ç, ğ, ş, ğ, ö, ü*, for a total of 29 letters including 8 vowels, thus using Latin characters.

[15] A derivative of the Arabic alphabet consisting of 38 letters and no distinct letters

[16] `https://pypi.org/project/Unidecode/#description`

will be. Surprisingly, Urdu alphabeth (as well as genral Arabic one) is prone to be qualitatively transliterate for our goal. For example the Turkish version of the russian name *Vitalij Churkin* is *Vitali Çurkin* whose transliteration is *Vitali Curkini*, which provides two more matches with respect to Levenshtein distance. For Urdu this approach is way more evident due a transition from Arabic to Latin alphabet: a personal name such as "*Seamus Heaney*", originally "شیماس ہیذیا", becomes "*shyms hyny*", while the entity "*Britannico*", originally "برطانویا", now becomes "*brTnwy*" being now effectively comparable to the original words.

Replacement Algorithm

Having collected our tools and data we now propose the algorithm used to replace our data with "*${DNT0}X*" tags by considering multiple-words-entities as a single words producing easier matching between source and target language. The details of the implementation are shown in Algorithm 1.

Given a pair of sentences X_a, X_b, we parse them and produce updated version R_1, R_2 of the two. First, for each multi/single entity $s \in S$ we initialize a distance list D which will be iteratively populated. By parsing each entity pointed by alignment $A^{(i)} \in A$ we measure the distance between the source entity and the target one; if the distance is lower than a given η treshold we add the couple target entity t, distance l to D.

Additionally, eveytime DNTs are found and thus replaced in the parallel sentences, we will also include the original sentences to the training data in order to provide the model the context eventually lost by instering tags instead of the original words. Since FastAlign may provide misleading alignments we usually set η treshold to be quite low for single-word entities (3/4) with an increased margin for multiple-words entites (10/12) in order to filter out bad matches. Then, we repeat the process on all available entities in the target sentences so to make really close entities to eventually emerge. Once D has been populated, we sort the list by ascending distanaces and we pop out the first element representing the closest entity.

Finally, the entities belonging to a match are replaced with the DNT tag inside the updated version of orignal sentences R_1 and R_2 and eventually returned as the output of the script. Since this process involves inferences from upstream tools, for quality reasons the ranking process only allows a sub-set of the overall alignments and entities to be effectively accepted and compared, and it must be said that this is a sub-optimal output with respect of having a flawless mechanism. However, performances were shown to be quite adequate.

Algorithm 1 Replacement Algorithm

Input:

X - List of tagged sentence pairs

A - List of alignments

η - Levenshtein threshold

Output:

R - List of replaced sentence pairs

function LEVENSHTEIN(s_1, s_2)
    return Levenshtein distance between transliterated s_1 and s_2
end function
function SORTBYDISTANCE(D)
    return list D sorted by ascending distances
end function
$R \leftarrow \emptyset$ ▷ Initialize list for replaced sentence pairs
for each pair (X_a, X_b) in X **do**
    $R_1 \leftarrow X_a$ ▷ Copy of the first sentence
    $R_2 \leftarrow X_b$ ▷ Copy of the second sentence
    $S \leftarrow$ source entities from X_a
    $T \leftarrow$ target entities from X_b
    for each s in S **do**
        $D \leftarrow \emptyset$ ▷ List for entities and their distances
        for each alignment $A^{(i)}$ in A **do**
            $A_t^{(i)} \leftarrow$ entity aligned in $A^{(i)}$
            $\mathcal{L} \leftarrow$ LEVENSHTEIN($s, A_t^{(i)}$)
            if $\mathcal{L} < \eta$ **then**
                $D \leftarrow D \cup \{(A_t^{(i)}, \mathcal{L})\}$
            end if
        end for
        for each t in T **do**
            $\mathcal{L} \leftarrow$ LEVENSHTEIN(s, t)
            if $\mathcal{L} < \eta$ **then**
                $D \leftarrow D \cup \{(t, \mathcal{L})\}$
            end if
        end for
        $D \leftarrow$ SORTBYDISTANCE(D)
        $(d_{closest}, \_) \leftarrow$ first element from D
        Replace appropriate entity in R_1 and R_2 with $d_{closest}$
    end for
    $R \leftarrow R \cup \{(R_1, R_2)\}$ ▷ Add replaced pair to list
end for
return R =0

3.4 Model Training

Once the dataset is populated by DNTs tags, a Machine Translation engine can be created and eventually fine-tuned, as shwon in Figure 3.4.

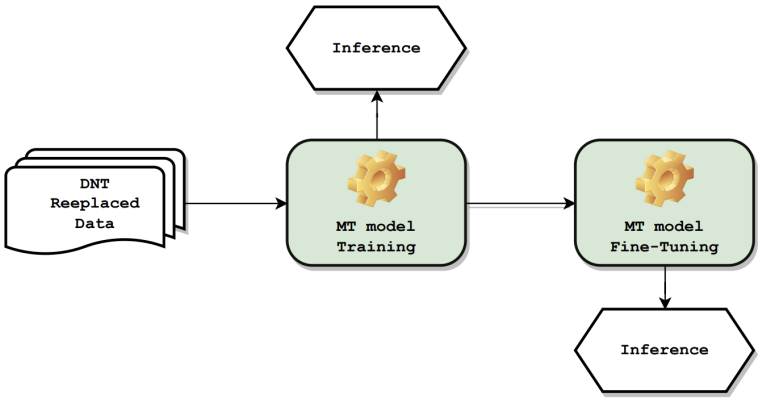


Figure 3.4: MT Engine Pipeline

The engines have to produce accurate translations with recpect to a given domain and, more importantly, to return DNTs tags during the inference, so that we can retrieve original entities back and substitute them. For this task we have used a specific MT framework named ModernMT[17]. It is is made on top of current SOTA architectures for Neural Adaptive Machine Translation. The framework is based on Fairseq Transfomer model[18], a sequence modeling toolkit written in PyTorch for research purposes. It provides a versatile solution for machine translation domain being designed to be scalable, customizable and fast to use. This framework provides some already-optimized model architectures to laverage for Machine Translation processes, along with different Transformers models of different sizes with reference to the parameters of the neural networks. For our experiments, we opted to use the *transformer_mmt_base* model as it represents a good tradeoff between performance and speed; the structure of the model, in terms of layers and parameter settings can e found in Appendix A. We then first create our model, which is easily done via Linux's Command Line Interface (CLI):

```
./mmt create ${source_lan} ${target_lan} ${data_folder}/ \
    --engine ${engine_name} \
    --arch transformer_mmt_base \
    --max-tokens 2500 \
    --update-freq 8 \
    --lr 0.0005 \
    --warmup-updates 8000 \
    --train-steps 250000 \
    --tensorboard-port 8067 \
    --no-test --debug

```

Listing 3.1: Creating ModernMT MT model

[17] https://github.com/modernmt/modernmt
[18] https://fairseq.readthedocs.io/en/latest/index.html

The parameters of the model initialization can be described as follows:

- **engine:** is the name given to the engine; with the same name, two subdirectories will be created, one working under the runtime/ dir and one with the final engine under engines/
- **arch:** this option is used to specify one of the architectures known to the package; the most relevant for the company are the three *transformer_mmt_x* where x stands for big, base and tiny corresponding to models similar in topology but of decreasing size (210, 61 and 38 million parameters).
- **max-tokens:** is the maximum number of tokens (minimum processing units) in a batch
- **update-freq:** parameters are updated by accumulating the counters for the number of batches specified with this option (default: 1). It is used to simulate larger batches when the hardware (RAM of the GPU) does not allow its direct use
- **lr:** learning rate (maximum)
- **warmup-updates:** is the number of updates by which the learning rate is increased linearly from 0 to the maximum value specified with the lr option. After that, the value decays according to the inverse of the square root at each iteration. This is the default trend imposed by ModernMT (inverse_sqrt). Other functions are available in fairseq- train via the lr-scheduler option.
- **train-steps:** is the number of training iterations. If it is not specified, there is an early-stopping procedure to decide for itself when to end the training.
- **tensorboard-port:** is the port on which the tensorboard makes available information on the progress of the training.
- **no-test:** avoids the extraction from the training data of a test set for internal use
- **debug:** avoids the final removal of all temporary files created during training. It can be very useful for control purposes and is necessary if you want to be able to continue training or perform a subsequent adaptation, bearing in mind that the space occupied on disk will be significant (order of hundreds of gigs depending on the model)

This script will run few operations that ModernMT performs before starting the training itself:

- **Cleaning:** Additional internal cleaning step is performed with respect to training parallel data.

- **BPE Tokenization:** Subword-based tokenization algorithm (BPE) is the perfect solution between word and character-based tokenization, specifically introduced for Neural Machine Translation. The main idea is to solve the issues faced by word-based tokenization (very large vocabulary size, large number of OOV tokens, and different meaning of very similar words) and character-based tokenization (very long sequences and less meaningful individual tokens). Firstly presented by Philip Gage in 1994 [24], it is a simple form of data compression algorithm in which the most common pair of consecutive bytes of data is replaced with a byte that does not occur in that data, defacto overcoming the aforementioned problems of OOV and particularrly big vocabularies.
- **Vocabulary binarization:** Unlike hand-made model the vocabulary is here automatically created and binarized. As previously stated, ModernMT includes a set of static tokens to their models, adding them up to the originate one.

Aside from the network-level hyperparameters, those listed above directly override those already set a priori by ModernMT. An interesting work on the impact that these hyper-parameters have on the final quality of transformer models can be found in the work of Martin Popel, Ondej Bojar [65] where they are extensively analysed and contextualised, providing valuable suggestions on which to base the creation of the transformer engines used in this work.

ModernMt is a very high-level framewowrk, and it allows rapid scalability in the creation of further models.

The only difference made in the creation of the Urdu model actually lies in the number of *training-steps* (training-steps:= 250k), having previously observed with Turkish that 300k steps were not strictly necessary. In fact, it must be considered that the creation (training) of such a MT model, in the order of tens of billions of parameters, would require more than seven days of training with our resources, and that launching several experiments per language would take too long; lowering the maximum number of training-steps accordingly saves useful time.

3.4.1 Fine-Tuning

Fine-Tuning is a process in which certain representations of the pre-trained model are slightly adjusted to make it more relevant to the problem in question. This avoids having to define the structure of the neural network and train it from scratch, basically using the weights of an already trained network as the starting values for training a new network. ModernMT does not expose a direct procedure for adapting a pre-trained model. However, it is possible to do so by using the option to continue an interrupted or terminated training provided it has been run by specifying the *–debug* option that inhibits the removal of temporary files. In this case, re-launching the created ./mmt with the same options plus the *–resume* and a greater number of iterations, allows the training to resume from where it had arrived and continue it until the new total number of iterations is reached. This mechanism can also be exploited for fine-tuning by replacing the original training data with the adaptation data, as described below. Once the original generic model is therefore saved, data binarized and the original vocabulary provided, we can now start fine-tuing the model on adapting data. In 3.2 a representative example.

```
./mmt ${source_lan} ${target_lan} ${data_folder}/ \
    -e ${engine_name} \
    --arch transformer_mmt_base \
    --max-tokens 2500 \
    --update-freq 8 \
    --lr 0.0001 \
    --warmup-updates 8000 \
    --train-steps 254000 \
    --tensorboard-port 8066 \
    --no-test --debug \
    --resume --save-interval-updates 200

```

Listing 3.2: Model Fine-Tuning

Compared to the engine creation command, note that:

- the learning rate (maximum) has been lowered by a factor of 5
- the number of iterations has been increased by 4000; assuming 250 thousand to be the previous model ending steps, specifying the value 4000 it means to continue training for another 4000 iterations, resulting in a final model trained for 254 thousand steps.
- **resume:** this option is used to resume training that has been interrupted or terminated.
- **save-interval-updates:** this option specifies the interval (in number of iterations) between one model save and the next; compared to the value used in the original creation (the default 1000) here the saves are thicker; remember that only the 10 most recent models are kept.

This was only a snapshot of the fine-tuning process, hence we are going to produce multiple adapted models based on diffrent itearion counts $X = \{1000,2000,3000,4000\}$. Once the adaptation is concluded we are provided alternatives for both generic and adapted domain, being ready to evaluate the models.

3.5 Evaluation

Before proceeding to evaluate our models we first have to produce the translation. For this task we take advantage of an API call for ModernMT services to provide us the translation, which is supposed to return our DNTs during its inference. Finally, we can evaluate the translation produced by our models by means of different metrics.

3.5.1 Bleu Score

BLEU, which stands for Bilingual Evaluation Understudy, is an algorithm used to assess the quality of machine-translated text from one language to another. It measures the similarity between machine-generated translations and those produced by humans, with the premise that the closer a machine translation aligns with a professionally translated human text, the better its quality is considered. BLEU

was among the pioneering metrics to establish a strong correlation with human evaluations of translation quality also being cost-effective, and automated evaluation method. First introduced in [60], it is still a widely adopted metric for MT.

The MT output is scored by the weighted average of the number of N-gram overlaps with the human translations. BLEU is a modified precision metric and it computes the N-gram matches for each candidate sentence C and sums the clipped counts over all the candidate sentences:

$$p_n = \frac{\sum_{C \in \{ \text{ Candidates } \}} \sum_{\text{n-gram } \in C} \text{ Count }_{\text{clip}} (\text{ n-gram })}{\sum_{C' \in \{ \text{ Candidates } \}} \sum_{n\text{-gram } ' \in C'} \text{ Count } (\text{ n-gram } ')} \tag{3.3}$$

where n is the length of the n-grams, generally being unigrams, bigrams, trigrams and often quadrigrams. Futhermore, the geometric average of the modified n-gram precisions, p_n is computed using n-grams up to length N and positive weights w_n summing to one. Then, let c be the total length of the candidate translation corpus and r be the effective reference length obtained by summing, for each candidate sentence, the length of the best-matching reference sentence, Brevity Penality is computed as:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{3.4}$$

And then:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n \right) \tag{3.5}$$

While this metric is widely used, it must be taken into account that it comes with several well-documented limitations. Key limitations refer to a lack of Semantic Understanding since BLEU primarily relies on n-gram overlap between the reference and candidate translations and it doesn't take into account the semantic meaning or coherence of the translations. There are evident length biases since BLEU tends to favor shorter translations because shorter sentences are more likely to have overlapping n-grams with the reference translations. Also, a lack of Cross-Dataset and Cross-Language Comparability due to score variability with respect to different datasets and languages must be noted. This makes it challenging to compare translation systems across different languages. Moreover BLEU is very sensitive to the parameter choiches since its computation involves several parameters, such as the choice of n-grams and their weights and small changes in these parameters can lead to significantly different BLEU scores. Although very important, these topics are out-of-scope for this thesis, but still relevant to know, also referring to what has been said in 2.3.4.

3.5.2 F-Scores

In addition to BLEU, and in order to quantitatively evaluate results with respect to DNTs translation, we also adopt more general Machine Learning F-Metrics to understand the behaviour of our translation models. We create a script to compare

DNTs tags between a reference and a hypotesis file, line by line, treating the reference DNTs as the *ground truth* and the hypotesis DNTs as the *predicted values*.

Precision (P)

measures the accuracy of positive predictions, i.e., the proportion of correctly predicted positive instances out of all instances predicted as positive. It is defined as:

$$P = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Where, in our script:

- TP (True Positives) is the number of instances correctly predicted as positive, considered as matching DNTs
- FP (False Positives) is the number of instances incorrectly predicted as positive, that is DNTs in hypothesis that are not in the reference = DNTs in hyp - Matching DNTs

So, the calulation is:

$$Precision = \frac{\text{Matching DNTs}}{\text{DNTs in hyp}} \tag{3.6}$$

Recall (R)

also known as Sensitivity or True Positive Rate, measures the ability of the classifier to identify all positive instances. It is defined as:

$$R = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Where:

- TP (True Positives) refers again to Matching DNTs
- FN (False Negatives) is the number of instances incorrectly predicted as negative when they were actually positive, namely DNTs in reference that are not in the hypothesis = DNTs in ref - Matching DNTs

Therefore, the calculation is:

$$Recall = \frac{\text{Matching DNTs}}{\text{DNTs in ref}} \tag{3.7}$$

F-score (F1-score)

is a harmonic mean of precision and recall. It combines both metrics into a single score, providing a balanced assessment of the model performance.

It is defined as:

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.8}$$

The F1-score is especially useful when you want to consider both false positives and false negatives, as it seeks a balance between precision and recall. The F1-score ranges between 0 (worst) and 1 (best), with higher values indicating better performance.

Summary

This chapter outlined the methodology employed to achieve our results to be shown in the following section of this thesis. In 3.1 it is initially provided an overview of the sources from which data has been gathered. Then, the chapter delved into specific details on data processing and manipulation in 3.2, followed by a panning in 3.3 on the tools exploited to create and link entities as well as the algorith with whom the corpora have been augmented. Subsequently, in 3.4 the detalis of model training and fine-tuning has been provided and finally a presentation of the adopted metrics to evaluate the quality of a Machine Trasnlation model is presentend in 3.5, coupled with the metrics exploited for DNTs assestment. In addition, the aim of this work is to maximise the recall values in order to filter out, among the trained and adapted models, those that manage to correctly translate the DNTs' tags, regardless of any hallucinations then recorded by precision. In order to copute these metrics, a script is created that ignores the order of the DNTs within the translations for obvious linguistic reasons (different language pairs may present a different syntactic order).

Chapter 4

Experiments and Results

In this chapter we are going to explore the experiments done on both languages exploiting the process presented in Chapter 3 with respect to engine creation and adaptation. Specifically, we will report the baseline models developed by PerVoice's IT team and available at the beginning of the internship and will provide results of newly made ones. We will distinguish between two kinds of evaluation operated on our data: BLEU score for translation quality evaluation and general Machine Learning F-score to bring out our models behaviours with respect of DNTs handling during inference mode.

4.1 Experiments

In order to have models suitable for the translation of DNTs, in addition to the creation of new engines specifically made from scratch with the help of datasets containing DNTs themselves, we will also consider a pair of models already trained by the company before the internship beginning. As a matter of fact, two models, one for Turkish and one for Urdu were already available, and are in fact being used as a baseline for the purpose of comparison with better performing models and we name these models `GENERIC.337M` and `GENERIC.150M`.

Model	Language	Type	Max Tokens	Learning Rate	Warmup Updates
GENERIC.337M	TR	Transf. Base	2500	0.0005	8000
GENERIC.150M	UR	Transf. Base	2500	0.0005	8000

Table 4.1: Baseline GENERIC models for Turkish and Urdu

As stated in 3.4, these configurations, including max-tokens, learning rates and warmup-updates values, have been extrapolated from [65] and fixed for comparability purposes. The only real difference, based on the data used for trainining, is indeed the quantity of the input words, precisely 337M for Turkish and 150M for Urdu, a substantial difference arising from treating respectively medium and low-resources languages. These models are not able to translate DNTs tags since there's

no trace yet of the latter in the training data, thus reporting not considerable values with respect to precision and recall. We remind that DNT tags are based on a prefix "${DNT0}" and a suffix "$x$" where x stands for a variable number used for enumeration purpose. While these models eventually recognize a limited quantity of numbers coming from the enumerative part of the DNT tag, errors at inference time are rather frequent, especially for low-resource languages, such as Urdu, where arabic numbers are not even adopted in the relative alphabeth. Thus, a corpus devoid of those tags leads to a model that trudges to deal with them. Therefore, the same models have been adapted by means of an adaptation set directly extracted from data provided by the customer, which has the double function of introducing the DNTs, as well as provide the adjustment towards the specific legal domain of customer documents, leading to more robus translation in terms of BLEU score and DNTs handling. To create this model it has been chosen a specific combination of learning rates and adapting steps due to a limited quantity of adatpation data. Thus, the latter consists in a quite small corpus of 2000 parallel sentences for source and target extracted from documents provided by the customer itself. A similar amount of adaptation data expects the fine-tune taks not to exceede in terms of lr and ft paramentes: larger learning rates would result in overfitting[1], as well as lower rates would lead to non-existent improvements on BLEU scores; Same concept may be applied to Fine-Tuning steps: these go hand in hand with the amount of data provided to the model where, considering batches[2] imposed by ModernMT framework, higher steps would again lead to a overfitting situation where the same sentences a repeatedly seen and trained on, while a right amount would only let the adaptation data to be seen few epochs[3]. Due to these considerations, the Learning Rates values and Fine-Tuning steps are set to $lr = \{5 \times 10^{-5}, 10^{-5}, 5 \times 10^{-6}, 10^{-6}\}$.

Model	**Language**	**Learning Rate**	**Fine-Tuning Steps**
PDF_Baseline	TR	0.00005	2.000
PDF_Baseline	UR	0.00005	2.000

Table 4.2: Baseline PDF models for Turkish and Urdu

These models were provided by the IT team of PerVoice which was working on this project and are reported in table 4.2 and whose scores are reported in the following sections. Aside the provided specifications, we have no information regarding other experiment conducted on those with respect to learning rates and training steps. This is due to the fact that we were only provided of the best performing models obtained from prior evaluation. Thus, we can assume that other tested model performed worse in terms of DNTs. For this reason we chose to create new

[1] A machine learning model may adapt way too much on training set characteristics which are not reflected in the typical distribution of the rest of the cases, leading to a bad generalization capability.

[2] The number of samples to work through before updating the internal model parameters.

[3] The number times that the learning algorithm will work through the entire training dataset.

adapted engines to both better recognize DNTs and increase as much as possible the BLEU score on the PDF domain, still prioritizing Recall.

4.1.1 Turkish

For this language a newly made GENERIC models has been created by means of the process described in chapter 3. Two generic models have been generated: 317M and 378M; the difference betweeen them lies in the amount of DNTs found inside the input training data. In the first case there will be less DNTs. As stated in chapter 3.3.3, eveytime DNTs are found and thus replaced in the parallel sentences, the original sentence is also included in the training data in order to provide the model the context possibily lost by instering tags instead of the original words. The different amount of DNTs are given by the fact that `GENERIC.317M` was an early prototype model including DNTs. Thus, we chose not to exceed in replacements in order to evaluate the performances. We therefore insert a probability of 50% for each DNTs to be found and so replaced in the original sentence, leading to a smaller amount of DNTs being insterted and, as shown next, poorer performances. Differently, the 378M model were not applying any further restriction. This lead to it being a more robust model.

Model	DNTs	BLEU Flores	BLEU Test Set	PRECISION Test Set	RECALL Test Set
GENERIC_Baseline	0	19.3	13.4	0	0
GENERIC.317M	3.6M	19.1	15.0	1.0	0.987
GENERIC.378M	6.5M	**19.5**	**15.3**	1.0	0.99

Table 4.3: Generic Turkish Models

Table 4.3 reports preliminar results for those models. First, we observe that for both Flores set and Test set the BLEU score register an increase; this may be due to the insertion of a double copy of the same sentence in the training set. In fact, it is possible that when single or multiple DNTs are found in the parallel representation of the same sentence, the latter may potentially be qualitatively better than those discarded, due to the restriction applied in the methodology to find proper matching entities. Thus both the newly made generic models are scoring better results in terms of translation quality. Additionaly, the baseline model is not capable of translating DNTs, while our models report optimal results: in both cases the best possibile Precision score is achieved, suggesting that all the translated tags are not repeated, while Recall gets close to 100%. Altough these are only general models they can perfectly handle DNTs, bringing us to concentrate on creating adapted models to improve the translation quality while mantaining a the possibile DNTs translation capability.

Adapting Results

Here we report the results obtained by adapting the model to the domain-specific corpus extracted from the document provided by the customer.

Model	Learning Rate	BLEU Flores	BLEU Test set	Precision Test set	Recall Test set	F1 Score Test set
PDF_Base		17.0	19.1	0.993	0.97	0.9814
317M.FT1K	5×10^{-5}	18.4	18.9	1.0	0.987	0.9934
	10^{-5}	19.0	17.6	0.987	0.997	0.9920
	5×10^{-6}	19.0	14.7	0.99	0.997	0.9935
	10^{-6}	19.2	14.6	1.0	0.987	0.9934
317M.FT2K	5×10^{-5}	16.8	18.8	0.997	1.0	0.9985
	10^{-5}	18.9	18.6	0.993	0.99	0.9915
	5×10^{-6}	19.0	18.3	0.99	0.993	0.9915
	10^{-6}	19.2	14.2	0.997	0.987	0.9920
317M.FT3K	5×10^{-5}	15.5	18.8	1.0	0.993	0.9965
	10^{-5}	18.3	18.9	1.0	0.987	0.9934
	5×10^{-6}	18.9	18.4	0.993	0.993	0.9930
	10^{-6}	19.1	13.8	0.987	0.993	0.9900
317M.FT4K	5×10^{-5}	14.0	18.7	0.987	0.993	0.9900
	10^{-5}	18.0	18.6	1.0	0.993	0.9965
	5×10^{-6}	18.9	18.6	0.997	0.99	0.9935
	10^{-6}	19.0	14.6	0.983	0.997	0.9900
378M.FT1K	5×10^{-5}	18.6	**19.2**	1.0	0.987	0.9934
	10^{-5}	19.4	18.1	1.0	0.993	0.9965
	5×10^{-6}	19.6	18.1	1.0	0.99	0.9950
	10^{-6}	**19.8**	14.8	1.0	0.987	0.9934
378M.FT2K	5×10^{-5}	17.4	19.0	1.0	0.997	0.9985
	10^{-5}	19.0	18.6	1.0	0.99	0.9950
	5×10^{-6}	19.3	18.6	1.0	0.993	0.9965
	10^{-6}	19.5	16.9	**1.0**	**1.0**	1.0
378M.FT3K	5×10^{-5}	16.7	18.8	**1.0**	**1.0**	1.0
	10^{-5}	18.5	18.9	1.0	0.99	0.9950
	5×10^{-6}	19.3	18.4	0.997	0.997	0.9970
	10^{-6}	19.6	17.2	1.0	0.997	0.9985
378M.FT4K	5×10^{-5}	15.5	15.5	1.0	0.99	0.9950
	10^{-5}	18.2	18.8	**1.0**	**1.0**	1.0
	5×10^{-6}	19.0	18.7	1.0	0.997	0.9985
	10^{-6}	19.4	18.0	1.0	0.983	0.9915

Table 4.4: TURKISH Fine-Tuning Result

PerVoice team had already created an adapted baseline engine, namely `PDF_Base` based on the relative Generic baseline model in order to increase generalization capability on specific legal domain and expand the generic version capability to handle DNTs; while this adapted model is performing very good in terms of BLEU for the Test set, there is a margin of improvement for Precision and Recall on DNTs. Thus, to create more robust adpted engines we fine tune our Generic models (see Table 4.3). We evaluate different learning rates and number of steps for fine-tuning. Table 4.4 reports the results. By considering BLEU scores, we assist to a controversial trend: BLEU scores on Flores set perform almost always better than the adapted baseline; this is surely due to having a more robust training set, independently of the adaptation set, leading to a better natural language capability. On the Test set the behavior is the opposite: only few of the tested configurations

come close to the score reported by the adapted baseline engine. This appears to be a language-dependent behaviour, as results on Urdu (see Sec. 4.1.2) follow a different trend. Nevertheless, note that the main goal of this work is to properly handle DNTs.

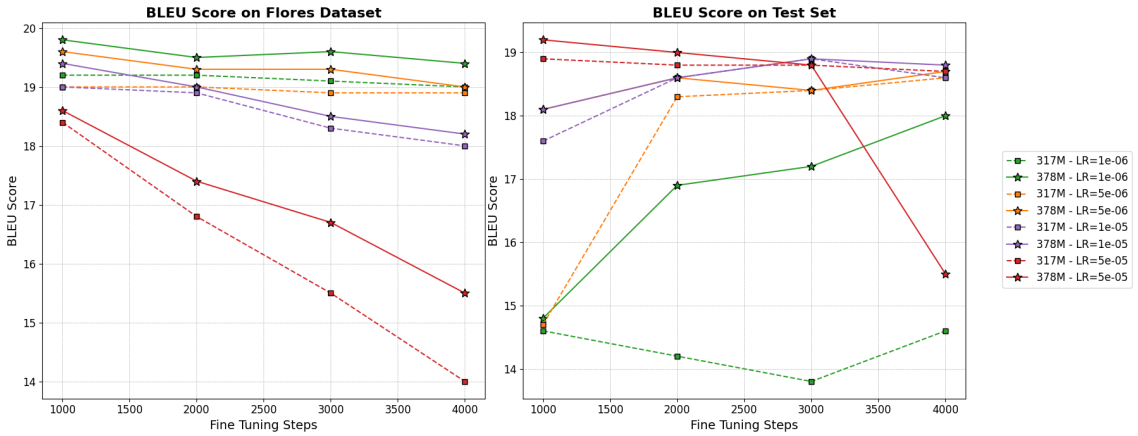


Figure 4.1: Turkish (a) BLEU Scores Flores (b) BLEU Scores Test set

We can also consider how learning rates and fine tuning steps actually affect the performances of the final models. Figure 4.1 (a) shows BLEU scores for Flores set for increasing fine-tuning steps. Generally speaking, the more the steps, the biggest the loss in terms of translation quality for natural generic domain. This suggests that the adaptation set is not contributing much in this case. Moreover, If we consider different learning rates it can be observed that for higher learning rates the adaptation is more "aggressive", probably leading to overfitting: by looking at the red lines, representing the highest LR configuration (5×10^{-5}) we can indeed observe that the translation quality follows a continuous and rigid drops. We can speculate that over-adaptation with such high learning rate doesn't really produce any benefit. Results on the Test Set (Fig. 4.1(b)) are less straightforward to interpret. In this case, neither LR nor training steps alone appear to affect BLEU results. Instead, we see more interactions between them on the final result:: a medium learning rate in a scenario in which lot more FT steps are given, may eventually lead to better translation quality. We can speculate that, for adaptation purposes, it may be better for the model to iterate more over the adaptation set and let it learn more slowly from it.

4.1.2 Urdu

GENERIC engines for Urdu were created following the same procedure as for Turkish. A generic 182M-word model and then a 194M-word model are produced, again taking into account the word count of the Italian text. In this case, the difference between the two also lies in the quantity of DNTs within the corpora. This results from the use of a different threshold in applying the Levenshtein distance (see Chapter 3.3.3), with a more flexible margin in the 194M model in order to test the behaviour of the engine with a greater amount of matching between the two parallel corpora, and consequently a greater number of DNTs replacement in the training set.

Model	DNTs	BLEU Flores	BLEU Test Set	PRECISION Test Set	RECALL Test Set
GENERIC_Baseline	0	13.7	17.3	0	0
GENERIC.182M	1.3M	13.9	**18.6**	1.0	0.997
GENERIC.194M	1.7M	**14.0**	17.0	1.0	0.997

Table 4.5: Generic Urdu Models

As can be seen from the Table 4.5, having filled the training data with DNTs tags allows the two engines shown here to obtain optimal results in the Precision and Recall metrics. In fact, almost all DNTs are correctly translated from the Italian language into Urdu, even considering the fact, as already pointed out, that the Urdu language does not use numbers in its alphabet, let alone words in Latin characters like our starting tag, belonging to an Arabic-type alphabet. The BLEU score with respect to the generic-domain text belonging to the Flores corpus, tends to improve as the DNT increases. This do not happen for the customer's Test Set. In order to create a more robust model in the specific domain of the documents assigned to us by the customer itself, several adapted models were created, again at varying learning rate values and training steps, thus emulating the adapted models of Turkish.

Adapting Results

This section discusses the results of the two adapted models in relation to the different variants realised. Again, the PDF_Base model represent the one provided by the company as a baseline, shown in the table 4.6. Specifically, we are dealing with a model that exceeds the performance of its Generic version in terms of BLUE score on both Flores and Testset, but once again presents inaccuracies in the recognition of the special DNTs tags, requiring the creation of a more robust engine for the translation of the DNTs themselves. The two generic models presented above are therefore adapted to improve the already good performance obtained by the `PDF_Base` model.

Model	Learning Rate	BLEU Flores	BLEU Test set	Precision Test set	Recall Test set	F1 Score Test set
PDF_Base		13.0	18.4	1.0	0.943	0.9691
182M.FT1K	5×10^{-5}	14.4	20.6	1.0	0.997	0.9985
	10^{-5}	13.9	19.0	1.0	0.997	0.9985
	5×10^{-6}	13.5	18.7	1.0	0.993	0.9965
	10^{-6}	13.2	18.8	1.0	0.993	0.9965
182M.FT2K	5×10^{-5}	**15.0**	20.8	0.997	0.997	0.997
	10^{-5}	14.6	19.2	1.0	0.997	0.9985
	5×10^{-6}	14.4	20.6	**1.0**	**1.0**	1.0
	10^{-6}	14.2	20.5	**1.0**	**1.0**	1.0
182M.FT3K	5×10^{-5}	14.7	**21.5**	0.997	1.0	0.9985
	10^{-5}	14.9	18.8	1.0	0.997	0.9985
	5×10^{-6}	14.7	19.0	1.0	0.993	0.9965
	10^{-6}	14.6	19.8	**1.0**	**1.0**	1.0
182M.FT4K	5×10^{-5}	14.2	19.6	0.997	1.0	0.9985
	10^{-5}	14.5	20.2	0.993	1.0	0.9965
	5×10^{-6}	14.5	20.5	0.987	1.0	0.9935
	10^{-6}	14.6	20.8	0.983	1.0	0.9914
194M.FT1K	5×10^{-5}	14.4	20.1	1.0	0.987	0.9935
	10^{-5}	13.9	18.6	0.997	1.0	0.9985
	5×10^{-6}	13.5	18.1	1.0	0.997	0.9985
	10^{-6}	13.3	18.0	1.0	0.993	0.9965
194M.FT2K	5×10^{-5}	14.9	20.9	0.993	0.997	0.9950
	10^{-5}	14.9	19.6	1.0	0.99	0.9950
	5×10^{-6}	14.5	19.9	1.0	0.99	0.9950
	10^{-6}	14.4	19.8	1.0	0.987	0.9935
194M.FT3K	5×10^{-5}	14.7	20.1	**1.0**	**1.0**	1.0
	10^{-5}	14.8	19.0	0.997	1.0	0.9985
	5×10^{-6}	**15.0**	18.9	0.997	0.997	0.997
	10^{-6}	14.9	19.4	1.0	0.993	0.9965
194M.FT4K	5×10^{-5}	14.3	17.2	1.0	0.993	0.9965
	10^{-5}	14.5	18.0	1.0	0.997	0.9985
	5×10^{-6}	14.7	19.0	1.0	0.997	0.9985
	10^{-6}	14.7	19.4	**1.0**	**1.0**	1.0

Table 4.6: URDU Fine-Tuning Results

Table 4.6 reports the scores obtained by fine-tuning the engines on the adaptation data. Starting from the baseline represented by the model `PDF_Base` we witness increments with respect of the BLEU scores on both datasets. On the one hand the engines capability to produce more natural translation on generic domain is increased. On the other hand the engine is able to better produce domanin-specific translation with respect to customer's documents.

Figure 4.2 shows the trends of the 182M and 194M engines with respect of BLEU scores considering Learning Rates and training steps. BLEU Scores on Flores dataset (a) report higher values increasing learning steps by at least 2000, meaning that the adaptation set adds information to the generic version of the model, resuting in more robust translations; adding up more training steps to the models do not

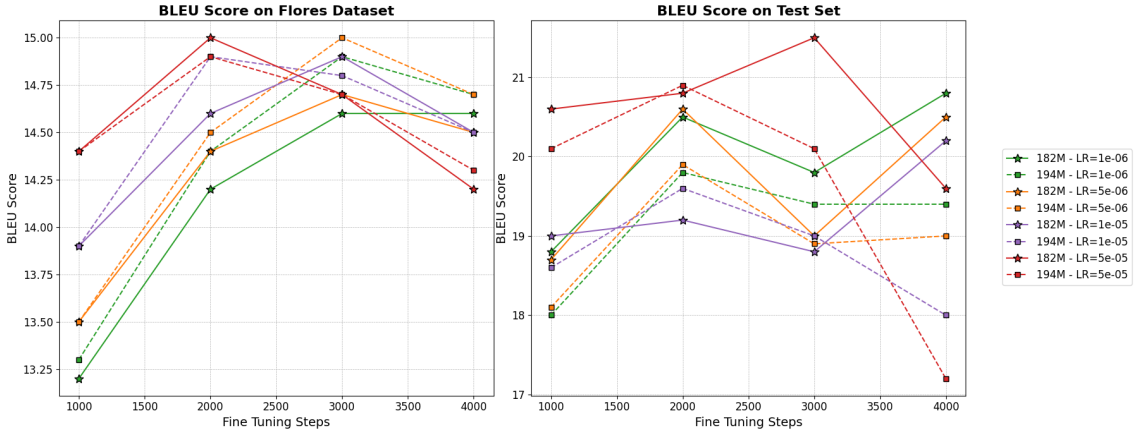
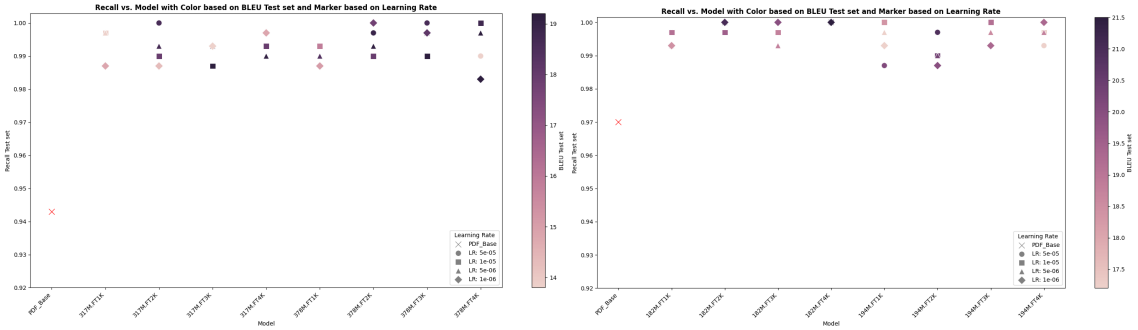


Figure 4.2: Urdu (a) BLEU Scores Flores (b) BLEU Scores Test set

increase performances, probably due to overfitting. Additionaly, we se that the highest learning rate (5×10^{-5}) displays the same overfitting behaviour. For BLEU Scores on Test Set (b) instead we have no linear correlation between fine-tuning steps and overall scores, showing a noisy trend with respect to increasingly higher fine-tuning steps, we register a peak value of 21.5 scored by the `182M.FT3K` engine. Additionally, we can note that the 182M appears to perform best in a variety of conditions. In fact, across different fine-tuning steps, it consistently outperform the 192M model. This leads to reflect on the fact that there is not necessarily a positive correleation between DNTs count and performances with respect to BLEU scores.

4.1.3 Performances on DNTs

This section relates directly to the results shown in Sections 4.1.1 and 4.1.2 in relation to the F-score values achieved. Recall that the aim of this work, along with generating qualitatively valid translations, prioritises generating engines that respond as well as possible to the handling of DNT, specifically by generating high values of Recall, a metric useful for assessing how many DNT are correctly translated.



(a) Turkish **(b)** Urdu

Figure 4.3: Recall by model and LR, with BLEU score

Figure 4.3 displays Recall performances by model and Learning Rate, including the adapted baseline model for comparison. We can make two considerations with

respect to the Figures. First, all models outperform the `PDF_Baseline` models by a relatively wide margin, highlighting their robustness to DNT management. Second, no clear trends emerge with respect to the Learning Rate, further indicating that there are other relevant aspects (e.g., number of training steps) that also affect the final outcome of the model. Therefore, we can state that they ultimately succeed in solving the task on which this work is based.

4.2 Reproducibility on other langauges

The work developed in this thesis is based on the creation of models capable of ensuring an adequate translation quality from a source language to a target language, but above all capable of managing the words that should not be translated, namely DNTs. Turkish and Urdu were used as a proof of concept for this approach since the former is part of a family of Latin alphabets, while the latter is part of Arabic and Oriental alphabets. The Latin alphabet is used in many European languages, including English, French, German, Spanish, Italian, as well as in the Americas and other parts of the world, totalling over 100 languages worldwide. On the other hand, the Arabic alphabet is mainly used in Arabic languages, which include Standard Arabic, as well as many regional variants and dialects used mainly in the Middle East and North Africa, counting a total of more than 30 languages belonging to this alphabet[4]. Therefore, working with this pair of languages made it possible to create a methodology applicable on a large number of languages. The work carried out during the internship focused only on the previously mentioned languages, leaving further scalability for future work to be carried out in the future. Nevertheless, a digression on this subject was made in terms of the reproducibility of the process on other languages. For logistical and computational reasons it was impossible to fully address this issue. However, a preliminarly analysis was carried out on the question of scalability towards other languages. Specifically, we considered the performance of DeepPavlov in recognising tags in languages with limited resources. In fact, let us recall that the entire DNT replacement system is based on searching for matching regarding entities in pairs of languages and then applying a replacement algorithm, making the NER system fundamental to the success of the work.

For this purpose, a number of Latin and Arabic languages were chosen, which differed from each other although they belonged to the same family of alphabets, in addition to a limited number of other languages belonging to alphabets of a different nature. For each of them, a sample of 1,000 lines was extracted from the corpus available from Opus, and then some statistics were calculated.

Table 4.7 shows a total of 19 different languages, including Turkish and Urdu, belonging to different alphabets.

The columns "*Tags*" and "*F. Tags*" differ by considering in the first case all the available tags in DeepPavlov, while in the second case only those referred to the table 3.5. With "*Ratio*" we intend instead to normalise the difference between the tags found in the source language and those in the target language, where a value of 1 indicates a perfect matching while a value tending towards zero indicates that

[4] `https://it.wikipedia.org/wiki/Alfabeto_arabo`

Table 4.7: Language and Alphabet Data

Language	Alphabet	ISO	SRC Tags	TRG Tags	Ratio	SRC F.Tag	TRG F.Tag	Ratio F.	Corpus	Len Corpus	DNTs
Português	Latin	**PT**	6480	6853	0.946	3239	3696	0.876	wikimedia	1000	1576
Russian	Cyrillic	**RU**	6027	6801	0.886	3198	3355	0.953	wikimedia	1000	1012
llinika	Greek	**EL**	3004	3147	0.955	1924	2042	0.942	GlobalVoices	1000	918
Deutsch	Latin	**DE**	2954	2926	0.991	1610	1623	0.992	WMT-News	1000	632
Turkish	Latin	**TR**	2180	2226	0.979	1042	1091	0.955	Generic	1000	418
Polish	Latin	**PL**	1362	1264	0.928	618	546	0.883	TED2020	1000	286
Bosniaco	Latin	**BS**	1145	1190	0.962	492	533	0.923	TED2020	1000	255
Albanese	Latin	**SQ**	1033	1033	1	482	443	0.919	TED2020	1000	218
Chewa	Latin	**NY**	4151	3903	0.94	480	620	0.774	MultiCCAligned	1000	193
Arabic	Arabic	**AR**	1362	1118	0.821	618	444	0.718	TED2020	1000	166
Zulu	Latin	**ZU**	3040	2713	0.892	437	516	0.847	MultiCCAligned	1000	125
Farsi	Arabic	**FA**	1091	1001	0.918	469	332	0.708	TED2020	1000	114
Kartuli	Georgians	**KA**	737	568	0.771	335	228	0.681	TED2020	1000	101
Urdu	Arabic	**UR**	1036	836	0,807	271	244	0,899	Generic	1000	88
Hausa	Latin	**HA**	190	324	0.586	98	324	0.302	Tanzil	1000	24
Kurdí	Arabic	**KU**	847	78	0.092	361	64	0.177	TED2020	1000	19
Bangla	Bengali	**BN**	999	33	0.033	503	1	0.002	TED2021	1000	0
Nepl	Devanagari	**NE**	3255	501	0.154	1882	22	0.012	GlobalVoices	1000	0
Tigrinya	Tigrinya alphabet	**TI**	185	5	0.027	62	0	0	NLLB	1000	0

there are many more tags in one language than in another. The number of DNTs found for each languages is provided, and the table is sorted based on those values. Additionally, few plots are provided to show some statistics of the entire case, highlighting advantages and critical issues of our method.

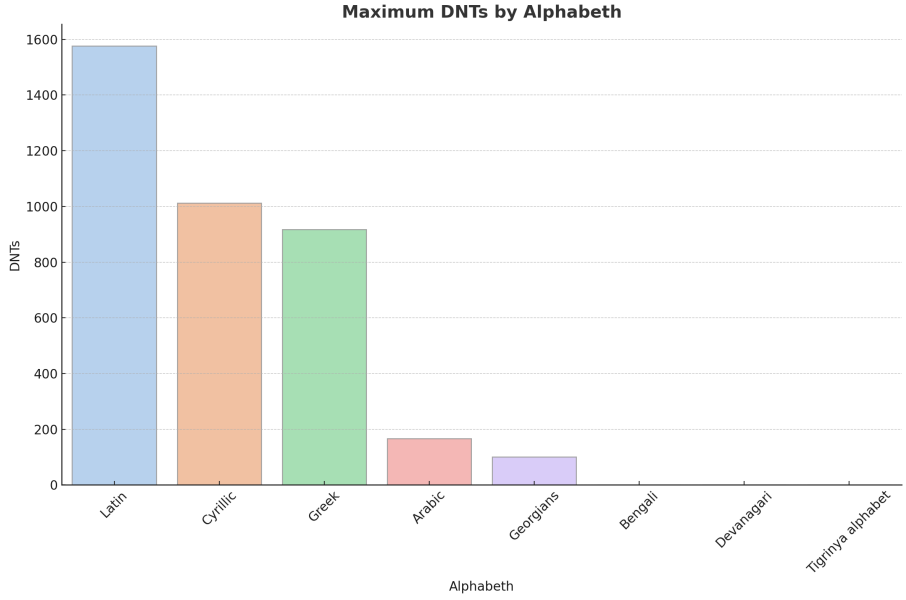


Figure 4.4: Maximum DNT by alphabeth

Figure 4.4 shows, for each alphabet presented, the maximum number of DNTs found within a sample, showing how the Latin alphabet generally promises a more substantial applicability of the method presented in this work. The Arabic alphabet, albeit in a reduced form, responds sufficiently well to the generability of DNTs by DeepPavlov, allowing it to work with languages belonging to this family. On the other hand, a peculiar situation is shown for other alphabets. Greek and Cyrillic show a great responsiveness, making us assume that the combination of tag and translitteration works well for languages of this type. Languages such as Tigrinya, Nepali and Bangla are expected to respond poorly to the methodology. These languages in fact report values close to 0. This is because on the one hand DeepPavlov

fails to acquire useful tags for the aforementioned languages, on the other hand because finding words whose transliteration corresponds to the original language does not seem to be an optimal solution; an alternative management at the entity level could lead to different results, as will be mentioned later.

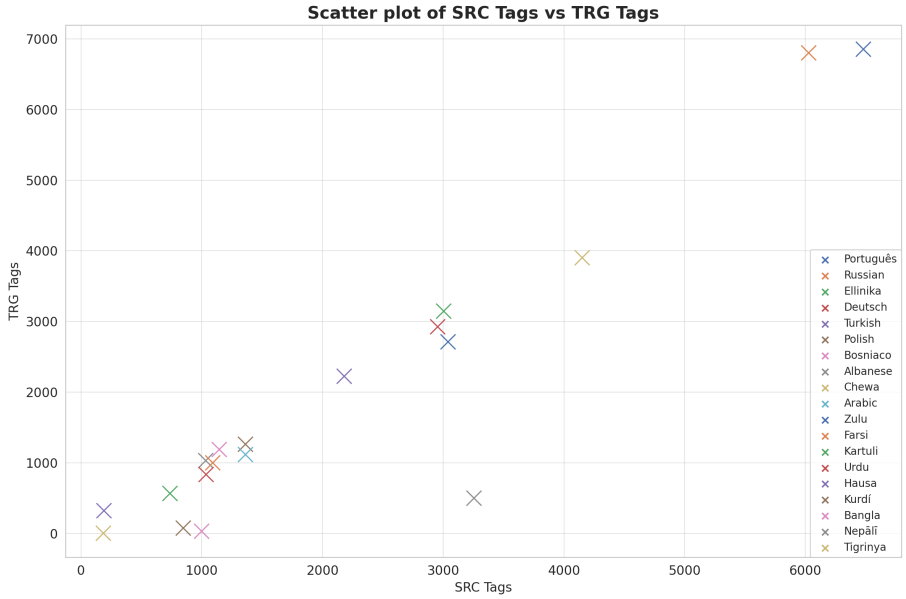


Figure 4.5: SRC Tags vs TRG Tags per language

Figure 4.5 focuses more specifically on the issue of the NER system. Reinforcing what was said earlier, it can be observed that Latin languages tend to maintain a primacy in terms of the number of tags found by DeepPavlov in contrast to others with more limited resources where the quantity tends to shrink.

Metric	Value
Pearson Correlation Coefficient	0.9418
R^2 Coefficient	0.8869
Spearman Correlation Coefficient	0.8620
Kendall Correlation Coefficient	0.7546

Table 4.8: Correlation statistics between "SRC Tags" and "TRG Tags"

As an aid to the graphical representation, Table 4.8 is provided, which calculates the ratio of the two variables in relation to different coefficients. All these coefficients reach values close to 1, effectively indicating a strong positive linear correlation between the two variables. Specifically, we find values mainly in the bisector of this pattern, indicating that the number of SRC tags tends to be similar to the number of TRG tags in most cases. There are also languages such as Nepali, Kurdí and Bangla that develop more along the x-axis, indicating an increasing imbalance between the sourgente and target languages with respect to the tags generated by DeepPavlov. It is also important to note, irrespective of the source language, that many languages being particularly resource-poor do not reach high levels of generated tags, due to

the fact that parallel texts, as explained in the chapter 3.2.2, are mostly inherent to religious domains with very verbose texts, where it is consequently difficult to find sufficient quantities of tags.

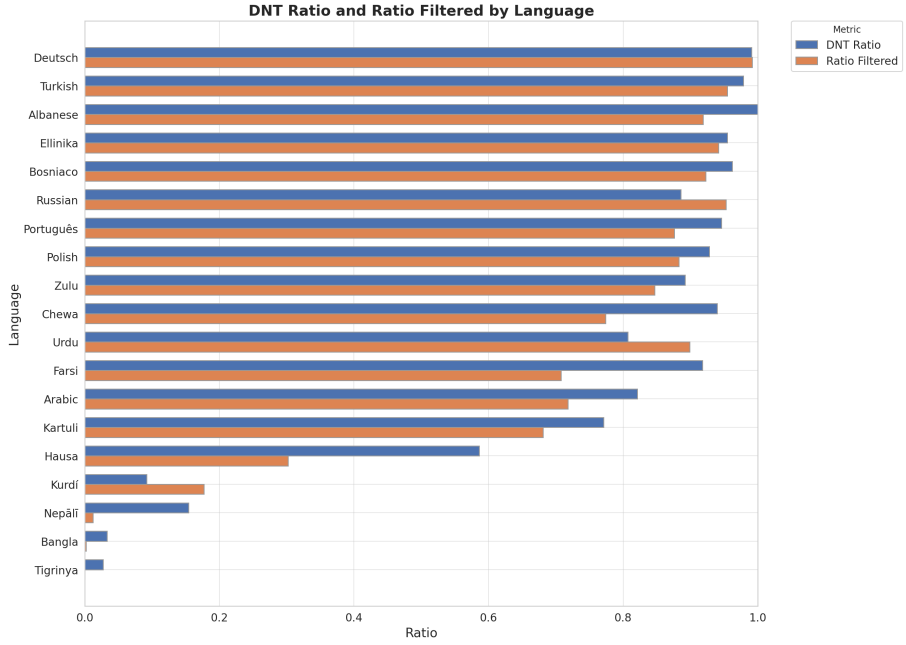


Figure 4.6: DNT Ratio and Ratio Filtered by Language

Finally, Figure 4.6 shows precisely the ratio between the source language and target language as well as the same in relation to the tags of interest in this work. It is interesting to note that for languages with greater resources, the difference between the blue bar (general rate) and orange bar (filtered tag rate) does not diverge much, where instead, going down the graph, these values tend to differ more and more, reinforcing the concept that more common domains contain text that is not very useful in this context.

Ultimately, this analysis does not take into account languages that are known to deviate semantically from Latin languages. Despite the fact that these may not be part of the family of resource-limited languages, it may be the case that is impossible to compare the original entity or its transliteration by using a reasonable Levenshtain treshold to produce quality matching, due to the different nature of the alphabeths. Some other languages also deviates from the classical Latin alphabet using single characters that integrate more complex meanings within them. An example is languages based on kanjis (Japanese and Chinese). Chinese uses more than 2166 kanjis, which are not real letters but rather words with their own meanings, in very large numbers precisely to cover as many objects as possible[5]. That said, with the system we have developed and for particularly complex languages the comparison between entities becomes impossible and, as suggested in the next chapter, an interesting implication for related works based on entity recognition may include testing different ways of comparison metrics, going beyond what has been done in this thesis.

[5]`https://it.wikipedia.org/wiki/Kanji`

Summary

This chapter outlined the main experiments developed to obtain performing Machine Translation engines with respect to both the BLEU score on translation quality and DNT handling. In section 4.1, various baseline models and their characteristics are reported by delving into the fine-tuning approach with respect to Turkish on the one hand and Urdu on the other, reporting the results of the best-performing models. In Section 4.2, we additionally depict the situation regarding the issue of reproducibility of the work discussed in this thesis, highlighting its potential and possible limitations.

Chapter 5

Conclusions

In this thesis, we proposed an approach for handling Do Not Translate (DNTs) Terms in the context of Machine Translation with respect to medium and low-resource languages. By DNTs we mean a variety of terms which, depending on the context, it is necessary not to translate: we can consider for example proper people name, places, dates, but also any part of the text for which we intend to leave intact in the translation phase. By low-resource languages instead we mean all those for which it is difficult to obtain an adequate number of data to work on and which highlight resources closely related to religious domains.

This work was developed in collaboration with PerVoice, a company that deals with research and development in the field of Automatic Speech Recognition and Automatic Translation. Together with PerVoice, we proposed an approach to solving the problem of DNTs, which arose from a customer's need to create effective multilingual translations for documents for which certain entities needed to be treated as terms not to be translated. We considered two languages as a proof of concept for our method, Turkish and Urdu. We created an end-to-end process that would provide Machine Translation engines capable of generating translations that were as correct as possible according to the domain they belonged to, and that would optimally manage the DNTs in inference. Therefore, starting with the collection, manipulation and filtering of data, the sources and characteristics of which are shown throughout the thesis, a methodology is proposed to be applied to resource-limited languages for the management of DNTs. A first step, by means of Named Entity Recognition (NER) and alignments matching between parallel corpora, involves identifying and replacing within the initial corpora certain entities with a specific enumerated tag for DNTs, in order to allow the engine in the training phase to modify the network weights in favour of the tags themselves. Specific techniques are then defined for the replacement of the terms not to be translated. We considered a methodology specifically tailored for Turkish and Urdu that includes the use of distance metrics, namely Levenshtein distance, and a transliteration approach to create a *Ranking System* to obtain a precise matching between entities present in parallel corpora, in order to replace them with DNTs tags. For the creation of the models, a PyTorch-based framework, called ModernMT, is proposed, which easily manages the creation, evaluation and execution of machine translation models, and of which the parameters used for these activities, as well as the characteristics of the models, are highlighted. Exploiting this tool and starting from models previously created

by the company, engines are then generated that produce translations adapted to the domain of the documents (measured by BLEU score), as well as manage DNTs (evaluated by F-Scores).

The obtained results have been quite encouraging, and may make a small yet significant contribution to the field of study. With respect to the scores reported in relation to the BLUE score, one of the most commonly used metrics in MT, we saw that our approach obtain an improvement, albeit marginal, on the performances, reporting higher percentage scores. We must consider that this work takes into consideration languages that are not particularly rich in resources, thus, any performance improvement can be considered relevant. With respect to the handling of DNT by machine translation systems, we saw that in almost all cases the adapted models obtain near perfect results, regardless of the aggressiveness of the fine tuning applied to them. In our case we are mostly interested in Recall, since our goal is to recognize as many DNTs as possible, and several trained models achieve this.

This work was set up from the outset to focus on a restricted set of languages. However, the question of reproducibility on other languages was considered as well in Chapter 4.2. Our findings suggest that the method could be effectively applied to languages using Latin, Arabic and cyrillic alphabets, while more complex languages, such as Asian ones using kanji, may pose a more challenging problem. However, the use of a ranking system to manage the various alignments generated by Fast Align, as well as the transliteration system to create comparable entities, is limited to a defined, albeit high, number of languages.

To overcome this limitation with a view to the future use of our methodology, we can point out several possible strategies:

Improvement on Named Entity Recognition : as often defined in the course of this thesis, the system we use for the NER, called DeepPavlov, is part of a long list of open source or otherwise available resources, in fact we cite NLTK[1] or Spacy[2] as direct competitors of DeepPavlov, which also provide the same models but, in this specific case limited to a smaller number of languages, leaving DeepPavlov the primacy in terms of scalability. An interesting task would be to extend the existing models, fine-tuning them or training new ones in order to allow the extraction of identities for those languages not yet present in the currently available models. In the domain of resource-constrained languages, this task would be a matter of constructing datasets within which texts and their entities are inserted, but unfortunately it turns out to be a particularly difficult task due to the fact that for obvious reasons it is not easy to find corpora for languages of this kind or to find those who would manually have to annotate them with the right entities. Fortunately, DeepPavlov as well as the models proposed in the library are already a few years old, and since there is much academic as well as commercial research in favour of NLP and multi-language domains, it is to be hoped that several datasets for particularly problematic languages will be created, if there are none already, enabling the list of processable languages to be expanded.

[1] `https://www.nltk.org/`
[2] `https://spacy.io/`

Alternative methods of textual comparison : in chapter 3.3.3, a ranking system was introduced with respect to entities based on the distance between words, using a metric very popular in NLP and infromation retrieval known as Levenshtein's distance or Edit's distance. This is a measure for quantifying the difference between two strings, and was chosen for efficiency issues in the languages of this work, as much as for the efficiency of algorithmic terms. However, it suffers from limitations arising from the fact that if one is working on languages belonging to different alphabets, there are bound to be problems in the letter-by-letter comparison on which this algorithm is based and for which transliterations are exploited. In the field of NLP, Levenshtein's formula is considered an "Edit Base similarity" formula. Although simple and quite effective on relatively short strings, it does not offer as good results when applied to longer strings. In fact, we have previously pointed out that the threshold used for the creation of matches between entities belonging to different texts has been set relatively low. There are, however, several categories of distances applicable in the domain of textual data. Specifically, "Token based Similarities" could be considered, which compare two or more strings directly based on words or n-grams, e.g. Jaccard index or Cosine Similarity. "Sequence Based" metrics which compare two strings at the level of string sub-sequences, could be applied as well. Alternatively, for languages that are not as resource-limited, an approach involving the use of pre-trained transfomers to compare the entities themselves could be attempted. All these metrics could be viable alternatives to Levenshtein's distance, assuming qualitatively better scenarios for possible future work.

Alternative frameworks for the creation of models : in this work, a very high-level ModernMT framework was exploited, which is able to create high-performance machine translation engines with not too much effort on the part of the user, and is certainly preferable in a more commercial context. From a more experimental point of view, there are other alternatives, such as Nvidia NeMo[3], which exploits deep-learning models based on PyTorch, optimised for proprietary hardware, and which is proposed as a valid alternative for natural language processing applications such as machine translation. This framework is at a lower level than ModernMT in that it allows the modification of a greater number of parameters than ModernMT permits, leaving room for greater customisation, which is especially useful in research. Exploiting different environments for the creation of machine translation engines would certainly bring new perspectives and new approaches, in continuation of what has been seen in this work.

In conclusion, given the lack of literature on the subject, it can be said that this work, with its quite encouraging results, can certainly represent a starting point for work on free entity management, both in the specific field of Machine Translation and in a more general NLP context.

[3] `https://developer.nvidia.com/nemo`

Appendix A

Metodology report

A.1 OPUS Corpora references

Corpus	Version	Reference
MultiCCAligned	1.1	statmt.org, [20]
OpenSubtitles	v2018	opensubtitles.org, [87]
WikiMatrix	1.0	github.com/facebookresearch/WikiMatrix, [78]
CCMatrix	1.0	github.com/facebookresearch/CCMatrix, [79]
TED2020	1.0	opensubtitles.org, [71]
QED	2.0a	Ahmed Abdelali et al. [1]
NeuLab-TedTalks	1.0	phontron.com
LinguaToolsWiki	v2014	opus.nlp.eu
XLEnt	1.2	statmt.org/xlent, [21]
GNOME	1.0	gnome.org
Tanzil	1.0	tanzil.net
KDE4	2.0	opus.nlp.eu
Bible-uedin	1.0	github.com/christos-c/bible-corpus
EUbookshop	2.0	opus.nlp.eu
Mozilla-I10n	1.0	github.com/mozilla-l10n
PHP	1.0	php.net
Tatoeba	v2023-04-12	tatoeba.org
GlobalVoices	v2018q4	casmacat.eu
wikimedia	v20230407	dumps.wikimedia.org
ELRC-wiki_health	1.0	elrc-share.eu
KDEdoc	1.0	opus.nlp.eu

Table A.1: Opus corpora references

A.2 ModernMT Transformer Base-Model

```
      TransformerModel(
(encoder): TransformerEncoder(
  (embed_tokens): Embedding(32664, 512, padding_idx=1)
  (embed_positions): SinusoidalPositionalEmbedding()
  (layers): ModuleList(
    (0): TransformerEncoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (1): TransformerEncoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (2): TransformerEncoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (3): TransformerEncoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
(self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (4): TransformerEncoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (5): TransformerEncoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
  )
)
```

```
(decoder): TransformerDecoder(
  (embed_tokens): Embedding(32664, 512, padding_idx=1)
  (embed_positions): SinusoidalPositionalEmbedding()
  (layers): ModuleList(
    (0): TransformerDecoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (encoder_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (encoder_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (1): TransformerDecoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (encoder_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (encoder_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (2): TransformerDecoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (encoder_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (encoder_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (3): TransformerDecoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (encoder_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (encoder_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=512, out_features=2048, bias=True)
      (fc2): Linear(in_features=2048, out_features=512, bias=True)
      (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    )
    (4): TransformerDecoderLayer(
      (self_attn): MultiheadAttention(
        (out_proj): Linear(in_features=512, out_features=512, bias=True)
      )
      (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
```

```
        (encoder_attn): MultiheadAttention(
          (out_proj): Linear(in_features=512, out_features=512, bias=True)
        )
        (encoder_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=
    True)
        (fc1): Linear(in_features=512, out_features=2048, bias=True)
        (fc2): Linear(in_features=2048, out_features=512, bias=True)
        (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      )
       (5): TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): Linear(in_features=512, out_features=512, bias=True)
        )
        (self_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=
    True)
        (encoder_attn): MultiheadAttention(
          (out_proj): Linear(in_features=512, out_features=512, bias=True)
        )
        (encoder_attn_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=
    True)
        (fc1): Linear(in_features=512, out_features=2048, bias=True)
        (fc2): Linear(in_features=2048, out_features=512, bias=True)
        (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
)

```

Listing A.1: ModernMT Transformer architecture

Bibliography

[1] Ahmed Abdelali, Francisco Guzman, Hassan Sajjad, and Stephan Vogel. The AMARA corpus: Building parallel language resources for the educational domain. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1856–1862, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA).

[2] Jaimeen Ahn and Alice Oh. Mitigating language-dependent ethnic bias in BERT. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 533–549, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[3] L. T. Anh, M. Y. Arkhipov, and M. S. Burtsev. Application of a hybrid bi-lstm-crf model to the task of russian named entity recognition, 2017.

[4] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[5] Leonard E. Baum and Ted Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554 – 1563, 1966.

[6] Iz Beltagy, Arman Cohan, Robert Logan IV, Sewon Min, and Sameer Singh. Zero- and few-shot NLP with pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 32–37, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[7] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. 3(null):11371155, mar 2003.

[9] Ondřej Bojar, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, and Christof Monz. Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 272–303, Belgium, Brussels, October 2018. Association for Computational Linguistics.

[10] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[11] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

[12] Jason P. C. Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns, 2016.

[13] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators, 2020.

[14] Paula Czarnowska, Yogarshi Vyas, and Kashif Shah. Quantifying Social Biases in NLP: A Generalization and Empirical Comparison of Extrinsic Fairness Metrics. *Transactions of the Association for Computational Linguistics*, 9:1249–1267, 11 2021.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[16] Greg Durrett and Dan Klein. A joint model for entity analysis: Coreference, typing, and linking. *Transactions of the Association for Computational Linguistics*, 2:477–490, 2014.

[17] Chris Dyer, Victor Chahuneau, and Noah A. Smith. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[18] Pnar Dönmez. Introduction to machine learning, 2nd ed., by ethem alpaydn. cambridge, ma: The mit press2010. *Natural Language Engineering*, 19(2), 2013.

[19] Christina Eberharter. *Le problematiche e le sfide della traduzione automatica. Un confronto tra Google Translate, Bing Translator e DeepL.* GRIN Verlag, 2020.

[20] Ahmed El-Kishky, Vishrav Chaudhary, Francisco Guzmán, and Philipp Koehn. CCAligned: A massive collection of cross-lingual web-document pairs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020)*, November 2020.

[21] Ahmed El-Kishky, Adithya Renduchintala, James Cross, Francisco Guzmán, and Philipp Koehn. Xlent: Mining a large cross-lingual entity dataset with

lexical-semantic-phonetic word alignment. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10424–10430, 2021.

[22] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*. Association for Computational Linguistics, 2017.

[23] Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994.

[24] Philip Gage. A new algorithm for data compression. *The C Users Journal archive*, 12:23–38, 1994.

[25] R. Gandy. Human versus mechanical intelligence. In Peter Millican and A. Clark, editors, *Machines and Thought*. Oxford University Press, 1996.

[26] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning, 2017.

[27] U. Germann, E. Barbu, L. Bentivogli, N. Bertoldi, N. Bogoychev, C. Buck, D. Caroselli, L. Carvalho, A. Cattelan, R. Cettolo, M. Federico, B. Haddow, D. Madl, L. Mastrostefano, P. Mathur, A. Ruopp, A. Samiotou, V. Sudharshan, M. Trombetti, and Jan van der Meer. Modern MT: a new open-source machine translation platform for the translation industry. In *Proceedings of the 19th Annual Conference of the European Association for Machine Translation: Projects/Products*, Riga, Latvia, May 30–June 1 2016. Baltic Journal of Modern Computing.

[28] Yoav Goldberg. *Neural Network Methods for Natural Language Processing*. Springer International Publishing, 2017.

[29] Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, and Angela Fan. The flores-101 evaluation benchmark for low-resource and multilingual machine translation. 2021.

[30] Jiatao Gu, Kyunghyun Cho, and Victor O. K. Li. Trainable greedy decoding for neural machine translation, 2017.

[31] Francisco Guzmán, Peng-Jen Chen, Myle Ott, Juan Pino, Guillaume Lample, Philipp Koehn, Vishrav Chaudhary, and Marc'Aurelio Ranzato. Two new evaluation datasets for low-resource machine translation: Nepali-english and sinhala-english. 2019.

[32] Tian Han, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. Alternating back-propagation for generator network. In *AAAI Conference on Artificial Intelligence*, 2016.

[33] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

[34] Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mu Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, and Ming Zhou. Achieving human parity on automatic chinese to english news translation, 2018.

[35] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[37] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2020.

[38] John Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79:2554–8, 05 1982.

[39] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[40] Prashant Johri, Sunil K. Khatri, Ahmad T. Al-Taani, Munish Sabharwal, Shakhzod Suvanov, and Avneesh Kumar. Natural language processing: History, evolution, application, and future work. In Ajith Abraham, Oscar Castillo, and Deepali Virmani, editors, *Proceedings of 3rd International Conference on Computing Informatics and Networks*, pages 365–375, Singapore, 2021. Springer Singapore.

[41] Pratik Joshi, Sebastin Santy, Amar Budhiraja, Kalika Bali, and Monojit Choudhury. The state and fate of linguistic diversity and inclusion in the NLP world. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6282–6293, Online, July 2020. Association for Computational Linguistics.

[42] Jugal Kalita. *Machine Learning: Theory and Practice*. CRC Press, 2022.

[43] Andrej Karpathy. Neural networks part 1: Setting up the architecture. notes for cs231n convolutional neural networks for visual recognition, 2016.

[44] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[45] V. Klema and A. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, 1980.

[46] Surafel M. Lakew, Alina Karakanta, Marcello Federico, Matteo Negri, and Marco Turchi. Adapting multilingual neural machine translation to unseen languages, 2019.

[47] Johannes Lederer. Activation functions in artificial neural networks: A systematic overview. *ArXiv*, abs/2101.09957, 2021.

[48] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020.

[49] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[50] Alexandre Magueresse, Vincent Carles, and Evan Heetderks. Low-resource languages: A review of past work and future challenges. *CoRR*, abs/2006.07264, 2020.

[51] Mieradilijiang Maimaiti, Yang Liu, Huanbo Luan, and Maosong Sun. Enriching the transfer learning with pre-trained lexicon embedding for low-resource neural machine translation. *Tsinghua Science and Technology*, 27(1):150–163, 2022.

[52] Andrzej Makiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers Geosciences*, 19(3):303–342, 1993.

[53] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[54] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[55] Wilhelmina Nekoto, Vukosi Marivate, Tshinondiwa Matsila, Timi Fasubaa, Taiwo Fagbohungbe, Solomon Oluwole Akinola, Shamsuddeen Muhammad, Salomon Kabongo Kabenamualu, Salomey Osei, Freshia Sackey, Rubungo Andre Niyongabo, Ricky Macharm, Perez Ogayo, Orevaoghene Ahia, Musie Meressa Berhe, Mofetoluwa Adeyemi, Masabata Mokgesi-Selinga, Lawrence Okegbemi, Laura Martinus, Kolawole Tajudeen, Kevin Degila, Kelechi Ogueji, Kathleen Siminyu, Julia Kreutzer, Jason Webster, Jamiil Toure Ali, Jade Abbott, Iroro Orife, Ignatius Ezeani, Idris Abdulkadir Dangana, Herman Kamper, Hady Elsahar, Goodness Duru, Ghollah Kioko, Murhabazi Espoir, Elan van Biljon, Daniel Whitenack, Christopher Onyefuluchi, Chris Chinenye Emezue, Bonaventure F. P. Dossou, Blessing Sibanda, Blessing Bassey, Ayodele Olabiyi, Arshath Ramkilowan, Alp Öktem, Adewale Akinfaderin, and Abdallah Bashir. Participatory research for low-resourced machine translation: A case study in African languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2144–2160, Online, November 2020. Association for Computational Linguistics.

[56] James Cross Onur Çelebi Maha Elbayad Kenneth Heafield Kevin Heffernan Elahe Kalbassi Janice Lam Daniel Licht Jean Maillard Anna Sun Skyler Wang Guillaume Wenzek Al Youngblood Bapi Akula Loic Barrault Gabriel Mejia Gonzalez Prangthip Hansanti John Hoffman Semarley Jarrett Kaushik Ram Sadagopan Dirk Rowe Shannon Spruit Chau Tran Pierre Andrews Necip Fazil Ayan Shruti Bhosale Sergey Edunov Angela Fan Cynthia Gao Vedanuj Goswami Francisco Guzmán Philipp Koehn Alexandre Mourachko Christophe Ropers Safiyyah Saleem Holger Schwenk Jeff Wang NLLB Team, Marta R. Costa-jussà. No language left behind: Scaling human-centered machine translation. 2022.

[57] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[58] Yirong Pan, Xiao Li, Yating Yang, and Rui Dong. Exploiting linguistic knowledge for low-resource neural machine translation. *Preprints*, March 2020.

[59] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[60] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[61] Sagarika Pattnaik, Ajit Kumar Nayak, and Srikanta Patnaik. A semi-supervised learning of hmm to build a pos tagger for a low resourced language. *Journal of Information & Communication Convergence Engineering*, 18(4), 2020.

[62] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[63] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[64] Marcin Pietrzykowski and Wojciech Saabun. Applications of hidden markov model: state-of-the-art. 07 2014.

[65] Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70, apr 2018.

[66] Marcelo O. R. Prates, Pedro H. C. Avelar, and Luis Lamb. Assessing gender bias in machine translation – a case study with google translate, 2019.

[67] Jorgensen M. Pritchard, J. The systran english-to-french machine translation system. *Proceedings of the Conference on Applied Natural Language Processing*, pages 2–17, 1982.

[68] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[69] Surangika Ranathunga, En-Shiun Annie Lee, Marjana Prifti Skenduli, Ravi Shekhar, Mehreen Alam, and Rishemjit Kaur. Neural machine translation for low-resource languages: A survey, 2021.

[70] Flo Reeder and Dan Loehr. Finding the right words: an analysis of not-translated words in machine translation. pages 356–363, October 28-31 1998.

[71] Nils Reimers and Iryna Gurevych. Making monolingual sentence embeddings multilingual using knowledge distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2020.

[72] Xin Rong. word2vec parameter learning explained. 2016.

[73] F. Rosenblatt. The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957.

[74] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

[75] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[76] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[77] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. pages 5149–5152, 2012.

[78] Holger Schwenk, Vishrav Chaudhary, Shuo Sun, Hongyu Gong, and Francisco Guzmán. Wikimatrix: Mining 135m parallel sentences in 1620 language pairs from wikipedia, 2019.

[79] Holger Schwenk, Guillaume Wenzek, Sergey Edunov, Edouard Grave, and Armand Joulin. Ccmatrix: Mining billions of high-quality parallel sentences on the web, 2020.

[80] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data, 2016.

[81] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[82] Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA, August 8-12 2006. Association for Machine Translation in the Americas.

[83] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[84] Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. Fast and accurate entity recognition with iterated dilated convolutions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2670–2680, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[85] Sandeep Subramanian, Oleksii Hrinchuk, Virginia Adams, and Oleksii Kuchaiev. NVIDIA NeMo's neural machine translation systems for English-German and English-Russian news and biomedical tasks at WMT21. In *Proceedings of the Sixth Conference on Machine Translation*, pages 197–204, Online, November 2021. Association for Computational Linguistics.

[86] Tony Sun, Andrew Gaut, Shirlyn Tang, Yuxin Huang, Mai ElSherief, Jieyu Zhao, Diba Mirza, Elizabeth Belding, Kai-Wei Chang, and William Yang Wang. Mitigating gender bias in natural language processing: Literature review. pages 1630–1640, jul 2019.

[87] Jörg Tiedemann. Finding alternative translations in a large corpus of movie subtitle. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3518–3522, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

[88] Jörg Tiedemann. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA).

[89] P. Toma. Systran as a multilingual machine translation system. *Proceedings of the Third European Congress on Information Systems and Networks, Overcoming the language barrier*, page 569581, 1977.

[90] Alan M Turing. Computing machinery and intelligence (1950). *The Essential Turing: the Ideas That Gave Birth to the Computer Age*, pages 433–464, 2012.

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[92] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9, 04 2022.

[93] Warren Weaver. Rededication to liberal decency. *Science*, 121(3138):15A, 1955. Cited by: 0; All Open Access, Bronze Open Access.

[94] Emil Winder. Natural language processing algorithms, 2023.

[95] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[96] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, ukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[97] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning – a comprehensive evaluation of the good, the bad and the ugly, 2020.

[98] Poorya Zaremoodi, Wray Buntine, and Gholamreza Haffari. Adaptive knowledge sharing in multi-task learning: Improving low-resource neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 656–661, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[99] Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. Transfer learning for low-resource neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1568–1575, Austin, Texas, November 2016. Association for Computational Linguistics.