UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

Streaming neural machine translation systems from European languages into English

End of Degree Project

Bachelor's Degree in Informatics Engineering

AUTHOR: Sarvazyan , Areg Mikael

Tutor: Civera Saiz, Jorge

Cotutor: Iranzo Sánchez, Javier

Experimental director: GARCES DIAZ-MUNIO, GONZALO VICENTE

ACADEMIC YEAR: 2021/2022

Resum

La traducció automàtica (MT, de l'anglès Machine Translation) és una dels àrees més actives dins de la intel·ligència artificial, particularment en el camp de l'aprenentatge automàtic. Recentment, aquesta àrea ha sigut el focus d'atenció per part d'importants figures tecnològiques com Google, Facebook, Microsoft, etc. a causa de les millores de rendiment obtingudes per aquesta tecnologia gràcies a la incorporació de xarxes neuronals artificials. Un dels principals motius que explica està atenció és l'enorme creixement de plataformes de difusió de continguts audiovisuals en streaming (per exemple YouTube i Twitch) i vídeo-conferencia (per exemple Zoom i Webex). En aquest context, un aspecte molt important és l'adaptació de tècniques i models convencionals per al seu ús en streaming, això és, per a un flux d'entrada a traduir constant i eixida ajustada per davall d'un temps de resposta (latència) donat. En aquest treball es proposa estudiar i implementar models avançats de MT neuronal en streaming, de llengües europees a l'anglès. Per a això, es farà ús de dades, tecnologia i experiència del grup MLLP del VRAIN, adquirits en el marc de projectes d'investigació i transferència tecnològica desenvolupats en els últims anys.

Paraules clau: Traducció automàtica; Traducció automàtica neuronal; Traducció automàtica en temps real

---

Resumen

La traducción automática (MT, del inglés Machine Translation) es una de las áreas más activas dentro de la inteligencia artificial, particularmente en el campo del aprendizaje automático. Recientemente, esta área ha sido el foco de atención por parte de importantes figuras tecnológicas como Google, Facebook, Microsoft, etc. debido a las mejoras de rendimiento obtenidas por esta tecnología gracias a la incorporación de redes neuronales artificiales. Uno de los principales motivos que explica esta atención es el enorme crecimiento de plataformas de difusión de contenidos audiovisuales en streaming (por ejemplo YouTube y Twitch) y video-conferencia (por ejemplo Zoom y Webex). En este contexto, un aspecto muy importante es la adaptación de técnicas y modelos convencionales para su uso en streaming, esto es, para un flujo de entrada a traducir constante y salida ajustada por debajo de un tiempo de respuesta (latencia) dado. En este trabajo se propone estudiar e implementar modelos avanzados de MT neuronal en streaming, de lenguas europeas al inglés. Para ello, se hará uso de datos, tecnología y experiencia del grupo MLLP del VRAIN, adquiridos en el marco de proyectos de investigación y transferencia tecnológica desarrollados en los últimos años.

Palabras clave: Traducción automática; Traducción automática neuronal; Traducción automática en tiempo real

---

Abstract

Machine translation is one of the most active areas within Artificial Intelligence, particularly in the field of Machine Learning. Recently, this area has been the focus of attention by major technology figures such as Google, Facebook, Microsoft, etc. due to the performance improvements obtained by this technology thanks to the incorporation of artificial neural networks. One of the main reasons for this attention is the enormous growth of streaming audiovisual content platforms (for example, YouTube and Twitch) and video-conferencing (for example, Zoom and Webex). In this context, a very important aspect is the adaptation of conventional techniques and models to be used in the

streaming scenario, that is, the translation of a constant input flow under response time (latency) constraints. In this work it is proposed to study and implement advanced models of neural MT in streaming, from European languages into English. For this, data, technology and experience of the VRAIN MLLP group, acquired in the framework of research and technology transfer projects developed in recent years, will be used.

Key words: Machine Translation; Neural Machine Translation; Streaming Machine Translation

Contents

Contents v
List of Figures vii
List of Tables vii

1 Introduction 1
1.1 Motivation . . . 1
1.2 Framework . . . 2
1.3 Goals . . . 2
1.4 Document Structure . . . 2
2 Preliminaries 5
2.1 Machine Learning . . . 5
2.1.1 Supervised Learning . . . 5
2.1.2 Bayes classifier . . . 6
2.2 Statistical Machine Translation . . . 6
2.3 Neural Machine Translation . . . 8
2.3.1 Computational graphs . . . 11
2.3.2 RNNs, The Encoder-Decoder Architecture, and Attention . . . 11
2.3.3 The Transformer . . . 15
2.3.4 Decoding . . . 20
2.3.5 Evaluation . . . 21
3 Data Processing 23
3.1 Corpora . . . 23
3.1.1 General Domain Corpora . . . 24
3.1.2 In-Domain Corpora . . . 25
3.2 Processing Steps . . . 26
3.2.1 Filtering . . . 27
3.2.2 Tokenization . . . 28
3.2.3 Lowercasing and Truecasing . . . 28
3.2.4 Subword Segmentation . . . 28
4 Domain Adaptation 31
4.1 Backtranslations . . . 31
4.2 Fine-tuning . . . 32
5 Streaming Machine Translation 35
5.1 Formal Definition . . . 36
5.2 Models . . . 36
5.2.1 Wait-k . . . 36
5.2.2 Test-Time Wait-k . . . 37
5.2.3 Multi-Path Wait-k . . . 37
5.3 Evaluation . . . 38
5.3.1 Average Proportion . . . 38
5.3.2 Average Lagging . . . 39
5.3.3 Differentiable Average Lagging . . . 40

6 Experiments **41**
6.1 Experimental Setup 41
6.1.1 Fairseq 41
6.2 General Domain Systems 43
6.3 In-Domain Systems 45
6.4 Streaming MT Systems 47
7 Conclusions **49**
Bibliography **51**

Appendices
Appendix: Sustainable Development Goals **59**

List of Figures

2.1 The IBM Model 1 co-occurrence learning process. 8
2.2 A simple multi-layer perceptron. 10
2.3 Example of a computational graph and automatic differentiation. 12
2.4 The full Transformer architecture. 16
2.5 Transformer encoder and decoder layers. 16
2.6 Positional Encodings. 19
2.7 Search graph for beam search decoding using $\beta = 4$. 20

3.1 Text processing pipelines applied to a phrase. 30

4.1 The process of obtaining a backtranslated corpus from monolingual data. 32

6.1 The general experimentation pipeline. 42
6.2 Change in BLEU w.r.t the number of updates in fine-tuning time. 46
6.3 Latencies and BLEU scores of wait-k MT systems on WMT13. 47
6.4 Latencies and BLEU scores of wait-k MT systems on CN21. 48

List of Tables

3.1 Contents of the general domain corpora for the language pair Fr-En. . . . 24
3.2 Contents of the CDS monolingual English corpus. 26
3.3 Statistics of the CERN News parallel corpus. 27

6.1 BLEU, chrF2, and TER scores on WMT13 Fr→En, and WMT14 Fr→En. . . 44
6.2 BLEU, chrF2, and TER scores on CN21 Fr→En, and CN22 Fr→En. 46
6.3 Results of V3-FT-CN with 1000 fine-tuning updates evaluated on WMT14 and CN22. 47
6.4 Results of V3-MP with wait-7 decoding, evaluated on WMT14 and CN22. 48

CHAPTER 1
Introduction

In this introductory chapter we present the motivation, framework and goals of this work, as well as its context within the current state-of-the-art in machine translation (MT). We illustrate how this work studies the field of MT by developing translation systems from French to English, for use in both offline and streaming scenarios, while also using a set of techniques to adapt MT systems to the domain of physics. Lastly, we describe how the contents of this work are organized.

1.1 Motivation

Recent advancements in computational power have made many technological dreams a reality. From automatic object detection to speech-to-speech translation to prompt-based image generation, the field of artificial intelligence is revolutionizing our daily lives. More specifically to this work, there is a current necessity to develop translation systems for offline and real-time scenarios that perform reliably in a given domain. Not only do these systems allow for greater accessibility to non-speakers of a language, but the application to a real-time scenario such as live-streams or virtual meetings can provide a critical role in our daily lives, and the focus towards a particular domain aids the models to perform reliably while translating technical topics.

There are many out-of-the-box multilingual MT services currently offered to the general public: Google Translate[1], DeepL[2], Microsoft Translator[3], Watson Language Translator[4], etc. While these services might be adequate for daily use by the general public, there are many instances where robust translation systems for particular domains are necessary. Additionally, current trends in streaming content generation, virtual meetings, metaverses, and the virtualization of our world as a whole, give rise to new challenges in effective communication between people who speak different languages. There is a demand for systems that automatically and reliably translate what is spoken or written in real-time, so that conversations can flow and listeners can understand new content that was previously inaccessible.

In this regard, multinational organizations are making an effort to lower language barriers so that communication across languages is a reality. Public organizations such as the European Organization for Nuclear Research (CERN), who store thousands of hours of recorded conferences and seminars and hold hundreds of multinational meetings every day, are pushing to make this multimedia content more accessible to the research

[1] https://translate.google.com/
[2] https://www.deepl.com/translator
[3] https://www.microsoft.com/en-us/translator/business/trial/
[4] https://www.ibm.com/cloud/watson-language-translator

community. However, the idea of letting mainstream MT providers, such as those mentioned above, process internal multimedia content is inconceivable, not only regarding its technical specificity, but also due to privacy concerns. This opens the possibility of deploying domain-adapted on-premises MT technology, developed by small and medium-sized research organizations, that perfectly fits the needs of these public organizations.

1.2 Framework

This work is part of the author's research internship at the Valencian Research Institute for Artificial Intelligence's Machine Learning and Language Processing (VRAIN MLLP) research group of the Universitat Politècnica de València (UPV). It is contextualized on a technology-transfer contract between the MLLP and the CERN. The project consists in the development of highly accurate automatic captioning and translation systems to be applied on CERN's multimedia content catalog, as well as low latency streaming speech-to-speech translation systems to be integrated into CERN's internal and external communication systems for live captioning of webcasts and video conference meetings. This work presents the experiments and results obtained in the development of the MT systems that will be used as part of offline and streaming speech-to-speech translation systems in the French to English language direction, adapted to the domain of high energy physics, the main domain of interest to CERN.

1.3 Goals

The main goals of this work, built upon the motivation and framework previously described, are the following:

- To study the current state-of-the art approaches for offline and real-time MT, including domain adaptation techniques and automatic evaluation.
- To apply the most important tools employed in MT research for data processing, model training, inference and evaluation.
- To explore and refine data filtering and processing techniques to improve the quality of MT systems.
- To develop general domain and in-domain MT systems that provide accurate enough translations for the purposes of CERN.
- To develop simultaneous MT systems for the streaming MT scenario with low latency and good enough quality.

1.4 Document Structure

In addition to this introductory chapter, the following chapters are structured as follows. Chapter 2 introduces the preliminary concepts needed to understand this work. It formalizes MT as a Supervised Learning task in the context of Machine Learning, while also introducing the historical context of the field. Here we dive deep into the state-of-the-art neural-based architecture that powers today's translation services by paving the road from the first neural model, the Perceptron, to the Transformer architecture that will be used in the work. In this chapter we also present the different evaluation metrics that

we use throughout this work. Chapter 3 is concerned with the general domain and in-domain corpora that will be used to develop MT systems, as well as the various data filtering and processing techniques that can be applied to said corpora. In Chapter 4 we describe the techniques that we use in this work for domain adaptation to the domain of high energy physics. More specifically we illustrate how we obtain synthetic data from a monolingual in-domain corpus, and the general fine-tuning framework with which we adapt general domain MT systems to this domain. We conclude the theoretical aspects of the work with Chapter 5, where we describe the problems of simultaneous and streaming translation, presenting simultaneous MT models that include latency constraints, as well as the latency-based evaluation metrics we use to evaluate simultaneous MT models. Afterwards, Chapter 6 presents the set of experiments and results of the general domain, in-domain, offline and online MT systems that were developed during the time this work was carried out. Finally, we summarize the contents of the work and envision future work in Chapter 7.

CHAPTER 2
Preliminaries

MT is a task encompassed within the fields of Natural Language Processing and Computational Linguistics, which themselves are included partly in the field of Machine Learning. Before diving deep into the current approaches of the MT task, we must first introduce the field of Machine Learning (ML), as well as the different approaches to MT in history. In this chapter we will illustrate the different types of tasks that exist in ML, defining important concepts such as Bayes' classifier, explaining how MT is an instance of a particular type of ML task, giving an overview of the historical advancements of MT, and finally delving into the details of how the current state-of-the-art approach for MT works.

2.1 Machine Learning

ML is a subfield of Artificial Intelligence that encompasses all tasks in which a program gradually learns to perform better. More specifically, as Tom Mitchell [53] defined:

"A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Essentially, ML is concerned with mathematical models that learn to execute tasks without a human's explicit instruction regarding the steps necessary to achieve them.

These tasks can be grouped into different categories: *supervised learning* tasks, where the model learns a mapping between a set of observations and their corresponding labels, *unsupervised learning* tasks, where the model only has access to observations and attempts to find relationships or groups in the data, and *reinforcement learning* tasks, where instead of accessing data, the model learns to perform a task by exploring its environment.

In this work we will mainly concern ourselves with supervised learning tasks. In the following sections we will expand on how MT can be modeled as a supervised learning task, and introduce other important concepts needed to understand this work. More specifically, we will be working on an instance of a classification problem.

2.1.1. Supervised Learning

Supervised Learning is a learning methodology applied to types of task T where the experience E is given by the model's ability to observe N labeled data pairs $\mathcal{S} = \{\mathcal{X} \times \mathcal{C}\} = \{(\mathbf{x}_n, \mathbf{c}_n)\}$, where $\mathcal{X}$ is the set of object representations and $\mathcal{C}$ is the set of class labels. In supervised learning, the model learns a mapping function from $\mathcal{X}$ to $\mathcal{C}$ by continuously

observing pairs in $\mathcal{S}$ through a process called *training*. In this case, a performance measure is needed to correct the model's hypotheses: a loss function L that computes the compatibility between the models output $\hat{\mathbf{c}}$ and the correct label $\mathbf{c}$ for a given datum $\mathbf{x}$.

There are two main types of supervised learning tasks: *regression*, where the model learns to predict the outcome of a set of continuous variables; and *classification*, where the model predicts the probability that an observation belongs to one or many categories.

2.1.2. Bayes classifier

If the label of the task at hand is categorical, i.e. it only has a limited discrete number of possible values, then the model must learn to classify elements in $\mathcal{X}$ into one or many of these categories. We call this supervised learning task a classification task.

According to the statistical decision theory framework, we should take the decision of classifying every $\mathbf{x}$ in the class $\hat{c}$ that minimizes the probability of error, that is,

$$\hat{c} = \underset{c \in \mathcal{C}}{\operatorname{argmin}} \; p_c(error \mid \mathbf{x}). \tag{2.1}$$

This is achieved by classifying $\mathbf{x}$ into the class with maximum posterior probability

$$\hat{c} = \underset{c \in \mathcal{C}}{\operatorname{argmin}} \; 1 - p(c \mid \mathbf{x}), \tag{2.2}$$

which is the well-known *Bayes classifier*

$$\hat{c} = \underset{c \in \mathcal{C}}{\operatorname{argmax}} \; p(c \mid \mathbf{x}). \tag{2.3}$$

Applying Bayes' rule, and discarding the denominator (since it is not involved in the maximization), we rewrite Equation 2.3 as

$$\hat{c} = \underset{c \in \mathcal{C}}{\operatorname{argmax}} \; p(\mathbf{x} \mid c)\, p(c), \tag{2.4}$$

where the posterior probability distribution $p(c \mid \mathbf{x})$ is factorized into a class-conditional probability $p(\mathbf{x} \mid c)$ and a class-prior probability $p(c)$.

2.2 Statistical Machine Translation

MT has been a task of interest for many years. The interest rose from the dream of automatically translating phrases from one language to another, and how this would revolutionize communication between different cultures. Thus, a significant amount of resources have been invested into MT research, and many important approaches and breakthroughs have been made throughout history. The more successful approach to the task of MT has been Statistical MT (SMT).

Historically, there was great optimism on MT in the 1950s, when translation from Russian to English was demonstrated in the Georgetown Experiment [32]. However, the first quality-wise usable approximations were made in the 1970s, where translation was modeled as an decryption task from any language into English. This was a popular due to the advancements in decryption in war times. At this point there were many commercial systems in use: Météo, Systran, Logos, METAL, Trados. However, these systems were used as computer-aided translation systems for human translators, as a way to ease their burden of translating whole texts at once. In the 1980 there was a trend to use an abstract

meaning representation approach: *interlingua*. The objective was to translate text from any language into a intermediary language, such that a translation from any language to any other would go through interlingua. Ideally we would only have one system with interlingua as a source, and another with it as a target: the translation onto other human languages would be carried out transitively. Sadly, even though it sounded good on paper, it did not work quite as well: the problem of abstract meaning representation is still one of the greatest challenges in artificial intelligence. There were also data-driven methods that would translate by retrieving and editing similar examples from memory systems.

Most of the successful SMT models use n-gram[1] statistics to approximate probability distributions and generate translations. Applying Equation 2.3 to the context of SMT, given a source sentence $\mathbf{x} = x_1, \ldots, x_j, \ldots, x_J$ where each word x_j is drawn from a source vocabulary $\mathcal{X}$, we search for the most probable translation $\hat{\mathbf{y}} = y_1, \ldots, y_i, \ldots, y_I$ according to

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}^*}{\operatorname{argmax}}\, p(\mathbf{y}|\mathbf{x}) \tag{2.5}$$

with $\mathcal{Y}$ and $\mathcal{Y}^*$ denoting the target vocabulary and translation space, respectively. Following the same procedure used in Section 2.1.2 to obtain Equation 2.4, we can use Bayes' rule to decompose the posterior probability into a product of prior and conditional probabilities

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}^*}{\operatorname{argmax}}\, p(\mathbf{x}|\mathbf{y})p(\mathbf{y}) \tag{2.6}$$

where $p(\mathbf{x}|\mathbf{y})$ and $p(\mathbf{y})$ are translation and language models, respectively. The *language model* provides the probability distribution of a sequence of words. Similarly, the *translation model* approximates the probability of a sentence $\mathbf{y}$ being the translation of another sentence $\mathbf{x}$. The model resulting in the combination of the language and translation models is the *noisy channel model*. It can be loosely interpreted as the noise $p(\mathbf{x}|\mathbf{y})$ being applied to $p(\mathbf{y})$, obtaining the noisy channel model $p(\mathbf{y}|\mathbf{x})$.

This statistical approach to the problem of MT was followed by IBM research [9]. They successfully proved that statistical methods could be applied to model the translation problem. A series of statistical models of increasing complexity were defined and hereafter called IBM Models. The simplest one, the IBM Model 1, is a lexical translation model that learns to translate by learning the *alignments* of source words to their respective translations in the target language. Essentially, it uses word level statistics obtained from a parallel corpora (a collection of source-target sentence pairs) to find pairs of words that co-occur, learning a correlation between them. An example of how this process is carried out can be seen in Figure 2.1. Here, we see how the model learns the co-occurrence between words that are closely aligned, even though it observes them in different contexts. It is important to note that this process is carried out over millions of sentence pairs to approximate alignment and co-occurrence probabilities precisely. The most probable alignment between source and target words is learned through the *Expectation-Maximization* (EM) algorithm [16].

However, researchers found that this system was lacking the natural bias of language: contextual information, high frequency n-grams, etc. Throughout the years, many approaches were tried. An incomplete list of them would include: syntax-enhanced systems [51], phrase-based systems [43], probabilistic smoothing techniques [55], tree-based models [1], re-ranking of hypotheses for decoding [70], etc. As time went by and these systems became popular, even though there were many advancements being made, there was no significant progress in terms of translation quality.

[1] An n-gram is a contiguous sequence of n tokens.

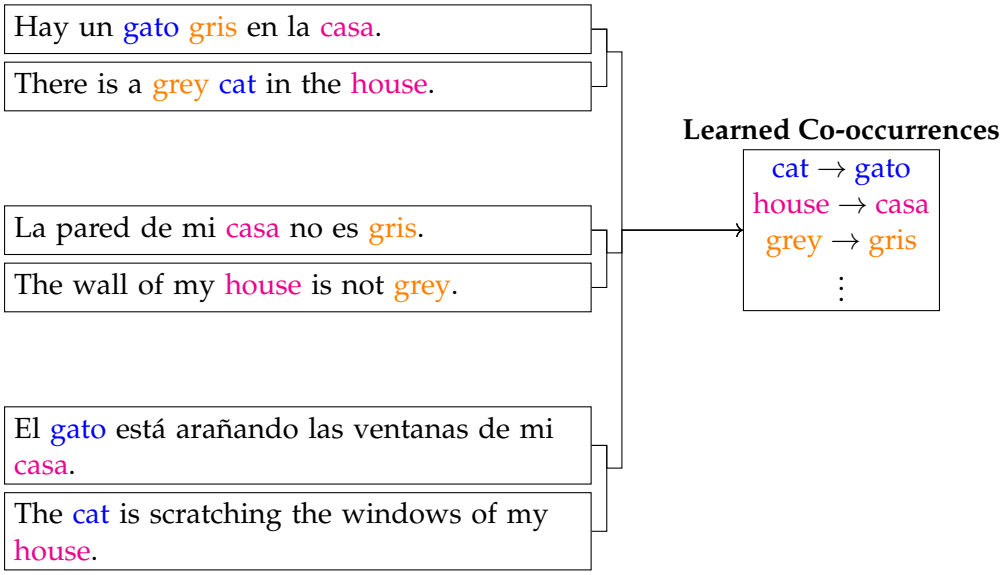


Figure 2.1: The IBM Model 1 co-occurrence learning process.

At this point, hardware was getting cheaper and faster, and there was another method starting to perform better than statistical methods in different ML tasks. Together with advancements in GPU technology, as well as the development of automatic differentiation techniques, neural based models were gaining popularity. They started to become useful in the field of automatic speech recognition (ASR), with a better approach to first modeling the acoustic signal and then the sequence of words. In the 2010s, Deep Neural Networks were being used for MT, and as time went by, they established what would be a new era for MT. These models were named *end-to-end* MT models, since they learned to approximate the posterior probability $p(\mathbf{y}|\mathbf{x})$ directly, as shown in Equation 2.5, instead of the popular SMT approach of decomposing it into parts as in Equation 2.6. Since these neural methods do not rely only on n-gram statistics and can observe whole sentences to see how important each word is when translating the phrase, they are able to learn to translate in a more generalized manner.

Nevertheless, both SMT and NMT methods had an important problem to deal with: *decoding*. Ideally, the model would always output the best possible translation for a source sentence. However, in reality there are various different target words that can effectively translate a given source word. Therefore, the last step of decoding is carried out, where a subspace of the translation space is searched for the best translation that the model can output. There are several methods for decoding, the most populars for MT being greedy decoding and beam search decoding. These decoding methods are different ways to instantiate the argmax operation from Equation 2.5 in practice without having to explore the full space of possible translations.

Now that we have introduced the historical background of MT, we will illustrate the history of neural networks in order to understand the inner-workings of the current state-of-the-art Neural MT systems that will be used throughout this work. Likewise, we will introduce the different decoding strategies used in MT and explain how quality is evaluated in MT.

2.3 Neural Machine Translation

As was mentioned in Section 2.2, throughout the years more computing power became available, which made it feasible to train deep networks using automatic gradient cal-

culations. Neural-based methods became more performant than SMT methods, revolutionizing the MT landscape through Deep Learning models. In order to understand the inner-workings of current NMT systems, we must first familiarize ourselves with the algorithms that cemented the foundations of these models. The following sections will introduce the main concepts surrounding neural networks, as well as explain the inner-workings of the current state-of-the-art architecture, the Transformer [80].

The Perceptron [65] was the first approximation of current neural methods. This first idea, together with the backpropagation algorithm [47, 66] and great advancements in computing power, led to multi-layer networks being applied as solutions to vast amounts of problems in the present.

One of the simplest and first multi-layer network that was tried is the Multi-Layer Perceptron (MLP), which essentially generalized the Perceptron to many layers. As can be seen in Figure 2.2, the MLP has an input layer, a set of hidden layers, and a final output layer. The hidden layers are particularly important. They apply transformations to the input projecting it into a higher dimensional manifold, and making it easier for the last layers to find a decision boundary that linearly separates the previously non-linearly-separable inputs. The idea was to create a universal function approximator by applying multiple compositions of affine transformations f and nonlinear functions g to the input:

$$\mathbf{y} = \left(g_{(L)} \circ f_{(L)} \circ g_{(L-1)} \circ f_{(L-1)} \circ \ldots \circ g_{(1)} \circ f_{(1)}\right)(\mathbf{x}) \tag{2.7}$$

The MLP's layers contain a variable number of *neurons*, the basic unit of neural networks, which are fully connected to all the neurons of the next layer. The expression for the next layer $\mathbf{h}^{(i+1)}$ of an MLP is the following:

$$\mathbf{h}^{(i)} = \begin{cases} g^{(i)}(\mathbf{W}^{(i)}\mathbf{x} + \mathbf{b}^{(i)}) & i = 0 \\ g^{(i)}(\mathbf{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}) & i > 0 \end{cases} \tag{2.8}$$

Where $g^{(i)}(\cdot)$ is any non-linearity, $\mathbf{W}^{(i)}$ is the weight matrix between layers i and $i+1$, $\mathbf{b}^{(i)}$ is the bias term, and $\mathbf{x}$ and $\mathbf{h}^{(i)}$ are the input and hidden layers respectively. Essentially, all layers apply a non-linearity to an affine transformation of the last layer. The affine transform maps the last layer onto a different space, and the non-linearity helps the MLP learn non-linear models.

Since MT is a classification task, where we classify the input into one of as many classes as words there are in the target vocabulary, we can apply the *softmax* activation function

$$\text{softmax}(\mathbf{z})_i = \frac{\exp z_i}{\sum_{i'} \exp z_{i'}}, \tag{2.9}$$

to the output layer to produce a probability distribution over all possible classes. For a network with L layers, the probability distribution is obtained as follows:

$$p(y_i|\mathbf{x}) = \text{softmax}(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}) \tag{2.10}$$

where y_i is the i-th word being translated. While this is not necessary, it helps to have a probabilistic perspective when dealing with deep learning models, since we can interpret probability distributions. To the intrigued reader we recommend [45] for a generalization of the probabilistic perspective onto energy based models.

The main group of optimization algorithms used to train MLPs and other neural models are *gradient based optimization* algorithms. What *gradient descent* attempts to do is to move the model's parameters towards ones with which the model's error is minimized. For this, it applies a forward pass, computing the model's hypothesis for a given input. Then, a backward pass is applied, where we use a loss function $\mathcal{L}$ to compute the

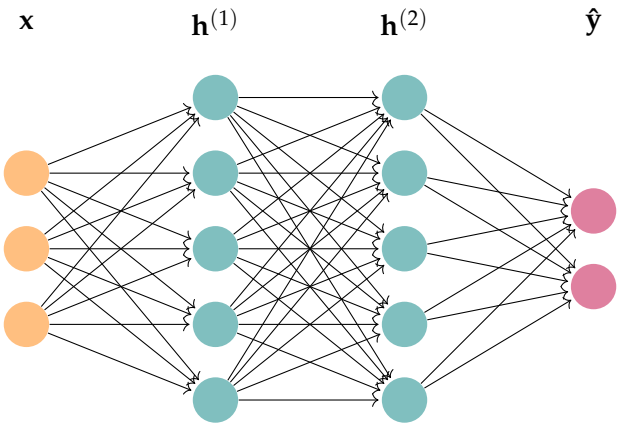


Figure 2.2: A simple multi-layer perceptron[2]. The edges represent weights between each layer. The blue nodes denote any non-linearity, while the purple ones represent typical classification activation functions such as the softmax.

error between the expected output and the model's hypothesis, and this error is propagated backwards from the output to the input layers, in order to tweak the weights of the model. We call this process *backpropagation*, and it iteratively computes gradients through the application of the chain rule, weighing the update of each parameter by how much it has contributed to the model's hypothesis. In Section 2.3.1 we illustrate how this is done automatically.

We define the gradient descent update rule for a set of parameters $\boldsymbol{\Theta}$ as

$$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \eta \nabla_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta}), \tag{2.11}$$

where η, the *learning rate*, specifies how big of a step to take in the negative gradient direction to reach the minimum without undershooting or overshooting. In practice, η can be specified statically or scheduled to rise and decay through time[3].

The loss function $\mathcal{L}$ measures the divergence between the MLP's predicted output $\hat{\mathbf{y}}$ and the golden standard $\mathbf{y}$ for a particular input $\mathbf{x}$. The main loss function for classification, and the one that will be used when training models in this work, is the *cross-entropy* loss:

$$\mathcal{L}_{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i \mathbf{y}_i \log \hat{\mathbf{y}}_i \tag{2.12}$$

However, gradient descent is not the only optimization method for neural network training. There are other methods such as particle swarm optimization [88], evolutionary learning through genetic algorithms [11, 74] and simulated annealing [64].

In practice, instead of calculating gradients after processing all the training data, the training data is divided into *batches*, and the gradient updates are done after processing a batch. This is the *mini-batch* variation of gradient descent. If these batches are also randomly subsampled from the training data, the algorithm is known as *Stochastic Gradient Descent*, SGD. Intuitively, SGD adds a noise ϵ to the traditional gradient descent updates, since it only observes a subset of the training data before updating the parameters.

$$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \eta \Big(\nabla_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta}) + \epsilon \Big) \tag{2.13}$$

[2] Adapted from https://latexdraw.com/drawing-neural-networks-in-tikz-short-guide/.

[3] Choosing the appropriate learning rate is very important. If one were to choose too small or too large of a learning rate, gradient optimization could take very long, or the minimum could be overshot. One could also explore the available learning rates using *hyperparameter optimization*.

The commonly used optimizers (Adagrad [19], Adadelta [84], Adam [38], and RMSProp [78]) are all variations of Stochastic Gradient Descent. The differences lie in the introduction of different terms into the update equations for faster or more robust convergence to the minimum.

Now that the basics of neural networks have been explained, we can move to how they can be scaled to *Deep Learning* models, i.e. multiple-layered models, to progressively obtain higher level representations of the input. The techniques explained in the following sections will be used for the development of state-of-the-art NMT systems.

2.3.1. Computational graphs

A computational graph is a tool used to define mathematical expressions, defined as a directed graph where the nodes correspond to mathematical operations, and the edges correspond to the inputs and outputs of a given operation. Given that a neural network is simply a set of mathematical operations applied to the input to obtain a prediction, we can model it as a computational graph. This way, the nodes represent different submodules of the network, and the edges show how the transformations made to the input are applied to obtain a prediction. The simple example shown in Figure 2.3 illustrates a typical computational graph, together with the gradient paths to the inputs. When networks have hundreds of modules, themselves composed of different submodules, constructing networks as computational graphs will allow for the use of *automatic differentiation*, a set of techniques used to calculate gradients automatically.

By iteratively applying the chain rule, automatic differentiation obtains derivatives of the loss function w.r.t. any parameter of the network. This way, automatic differentiation removes the worry of efficient neural network training. However, it also limits the network architecture: all operations must be partially differentiable, and the method for calculating the gradients must be defined. It is straightforward to see that otherwise, automatic differentiation would not work.

Most of the current neural network libraries, such as Tensorflow or PyTorch, construct computational graphs implicitly and use automatic differentiation to automatically calculate gradients[4]. Using these libraries, we can define a neural network by the operations applied to the input in order to obtain the output, delegating them with the construction of the computational graph and calculation of gradients.

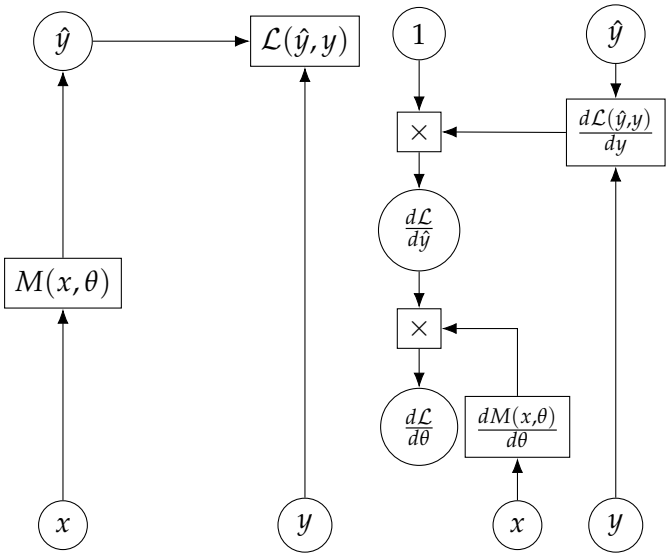
2.3.2. RNNs, The Encoder-Decoder Architecture, and Attention

MT can be modeled as a sequence to sequence (*seq2seq*) task, where the model's output is a sequence of words in the target language conditioned on the input, a sequence in the source language. This means that we can improve upon the standard MLP, creating a model that considers some number of previous or next words of the input as important contextual information when predicting which word to emit.

Recurrent Neural Networks

The simplest modification to apply to MLPs is to introduce cycles, such that the neurons have access to their previous states when emitting the next token. This modification is the essence of *Recurrent Neural Networks* [66, 29], where we modify Equation 2.8 to take

[4] `torch.autograd` in PyTorch, `tf.GradientTape` in Tensorflow



(a) The computational graph. **(b)** The gradient path.

Figure 2.3: Example of a computational graph and automatic differentiation[5].

into account different time steps

$$\mathbf{h}_t^{(i)} = g^{(i)}(\mathbf{W}^{(i)}\mathbf{h}_t^{(i-1)} + \mathbf{V}^{(i)}\mathbf{h}_{t-1}^{(i-1)} + \mathbf{b}^{(i)}), \tag{2.14}$$

where the hidden state $\mathbf{h}_{t-1}$ at the previous time step $t-1$ conditions the value of the current hidden state $\mathbf{h}_t$. The term $\mathbf{V}^{(i)}\mathbf{h}_{t-1}^{(i-1)}$ denotes the usage of the previous hidden state[6] transformed through a matrix $\mathbf{V}$.

In order to train an RNN, we can use a similar gradient-based method as the one used when training other types of models. However, since recurrent networks take into account a time component, this must also be considered at time of gradient computation. Thus, we use *backpropagation through time* [54], where we unroll the RNN through the time component by calculating gradients up to some time-step $t-h$, where t is the current time-step and h is the history, i.e. up to how far in the past to unroll the network. Note that in current deep learning libraries, this also done automatically.

While recurrent networks have some advantages for sequence tasks, the conditionality introduced by the cyclic connections also comes with the gradient vanishing and gradient explosion problems. When unrolling the network to calculate the gradient of the loss function at time t conditioned on an input in the past, a longer path from the input to the output can explode or vanish the gradient. The reasoning behind this is that when a gradient is much greater than 1, applying the chain rule multiple times yields bigger and bigger gradients for the following modules, destabilizing the training process; and when the gradient is close to 0, it becomes smaller until it is insignificant and no parameter updates are carried out.

Thus, there have been many variants to the traditional RNN that have been developed which attempt to solve this problem:

- Long Short-Term Memories (LSTM), a type of RNN that contain memory and three gates that control the flow of information [27]. The *input gate* stores some amount

[5] Adapted from https://atcold.github.io/pytorch-Deep-Learning/en/week02/02-2/.

[6] Time is understood as discrete: a time step t is the cardinal of the input (output) token that is being observed (emitted).

of the input information in the memory, the *forget gate* decides what information from memory should be forgotten, and the *output gate* produces a new short-term memory.

- Gated Recurrent Units (GRU), similar to LSTMs, but with two gates instead of three: an *update gate*, that determines the amount of information will be needed for the future, and the *reset gate* which decides how much of the hidden state to forget [14].

There are also other types of recurrent neural units, such as the Simple Recurrent Unit [46], Hopfield networks [29], etc.

Input Vectors and Embeddings

Neural Networks only work with numeric values, and since words are not composed of numbers, we need to figure out how to transform them such that we obtain an adequate numerical representation of them. Herein lies the motivation for this section.

One of the simplest ways to represent words as numbers is to simply map a number to each word in V, the vocabulary. In practice, given the vectorial nature of the network's layers' equations, instead of assigning a number to each word we compose a *one-hot encoding* representation of them. In one-hot encoding, we map the i-th word to a vector $\mathbf{x}_{OH} \in \mathbb{R}^{|V|}$, where each component x_j of $\mathbf{x}_{OH}$ is defined as follows:

$$x_j = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

meaning that the vector has a value of 0 everywhere but at the i-th component, the one associated to the word's position in $|V|$ when parsing the training data.

However, a one-hot encoding representation leads to vectors of high dimensionality. In order to combat this, we condense the representation by projecting the one-hot vectors to a lower dimension using a *word-embedding* matrix. Ideally, this projection should map similar words to a similar region of the embedding space. Since the model's outputs are also words, a similar process is applied to the output using another embedding matrix. Typically, these input and output embedding matrices are jointly learned.

Through the years, there have been multiple attempts to pre-train word-embeddings for ease of use in many NLP applications. These pre-trained embeddings are described in detail in [52, 59]. However, we will not be using them: NMT models that learn their own embedding matrices achieve better results. It is also important to note that since the inputs are represented as one-hot vectors, and these are vectorial representations of the cardinal of a word in the vocabulary, calculating the word-embedding in practice consists only of indexing a particular row in the embedding matrix.

The Encoder-Decoder Architecture

The encoder-decoder architecture is a type of seq2seq model, and it is the architecture that most current NMT systems use. As the name describes, it has an *encoder* component, a network that is responsible of encoding the information in the source by learning a representation that the second component, the *decoder*, will be able to decode into the emitted translation. Typically, these components are formed by LSTM or GRU cells, or Transformer layers, which we will describe in detail in Section 2.3.3, and are trained jointly. What follows is a brief introduction into one of the first encoder-decoder models used in NMT: the RNNSearch model as presented in [4].

In the RNNSearch model, the encoder is formed by bidirectional RNNs, *BiRNN* [67], a network that consists of forward and backward RNNs. The motivation behind this choice is that for some word x_i, ideally, the encoder should not only learn a representation based on the prefix $\mathbf{x}_{<i}$, but also consider the following words $\mathbf{x}_{>i}$. This is due to the fact that both the right and left context of x_i modify its meaning in the sentence.

Thus, the forward RNN $\overrightarrow{f}$ obtains a forward hidden state $\overrightarrow{\mathbf{h}_j}$ by reading the input sequence in the natural order $(x_1, \ldots, x_{|\mathbf{x}|})$.

$$\overrightarrow{\mathbf{h}_j} = \overrightarrow{f}(\overrightarrow{\mathbf{h}_j}, \mathbf{x}_j) \tag{2.16}$$

Conversely, the backward RNN $\overleftarrow{f}$ represents the reverse sequence $(x_{|\mathbf{x}|}, \ldots, x_1)$ by a backward hidden state $\overleftarrow{\mathbf{h}_j}$.

$$\overleftarrow{\mathbf{h}_j} = \overleftarrow{f}(\overleftarrow{\mathbf{h}_j}, \mathbf{x}_j) \tag{2.17}$$

Finally, the output of the encoder is formed by concatenating these hidden states, i.e. $\mathbf{h_j} = [\overrightarrow{\mathbf{h}_j}; \overleftarrow{\mathbf{h}_j}]$. Once $\mathbf{h}_j$ has been obtained, the decoder forms a context vector $\mathbf{c}_j$ by *attending* to different parts of $\mathbf{h}_j$. The attention mechanism that allows for this to work is introduced in the following section. This way, $\mathbf{c}_j$ together with the previous hidden state $\mathbf{s}_{j-1}$ and the (embedding of) the last emitted word y_{j-1} can be used to obtain the decoder's new hidden state:

$$\mathbf{s}_j = f(\mathbf{s}_{j-1}, y_{j-1}, \mathbf{c}_j) \tag{2.18}$$

This new hidden state $\mathbf{s}_j$ is transformed by some other fully-connected layer to get the conditional probability

$$p(y_j|\mathbf{y}_{<j}, \mathbf{x}) = g(y_{j-1}, \mathbf{s}_j, \mathbf{c}_j), \tag{2.19}$$

out of which we will emit the j-th word.

Attention

Introducing the attention mechanism into our architecture will enhance the model so it can choose how much each hidden state $\mathbf{h_j}$ contributes to the current translation, i.e. the importance of each input word for the current translation.

The attention mechanism can be described as a function that obtains a compatibility between a query vector $\mathbf{q}$ and a set of keys $\mathbf{K}$,

$$\alpha_{jk} = \text{softmax}(\mathit{Attention}(\mathbf{q}, \mathbf{K}))_j, \tag{2.20}$$

such that it can be used for a weighed sum over values $\mathbf{V}$

$$\mathbf{c}_j = \sum_k \alpha_{jk} \mathbf{V}_j, \tag{2.21}$$

essentially controlling the flow of information in our model.

In RNNSearch, the authors use $\mathbf{q} = \mathbf{s}_{j-1}$, $\mathbf{K} = \mathbf{h}_j$ and $\mathbf{V}_j = \mathbf{h}_j$ to obtain the context vector $\mathbf{c}_j$ as the weighed sum of the hidden states. This vector is what the decoder will use to emit the next translation.

$$\mathbf{c}_j = \sum_k \alpha_{jk} \mathbf{h}_j \tag{2.22}$$

While there are many types of attention, Bahdanau et. al. [4] propose

$$\mathit{Attention}(\mathbf{s}_{j-1}, \mathbf{h}_j) = \mathbf{v}_a \tanh(\mathbf{W}_a \mathbf{s}_{j-1} + \mathbf{U}_a \mathbf{h}_j), \tag{2.23}$$

where $\mathbf{v}_a$, $\mathbf{W}_a$, and $\mathbf{U}_a$ are trainable weights. However, in Section 2.3.3 we will see that the Transformer architecture uses scaled dot-product attention.

2.3.3. The Transformer

The Transformer architecture [80] was introduced in 2017, and it revolutionized the field of MT by achieving significant improvements both in quality and speed. It is currently the go-to model for MT and many more tasks. Presently, there are various large language models such as GPT-3 [10], its smaller open source variant GPT-NeoX-20B [8], and the newly released PaLM [15], Chinchilla [28] and OPT [87] models, all of which use a variant of this architecture, making the Transformer the state-of-the-art in NLP tasks. Additionally, Transformers are starting to dominate in fields where the tasks are modeled with non-sequential inputs, such as the Vision Transformer [18] in computer vision tasks.

The following sections will explain the inner-workings of the Transformer architecture in the way it was first introduced for NMT.

The Transformer Architecture

In a similar fashion to RNNsearch [4], the Transformer is also comprised of encoder and decoder layers. However, neither of these contain RNN components. In fact, these blocks are mainly formed by attention and fully connected layers. Figure 2.4 shows the full architecture as proposed in the original paper, and Figure 2.5 details the inner-workings of the encoder and decoder layers.

The source phrase is the input of the encoder side, as shown on the left of the figures. Here, the model represents each word with its word-embedding, and since word-embeddings by themselves do not account for positional information, positional encodings are also extracted from the phrase. Afterwards, the embedded representation of the phrase is fed into a stack of six encoder layers, where each layer transforms it using self-attention, residual connections and layer normalization modules, as shown in Figure 2.5. The multi-head self-attention layer learns how much of the context to take into account when learning a representation of a particular source sentence. Then, the result of the attention layer is added with a skip connection, and the parameters are normalized layer-wise, aiding in the training process. These submodules of the architecture are explained in the following sections. An MLP network obtains the embedded representation of the source input and maps it to another manifold, helping the model to have greater discriminative power. Afterwards, another process of adding and normalizing is applied. The target, which is the input to the decoder stack shown on the right of the figure, also goes through a similar set of transformations. However, each of the six decoder layers also obtains information from the last encoder layer through a cross-attention mechanism, as shown in Figure 2.5. This can be understood as the decoder layers obtaining the relevant information that relates the source phrase to its target. Finally, the model obtains its final representation of the target in the uppermost decoder layer. In training time, a softmax is applied and the token with maximal probability is emitted. This is in contrast to inference time where a decoding process has to be applied, and the target phrase with maximal probability is chosen to be emitted[7]. The main decoding technique, beam search, will be explained in Section 2.3.4.

There are two variants of the Transformer model. The main difference between these is the dimensionality of all the vectors that the model outputs (and therefore, all the matrices' dimensions too): The BASE model employs $d_{model} = 512$, while the BIG model

[7]When training in sequence tasks, instead of having the previous context of the decoder be what the model emitted previously, we apply *teacher forcing*, feeding it the real target for the previous translated words. This yields better results since the model uses the real translation as context, and allows us to omit the costly decoding process.

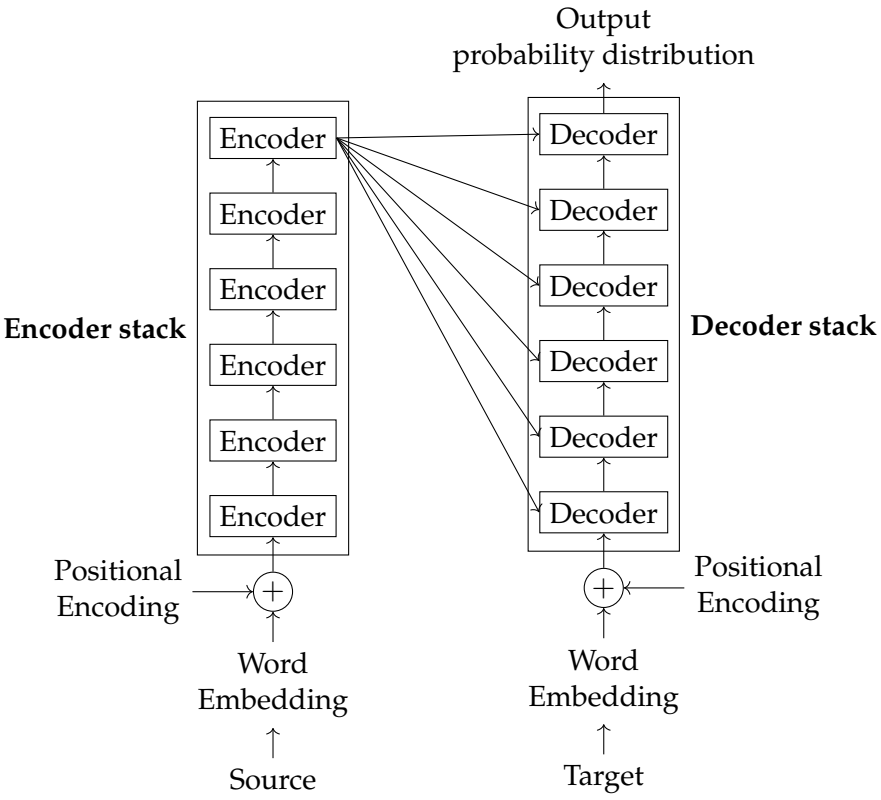


Figure 2.4: The full Transformer architecture as shown in [80].

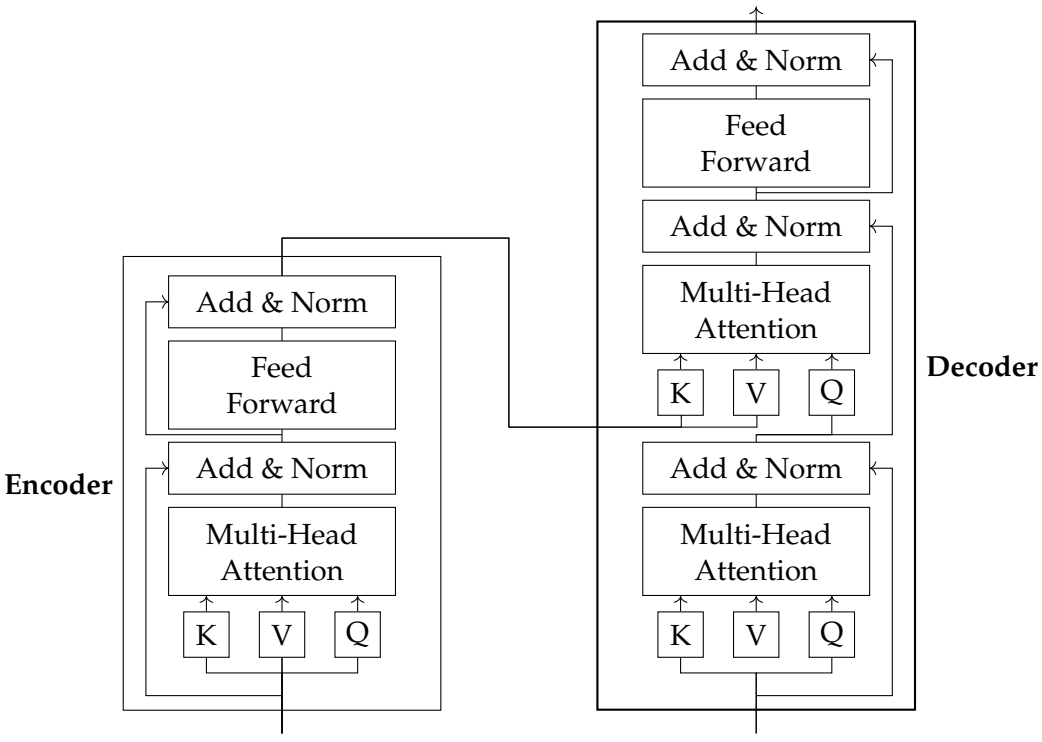


Figure 2.5: Transformer encoder and decoder layers as described in [80].

applies $d_{model} = 1024$. Both the encoder and decoder stack contain 6 encoder or decoder layers, respectively.

Layer Normalization

The Transformer architecture applies Layer Normalization [3] to the output of both the attention and feedforward sublayers. This method directly estimates normalization statistics of a layer's parameters,

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} h_i^l, \quad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(h_i^l - \mu^l)^2} \tag{2.24}$$

and then normalizes them accordingly:

$$LayerNorm(\mathbf{h}^l) = \frac{\mathbf{g}}{\sigma^l} \odot (\mathbf{h}^l - \mu^l) + \mathbf{b} \tag{2.25}$$

Here H denotes the number of hidden units in a layer, $\mathbf{h}^l$ is the output of said layer, $\mathbf{g}$ and $\mathbf{b}$ are the learned gain and bias vectors, and $\odot$ denotes the Hadamard product. In practice, the calculated statistics μ^l and σ^l are broadcast to have the same dimensions as $\mathbf{h}^l$ so that vectorial operators are used and the GPU's compute power is leveraged.

Residual Connections

Sometimes it can take a long time for model training to start, since the gradients are calculated over the random values to which the model's parameters are initialized. Residual Connections [26] attempt to facilitate the starting process of training, by introducing a *skip connection* from the input to the output non-linearity directly

$$ResidualConnection(\mathbf{x}, g) = g(\mathbf{x}) + \mathbf{x}, \tag{2.26}$$

where g is the layer on which the skip connection is being applied. In the gradient computation, and due to the fact that backpropagation is the iterative application of the chain rule, it can be seen from Equation 2.26 that the residual connection guarantees that the gradient of the above layer is passed to the one below through $\mathbf{x}$, since it's gradient is 1. This way, it is easier to optimize the residual mapping than the original one.

Multi-Head Self-Attention

In order to understand the Transformer architecture that was previously explained, we must describe how multi-head attention functions and why it is important.

As defined in Section 2.3.2, the attention mechanism needs a query $\mathbf{q}$, and a set of key-value pairs $\mathbf{K}$ and $\mathbf{V}$. In practice, we pack the set of queries into a matrix $\mathbf{Q}$.

The Transformer employs scaled dot-product attention. In this variant, the compatibility between $\mathbf{Q}$ and $\mathbf{K}$ is obtained by the dot-product operation:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}^T\mathbf{K}}{\sqrt{d_K}})\mathbf{V} \tag{2.27}$$

The scaling factor $\sqrt{d_K}$ is used so that the gradients are more stable.

The multi-head attention variant simply computes multiple attention mechanisms over the same queries and key-value pairs. In order to do this, each attention head's input

is a projected version of the queries and key-value pairs, so that the model can jointly attend to information from different representation manifolds, a property that single-head attention does not have:

$$\mathbf{head}_i = Attention(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{2.28}$$

The $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$ and $\mathbf{W}_i^V$ are parameter matrices that are learned in training time. To form the output vector of the multi-head attention sublayer, the attention heads' outputs are concatenated and projected back to the original subspace using $\mathbf{W}^O$, another learned projection matrix:

$$MultiHead(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Concat(\mathbf{head}_1, \ldots, \mathbf{head}_h)\mathbf{W}^O \tag{2.29}$$

The model proposed by the authors employs $h = 8$ parallel attention heads, using $d_K = d_V = d_{model}/h = 64$ in both the BASE and BIG variants, so the computational cost is similar to the single-head attention.

It is important to note that the naming distinction between an attention layer and a self-attention layer is only regarding the inputs the layer receives. Figure 2.5 shows that the self-attention layer in the decoder has $\mathbf{Q} = \mathbf{K} = \mathbf{V}$, meaning that in order to learn a representation for the input word x_i, the attention layer attends both to x_i and its surrounding words. This is also the same in the encoder layer's self-attention, where $\mathbf{V}$ is also the same as $\mathbf{Q}$ and $\mathbf{K}$, since there is no information leaking issue. However, the cross-attention layer in the decoder receives as input a $\mathbf{Q}$ matrix that is not the same as $\mathbf{V}$. Thus, the name change signifies whether the information that flows to the next layer is or is not in the query.

Embeddings

Similarly to other Deep Learning models, in the Transformer architecture the embedding layers that are inputs to the encoder and decoder stacks are learned, and the softmax is applied to the decoder's output to obtain a conditional probability over the output vocabulary. However, since self-attention layers attend to all of the words in the input, independently of their position with respect to that of the word to be emitted, there is no way for these layers to inherently represent positional information. Thus, the attention layers need to be aided by *positional encodings* in order to learn adequate representations of the inputs. An approach to this problem can be to simply map each position to a one-hot vector, and then learn a projection matrix $\mathbf{E}_p$, in the same fashion that the word embeddings are learned using $\mathbf{E}_v$, to represent positional information adequately. In practice, the position one-hot vector $\mathbf{p}_i$ and the vocabulary one-hot vector $\mathbf{v}_i$ would be concatenated such that only one embedding matrix $\mathbf{E}$ would have to be trained:

$$\mathbf{e}_i = [\mathbf{v}_i; \mathbf{p}_i]\mathbf{E} \qquad \mathbf{E} = [\mathbf{E}_v; \mathbf{E}_p] \tag{2.30}$$

However, the authors that presented the Transformer architecture proposed to model the position of the word using sinusoidal functions at different frequencies as a vector of d_{model} dimension. Essentially, these fixed encodings are modeled such that each component of the positional encoding vector is a sinusoid at a different frequency:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \qquad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.31}$$

where pos is the position and i is the dimension. An illustration with $d_{model} = 512$ and a sequence with 100 tokens is show in Figure 2.6. Here we observe how each row is different from the next, but close rows are more similar than those farther apart.

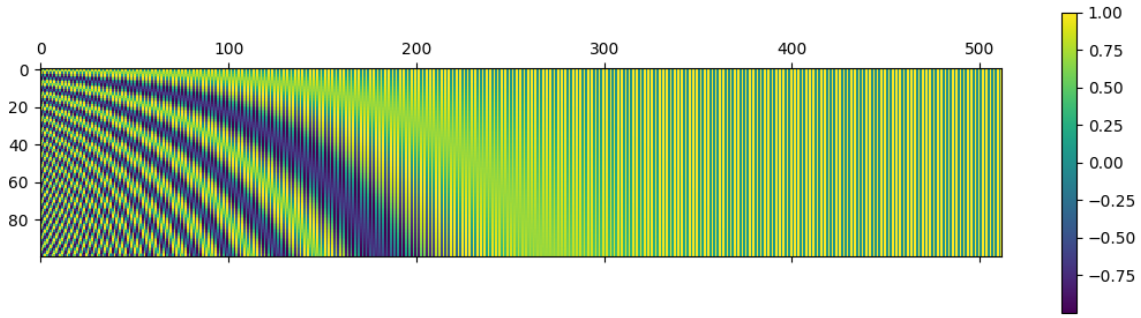


Figure 2.6: Positional Encodings. Row i is the position vector of the i-th token.

Once computed, these positional encoding vectors can be summed with the word embedding vectors. This summation can be interpreted as the positional encoding vectors "pushing" the word embedding vectors more towards some region of the subspace, grouping words that appear in similar positions throughout the training set. Once the input is formed this way, the self-attention layers can obtain positional information too.

Experiments in the original paper showed that using fixed positional encodings or learned positional encodings produced very similar results. The authors claimed that fixed positional encodings may allow the model to generalize to longer sequence lengths.

Weight Tying

Weight tying [62] is a technique that leverages the fact that both the input and output of the model is a word-embedding to greatly reduce the total number of parameters to train. The idea behind it is to share the embedding matrices of the source and target languages, as well as the embedding matrix of the softmax layer that is the output of the decoder, thereby having the model learn representations in of words in different languages in the same space.

Label Smoothing

When training models as big as Transformers, there can be a tendency for *overconfidence*. Overconfidence, the ability for the model to assign most of the probability mass to one of the classes in the output conditional probability distribution, is an important problem to tackle: there can be scenarios where multiple words could be translations of a given input word, so it is in our interest to flatten the probability distribution and not overly-favor any of these choices. Label smoothing is an attempt to redistribute the mass from high probability regions to lower ones, preventing zero probabilities and overconfidence. The technique consists in introducing noise to the label probabilities. Instead of using

$$\hat{y}_i = \underset{y_i \in \mathcal{Y}}{\operatorname{argmax}} \log p(y_i | \mathbf{y}_{<i}, \mathbf{x}) \tag{2.32}$$

we apply the label smoothing regularization technique to maximize:

$$\hat{y}_i^{LS} = \underset{y_i \in \mathcal{Y}}{\operatorname{argmax}} \left((1-\epsilon) \log p(y_i | \mathbf{y}_{<i}, \mathbf{x}) + \frac{\epsilon}{|\mathcal{Y}|} \right) \tag{2.33}$$

where ϵ is the noise introduced as a small constant, and $\mathcal{Y}$ is the target language vocabulary. In fact, if one were to wish not to use the label-smoothed variant of the probability distribution, simply setting $\epsilon = 0$ reverts p to the overconfident distribution.

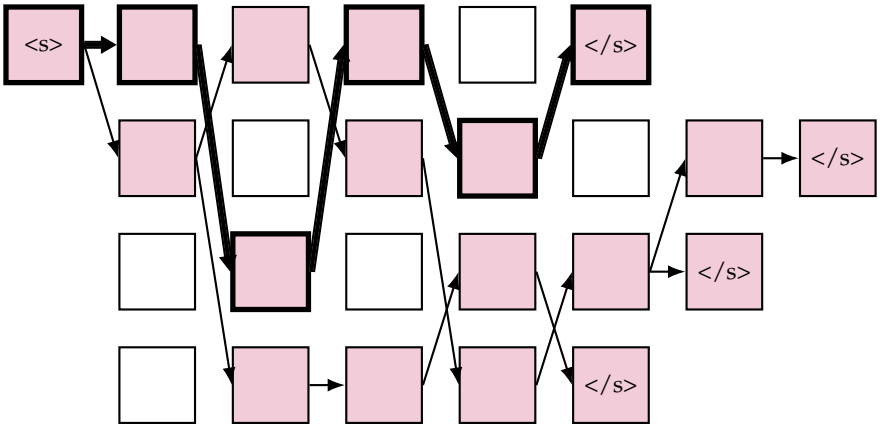


Figure 2.7: Search graph for beam search decoding using $\beta = 4$. Purple boxes denote tokens that are used in a hypothesis, while blank ones represent unused tokens. Thick borders and arrows denote the best hypothesis. Adapted from [41].

2.3.4. Decoding

We previously explained how MT systems emit the words in the target language that maximize the conditional probability distribution conditioned by the previous outputs and the source sentence. However, in inference time (when testing the model or using it in a production environment), issues can easily arise if this is the emission methodology followed: the model could simply output a wrong translation for some word, and since the next words to emit are conditioned on this one, the model could *hallucinate*, emitting incorrect translations.

Therefore, we need a better way to construct the translation to guarantee it is truly the most probable sequence. The *decoding* step is responsible for this.

There are many methods of decoding. The most naïve is *greedy decoding*, where the model emits the candidate with the best conditional probability, as was previously explained. However, why only output one word? Should it not be a better approximation to construct many different possible translation sentences and pick the most probable one? This is a better approach: *exhaustive search*, where all the words in the vocabulary are considered for all positions, forming a graph. In exhaustive search, the decoding step consists of searching over all possible translations to find the most probable one. However, it is clearly not a viable solution given its exponential complexity. We can modify exhaustive search by limiting the number of candidates for each position in the output. This is essentially what *beam search* does.

The beam search algorithm can be understood as a breadth-first search over a limited subset of possible words to emit. For each position where a word is to be emitted, the algorithm sorts and considers only the top β candidates, constructing a *beam*. The β parameter is known as the beam width. As shown in Figure 2.7, it uses the candidates in the beam as the previous context for the decoder model. When a beam is formed, beam search expands these states by continually using the members of the new beam to obtain the β best candidates to emit next, while also keeping track of which words were used as a previous context to obtain this new beam. In the end, the model's emitted translation is the one with the best joint probability.

2.3.5. Evaluation

The methods to evaluate the quality of MT systems has been a topic of debate among researchers, and it is still an open problem. There are three main methods of evaluation: manual evaluation, automatic evaluation using string-based metrics, and automatic evaluation with pre-trained models.

Manual evaluation has the advantage of being strongly reliable, since the translation quality is judged by human translators. However, it is clear that this method does not scale, given how unfeasible it would be to evaluate the outputs of new models every time they are trained.

Hence, different automatic string-based metrics have been developed. Automatic metrics compare the output of the system under evaluation with the expected output from the development and test sets, defined as the *reference translation*. A naïve metric can be the *precision*, that is, the percentage of correctly output words in the model's output w.r.t the reference translation. This metric does not take contextual information into account and does not penalize for short sentences, making it a bad choice for MT evaluation.

Thus, there are many automatic string-based metrics that also consider contextual information. Out of the many existing metrics, the prevalent ones in current research are BLEU, chrF, and TER. In the following sections we briefly explain how they work.

As for automatic evaluation with pre-trained models, there are many proposed metrics every year. We refer the inquisitive reader to [63] for the COMET metric, a pre-trained metric that is strongly correlated with human evaluation. However, we will only be using string-based metrics in this work.

BLEU

BLEU [58], the Bilingual Evaluation Understudy metric, measures the average precision of different n-grams at different levels.

$$AveragePrecision(N) = \frac{1}{N}\sum_{n=1}^{N} log p_n \tag{2.34}$$

It uses a clipped precision p_n that requires an n-gram to appear the same number of times both in the reference translation and in the candidate translation.

$$p_n = \frac{\text{matching n-grams}}{\text{total n-grams in output}} \tag{2.35}$$

It also penalizes short output sentences with the following brevity penalty.

$$BrevityPenalty = \begin{cases} 1 & |output| > |reference| \\ \exp\left(1 - \frac{|output|}{|reference|}\right) & |output| \leq |reference| \end{cases} \tag{2.36}$$

Traditionally, only n-grams up to 4-grams are considered. Thus, the full metric is defined as follows:

$$BLEU(4) = BrevityPenalty \times AveragePrecision(4) \tag{2.37}$$

The metric is calculated concatenating all test sentences, and is usually multiplied by 100 for better readability (the original score being in the interval $[0, 1]$), where bigger is better.

TER

Another commonly used metric is the Translation Edit Rate [72], which measures how much of the output would have to be edited in order to obtain the reference translation. The editing operations considered are: insertion of a word, deletion of a word, substitution of a word by another, and movement of a contiguous sequence of words to another region of the output.

$$TER = \frac{\text{word-level edit distance}}{|\text{reference}|} \tag{2.38}$$

The reported scores are also multiplied by 100 for better readability, and lower is better.

chrF

The character F-score, *chrF* [60], is an automatic metric based on character n-gram *precision* and *recall* scores. It is both language and tokenization independent, and essentially adapts the Fβ-score t a character-level scenario:

$$chrF\beta = (1+\beta)^2 \frac{chrP \times chrR}{\beta^2 \times chrP + chrR} \tag{2.39}$$

Here, $chrP$ is the character n-gram precision, obtained as the percentage of n-grams in the hypothesis that also appear in the reference. Similarly, $chrR$ is the character n-gram recall, the percentage of n-grams in the reference present in the hypothesis. As is also true in the typical F-score, the β parameter controls how much more importance is given to precision over recall, and if $\beta = 1$, they have the same importance. In this work, we will be using $\beta = 2$, i.e. the chrF2 metric.

CHAPTER 3
Data Processing

In this chapter, we will introduce the corpora available for training and evaluating MT models. We will explain the typical steps of data collection, processing and preparation that are used for current state-of-the-art models. Ideally, this data pipeline would yield data of good quality for the model to learn representations that accurately characterize textual information. More specifically, we will mention the general domain and in-domain corpora we will use to train NMT systems, while also describing the components that comprise a typical processing pipeline for text.

3.1 Corpora

In order to train NMT systems such as the Transformer architecture explained in Section 2.3.3, we need to feed it with millions of bilingual pairs (bitext), that is, source-target pairs where the target is the translated version of the source into a different language. Historically, there have been private and public initiatives to obtain bitexts, some of which have contributed to the release of free and open parallel corpora available on the web. These bitexts have been generated by professional translators, experts or volunteers. Therefore, the different data collection processes yield bitext in many language pairs, for various domains, and in differing levels of quality. Previously, the bitext was made available through public institutions such as the ONU, the EU Parliament, etc., but it is important to mention the endeavor to centralize the availability of popular parallel corpora in the OPUS [77][1] platform.

There exist two types of data: *general domain* and *domain specific*. Generally in NLP, general domain data refers to a corpus that contains text from various domains, one of which may be a domain of interest to us; where as the in-domain corpora are comprised of bitexts that are more closely related to the domain of interest for the final application. In this work, we will be using general domain corpora to obtain translation systems that perform well on average for most content, and then carry out a process of domain adaptation, where the in-domain data will be leveraged such that the model performs better when given sentences from that specific domain. This is in contrast to general domain translation systems such as Google Translate, DeepL, etc., where the main focus is to cater translations to the general public, thereby training models to perform well in general domain text. We will go on to briefly mention the general domain and in-domain corpora that will be used in this work.

[1] `https://opus.nlpl.eu/`

Corpus	Bilingual pairs	Words	
		English	French
WikiMatrix [68]	2.7 M	57.8 M	63.1 M
WikiMedia [76]	1.0 M	24.1 M	25.8 M
UNPC [89]	30.3 M	658.4 M	816.4 M
Giga Fr-En [76]	22.5 M	575.8 M	672.2 M
ParaCrawl [6]	216.6 M	3.7 G	4.1 G
EUBookshop [71]	10.8 M	224.6 M	244.5 M
CCAligned [21]	15.6 M	156.7 M	171.1 M
DGT-TM [75]	4.9 M	86.3 M	95.4 M
Europarl [40]	1.2 M	28.6 M	29.9 M
Europarl-ST [37]	96.5 K	2.3 M	2.6 M
News Commentary [76]	3.2 M	70.7 M	76.6 M
CommonCrawl[2]	0.1 M	4.1 M	4.7 M
Total	309.0 M	5.6 G	6.3 G

Table 3.1: Contents of the general domain corpora for the language pair Fr-En, where K=10^3, M=10^6 and G=10^9.

3.1.1. General Domain Corpora

General domain corpora refer to parallel text sourced from any topic and in any tone and style of the source and target languages. These texts tend to be used as training data for any NMT system used for any purpose. However, there are many types of general domain parallel texts, and in order to obtain an NMT system of high quality we must consider their differences. Many corpora from various sources have been compiled over the years. Not only does this mean that the general content of the texts could be outdated for certain applications, but it also implies that we must take into account tonal differences; source domain, which can range from a variety of topics; formality, given that there are differing tone formality levels between parliamentary speech transcriptions, movie subtitles, internet blogs, etc.; and other properties of the corpora.

After thorough consideration, the corpora that will be used throughout this work can be seen in Table 3.1, where we include datasets that contain millions of bilingual pairs. Most importantly, ParaCrawl contains one order of magnitude more sentences than others, meaning that the biases within its text will bias our model the most. It is also important to note how French sentences consistently contain more words than their English counterparts, as can be seen by the higher values in the French words column. This information will be of use to us in the streaming scenario. In the following sections we will briefly describe the contents of these corpora.

Sourced from the web

Many of the corpora that will be used in this work have been obtained from web-based sources, and the translation of the text was readily available paired with its counterpart in the original language. The corpora of this fashion that we will use in this work are:

- WikiMatrix and WikiMedia, corpora that contain parallel data extracted from Wikipedia articles and their respective translations,
- Common Crawl and CCAligned, bitext obtained by crawling the web and checking for languages and translations via the URLs,

[2] https://commoncrawl.org

- Paracrawl, one of the largest parallel text collection projects, which continually crawls the web, aligning and processing bitext,
- Giga Fr-En, a parallel corpus containing 10^9 words from phrases crawled from Canadian and European Union sources.

Since these corpora were obtained as a result of applying massive data collection schemes, this also involves the automation of the cleaning, filtering and processing steps needed to obtain parallel data of high quality. This process is thoroughly explained in Section 3.2.

From non-web sources

In contrast to previously mentioned corpora, these bitexts are not crawled from the internet. Rather, they are obtained via transcriptions of parliamentary meetings, books, news, etc. Thus, significant effort was made to translate these texts and obtain high quality translations, either by professional translators, or volunteers who showed fluency in both languages. These include:

- UNPC, the United Nations Parallel Corpus, is the first parallel corpus composed from United Nations documents. It consists of manually translated UN documents from the year range 1990 to 2014 for many languages.
- EUBookshop contains parallel data from the EU Bookshop, an online archive of publications from European institutions.
- Europarl and Europarl-ST include bitext extracted from the proceedings of the European Parliament.
- DGT-TM is a parallel corpus published by the European Comission's Directorate-General for Translation.
- News Commentary compiles translated news obtained by the CASMACAT[3] project.

WMT

The WMT (Conference in Machine Translation) holds annual MT conferences and features shared MT tasks to incite the advancement of the field. In order for the WMT to correctly evaluate the entries to the shared tasks, they must offer test corpora of high quality. It is standard practice in the field of MT to evaluate systems on the WMT translation task's test sets, since they are regarded of better quality than most other corpora. In this work we will be using the test sets of the translation tasks of the WMT13[4] and WMT14[5], which were extracted from the News Commentary corpora, as general domain development and testing sets when evaluating system performance.

3.1.2. In-Domain Corpora

If we wish for the NMT systems to perform well for a particular domain, we also need in-domain data to carry out a process of domain adaptation as will be described in Chapter 4. In our case, the NMT systems must be adapted to the domain of high energy physics from CERN. The following sections detail two datasets of different nature from the CERN domains.

[3] https://www.casmacat.eu/
[4] https://www.statmt.org/wmt13/translation-task.html
[5] https://www.statmt.org/wmt14/translation-task.html

	Objects	Sentences	Tokens	Vocab
Titles	519 K	519 K	4.6 M	228 K
Abstracts	130 K	652 K	15.6 M	393 K
Documents	296 K	48.9 M	1.1 G	10.9 M
Overall	945 K	50.0 M	1.1 G	11.0 M

Table 3.2: Contents of the CDS monolingual English corpus, where K=10^3, M=10^6, G=10^9.

CERN Document Server Monolingual Data

One corpora that will be used throughout this work for domain adaptation into the domain of physics is the CERN Document Server (CDS) monolingual corpus. Since the NMT system that we are developing is in the direction Fr→En, we can use the English monolingual CDS corpus for domain adaptation.

The CERN Document Server[6] contains published articles, preprints, scientific papers, reports, PhD theses, books and proceedings, etc. Out of these, the articles, preprints, books and proceedings were collected in the form of PDF files containing monolingual text in English to compile the CDS monolingual corpus.

Apache Tika[7] was used to extract plain text from source PDF files. Afterwards, a set of filtering techniques were applied to the text to obtain a higher quality version of the corpus. Table 3.2 shows a summary of the text that can be found in the CDS corpus. From here we observe that the corpus contains 50 million sentences in English, which our MT systems will leverage to learn better representations of our target language.

CERN News

The CERN News parallel corpus is another in-domain corpus that we will be using throughout this work. This corpus compiles news entries in the CERN news website[8], containing news in topics classified as Accelerators, At CERN, Computing, Engineering, Experiments, Knowledge sharing, and Physics. The bitext was crawled and aligned with the `hunalign`[9] sentence aligner [79], and split into training, validation and testing sets. The latter two were manually edited to obtain an evaluation set of higher quality. Regarding the splitting of the corpus, the training set is composed of news articles from 1970 to September 2021, the validation set contains documents from September to December 2021, and the remaining documents until May 2022 are compiled into the test set. Table 3.3 shows a set of summary statistics of the CERN News corpus. Interestingly, the observation that sentences in French tend to contain a higher quantity of words also holds in this case. Additionally, we note that we have 56 thousand in-domain source-target pairs with which we can use for the domain adaptation of general domain MT models to the domain of high energy physics.

3.2 Processing Steps

In order to feed the source and target sentences to the model so it can generalize better, we would like for the vocabularies of both languages to be small, so that there is less computing power needed to process them. Thus, we first need to *filter* the text, and then

[6] http://cdsweb.cern.ch/
[7] https://tika.apache.org/
[8] https://home.cern/news
[9] https://github.com/danielvarga/hunalign

Dataset	Bilingual pairs	Words		Documents
		English	French	
Training	51.9 K	841.9 K	909.4 K	3409
Validation	2.2 K	331.7 K	405.3 K	144
Testing	1.8 K	274.0 K	333.3 K	128
Total	55.9 K	1.5 M	1.7 M	3681

Table 3.3: Statistics of the CERN News parallel corpus, where K=10^3, M=10^6.

apply a set of transformations that will prepare the input: *tokenization*, *truecasing*, and *subword segmentation*. In this section we will describe the specifics of the aforementioned transformations.

3.2.1. Filtering

Data filtering is one of the most important cleaning steps for any NLP task. It denoises the data, removing text of lower quality, so that the model can learn the language in a more effective manner. This is especially interesting to us. Since most of the training corpora has been obtained by crawling the web, there is bound to be noise within it. Noise in text data for the translation task is manifested in multiple ways: text in a different language than the pair of interest to us, spelling and grammatical errors, shortened translations that might not correctly convey meaning, colloquialisms, abbreviations, phonetic substitutions, translterations, etc. Given that human language can be inherently noisy in our attempt to convey meaning using words, this is already enough of a challenge for our NMT systems. Additionally, we want the training data to be of high quality, not containing phrases that might veer the system's training towards an undesirable representation of the language pairs. Usually, filtering noisy data helps the model learn better representations, and thus perform better in inference time.

Filtering is defined as the process of going through the corpora and selecting which set of sentences is of good enough quality to be kept as part of the training corpus. There are many text filtering techniques that are applicable to both monolingual and bilingual corpora. These can include language identification, filtering by sequence lengths, deescaping unicode characters, cleaning the HTML/XML/etc. tags from documents, etc. In this work, we will be using the monolingual filtering techniques of *language identification*, filtering by *sequence length*, and one of the most used parallel data filtering techniques, discarding by a *source-to-target ratio* threshold:

- **Language Identification** is the process of identifying whether the parallel data contains text in a language other than the source or target, in their respective sides. This can usually be done either with ad-hoc methods of language identification, where a set of programmed rules are checked; or with the usage of language identification models. These models can be statistical models based on n-grams, or neural models.
- **Sequence Lengths** can be used to filter parallel data where either the source or target side sentence is very long. Very long sentences (say, 150+ words) can be a product of incorrect grammar, the author of the text being very verbose and redundant, or other causes. This means that we can filter the text by simply defining a threshold of sequence length to a particular value.
- **Source-to-target ratio** is the ratio of source words to target words. If this ratio is very small or very large, it usually implies that the translation either did not

convey as much meaning as the source, or that it was unnecessarily extensive and redundant. Thus, we filter source-target pairs where their ratio is above a threshold.

More precisely, the ParaCrawl corpus mentioned in Section 3.1.1 was filtered using the `bitextor-bicleaner`[10] and `bitextor-bifixer`[11] tools to be considered of good enough quality for translation tasks. Additionally, the CDS monolingual corpus was cleaned and filtered using `NLTK`[12], `langdetect`[13], and other software, in order to eliminate unnaturally short and long phrases, remove non-English text and unnecessary unicode-style encoded characters. Finally, there was a cleaning and filtering process applied to the CERN News corpus, where the validation and testing sets were manually reviewed so that the parallel text was correctly aligned, and a set of defined undesirable patterns that were observed in many news articles were removed.

3.2.2. Tokenization

The first and arguably most important step in the processing of any text is *tokenization*. Tokenization consists of identifying the different *tokens* that form a particular sentence, in such a way that said tokens are always the same. This process will allow for certain constructs such as `hello`, `hello.`, `hello?`, and other variants to all map to the same token `hello`. Clearly all of these constructs contain the word `hello`, so we would be wasting valuable space if we included them as different entries in the vocabulary. Instead, considering the word and the punctuation marks as separate tokens will accumulate information that can appear in slightly different forms into one token, thereby also making the vocabulary smaller and easier for the model to digest. In this work, the `Moses` [42] tokenizer will be used. An example of tokenization is shown in Figure 3.1, in which punctuation marks are separated from words.

3.2.3. Lowercasing and Truecasing

Once tokenization has been applied, lowercase and uppercase variants of the same word still exist in the corpus. Should `hello`, `Hello` and `HELLO` be considered as different words? Clearly not. To solve this problem, we could apply *lowercasing*, where the text is normalized into its lowercase version, or *truecasing*, where each token is replaced by its cased variant with highest frequency in the corpora. Lowercasing can be a very fast process, but it yields the disadvantage of the model disregarding capitalizations that might provide contextual information (e.g. formal pronouns in Spanish, Italian, and other Latin-descendant languages; nouns in the German language; or any named entity more generally). In order to keep said contextual information, we will be using truecasing in this work by training a truecase model, where the cased variants of highest frequency of all tokens are found, and the model replaces all occurrences of the tokens into their variants with maximal frequency of appearance in the corpus. More specifically, the `Moses` [42] truecaser will be used on all corpora. Figure 3.1 illustrates an example of truecasing.

3.2.4. Subword Segmentation

Once tokenization and truecasing has been applied, we could already feed the data to our MT model and start training. However, some issues might arise in inference time: what

[10] https://github.com/bitextor/bicleaner
[11] https://github.com/bitextor/bifixer
[12] https://www.nltk.org/
[13] https://github.com/Mimino666/langdetect

happens when the model is met with an input token it has not seen before, and therefore does not know how to translate? The naïve approach is to output a special `<UNK>` token denoting that said token is unknown to the MT system. Another, to simply back-off into a dictionary, is also an option that can be easy to implement. However, it would be more convenient if the model could attempt to approximate a translation of a rare token, using its learned knowledge about the source-to-target mappings. Here is where *subword segmentation* comes in. The usage of subword segmentation algorithms has proven to be crucial for elevating NMT systems' performance, becoming a necessary step in the data processing pipeline. In this section, we will describe two popular variants for subword segmentation in MT and NLP in general: Byte-Pair Encoding (BPE) and `SentencePiece`.

Byte-Pair Encoding

BPE, *Byte-Pair Encoding* [23, 69], is an algorithm that attempts to split words into subwords, by transforming rare words into frequent subwords, following our intuition that unknown words could be translated by translating the subwords they are made up of. BPE works by first splitting all the text into a sequence of the smallest tokens possible: characters. In this step, a special character is added to denote the end of a word. Then, a number of *merge operations* are iteratively applied, where the most frequent consecutive token pairs are replaced by their concatenation in all the corpora. This special character enforces that no merge operation is applied by concatenating the suffix of a word with its consecutive word's prefix.

In practice, the merge operations are learned as a table, and once learned, they are applied throughout the full corpora. As shown in Figure 3.1, segmented words will appear with the `@@` suffix so the original text is easily recoverable. This way, if the model is encountered with a rare word for which it has not learned a translation, it can split the word into subwords and concatenate their translations to form its hypothesis. `subword-nmt`[14] is an open source implementation of the BPE subword segmentation algorithm.

SentencePiece

`SentencePiece`[15] [44] is another subword segmentation algorithm that attempts to capture the most frequent subwords that appear in text, while also taking into account the diversity of subwords to maximize the number of words that can be generated, and minimize the number of redundant subwords. It is an unsupervised text tokenizer (and detokenizer) that implements subword segmentation algorithms such as BPE or unigram language models. Currently, it is considered as the alternative to the Byte-Pair Encoding implementation offered by `subword-nmt`, and tends to be the preferred option in the field.

As opposed to `subword-nmt`, its number of tokens is predetermined, and more importantly, it does not require tokenized data. Instead, it treats whitespace as another symbol, which allows it to segment raw text. In order to do this, it uses the unicode character U+2581 (denoted as _ in Figure 3.1), which it adds as a prefix to all words before segmenting them. `SentencePiece` also trains its own (de)tokenization models from the raw data, and since it treats sentences as sequences of characters (as opposed to sequences of words in `subword-nmt`), it can be used in a language agnostic manner.

As far as its implementation details are concerned, `SentencePiece` is comprised of four main components: the *normalizer*, which standardizes the raw text into their Unicode codes (i.e., sequences of type U+ABCD); the *trainer*, in charge of training the segmenta-

[14] https://github.com/rsennrich/subword-nmt
[15] https://github.com/google/sentencepiece

Raw text	`Thank you, Mr Segni, I shall do so gladly.`
Tokenization	`Thank you , Mr Segni , I shall do so gladly .`
Truecasing	`thank you , Mr Segni , I shall do so gladly .`
Byte-Pair Encoding	`thank you , Mr Se@@ gn@@ i , I shall do so gl@@ ad@@ ly .`
Raw text	`Thank you, Mr Segni, I shall do so gladly.`
Truecasing	`thank you, Mr Segni, I shall do so gladly.`
SentencePiece	`thank_ you ,_ Mr_ Seg ni ,_ I_ shall_ do_ so_ glad ly ._`

Figure 3.1: Text processing pipelines applied to a phrase. Underlined sections show the changes made at each processing step. For SentencePiece, only non-end-of-word added characters are underlined.

tion model by building a vocabulary based on the raw text; and the *encoder* and *decoder* components, which are responsible for applying the learned vocabulary of subwords to the raw text, to either obtain a segmented representation of the phrases or to return to the raw text from its segmented version, respectively.

In our case, we will be applying truecasing before `SentencePiece`, since it is undesirable if the same subword with different casing appears in the learned vocabulary. This difference between the processing pipeline and the one we will use with `subword-nmt` can be seen in Figure 3.1.

CHAPTER 4
Domain Adaptation

Since we are developing an MT system that we know will be used in a particular domain, it would be ideal for the probability distribution that the model learns to be slightly biased towards terms in this domain. The process by which the model learns this bias is *domain adaptation*.

There are many applications of translation systems that require models to have a better understanding of a particular domain, to incorporate terms belonging to said domain into the vocabulary, and to therefore have a domain bias towards said terms. There are domains for which significant effort has been made to obtain in-domain parallel data for the particular language pair (and domain) of the NMT system we are training. However, most of the time, in-domain data is not available. In these scenarios, we have two alternatives: invest a considerable amount of time and resources in the elaboration of a high quality in-domain parallel corpora for the language pairs we are interested in, or generate synthetic parallel data by *backtranslating* large amounts of monolingual text in the target language.

Given that we have both monolingual and bilingual in-domain text available, we will opt for fine-tuning by leveraging parallel corpora obtained through backtranslation, as well as fine-tuning on existing in-domain bitext. In this chapter we will explore the main techniques that we will use to adapt a pre-trained MT system into a particular domain. More specifically, we will describe the process of backtranslating monolingual text, a current state the state-of-the-art technique, as well as explain the process of fine-tuning a pre-trained MT model.

4.1 Backtranslations

Throughout history many state-of-the-art MT systems were only trained on parallel text, without a way to leverage monolingual data, of which there is a much higher availability. However, currently many state-of-the-art MT systems use backtranslations, i.e. synthetic parallel data where the target text is the original monolingual corpus and the source is the translation obtained by a reverse model. The idea is to translate in the opposite direction in order to maintain the quality of the target text for future training. Using backtranslations can be interpreted as enhancing the model's internal and implicit language model of the target, since it can observe more high quality data of the target language. This is what makes backtranslations the current state-of-the-art [48, 20]. Note that the application of backtranslations to be used as a domain adaptation technique was not of main concern to the field, whereas its application to obtain in-domain parallel text is very prevalent nowadays.

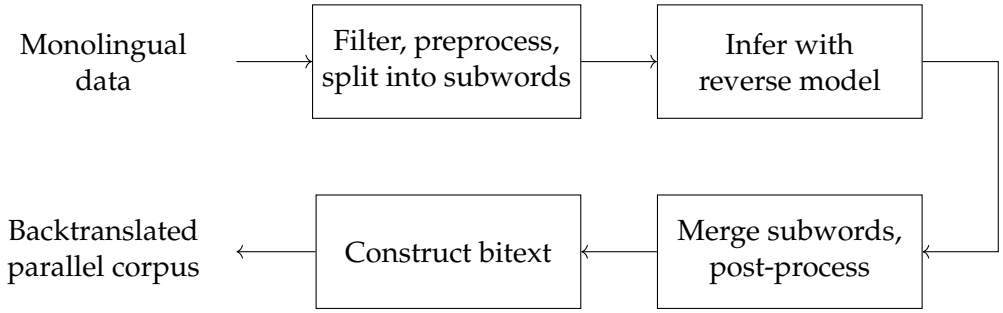


Figure 4.1: The process of obtaining a backtranslated corpus from monolingual data.

Clearly, training systems on general domain bitext would yield general domain systems. In order to obtain an MT system with high-quality performance on an in-domain test set we can backtranslate the in-domain monolingual text that we have available, and then use the synthetic bitext to adapt a model to this domain.

Figure 4.1 illustrates the steps to take in order to obtain synthetic parallel data in the form of backtranslations. Firstly, we must obtain the monolingual data and the translation model in the reverse direction. In our case, since we have monolingual data in English and the system that will make use of the backtranslated data translates in the direction Fr→En, the model to be used for backtranslating must be in the direction En→Fr. Next, we apply the same processing transformations to the monolingual data as the ones that were applied to the training data of the reverse model. This must be done so the reverse model can correctly understand and translate the monolingual data it is given in inference time. In this step we can also apply some filtering techniques. Afterwards, we feed the monolingual data to the reverse model, obtaining the hypotheses in the source language of our model. Finally, we post-process the model's output such that the translated text is in a user-presentable format, and construct the synthetic parallel corpus.

Now that we have a corpus of backtranslated data, we can opt to fine-tune an existing general domain NMT model using a subsampled version of the backtranslated corpus, or train from scratch by mixing the backtranslations together with general domain corpora.

4.2 Fine-tuning

Fine-tuning is a popular technique, typically used to adapt a general model to a particular task. In the context of MT, this technique is applicable when we have access to bitext in the particular domain and language pair that we are interested in, as well as to a general domain translation model for said language pair. We can fine-tune the model to improve its performance on in-domain test sets.

Fine-tuning can be framed as a *transfer learning*[1] technique, where we tune the parameters of an existing model using training data that defines a particular task. It is one of the most popular transfer learning approaches, since there are currently many well-performing general models, and an enormous amount of small datasets defining fine-grained tasks. A popular example of transfer learning includes the fine-tuning of a language model such as BERT [17] to a particular task such as question answering, text summarization, or story generation.

[1]Transfer learning refers to a set of techniques that attempt to transfer the knowledge of a model trained for some task A onto a different, possibly similar, task B, such that the model can use its learned knowledge to generalize on B with minimal training.

There are many approaches to fine-tune a pre-trained model: freezing all the layers except for the topmost classifier; freezing the top K layers of the model; training on all of the model's parameters, but using different learning rates on different layers (bigger ones on layers closer to the output); modifying the architecture by adding and training additional layers; or continually training the model without freezing any layer. Examples of these techniques can be found in [31, 30, 83, 56, 39]. These fine-tuning techniques attempt to tackle the problem of *catastrophic forgetting*, a phenomenon where a neural model quickly forgets what it previously had learned when it observes new information.

In this work, we will be using the fine-tuning approach of training on all of the model's parameters with the last learning rate of the scheduling procedure at training time, in order to adapt the network's learned information to the domain of physics. In Section 6.3 we describe how the fine-tuning process was carried out practically, as well as the results of evaluating on in-domain datasets.

CHAPTER 5
Streaming Machine Translation

Translating phrases that the model has full access to is not the only scenario where translation is applied. What about environments where a stream of text output by an ASR system must be translated? Or live chat environments where the system should translate text typed by the user in real-time with low latency? This is the more complex problem of *simultaneous translation*, where an MT system generates translations before having read the full source sentence. Simultaneous MT introduces the challenge of translating in real-time. It has many applications, and is especially difficult due to the latency constraint. While offline MT is concerned with translating text and maintaining as much of the meaning as possible, it does not have any latency requirements, we are only interested in the quality aspects of the translation. However, simultaneous MT, as it is currently understood and carried out, is done on a sentence-level. We can define *streaming translation* as a similar problem to that of simultaneous translation, but where we observe a stream of source tex, of potentially infinite length, and emit its translation in the target language in another potentially infinitely-long stream. In this scenario, we have the option to leverage previously observed and translated phrases as historical contextual information, thereby potentially improving the translation quality of following input sequences [34].

There are many differences in inference time between an offline setting and in a simultaneous or streaming scenario. In the latter two, in addition to emitting highly accurate translations, we are also interested in minimizing the latency of the process. Moreover, the model does not have access to the whole phrase that will be translated, so it may need to predict what the main verb of the phrase is, the object it might be referring to, etc. due to differences in the syntactic typology of the source and target languages[1]. Thus, we must optimize the set of parameters in the space of all possible MT systems, finding systems that exhibit a good trade-off between translation quality and translation time.

Furthermore, it is important to note that the systems we are developing will be integrated into a cascading speech-to-speech translation pipeline, where an ASR system converts an audio signal into text, then an MT system translates this text into a different language so that a speech-to-text system can generate a spoken audio of the translated text. This introduces another complexity to the streaming scenario: the output of the ASR system is not segmented into phrases. That is, the model is unaware of where a certain phrase ends and the next one begins. There are two main approaches to solving this problem, by either introducing a *segmentation model*, i.e. a model that is trained to recognize if a particular token denotes the end of a phrase [35, 36], or training end-to-end speech-to-speech systems and segmenting the incoming audio stream instead. Due to time and computing concerns, in this work we will not concern ourselves with the segmentation

[1] It is clear that when translating in real-time from a subject-object-verb language to a subject-verb-object language, there will be phrases for which the model will emit the translation of the main verb before reading it in the source.

model, instead assuming such a model already exists. In the following sections we will formalize the problem of MT in a streaming scenario, describe the different evaluation metrics, and explain how the state-of-the-art simultaneous MT systems function.

5.1 Formal Definition

As opposed to the formalization of offline MT explained in Section 2.2, in a streaming scenario the model does not have access to the whole phrase it is translating. Instead, it can only access the previous context and (potentially) a small window of the future words, since otherwise the latency will be too high. Thus, the formalization of the approach we will use is as follows: for source and target sequences $\mathbf{x}$ and $\mathbf{y}$, we must predict the best translation y_i, only having access to previously translated words $\mathbf{y}_{<i}$ and source stream $\mathbf{x}_{\leq g(i)}$ where $g(i)$ is the number of source tokens that are read when the system preforms a writing operation at a position i. This way, the best translation $\mathbf{y}$ for a source stream $\mathbf{x}$ with a function g optimized by maximum likelihood estimation is given by

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}^*}{\operatorname{argmax}}\, p_g(\mathbf{y} \mid \mathbf{x}) \tag{5.1}$$

where $\mathcal{Y}^*$ denotes the space of all possible translations. We can apply the product rule of probability to obtain Equation 5.1 as a product of the conditional probabilities of each token that is emitted

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}^*}{\operatorname{argmax}} \prod_i p(y_i \mid \mathbf{x}_{\leq g(i)}, \mathbf{y}_{<i}) \tag{5.2}$$

where $p(y_i \mid \mathbf{x}_{\leq g(i)}, \mathbf{y}_{<i})$ is the probability of y_i being the best translation to emit at step i having read $g(i)$ source tokens $\mathbf{x}_{\leq g(i)}$ and written $\mathbf{y}_{<i}$ in the output stream.

At a position i, our streaming MT model will use a *policy* to decide whether to do a reading operation, where it observes the next source token, or a writing operation, where it emits the next token in the translated output stream. This policy maps the current state of the streaming system's environment (i.e. current position, number of tokens read and written, etc.) to reading and writing action space. It can be learned through adaptive methods, usually using reinforcement learning techniques, or set to a static policy. In our work, we will be using the latter approach, as it is computationally less expensive and performs on par with current adaptive methods. For adaptive policies, we recommend the reader consult [85, 82].

5.2 Models

In order to achieve good performance in streaming MT, we must adapt the existing Transformer architecture into a model that takes latency requirements into account, while also generating accurate translations by only having access to a prefix of the full phrase. This implies that our model should use a reliable policy of reading and writing operations. In the following sections, we will define the non-adaptive wait-k policy that we use in this work, illustrate how a model that was not trained for a streaming scenario can still be leveraged by constraining it with a defined latency, and describe the fully streaming MT model that we will be using.

5.2.1. Wait-k

A wait-k policy is one of the simplest policies for streaming translation. It is inspired by observing how human simultaneous interpreters need to wait until they have enough

context to begin translating, always lagging behind the original speaker. This way, a wait-k policy consists of first reading k tokens from the input stream, then alternating between a reading and writing operation. If the model reads a token at position i, its translation will be emitted at position $i + k$.

We define the wait-k policy's g function as

$$g_{\text{wait}-k}(i) = \left\lfloor k + \frac{i-1}{\gamma} \right\rfloor, \tag{5.3}$$

where we simply add k to the $i - 1$ term that represents a wait-0 oracle. The flooring operation is added to follow the definition of g as a count of the number of reading operations. Importantly, a wait-0 oracle will always write the translation of the next token before performing the reading operation to observe what token it is. We also include γ as a *catch-up* term, obtained as the source-to-target ratio. For a source-target pair, we define

$$\gamma_n = \frac{|\mathbf{y}_n|}{|\mathbf{x}_n|}. \tag{5.4}$$

However, in a streaming scenario we must take into account the global catch-up γ, which can be estimated as the averages over all γ_n of the source-target pairs in the corpus. The catch-up γ conditions the model to take into account the fact that two languages differ in the number of tokens necessary to semantically describe the same idea, and therefore the source and target lengths will differ most of the time.

5.2.2. Test-Time Wait-k

If we do not have the resources to train our own streaming MT model, but we can access a trained offline MT model, we can use it in a streaming scenario. The idea is to leverage the offline MT model by introducing the k parameter as a latency constraint, modifying the decoding step to follow a wait-k policy. The model will not be able to anticipate the incoming source nor predict what the next translations would be in a precise manner. Nevertheless, for bigger k values, it has been observed [49] that test-time wait-k decoding performs on par with models trained using wait-k decoding as a policy and a modeling of the streaming translation task as a prefix-to-prefix task, as we defined in Section 5.1. This is the expected behavior, since as we pick higher values for k, the model's decoding phase approaches the offline decoding process, where the model has read the full source phrase.

5.2.3. Multi-Path Wait-k

While training a streaming MT model on a single wait-k decoding policy is a good first step towards achieving good performance, it introduces the restriction of having to choose a particular value for k in training time. What happens if this value is not appropriate for some streaming scenarios? Instead of having to train many models with different k values, the authors of [22] propose to train a single model with multiple decoding paths.

If we also consider the wait-k path as a latent variable $\mathbf{z}^k$, we can modify the product of conditional probabilities in Equation 5.2 to also condition on the latent.

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{z}^k) = \prod_i p(y_i \mid \mathbf{x}_{\leq \mathbf{z}_i^k}, \mathbf{y}_{<i}, \mathbf{z}_{<i}^k). \tag{5.5}$$

However, we can jointly optimize over many different wait-k paths,

$$\mathbb{E}_K[p(\mathbf{y} \mid \mathbf{x}, \mathbf{z}^k)] \approx \prod_{k \sim \mathcal{U}(K)} p(\mathbf{y} \mid \mathbf{x}, \mathbf{z}^k). \tag{5.6}$$

where we uniformly sample from a set K of values for k. By leveraging multi-path training, we obtain a more generalizable model that can also be used more reliably with any wait-k policy that it was trained with. In practice, we apply logarithms to both sides and obtain a sum over k, granting more numerical stability.

This multi-path wait-k model will be the one that we will be training as a streaming MT model. However, it is important to note that there are many other approaches to training simultaneous and streaming MT systems. For these, we recommend the reader consult [2, 50, 86].

5.3 Evaluation

There have been many proposals for evaluation metrics in the field of simultaneous and streaming MT. Using a good metric to evaluate systems is especially important, since they provide proof that a system is performing better than another. Additionally, since we are not only evaluating translation quality but also latency times, we cannot rely only on traditional MT metrics such as BLEU, TER, etc. Instead, we must also take into latency-based evaluation metrics.

At the dawn of simultaneous MT research, latency was reported in milliseconds [5, 81]. Since simultaneous MT was being used in streaming cascading speech-to-speech systems or real-time speech-to-text, the calculated latencies were obtained as the difference in milliseconds of the input and output streams.

Some time later, the traditional MT metrics were adapted to take latency into account. This is the example of LatencyBLEU [24], which obtains the average BLEU score of the partial translations recorded at different positions. However, this was not a good enough metric for latency evaluation, since it attempted to implicitly evaluate latencies while evaluating translation quality.

Instead, researchers realized that developing metrics that only evaluated latency was a necessity. The idea behind these metrics is to obtain precise and interpretable abstract measurements of the latency that an MT model is characterized by. while also guaranteeing the expected latency measurements to not depend on the environment and hardware with which the system was evaluated.

A popular metric that was proposed was Consecutive Wait (CW) [25], which obtained the latency between the observation of a token and its translation, measured in terms of number of words. It measured local latencies, thereby not taking into account the lagging terms. This feature of CW made it too naïve to be seriously considered as a good latency-based metric. Thus, new metrics were proposed, solving some of CW's issues and iteratively converging towards well-defined metrics for latency-based evaluation. The following sections will describe the AP, AL and DAL metrics that we will use throughout this work to evaluate the latency of streaming MT systems.

5.3.1. Average Proportion

Average Proportion (AP) [13] is one of the first metrics used to measure latencies. This metric averages the absolute delay to write each token for each phrase

$$AP = \frac{1}{|\mathbf{x}||\mathbf{y}|} \sum_{i=1}^{\mathbf{y}} g(i) \tag{5.7}$$

The averaging is done to keep the co-domain of the metric in the $[0, 1]$ interval, which allows for some interpretability. However, it denotes a percentage, which does not help

us understand what the actual delay of the system was in global terms. Another issue of AP that needed to be addressed was its sensitivity to the lengths of the inputs. From Equation 5.7 we can clearly see that if we use wait-1 system, i.e. a simultaneous MT system that always alternates between one reading and one writing operation, as we increase the lengths of the inputs, the sum grows linearly, while the $|\mathbf{x}||\mathbf{y}|$ term does not.

As described in [49], for a wait-k system with $k = 1$, its $g(i) = 1$. This means that if $|\mathbf{x}| = |\mathbf{y}| = 1$, we obtain $\sum_{i=1}^{\mathbf{y}} g(i) = 1$ since we are reading the only token in $\mathbf{x}$ before writing the translation of length 1. If we increase the lengths, $|\mathbf{x}| = |\mathbf{y}| = 2$, the first writing operation happens after one reading operation, but the second writing will happen after a cumulative 2 reading operations. $\sum_{i=1}^{\mathbf{y}} g(i) = g(1) + g(2) = 1 + 2 = 3$, but the normalization term $\frac{1}{|\mathbf{x}||\mathbf{y}|} = \frac{1}{4}$, so we get $AP = 0.75$. At the limit, $|\mathbf{x}| = |\mathbf{y}| \to \infty$, $AP \to 0.5$. This is undesirable, since we are theoretically only using half of the co-domain $[0, 1]$ of the function. Thus, other metrics such as Average Lagging and Differentiable Average Lagging were proposed.

5.3.2. Average Lagging

Average Lagging (AL) [49] is a latency metric that attempts to solve the issues that AP has. It is designed to measure the latency between reading and writing operations, taking into account the lag behind a wait-0 oracle policy.

$$AL_g = \frac{1}{\tau} \sum_{i=1}^{\tau} \left(g(i) - \frac{i-1}{\gamma} \right) \tag{5.8}$$

It measures the distance between a policy $g(i)$ and the wait-0 oracle $\frac{i-1}{\gamma}$ at every step i. A wait-0 oracle's AL is defined to be 0, since the emission of a translation is done before its respective source token is read, therefore having no lag.

If we replace $g(i)$ in Equation 5.8 by its definition in Equation 5.3 for a wait-k policy, we obtain

$$AL_{\text{wait}-k} = \frac{1}{\tau} \sum_{i=1}^{\tau} \left(\left\lfloor k + \frac{i-1}{\gamma} \right\rfloor - \frac{i-1}{\gamma} \right) \tag{5.9}$$

$$\approx \frac{1}{\tau} \sum_{i=1}^{\tau} \left(k + \frac{i-1}{\gamma} - \frac{i-1}{\gamma} \right) \tag{5.10}$$

$$= \frac{1}{\tau} \sum_{i=1}^{\tau} k = k \tag{5.11}$$

By dropping the flooring operation and simplifying, we see that AL is designed such that its value is k for a wait-k system[2]. The idea behind this is to make the metric more interpretable, since any wait-k system is k tokens behind the wait-0 oracle by definition.

AL also needs to take into account the differences between source and target lengths, which is why the authors introduce the γ catch-up term in Equation 5.8. Lastly, the summation is done up to τ, which denotes the position after which there are no more reading operations to be done, i.e. the source phrase $\mathbf{x}$ has been completely observed.

$$\tau = \tau_g(|\mathbf{x}|) = \min_{i:g(i)=|\mathbf{x}|} i \tag{5.12}$$

However, as we will see in the next section, while the main ideas of AL are well developed, introducing the minimization term to find the τ term makes in a non-differentiable function. This is what the Differentiable Average Lagging metric attempts to solve.

[2] In practice, it most likely will not be k exactly, since the source and target ratios may be different.

5.3.3. Differentiable Average Lagging

The main development introduced by the authors of the Differentiable Average Lagging (DAL) [12] metric is modification of the AL metric by removing the argmin operation, thus making the metric differentiable.

The authors of the DAL metric highlight that the usage of τ as the upper bound of the summation is a way for the AL metric to solve a much more important problem: the writing operations done after reading the whole source phrase $\mathbf{x}$ are done with cost 0. That is, AL maintains the incorrect assumption that writing after step τ is done instantaneously, where in reality if the system is lagging behind the source stream by some factor, it will continue lagging by that factor when the source phrase ends.

Therefore, the authors define the DAL metric by first introducing a function $g_d\prime$

$$g_d\prime(i) = \begin{cases} g(i) & i = 1 \\ \max\left(g(i), g_d\prime(i-1) + d\right) & i > 1 \end{cases} \tag{5.13}$$

that wraps g by also defining a minimal cost for writing operations d. The second term of the maximization guarantees a non-zero writing cost. We can see that $g_d\prime$ computes the latency before writing token i, accounting for writing costs. For some writing cost d, DAL is defined as follows:

$$DAL_d = \frac{1}{|\mathbf{y}|} \sum_{i=1}^{|\mathbf{y}|} \left(g_d\prime(i) - (i-1)d\right) \tag{5.14}$$

If we define

$$d = \frac{1}{\gamma} = \frac{|\mathbf{x}|}{|\mathbf{y}|} \tag{5.15}$$

as our writing cost, we can use $g_d\prime = g_{\frac{1}{\gamma}}\prime$ to define the DAL metric similarly to AL

$$DAL_{\frac{1}{\gamma}} = \frac{1}{|\mathbf{y}|} \sum_{i=1}^{|\mathbf{y}|} \left(g_d\prime(i)d - \frac{(i-1)}{\gamma}\right), \tag{5.16}$$

maintaining its idea of lagging behind the wait-0 oracle.

It is important to note that these transformations of the original AL metric remove any non-differentiable operations such as the argmin. To the interested reader we recommend [33, 34] for new developments in streaming MT metric research. In this work, we will report AP, AL and DAL metric scores in the development of streaming MT systems.

CHAPTER 6
Experiments

This chapter will introduce the experimental setup used for the development of the NMT systems. More precisely, we describe the toolkit used and the decisions taken for training and adaptation of different offline and simultaneous MT systems, as well as the results obtained on our validation and test corpora.

6.1 Experimental Setup

Figure 6.1 shows the pipeline which will be used for the development of NMT systems. As explained in Chapter 3, the first step consist in compiling the data and defining the training, validation and test sets. Here, a set of the processing steps are applied, where we use many of the scripts offered by `Moses`, `subword-nmt` and `SentencePiece`.

Afterwards, we develop our MT models using *fairseq* [57]. We binarize the data with `fairseq-preprocess` and train the NMT model using `fairseq-train`. The resulting model is used in inference mode with the `fairseq-interactive` tool in order to obtain the hypotheses for the validation and test sets. A detailed description of the functionalities of fairseq is provided in the next section.

Subsequently, the hypotheses are post-processed by reconstructing subwords into words, detruecasing, detokenizing, etc. so that the output of the model is comparable to the target of reference. Finally, the hypotheses and references are fed into `sacreBLEU` [61], a free software implementation of popular translation quality metrics, as well as Javier Iranzo-Sanchez's implementation of the latency metrics presented in Section 5.3. This way we evaluate our model and continually iterate, either changing model hyperparameters or refining the data cleaning and processing steps.

Having shown the experimental setup used in this work, we will describe the functionalities offered by fairseq, the deep learning toolkit that we will be using to develop NMT systems. Afterwards, we will present the baseline system that is in a production environment at the beginning of this work, and enumerate the different improvements that were assessed for subsequent iterations of the NMT system.

6.1.1. Fairseq

The main toolkit that was used to develop NMT systems is fairseq, a tool developed by Facebook AI Research for sequence modeling tasks such as text summarization, language modeling, and more specifically, translation. Fairseq is built on PyTorch, a tool also developed by the Facebook AI Research team as a python interface to the torch library originally developed in Lua. Together with TensorFlow and JAX/Flax, it is currently one

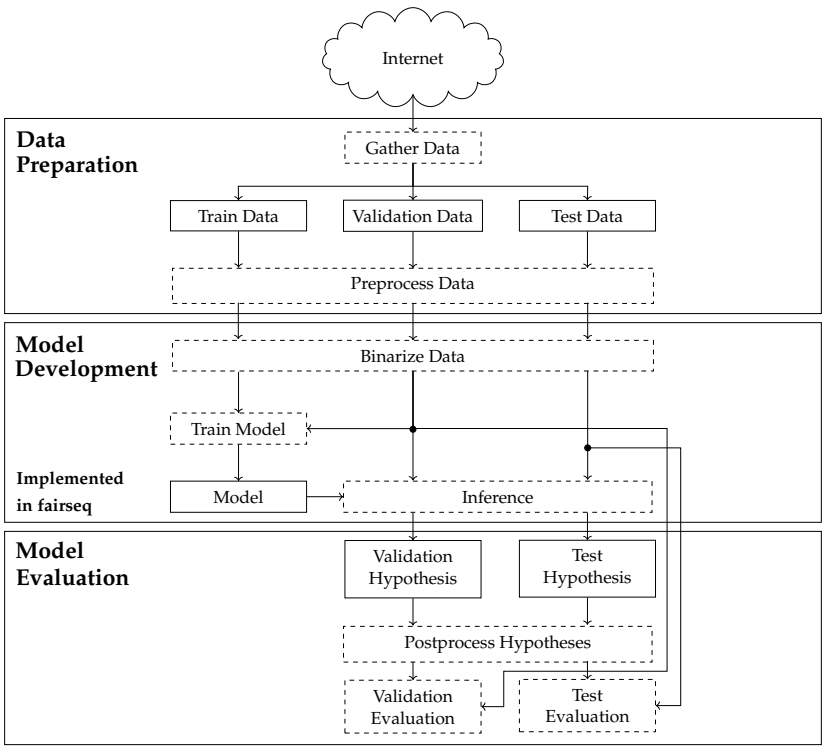


Figure 6.1: The general experimentation pipeline. Dashed rectangles denote processes, whereas plain bordered ones are objects obtained from said processes.

of the most popular open source ML frameworks. Similarly to its alternatives, it provides GPU accelerated tensor computing operations, as well as a set of implemented modules and an interface for easy definition of deep learning models through the usage of computational graphs for automatic differentiation.

Additionally, it incorporates its own training loops, predefined model architectures, functionalities for easy processing of sequence data for the training of big models in many GPUs, automatic saving of models at different checkpoints when training a model, as well as a robust logging system that enables researchers to quickly learn about potential problems in the training process.

Apart from the python API, fairseq offers command line tools that facilitate the development of large models, as well as the possibility to extend the toolkit by defining new models, metrics, etc. The main command line tools that will be used throughout this work are:

- `fairseq-preprocess`. It takes (preprocessed) parallel text as input and builds the vocabularies in a format that the model can parse, binarizing the training data for ease of use by the model. It takes into consideration a set of command line arguments and supports parallel computing for large datasets.
- `fairseq-train`. It trains the model on one or many GPUs. providing a way to configure the model and set the hyperparameters to use for training, as well as the binarized data path from `fairseq-preprocess`. It offers an implementation of the Transformer model as described in Section 2.3.3
- `fairseq-interactive`. It implements the beam search algorithm presented in Section 2.3.4, translating data with a trained model in inference mode. Within its parameters, we can specify the number of beams, inference batch size, the model to use, etc.

For training and inference of simultaneous MT systems we will use a fork of fairseq maintained by Javier Iranzo-Sanchez, which implements the simultaneous and streaming MT models, training schemes and decoding policies explained in Chapter 5.

These tools are executed via a job scheduler on a cluster with 10s+ GPUs. As is conventionally performed, we averaged the last 8 checkpoints of the models obtained in training time, recorded within 10000 gradient updates. The idea behind this technique is to ensemble the models, due to the propensity of big models to overfit after observing large amounts of data. In practice, it leads to better performance.

Finally, it is important to note that all our models have been trained with mixed-precision. Ideally, we would all our model's parameters to be 16-bit floating point numbers, in order to decrease our memory usage by a big margin. However, in practice this tends to lower the performance to a degree where it is undesirable. Thus, we use mixed-precision training, where the model's parameters are stored in 16-bit precision, but the computation of the loss, gradient, the backpropagation is done with 32-bit precision. Once the updated parameters are computed in 32-bit precision, they are converted to 16 bits to reduce memory.

6.2 General Domain Systems

In this section, we will describe the general domain systems that were trained for the language direction Fr→En. We will always be using the BIG variant of the Transformer model, the Adam optimizer with momentum betas $\beta_1 = 0.9, \beta_2 = 0.98$, a starting learning rate of $\eta = 1e-07$ with 16000 warm-up steps to an inverse square root learning rate scheduler and minimum $\eta_{min} = 1e-09$. We apply a drop-out with 0.1 probability, and the label-smoothing noise is set to 0.1. Additionally, for all systems, the inference process was carried out using a beam size of 6.

Our baseline MT system was trained in 2019, in the framework of the X5Gon[1] project, a European Union (EU) initiative to help students access cross-modal, cross-lingual, cross-domain, cross-cultural and cross-site educational resources. The system was trained on the Europarl-ST corpus, as well as the WMT10, WMT13 and WMT14 training sets for the translation tasks. The training data was cleaned using `langid`[2], an open-source language identification tool, and filtered with a source-target length ratio of 1.5. Afterwards, the corpus' punctuations were normalized, the text was tokenized and truecased using a set of scripts from `Moses`, and the BPE subword segmentation algorithm was applied with 40000 merge operations. For this baseline system and the subsequent systems we develop in this work, we present the general-domain evaluation results in Table 6.1, and the in-domain evaluation results in Table 6.2, where we evaluate system performance on general domain data with WMT13 as the validation set and WMT14 as the test set, whereas for in-domain evaluation we use CN21 and CN22 as validation and test sets, respectively. As shown in these tables, our baseline obtains BLEU scores of 34.7 and 39.4 on WMT13 and WMT14, and 35.3 and 36.8 on CN21 and CN22 respectively.

In order to understand what these scores entail, we provide context regarding the meaning of these scores based on metric interpretability research conducted on [73]. From this work, we know that a BLEU score of approximately 40 or better signifies that little post-editing is needed, that is, the translations are regarded of good quality. Additionally, scores nearing 30 BLEU correlate with translations where post-editing is still considered quicker than retranslating manually, but the translation quality is not very ad-

[1] https://www.x5gon.org/
[2] https://github.com/saffsd/langid.py

equate. The authors show similar correlations with TER scores around 45 for the former, and 50 for the latter.

System	WMT13			WMT14		
	BLEU	chrF2	TER	BLEU	chrF2	TER
X5Gon	34.7	**59.8**	54.1	39.4	64.5	47.0
V1	**35.1**	**59.8**	**53.6**	39.2	64.0	47.3
V2	**35.1**	**59.8**	53.9	**39.5**	**64.3**	47.2
V3	34.8	59.7	54.2	39.3	64.2	**46.9**
V4	34.6	59.6	54.5	39.1	64.0	47.2

Table 6.1: BLEU, chrF2, and TER scores on WMT13 Fr→En, and WMT14 Fr→En.

The first system that we trained used a naïve approach of simply increasing the training data size, without applying any data filtering techniques. We refer to this system as V1. Here, will be using the corpora mentioned in Section 3.1.1 as training data. We only apply basic data processing techniques such as tokenization, truecasing, and BPE of 40000 merge operations, but we do not filter the data in any way. Taking into account the much bigger size of the corpus with respect to the one used for our baseline, we train the V1 system for 1.5M parameter updates. This amounted to 2 epochs[3] of training. We note that the full process of developing this system by processing the data and training the model took close to three weeks. Having done this, we found an overall improvement in its evaluation. As shown in Table 6.1, V1 obtains a BLEU of 35.1 and 39.2 on WMT13 and WMT14, respectively. These scores are already on par with the baseline. Thus, the next general domain models will be trained after refining the data processing pipeline, in order to obtain a more robust and reliable MT system.

Now that we have obtained a first MT system that is comparable in quality with X5Gon, we will apply some of the data filtering techniques described in Section 3.2. We trained a V2 system on the same corpus as V1, filtering the training bitext with `langid`, which we use to identify and remove phrases in languages other than French and English, for their respective texts. In a similar fashion to our baseline, we filter out source-target pairs with a length ratio greater than 1.5 and define an upper bound the model's input sequence lengths of 150 tokens. The data processing pipeline was not modified from V1: we tokenize, truecase, and segment using BPE with 40000 merge operations. Having observed the extensive training time necessary for the model to complete 1.5M updates, we opted to train up to 1M updates for V2 and the following models. From Table 6.1 we see that V2 maintains the evaluation scores of V1 in the WMT13 test set, while improving on WMT14. Importantly, we note that V2 reaches the best quality scores on WMT14 of 39.5 BLEU and 64.3 chrF2, while not deviating from V1's scores on other metrics and corpora.

After having trained V1 and V2, we found some aspects of the training data we hypothesized were affecting the model's learning capability. Thus, we refined the preprocessing of apostrophes and HTML-escaped characters to improve the performance of our model. For this we used a set of scripts offered by `Moses`, and introduced `SentencePiece` as the subword segmentation algorithm. We trained a third system, V3, on the same training data as V2. Following V2, we also limited the sequence length of the inputs to the model to 150 tokens. We found that the translation quality scores of V3 slightly decreased, still being on par with the baseline X5Gon model as is shown in Table 6.1. At this point in time we also evaluated on the CERN News development and testing corpora, finding big improvements w.r.t. the baseline. From Table 6.2 we observe that V3 obtained BLEU scores of 36.9 and 38.6 on CN21 and CN22 respectively, amounting to an

[3] An *epoch* is completed when the model has observed the full training dataset once

increase of almost two points in the in-domain evaluation sets. Additionally, it's chrF2 score is increased by a big margin, and TER improved too. This shows the power of a robust data processing pipeline, where denoising, filtering, as well as correctly preparing the text for the model improves its performance, without the need to modify the model itself. Finally, we computed the 95% confidence intervals for V3's results, finding that the mean for bootstrapped resampling with 1000 samples is within the ranges of 34.8 ± 0.8, 39.2 ± 0.7 for general domain validation and test sets, and 36.9 ± 0.8, 38.6 ± 1.0 for in-domain validation and test sets, respectively. Thus, we find that while there are no statistically significant differences between these systems, our systems perform consistently better than the baseline on all metrics and test datasets.

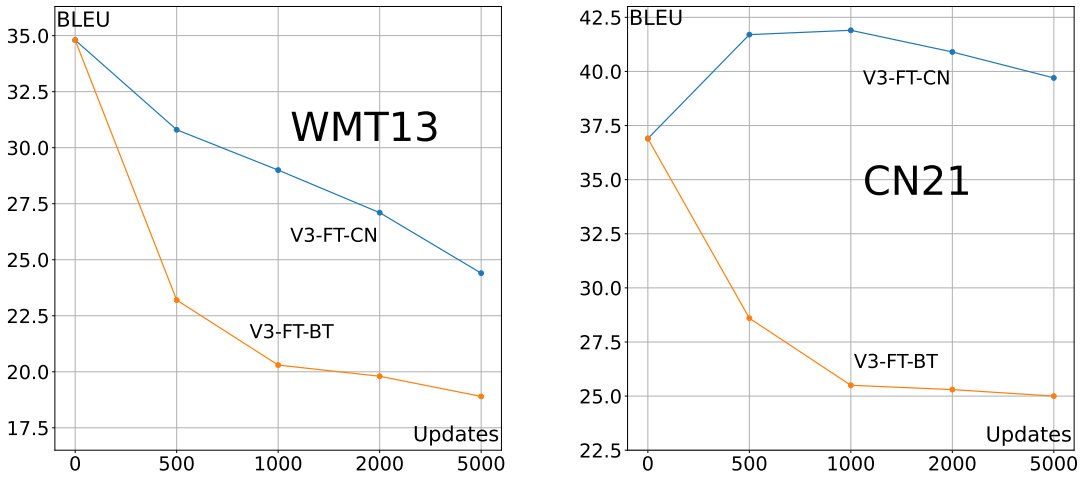
6.3 In-Domain Systems

Having trained well-performing general domain MT systems, we developed in-domain systems by either fine-tuning previously trained general domain systems, or training new systems from scratch by leveraging both monolingual backtranslated and bilingual in-domain text. We will maintain the same training hyperparameters used during general domain MT model training. For fine-tuned systems, we will define a fixed learning rate scheduler to a constant value, keeping the learning rate that was last used by the scheduler in general domain training.

To obtain the backtranslated synthetic parallel text, the monolingual CDS corpus described in Section 3.1.2 was used. The text was tokenized and truecased using a trained truecaser from `Moses`. Similarly, we applied `SentencePiece`'s BPE subword segmentation using trained subword segmentation models. Afterwards, we filtered the segmented text using a length threshold of 150 and backtranslated the processed monolingual data using a trained En→Fr MT system. The processing steps used this system's truecasing and subword segmentation models, so the model's inputs in inference time are processed in the same manner as in training-time. Lastly, we removed `SentencePiece`'s unknown tokens and built the synthetic backtranslated in-domain corpus. This process took over 10 days of compute time, yielding approximately 50 million new source-target pairs.

Having addressed the data-related issues in the development of our V3 system, and observing the performance increase of close to 2 BLEU on the in-domain evaluation sets, we opted to fine-tune this model. We assessed two approaches: using the backtranslated monolingual in-domain text, and fine-tuning on the bilingual in-domain corpus. Thus we fine-tuned V3 using the CERN News training data, referring to this as the V3-FT-CN system. Since the CERN News training data was comprised of approximately 50K examples, we obtained the V3-FT-BT system by fine-tuning V3 using 50K synthetic parallel pairs subsampled from the backtranslations mentioned previously. Figure 6.2 shows the progression of BLEU scores on the general domain and in-domain validation sets in the fine-tuning process. First, we see that V3-FT-BT's performance rapidly decreases. We hypothesize the decrease in performance may be due to how when fine-tuning on backtranslations we are essentially fine-tuning on in-domain monolingual data with source that potentially contains translationese[4], whereas using in-domain (non-synthetic) parallel data introduces bitext of high quality to the fine-tuning scheme; in addition to the domain mismatch between the CDS corpus, which contains highly technical scientific articles w.r.t. the in-domain evaluation sets, comprised mainly of STEM-related news. Additionally, we observe how V3-FT-CN quickly adapts to the in-domain evaluation sets, as the BLEU scores on CN21 and CN22 increase after the first 500 updates, dropping after

[4]In translated text, *tanslationese* is manifested as awkward, unidiomatic or unnatural aspects in translations, which are artifacts of the translator attempting to preserve the meaning of the source.



(a) Fine-tuning on CERN News.

(b) Fine-tuning on CDS.

Figure 6.2: Change in BLEU w.r.t the number of updates in fine-tuning time.

System	CN21			CN22		
	BLEU	chrF2	TER	BLEU	chrF2	TER
X5Gon	35.3	63.6	54.5	36.8	64.0	53.1
V3	**36.9**	63.9	**52.6**	**38.6**	64.5	**51.6**
V4	36.6	**64.6**	53.5	38.2	**65.0**	52.4

Table 6.2: BLEU, chrF2, and TER scores on CN21 Fr→En, and CN22 Fr→En.

another 2000 updates are carried out, and the general domain evaluation BLEU scores drop by approximately 5 points. Most importantly, we find that V3-FT-CN reaches an in-domain BLEU of 41.6 on CN21, improving V3's scores by over 10% relative, which is a statistically significant difference w.r.t. our general domain V3 system.

In addition fine-tuning an existing general domain MT system, we opted to train a V4 system from scratch by including backtranslated monolingual in-domain data as part of the training data. We followed the same data processing pipeline as the one described for the V3 model, only increasing the maximum sequence length threshold from 150 to 250, having seen that there were many longer phrases in the CERN News test sets. As reported in Tables 6.1 and 6.2 we observe that V4 also improves upon X5Gon's scores on the both WMT and CN. It obtains BLEU and TER scores on par with V3, while performing better in all evaluation corpora when measured with chrF2. Thus, we find that the set of 50 million backtranslations do not seem to significantly improve the results w.r.t. the V3 system. In a similar fashion to the fine-tuning using backtranslated text, our hypothesis is that the lack of improvements are due domain mismatch between the backtranslated bitext and the in-domain sets, and translationese.

Finally, having observed the behavior of V3-FT-CN as more updates are performed, we use the checkpoint at 1000 updates as our final in-domain system since this is the system that performs the best on our in-domain validation set. The results of evaluating this system on the general domain and in-domain test sets are presented in Table 6.3. We note that CERN's requirements for high-accuracy MT systems is of 40 BLEU or higher, and that our best in-domain system exceeds this value by 3 BLEU. Additionally, we find

over 11% relative increase in in-domain BLEU w.r.t. out best general-domain system in a similar fashion to the evaluation on the validation in-domain sets.

Corpus	BLEU	chrF2	TER
WMT14	31.7	57.8	54.7
CN22	42.9	66.4	47.9

Table 6.3: Results of V3-FT-CN with 1000 fine-tuning updates evaluated on WMT14 and CN22.

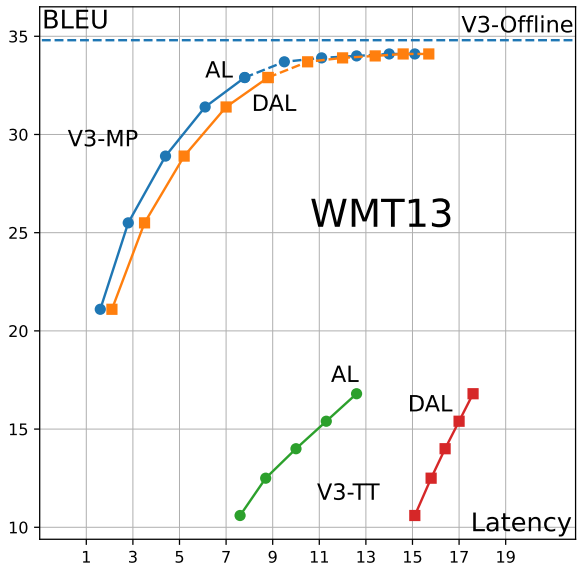


Figure 6.3: Latencies and BLEU scores of wait-k MT systems on WMT13.

6.4 Streaming MT Systems

Our first step of developing streaming MT systems was to introduce a wait-k decoding policy to our existing best performing system, V3, to consider the scenario where we do not have enough resources to train simultaneous MT systems from scratch. We did not expect this system to perform as well, given that it was trained by accessing the full source. In addition to the test-time wait-k system, we trained a simultaneous MT system with the multi-path wait-k training scheme. We used the same data processing pipeline as V3, given its better performance over the other models, only modifying the warm-up updates of the learning rate to 4000. For both of these systems, the inference process was carried out using a beam size of 6 and a catch-up of $\gamma = 1.125$. We will refer to the multi-path wait-k system as V3-MP, and the test-time wait-k system as V3-TT.

Figure 6.3 shows the BLEU and latency scores for both systems using wait-k decoding with $k \in \{1,3,5,7,9\}$ on WMT13, our general domain validation set. As seen in this figure, in a general domain scenario, the quality of V3-MP is consistently better than V3-TT, independently of the value of k that was used in decoding time. Likewise, using any latency metric, V3-MP consistently translates faster than V3-TT. This is expected, since V3-MP is an MT model trained for a real-time scenario, where as V3-TT is not. We note that the AP scores are range from 0.6 to 0.8 for V3-MP, and 0.8 to 0.9 for V3-

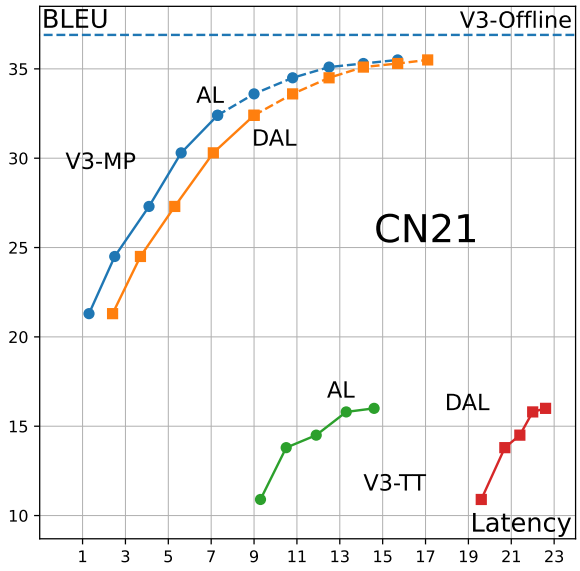


Figure 6.4: Latencies and BLEU scores of wait-k MT systems on CN21.

TT across k values, thereby also proving that V3-MP is better suited for streaming MT. No result reaches the offline quality score of 34.8 BLEU. However, as latencies increase, both systems have access to more source tokens and produce better translations, which explains how the increase in k results in better BLEU scores.

Corpus	BLEU	chrF2	TER	AP	AL	DAL
WMT14	33.8	61.8	54.7	0.7	5.9	7.1
CN22	30.9	61.1	63.3	0.7	5.5	7.1

Table 6.4: Results of V3-MP with wait-7 decoding, evaluated on WMT14 and CN22.

We see a similar behavior on Figure 6.4, which illustrates the relationship between BLEU and latency scores using the same k values for a wait-k policy-based decoding evaluated on CN21, the in-domain validation set. Here, V3-MP is consistently faster and more accurate, but does not reach the offline BLEU score of 36.9. Lastly, the dotted sections of the AL and DAL curves for the V3-MP, illustrated in both Figure 6.3 and 6.4, show V3-MP's convergence towards the offline BLEU score as k grows.

Having evaluated various streaming MT systems on both general and in-domain validation sets, we can now choose a value for k. We note that in order to choose the best k value, we would have to deploy various MT systems with different k values onto CERN's network, to then measure true response time using the execution environment, network and hardware within which the final system will be deployed, while also taking into account the latencies of the streaming ASR, segmentation and TTS models. Given that this complex process an take weeks or months to complete, we will present an illustrative best system by considering previous research in [36], also making sure to emit translations with BLEU scores of around 30, where post-editing is still quicker than retranslating [73]. Thus, we believe the V3-MP system with a k of 7 as a lower bound, is of good quality. The results of our V3-MP wait-7 system evaluated on the test sets is shown in Table 6.4. We observe that a V3-MP with wait-7 decoding obtains BLEU scores of 33.8 and 30.9 on WMT14 and CN22 respectively, proving that the system generates adequate translations.

CHAPTER 7
Conclusions

In this chapter we will provide a summary of the work, describing the obstacles overcome, lessons learned, goals met, and future work.

In Chapter 2 we briefly introduced the theory behind ML, while also providing historical accounting of MT research and describing the first statistical MT models. We also analyze how a basic artificial neural network functions and how sequence-to-sequence tasks are modeled, briefly touching upon complex encoder-decoder recurrent models such as RNNsearch, and most importantly, describing the inner-workings of the Transformer architecture.

Chapter 3 illustrated the different text corpora that are typically used in practice, as well as various approaches towards data processing. We explained what our general domain and in-domain training and evaluation sets are comprised of. Additionally, we studied how based on the source of the corpus, there is often need to denoise and filter the text, and the many filtering techniques that can be applied to text. We also described tokenization, truecasing and subword segmentation, typical data processing techniques in NLP.

In Chapter 4 we explained typical domain adaptation approaches, explaining how in-domain data can be leveraged to obtain high-accuracy in-domain MT systems. We described the steps to backtranslate monolingual text in order to obtain synthetic parallel data, and the process of fine-tuning an existing general domain MT system to a particular domain.

Furthermore, in Chapter 5 we defined the tasks of simultaneous and streaming translation. Here we explained how offline MT systems can be used in a streaming scenario by using wait-k policies, and presenting an approach to train simultaneous MT systems from scratch using the multi-path wait-k training scheme. Additionally, we provided a brief historical account of simultaneous MT latency evaluation, together with the latency-based metrics currently used to evaluate an MT system's response time

Lastly, Chapter 6 explained how the general experimental environment was set-up with the usage of `fairseq`, a job scheduler on a cluster of many GPUs for efficient parallel training, together with other open-source software for the data processing and evaluation steps. Here we described the results of a series of experiments carried out to obtain general domain MT systems, in-domain MT systems, as well as streaming MT systems. For general domain MT, we observed how a robust data processing pipeline considerably improved the quality of the MT system. Afterwards, we experimented with fine-tuning techniques to adapt a general domain MT system to the domain of interest to CERN using backtranslated synthetic parallel data and an existing in-domain corpus. Our best experiments resulted in an in-domain MT system that reaches BLEU scores that improve the baseline system by a relative 11%. We also carried out experiments for streaming MT,

where we observed that test-time wait-*k* systems produced low quality translations with high response times, finding that using a multi-path training scheme greatly improved the quality of the translation, whilst also guaranteeing lower latency times.

Regarding future work, there were many aspects that were left out due to time constraints and the limited availability of computing resources. Firstly, we believe that training with the exclusion of some training datasets could be beneficial. This is due to inherent noise in data scraped from the web such as ParaCrawl or CommonCrawl. Secondly, we did not have access to in-domain highly technical parallel text to evaluate in-domain systems. We believe that additional more technical in-domain evaluation sets could aid our decision making for future experiments in the iterative refinement process of training and evaluating MT systems.

Thirdly, due to computing and time constraints we were unable to experiment with other state-of-the-art domain-adaptation techniques. An example is the usage of adapter layers [30, 7] where additional layers are introduced to the pre-trained general domain MT model. These layers usually project the input to a lower dimension, applying a set of non-linearities, and then projecting back to the original dimension. The idea behind them is to provide more parameters to the network and only train on the new parameters, freezing the rest of the model's weights. This means that adapters can be used in a plug-in fashion, by including them in inference time for in-domain phrases, but using the original pre-trained model otherwise.

Lastly, since simultaneous MT is a hectic research area, there are many new promising approaches being published. More specifically, due to computing constraints we did not have the opportunity to experiment with adaptive decoding policies, that is, policies where the model learns when to perform reading or writing actions. We believe that adapting these approaches to a streaming scenario could yield good results. These include models trained to identify *meaningful units*, chunks of text that can be translated atomically, without the model needing to access future or previous context [85]; or augmenting the training corpus by including adaptive prefix-to-prefix pairs to emulate wait-*k* training without having to pay its high computational cost [82].

Our next immediate steps are to deploy the offline general domain and in-domain MT systems so they can be used translate CERN's digital multimedia material, train a multi-path system with general domain and backtranslated parallel data and deploy it as a step in the cascading speech-to-speech translation pipeline being used in CERN's real-time communication channels. Additionally, we want to experiment with fine-tuning this new simultaneous MT system using the CERN News corpus, with which our developed offline MT systems achieved great results.

Bibliography

[1] Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. Learning dependency translation models as collections of finite-state head transducers. *Computational linguistics*, 26(1):45–60, 2000.

[2] Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, Chung-Cheng Chiu, Semih Yavuz, Ruoming Pang, Wei Li, and Colin Raffel. Monotonic infinite lookback attention for simultaneous machine translation. *arXiv preprint arXiv:1906.05218*, 2019.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Srinivas Bangalore, Vivek Kumar Rangarajan Sridhar, Prakash Kolan, Ladan Golipour, and Aura Jimenez. Real-time incremental speech-to-speech translation of dialogs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 437–445, 2012.

[6] Marta Bañón, Pinzhen Chen, Barry Haddow, Kenneth Heafield, Hieu Hoang, Miquel Esplà-Gomis, Mikel L Forcada, Amir Kamran, Faheem Kirefu, Philipp Koehn, et al. Paracrawl: Web-scale acquisition of parallel corpora. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4555–4567, 2020.

[7] Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. Simple, scalable adaptation for neural machine translation. *arXiv preprint arXiv:1909.08478*, 2019.

[8] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

[9] 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.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[11] Erick Cantú-Paz et al. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

[12] Colin Cherry and George Foster. Thinking slow about latency evaluation for simultaneous machine translation. *arXiv preprint arXiv:1906.00048*, 2019.

[13] Kyunghyun Cho and Masha Esipova. Can neural machine translation do simultaneous translation? *arXiv preprint arXiv:1606.02012*, 2016.

[14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[15] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[16] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[19] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[20] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*, 2018.

[21] 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)*. Association for Computational Linguistics, November 2020.

[22] Maha Elbayad, Laurent Besacier, and Jakob Verbeek. Efficient wait-k models for simultaneous machine translation. *arXiv preprint arXiv:2005.08595*, 2020.

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

[24] Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Proceedings of the 2014 Conference on empirical methods in natural language processing (EMNLP)*, pages 1342–1352, 2014.

[25] Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor OK Li. Learning to translate in real-time with neural machine translation. *arXiv preprint arXiv:1610.00388*, 2016.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[28] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[29] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[30] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[31] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[32] W John Hutchins, Leon Dostert, and Paul Garvin. The georgetown-ibm experiment. In *In*. Citeseer, 1955.

[33] Javier Iranzo-Sánchez, Jorge Civera, and Alfons Juan. Stream-level latency evaluation for simultaneous machine translation. *arXiv preprint arXiv:2104.08817*, 2021.

[34] Javier Iranzo-Sánchez, Jorge Civera, and Alfons Juan. From simultaneous to streaming machine translation by leveraging streaming history. *arXiv preprint arXiv:2203.02459*, 2022.

[35] Javier Iranzo-Sánchez, Adrián Giménez Pastor, Joan Albert Silvestre Cerdà, Pau Baquero-Arnal, Jorge Civera Saiz, and Alfons Juan. Direct segmentation models for streaming speech translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2599–2611. Association for Computational Linguistics, 2020.

[36] Javier Iranzo-Sánchez, Javier Jorge, Pau Baquero-Arnal, Joan Albert Silvestre-Cerdà, Adrià Giménez, Jorge Civera, Albert Sanchis, and Alfons Juan. Streaming cascade-based speech translation leveraged by a direct segmentation model. *Neural Networks*, 142:303–315, 2021.

[37] Javier Iranzo-Sánchez, Joan Albert Silvestre-Cerda, Javier Jorge, Nahuel Roselló, Adria Giménez, Albert Sanchis, Jorge Civera, and Alfons Juan. Europarl-st: A multilingual corpus for speech translation of parliamentary debates. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8229–8233. IEEE, 2020.

[38] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[39] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[40] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of machine translation summit x: papers*, pages 79–86, 2005.

[41] Philipp Koehn. *Neural machine translation*. Cambridge University Press, 2020.

[42] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180, 2007.

[43] Philipp Koehn, Franz J Och, and Daniel Marcu. Statistical phrase-based translation. Technical report, University of Southern California Marina Del Rey Information Sciences Inst, 2003.

[44] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[45] Yann LeCun and Fu Jie Huang. Loss functions for discriminative training of energy-based models. In *International Workshop on Artificial Intelligence and Statistics*, pages 206–213. PMLR, 2005.

[46] Tao Lei, Yu Zhang, Sida I Wang, Hui Dai, and Yoav Artzi. Simple recurrent units for highly parallelizable recurrence. *arXiv preprint arXiv:1709.02755*, 2017.

[47] Seppo Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki, 1970.

[48] Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. Very deep transformers for neural machine translation. *arXiv preprint arXiv:2008.07772*, 2020.

[49] Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, et al. Stacl: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. *arXiv preprint arXiv:1810.08398*, 2018.

[50] Xutai Ma, Juan Pino, James Cross, Liezl Puzon, and Jiatao Gu. Monotonic multihead attention. *arXiv preprint arXiv:1909.12406*, 2019.

[51] Yanjun Ma, Sylwia Ozdowska, Yanli Sun, and Andy Way. Improving word alignment using syntactic dependencies. Association for Computational Linguistics, 2008.

[52] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[53] Tom M Mitchell and Machine Learning. Mcgraw-hill. *New York*, pages 154–200, 1997.

[54] Michael C Mozer. A focused backpropagation algorithm for temporal. *Backpropagation: Theory, architectures, and applications*, 137, 1995.

[55] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38, 1994.

[56] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[57] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[58] 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 on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[59] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[60] Maja Popović. chrf: character n-gram f-score for automatic mt evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, 2015.

[61] Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels, October 2018. Association for Computational Linguistics.

[62] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[63] Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. Comet: A neural framework for mt evaluation. *arXiv preprint arXiv:2009.09025*, 2020.

[64] LM Rasdi Rere, Mohamad Ivan Fanany, and Aniati Murni Arymurthy. Simulated annealing algorithm for deep learning. *Procedia Computer Science*, 72:137–144, 2015.

[65] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[66] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[67] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[68] Holger Schwenk, Vishrav Chaudhary, Shuo Sun, Hongyu Gong, and Francisco Guzmán. Wikimatrix: Mining 135m parallel sentences in 1620 language pairs from wikipedia. *arXiv preprint arXiv:1907.05791*, 2019.

[69] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[70] Libin Shen, Anoop Sarkar, and Franz Josef Och. Discriminative reranking for machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 177–184, 2004.

[71] Raivis Skadiņš, Jörg Tiedemann, Roberts Rozis, and Daiga Deksne. Billions of parallel words for free: Building and using the eu bookshop corpus. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1850–1855, 2014.

[72] Matthew Snover, Bonnie Dorr, Richard 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, 2006.

[73] Lucia Specia, Marco Turqui, Zhuoran Wang, John Shawe-Taylor, and Craig Saunders. Improving the confidence of machine translation quality estimates. In *Proceedings of Machine Translation Summit XII: Papers*, 2009.

[74] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[75] Ralf Steinberger, Andreas Eisele, Szymon Klocek, Spyridon Pilos, and Patrick Schlüter. Dgt-tm: A freely available translation memory in 22 languages. *arXiv preprint arXiv:1309.5226*, 2013.

[76] Jörg Tiedemann. Parallel data, tools and interfaces in opus. In *Lrec*, volume 2012, pages 2214–2218. Citeseer, 2012.

[77] Jörg Tiedemann and Lars Nygaard. The opus corpus-parallel and free: http://logos. uio. no/opus. In *LREC*. Citeseer, 2004.

[78] T Tieleman and Geoffrey Hinton. Divide the gradient by a run- ning average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

[79] Dániel Varga, Péter Halácsy, András Kornai, Viktor Nagy, László Németh, and Viktor Trón. Parallel corpora for medium density languages. *Amsterdam Studies In The Theory And History Of Linguistic Science Series 4*, 292:247, 2007.

[80] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[81] Wolfgang Wahlster. *Verbmobil: foundations of speech-to-speech translation*. Springer Science & Business Media, 2013.

[82] Guangxu Xun, Mingbo Ma, Yuchen Bian, Xingyu Cai, Jiaji Huang, Renjie Zheng, Junkun Chen, Jiahong Yuan, Kenneth Church, and Liang Huang. Data-driven adaptive simultaneous machine translation. *arXiv preprint arXiv:2204.12672*, 2022.

[83] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[84] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[85] Ruiqing Zhang, Chuanqiang Zhang, Zhongjun He, Hua Wu, and Haifeng Wang. Learning adaptive segmentation policy for simultaneous translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2280–2289, 2020.

[86] Shaolei Zhang and Yang Feng. Gaussian multi-head attention for simultaneous machine translation. *arXiv preprint arXiv:2203.09072*, 2022.

[87] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models. *ArXiv*, abs/2205.01068, 2022.

[88] Yudong Zhang, Shuihua Wang, and Genlin Ji. A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical problems in engineering*, 2015, 2015.

[89] Michał Ziemski, Marcin Junczys-Dowmunt, and Bruno Pouliquen. The united nations parallel corpus v1. 0. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3530–3534, 2016.

Appendix

SUSTAINABLE DEVELOPMENT GOALS

Degree to which the work is related to the Sustainable Development Goals (SDGs).

Sustainable Development Goals	**High**	**Medium**	**Low**	**Not applicable**
SDG 1. **No poverty.**				**X**
SDG 2. **Zero Hunger.**				**X**
SDG 3. **Good Health and Well-being.**			**X**	
SDG 4. **Quality Education.**	**X**			
SDG 5. **Gender Equality.**				**X**
SDG 6. **Clean Water and Sanitation.**				**X**
SDG 7. **Affordable and Clean Energy.**				**X**
SDG 8. **Decent Work and Economic Growth.**		**X**		
SDG 9. **Industry, Innovation and Infrastructure.**		**X**		
SDG 10. **Reduced Inequality.**	**X**			
SDG 11. **Sustainable Cities and Communities.**				**X**
SDG 12. **Responsible Consumption and Production.**				**X**
SDG 13. **Climate Action.**				**X**
SDG 14. **Life Below Water.**				**X**
SDG 15. **Life On Land.**				**X**
SDG 16. **Peace, Justice, and Strong Institutions.**			**X**	
SDG 17. **Partnerships for the Goals.**			**X**	

Thoughts on the relationship of the TFG/TFM regarding the SDGs and the more related SDG(s).

The United Nation's (UN) 2030 Agenda for Sustainable Development presents 17 Sustainable Development Goals seeking to balance the social, economical and environmental aspects for sustainable development. Within this framework, we wish to distinguish the following for this work:

- **SDG 4: Quality Education.** MT technology is leveraged daily to provide accessibility to vast amounts of educative multimedia content. In addition to this, a significant number of students across the globe use MT systems for various academic purposes, as an aid in document writing, foreign language learning, etc. The MT systems developed in this work will be used to translate technical material in the field of high-energy physics, providing accessibility to this content to non-native language speakers with interest in the subject.

- **SDG 8: Decent Work and Economic Growth**. One of the main areas of application of MT systems is aiding human translators. Here, MT software offers tools that suggest and predict translations, including translation memories and terminology banks, etc. which decrease the workload of human translators and improve their working conditions.

- **SDG 9: Industry, Innovation and Infrastructure.** This work is concerned with the application of state-of-the-art ML techniques for the development of MT systems that will be used in the CERN as a tool for more effective communication between researchers from different cultures, lowering language barriers and encouraging collaborative work. Our plan is to continue developing and improving upon our current MT systems by introducing innovative ideas in cutting-edge MT research.

- **SDG 10: Reduced Inequality.** MT systems play a key role in our lives, bridging the communication gap between speakers of different languages. In doing so, MT systems lower language barriers, reducing inequality and making communication across languages a reality.

While this work does not concern itself primarily with these, we wish to distinguish its connections with **SDG 3: Good Health and Well-being**, **SDG 16: Peace, Justice and Strong Institutions**, and **SDG 17: Partnership for the goals**. Regarding **SDG 3**, MT systems can be applied as tools to improve communication within the medical field, thereby motivating new research, while also aiding in the well-being of non-native speakers by providing tools that lower language barriers for their effective communication with medical professionals. Additionally, we note the connection to **SDG 16** and **SDG 17**, given that MT systems can be leveraged for meetings between governments of different countries, providing fast and accurate translations for exchanges between their authorities, as well as improving cross language communication as a way to motivate partnerships between institutions from different cultures.

We believe the remaining SDGs are not connected to this work, either due to them not being linked to MT, language technologies in general, or the research carried out in this work.