UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

A comparative study of Neural Machine Translation frameworks for the automatic translation of open data resources

DEGREE FINAL WORK

Degree in Computer Science

Author: Javier Iranzo Sánchez

Tutor: Jorge Civera Saiz
Cotutor: Alfons Juan Ciscar
Experimental director: Adrià Giménez Pastor

Course 2017-2018

Resum

La traducció automàtica (MT) és una de les àrees més actives dins de la intel·ligència artificial, particularment en el camp del reconeixement de formes. Recentment, aquesta àrea ha estat el focus d'atenció d'importants figures tecnològiques com Google, Facebook, Microsoft, etc. a causa de les millores de rendiment aconseguides per aquesta tecnologia gràcies a la incorporació de xarxes neuronals artificials, que formen l'espina dorsal de l'aprenentatge profund. En aquest sentit, la nova era de sistemes MT basats en xarxes neuronals (NMT) ha comportat traduccions automàtiques suficientment precises per facilitar la comunicació humana en una societat multilingüe. En aquest treball s'expliquen els models teòrics que formen la base de NMT, així com els coneixements previs necessaris per proporcionar una visió completa de l'àrea. A més a més, es desenvolupen sistemes NMT d'última generació utilitzant dos coneguts *frameworks* d'aprenentatge automàtic, Tensorflow i MXNet. Aquests sistemes s'entrenen amb les dades de dos cassos d'estudi. El primer és el *Workshop on Machine Translation* per a la traducció de comunicats de notícies y debat del Parlament Europeu d'alemany a anglés. El segon cas és poliMedia, el repositori principal de la UPV de vídeos educatius, que en aquest treball s'utilitzen per a la traducció d'espanyol a anglés. Els sistemes desenvolupats s'avaluen i es comparen pel que fa a la seua qualitat i eficiència, abans de ser desplegats en un entorn de producció real com poliMedia.

Paraules clau: reconeixement de formes, aprenentatge automátic, traducció automática, aprenentatge profund, comparació

Resumen

La traducción automática (MT) es una de les áreas más activas dentro de la inteligencia artificial, particularmente en el campo del reconocimiento de formas. 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, que constituyen la espina dorsal del aprendizaje profundo. En este sentido, la nueva era de sistemas MT basados en redes neuronales (NMT) ha traído consigo traducciones automáticas lo suficientemente precisas como para facilitar la comunicación humana en una sociedad multilingüe. En este trabajo se explican los modelos teóricos que forman la base de NMT, así como los conocimientos previos necesarios para poder proporcionar una visión completa del área. Además, se desarrollan sistemas NMT de última generación utilizando dos conocidos *frameworks* de aprendizaje automático, Tensorflow y MXNet. Estos sistemas se entrenan con los datos de dos casos de estudio. El primero es el *Workshop on Machine Translation* para la traducción de comunicados de noticias y debates del Parlamento Europeo de alemán a inglés. El segundo caso es poliMedia, el repositorio principal de la UPV de videos educativos, que en este trabajo se usa para la traducción de español a inglés. Los sistemas desarrollados se evalúan y comparan con respecto a su calidad y eficiencia, antes de ser desplegados en un entorno de producción real como poliMedia.

Palabras clave: reconocimiento de formas, aprendizaje automático, traducción automática, aprendizaje profundo, comparación

Abstract

Machine Translation (MT) is one of the most active areas in Artificial Intelligence, particularly in Pattern Recognition. MT has recently received a great deal of attention by key technology players such as Google, Facebook, Microsoft, etc. due to the performance boost experienced by MT technology thanks to the incorporation of artificial neural networks that constitute the backbone of deep learning. In this respect, the new era of Neural-based MT (NMT) systems has brought about accurate enough automatic translation to ease human communication in a multilingual society. In this work, conventional theoretical models behind NMT are introduced together with the required background to provide a comprehensive view. In addition, state-of-the-art NMT systems are built on top of two well-known frameworks for machine learning, Tensorflow and MXNet. These systems are trained using the data from two case studies. The first case is the Workshop on Machine Translation for the translation of broadcast news and European Parliament debates from German into English. The second case is poliMedia, UPV's main repository for educational video lectures, which in this work is used for translation from Spanish into English. The developed systems are evaluated and compared with respect to their quality and efficiency, before being deployed in a real-world production environment such as poliMedia.

Key words: pattern recognition, machine learning, machine translation, deep learning, comparison

Contents

Contents v
List of Figures vii
List of Tables vii

1 Introduction 1
1.1 Motivation and goals . . . 1
1.2 Pattern Recognition . . . 2
1.3 Machine Translation . . . 3
1.3.1 Word-Based Models . . . 5
1.3.2 Phrase-Based Models . . . 7
1.3.3 Neural-Based Models . . . 9
1.3.4 Computational graphs . . . 11
1.3.5 Evaluation of results . . . 12
1.4 Framework of this work . . . 13
1.5 Document structure . . . 13

2 Data processing 15
2.1 Data preparation . . . 15
2.1.1 Preprocessing steps . . . 15
2.1.2 Data filtering . . . 16
2.1.3 Byte Pair Encoding . . . 17
2.2 Data collection . . . 18
2.2.1 WMT2017: German → English . . . 19
2.2.2 poliMedia: Spanish → English . . . 20

3 Attention-based RNN NMT models 23
3.1 Overview of NMT models . . . 23
3.2 RNN-Attention . . . 24
3.2.1 Attention Mechanism . . . 25
3.3 Computer tools . . . 26
3.3.1 Tensorflow-nmt . . . 27
3.3.2 Sockeye . . . 27
3.4 System description . . . 28
3.5 Results . . . 29

4 Transformer Self-Attention NMT models 35
4.1 Transformer . . . 35
4.2 System description . . . 42
4.2.1 Data filtering . . . 42
4.3 Results . . . 43

5 System integration into a transcription and translation platform 49
5.1 TLP . . . 49
5.2 Translation generation . . . 51
5.3 Performance evaluation . . . 52

6 Conclusions 55

Bibliography **57**

List of Figures

1.1 Example of a typical Pattern Recognition Pipeline. 4
1.2 Example of the phrase segmentation used by a SMT model when translating sentence from the WMT corpus. 7
1.3 Computational graph representation of the operations carried out in one of the hidden layers of a neural network. 11

2.1 Example of tokenization with Moses. 16
2.2 Example of BPE segmentation of a text. 18
2.3 Screenshot of a poliMedia video. 20

3.1 Encoder/decoder architecture. 27

4.1 Figure of the Transformer architecture. 39
4.2 Transformer Encoder Block. 39
4.3 Transformer Decoder Block. 40
4.4 Number of sentences with perplexity under a given threshold. 45

5.1 Example of some of the videos uploaded to TTP. 50
5.2 General TLP architecture. 50
5.3 TLP Workflow. 51

List of Tables

2.1 WMT 2017 corpus statistics. 19
2.2 Paracrawl corpus statistics. 20
2.3 poliMedia corpus statistics. 21

3.1 Results obtained for models trained with the poliMedia corpus. 30
3.2 Resource consumption for systems trained with the poliMedia corpus. . . 32
3.3 Results obtained for models trained with the WMT corpus. 32
3.4 Resource consumption for WMT models. 34

4.1 Results obtained for Transformer models trained with the poliMedia corpus. 43
4.2 Results obtained for Transformer models trained with the WMT corpus. . 44
4.3 Resource consumption for the WMT task. 44
4.4 Results obtained for different amounts of filtered sentences, WMT18 task. 47
4.5 WMT18 news track results. 47

5.1 Resource consumption and system speed with respect to different batch sizes. 53

5.2 Translation samples comparing the performance of SMT models, WMT corpus. 54
5.3 Translation samples comparing the performance of SMT models, poliMedia corpus. 54

CHAPTER 1
Introduction

This work studies and compares different solutions to the problem of Machine Translation (MT), with the aim of obtaining a system that is able to produce accurate translations in an efficient way, in order to be used for the translation of Open Educational Resources (OER). This chapter introduces the motivation and context of this work, as well as the key concepts that will be necessary for the reader to understand the rest of this work.

Motivation and goals

Due to recent technological development, the idea that machines could be able to automatically generate translations has become a reality. A few years ago, most people did not imagine that MT systems that achieved good performance in a variety of domains could be developed in a short period of time. And yet, thanks to the combination of a series of research developments coming both from the academia and the industry, the performance of MT systems has surpassed many of its previously believed limits. While there are still many challenges and research areas to explore, we are now at a moment where MT has achieved a level of performance that makes it possible to leverage this technology in a variety of ways that were not possible until now. This opens up new avenues for the development of systems that are able to provide cheap solutions for the translation of massive amounts of text, either as standalone systems or as a tool to reduce time for human translators. The importance of this technology and the business value that it can bring has not been ignored by key technology players. Nowadays, all technological giants such as Google[1], Facebook [2], Microsoft[3], eBay[4] and Amazon[5] have all developed their own MT systems that they use in their day-to-day operations, and some of them also offer the use of those translation systems to their clients. Many organizations have adapted their processes to use MT services in order to translate massive quantities of text as required by their business needs. In a way, it can be said that MT is quickly becoming ubiquitous, and this trend will continue to increase in the future as new research developments continue to improve the quality and reduce the cost of generating these automatic translations.

The previously mentioned advances in automatic translation become even more interesting when looking at the recent trends in education. In a trend that started in the

[1] https://translate.google.com

[2] https://code.facebook.com/posts/289921871474277/transitioning-entirely-to-neural-machine-translation/

[3] https://www.microsoft.com/en-us/translator

[4] http://labs.ebay.com/research-areas/research-machine-translation

[5] https://aws.amazon.com/es/translate/

early 2000, and whose speed has only been increasing, new educational models that include media resources into their programs have been incorporated into most modern educational institutions. The most prominent example of this trend is the possibility of online education, either as a way of supplementing traditional learning, or as a complete replacement to classroom courses. Online platforms like edX, Coursera or Udacity are some of the examples of this new learning paradigm. Higher-education institutions such as universities have placed an ever increasing amount of effort into developing and disseminating OER, which allows many individuals to gain access to high-quality educational content that might not be available to them otherwise.

This work aims to find synergies between the two previously explained developments, in order to boost the beneficial impacts of these technologies. Our goal is to bring the recent developments of MT into the OER world. Therefore, the goals of this work can be summarized as follows:

- To become familiar with MT techniques and the current state of the art, with the aim of understanding the key principles of this field and the methods used to build MT systems and to be able to explain them.
- Identify available tools for building MT systems, learn how to use them and compare their advantages and disadvantages, while gaining familiarity with the complexity and methods used in their training.
- Identify and understand new research proposals to improve the performance of these systems, and how to use, integrate or implement them into working systems.
- Learn to develop a system that using open tools is able to compete with the current best MT systems, and to empirically compare our performance with them.
- Integrate the developed MT systems into a real production environment tasked with the translation of OER.

Pattern Recognition

Pattern recognition is the field of science that is "concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories" [6].

Today there are many examples of successful applications of this field to diverse problems that range from detecting which objects appear in a photo [36] to teaching a computer how to play a video-game [29] to the automatic translation of texts, the goal of this work.

The goal of this field is to construct a system that is able to make the correct decisions or actions (according to a certain criterion) for a given scenario. The system decision is computed by a mathematical model whose output depends on a series of parameters or weights. These weights are selected by using a learning algorithm that learns them from a set of training data. The algorithm is supplied with a set of samples that are used to learn how to act by extracting knowledge from the data.

This training data is the result of an acquisition process that transforms a real-world instance of a problem into a digital representation ready to be used by the learning algorithm. It is desirable that this representation contains as much useful information as possible, disregarding useless features, applying transformations and sometimes generating new features that are better suited for the algorithm, a process known as feature extraction.

The output of this pre-processing step is a series of feature vectors that contain the numerical representation of the data that will be used to take a decision. A feature vector is usually denoted by the random variable $\mathbf{x}$.

One common approach to pattern recognition is that of *supervised learning*, where the algorithm is not only given a data sample, but also a label, typically denoted as $\mathbf{y}$, that tells the algorithm what is the desired output for that data sample $\mathbf{x}$. This means that the algorithm is provided with the input and the correct output for that sample, so it can then use that information in order to match its action with the desired one.

When presented with an $\mathbf{x}$, $d \in D$ represents the decision taken by the system. If we define $p(d|\mathbf{x})$ as the probability of that decision being right[6], the probability of error if the system takes decision d is defined as:

$$p_d(error|\mathbf{x}) = 1 - p(d|\mathbf{x}) \tag{1.1}$$

The best probability of error, that is, the one that minimizes the probability of error, can be developed as:

$$p_*(error|\mathbf{x}) = \min_{d \in D} p_d(error|\mathbf{x}) = 1 - \max_{d \in D} p(d|\mathbf{x}) \tag{1.2}$$

The minimum error classifier takes a decision $\hat{d}$ such that:

$$\hat{d}(\mathbf{x}) = \arg\max_{d \in D} p(d|\mathbf{x}) \tag{1.3}$$

Due to the difficulties involved in computing $p(d|\mathbf{x})$, it is common to apply Bayes' rule to rewrite the formula as:

$$\hat{d}(\mathbf{x}) = \arg\max_{d \in D} p(d|\mathbf{x}) = \arg\max_{d \in D} \frac{p(x|d)p(d)}{p(x)} \tag{1.4}$$

Since the denominator does not depend on d, it can be dropped:

$$\hat{d}(\mathbf{x}) = \arg\max_{d \in D} p(x|d)p(d) \tag{1.5}$$

For example, in the case of a classification problem, d represents the decision of assigning a certain class label (usually written as c) to a sample.

The learning algorithm is used to estimate these probability distributions based on the provided training data, with the goal of obtaining a system that is able to *generalize* this knowledge, that is, to be able to make the right decision when being presented with new samples that were not shown to the system during the training process. The typical process of a Pattern Recognition system that we have explained is shown graphically in Figure 1.1.

Machine Translation

MT can be seen as an application of Pattern Recognition that seeks the development of computer systems that are able to automatically translate texts. When given a sentence, the system should produce a good translation of it into another language, with the goal of preserving the original meaning as much as possible.

[6] We will assume that a decision is either right, and therefore has cost 0, or it is wrong and has cost 1.

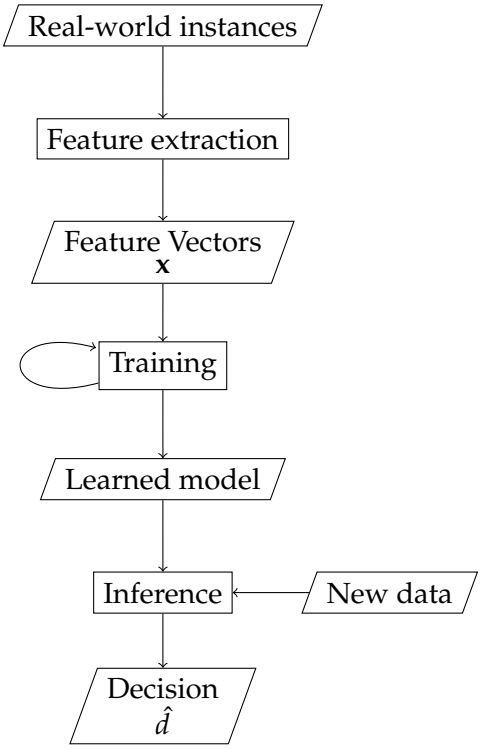


Figure 1.1: Example of a typical Pattern Recognition Pipeline.

Formally, given a sentence in one language, $\mathbf{x} = x_1, x_2 \ldots, x_I$, the goal is to find the best translation, that is, the $\mathbf{y} = y_1, y_2 \ldots, y_I$ that maximizes $p(\mathbf{y}|\mathbf{x})$.[7]

This probability is learned from parallel corpora, collections of text that contain sentences in one language paired with their translations into another language. This is then used by the model to obtain the appropriate translation knowledge. The system is presented with sentences from the source language (the language we are translating from), and their translations into the target language (the language we are translating into), and learns the relationship from there. This means that we obtain systems that are able to translate only from one language into another, for example, from Spanish into English, but we would have to train another system to translate from English into Spanish.[8]

The same way that in the general case, it it usual to decompose the translation probability using Bayes' rule:

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{y})p(\mathbf{y})}{p(\mathbf{x})} \tag{1.6}$$

One can imagine that a sentence was originally written in the language we want, but it somehow got corrupted along the way, and we ended up with a sentence in the source language. This means, that for some possible translation $\mathbf{y}$ the probability of that sentence being the translation, $p(\mathbf{y}|\mathbf{x})$ can then be interpreted as the probability that the sentence was the one that was emitted originally, $p(\mathbf{y})$, multiplied by the probability that this sentence was disrupted and became the sentence in the source language, $p(\mathbf{x}|\mathbf{y})$. This model is known as the *noisy-channel model* [40], $p(\mathbf{y})$ as the *language model*, and $p(\mathbf{x}|\mathbf{y})$ as the *translation model*.

Learning a conditional probability distribution for translations is not enough for a fully functional MT system, since by itself, knowing how good of a translation is a certain

[7] It is common to use the letters f and e instead of x and y in MT texts, both notations are equivalent.

[8] There exists research both for training multilingual models and for unsupervised learning, but they both remain open research areas.

sentence does not provide us enough information to know if it is the best one, as there could be others that are much better. Our ideal goal is to find a translation $\hat{\mathbf{y}}$ such that:

$$\hat{\mathbf{y}} = \arg\max_{y} p(\mathbf{y})p(\mathbf{x}|\mathbf{y}) \tag{1.7}$$

As in the previous case, we have dropped $p(\mathbf{x})$ from the equation since it does not depend on y.

We require an additional step, known as *decoding*, that tries many possible translations in order to search for the best final translation. Since there are infinitely many possible translations in the search space, the search is carried out only in a subset of this space, with the hope that the best translation, or at least one that is good enough, is present on that subset. Techniques such as stack decoding [15] or beam-search [21] are used to carry out this task.

The following sections explain the main three models that have been historically used in MT: word-based models, phrase-based models and neural network models. The models are explained in chronological order.

Word-Based Models

Translation model

Word-based models [7, 8] estimate the translation probability by assuming that translations are produced by individually translating each word of the sentence. We will now explain a word-based model known in the literature as IBM-1. Due to the application of Bayes' rule, the translation direction has been inverted, but we will refer to $\mathbf{x}$ as the source sentence and $\mathbf{y}$ as the target sentence in order to maintain coherence with the nomenclature used in the rest of this work. This model uses two main concepts, a *lexical translation* model, that gives the probability of each possible translation for a certain word, and an *alignment*. An alignment indicates, for each of the source positions, which is the target position that corresponds to the word they are a translation of. We can imagine this alignment as a function that returns a position i for every position j given as input.

A word-based model for producing a translation $\mathbf{x}$ given an input sentence $\mathbf{y}$ can be defined as:

$$p(\mathbf{x}|\mathbf{y}) = \sum_{\mathbf{a}} p(\mathbf{x}, \mathbf{a}|\mathbf{y}) \tag{1.8}$$

$\mathbf{a}$ is an alignment vector that indicates, for each of the source words, which is the target word that produced them. If $a_j = n$, it means that the word in position j is a translation of the target word in position n. A NULL token is appended to position 0 of the target sentence to allow for source words that are not aligned with any of the other target words. The alignment vector provides us with a mapping that we can consult in order to know which is the target word that corresponds with a certain source word.

For some fixed alignment $\mathbf{a}$ and target sentence $\mathbf{y}$, the probability of a translation $\mathbf{x}$ is defined as:

$$p(\mathbf{x}, \mathbf{a}|\mathbf{y}) = \prod_{j=1}^{J} p(x_j, a_j|x_1^{j-1}, a_1^{j-1}, \mathbf{y}) = \prod_{j=1}^{J} p(a_j|x_1^{j-1}, a_1^{j-1}, \mathbf{y})p(x_j|x_1^{j-1}, a_1^{j}, \mathbf{y}) \tag{1.9}$$

We are going to make a series of assumptions in order to compute this probability. First, we assume that the alignment probability between target and source words is given

by a uniform probability distribution:

$$p(a_j|x_1^{j-1}, a_1^{j-1}, \mathbf{y}) := \frac{1}{I+1} \tag{1.10}$$

Additionally, and because we are working with a word-based model, we assume that the translation of each word of the source sentence only depends on the corresponding target word it is aligned to.

$$p(x_j|x_1^{j-1}, a_1^{j}, \mathbf{y}) := p(x_j|y_{a_j}) \tag{1.11}$$

The previous two assumptions allow us to obtain a simplified computation for Equation 1.9:

$$p(\mathbf{x}, \mathbf{a}|\mathbf{y}) = \frac{1}{I+1} \prod_{j=1}^{J} p(x_j|y_{a_j}) \tag{1.12}$$

Provided with an alignment for every sentence, we can estimate the lexical translation probabilities for pair of words by counting the number of times it has been translated and normalizing the results.

$$p(u|v) = \frac{N(u,v)}{\sum_{u'} N(u',v)} \tag{1.13}$$

$N(u,v)$ is a count function that computes the number of times a word v has been translated as u. For a single sentence pair and its alignment vector $\mathbf{a}$, this function is computed as:

$$N(u,v) = \sum_{j=1}^{J} \delta(x_j = u)\delta(y_{a_j} = v) \tag{1.14}$$

The previous function can be extended to a corpus containing N parallel sentences in a straightforward way. The problem is that our parallel corpus does not provide us with an alignment between words, so we can not directly compute the lexical translation probability. In this case, the alignment acts as a hidden variable. The parameters of the model (the lexical translation probability distribution) can be trained with the expectation maximization algorithm (EM) [12], an iterative algorithm that computes an estimation of the alignment on each step.

Once we have obtained our translation probability distribution, the probability of translating an entire sentence, $p(\mathbf{x}|\mathbf{y})$, can be computed as:

$$p(\mathbf{x}|\mathbf{y}) = \frac{1}{(I+1)^J} \prod_{j=1}^{J} \sum_{i=1}^{I} p(x_j|y_i) \tag{1.15}$$

Language model

A language model estimates the a priori probability that a sentence occurs in a language, $p_{LM}(\mathbf{y})$.

The inclusion of a language model provides a MT system with information about the expected structure of sentences in a certain language. This addition means that the system has some knowledge about what a "proper" sentence in that language is, and this can provide a more fluent output and help to decide in case of ambiguities in translation.

src: 28 | @-@ jähriger | Koch | in San | Francisco | Mall | tot | aufgefunden
tgt: the 28 | @-@ year @-@ old | Koch | in San | Francisco | mall | found | dead .

Figure 1.2: Example of the phrase segmentation used by a SMT model when translating sentence from the WMT corpus.

As far as language models are concerned, a sentence is made up of words, so one initial approach to language modelling consists in using the chain rule to decompose the probability of a sentence.

$$p_{LM}(y_0^I) = \prod_{i=1}^{I} p(y_i|y_0^{i-1}) \tag{1.16}$$

In practice, this theoretical assumption is not used due to the exponential growth of possible histories. One option in order to avoid this exponential growth is to make the assumption that a word only depends on the n preceding words, therefore:

$$p(y_i|y_0^{i-1}) \simeq p(y_i|y_{i-(n-1)}^{i-1}) \tag{1.17}$$

This model is known as the *n-gram model*, and it has a very simple estimation based on counting the number of appearances of sequences of n-words.

$$p(y_i|y_{i-(n-1)}^{i-1}) = \frac{N(y_{i-(n-1)}, \ldots, y_{i-1}, y_i)}{\sum_v N(y_{i-(n-1)}, \ldots, y_{i-1}, v)} \tag{1.18}$$

Language models are usually evaluated in terms of perplexity computed over a text $y_1, y_2, \ldots, y_n$.

$$PP = 2^{-\frac{1}{N}\log p(y_1, y_2, \ldots, y_n)} \tag{1.19}$$

The perplexity is a estimation on how many different words on average can follow a given word according to the language model. The lower this number is, the more confident the model is about what the next word will be.

Phrase-Based Models

Phrase-based models [25] translate sentences by decomposing them into phrases, a small set of contiguous words, and translating each of those phrases in order to obtain the translated sentence. Words can only belong to a single phrase, so there is no overlap between phrases. The use of phrases as the basic unit of translation instead of words allows these models to achieve greater performance, and until the arrival of neural-based systems, they were the state of the art.

Figure 1.2 shows the phrase segmentation selected by a phrase-based model built using the Moses toolkit, for the translation of a sentence from German into English.

Log-Linear Models

We have previously used Bayes' rule in order to split the computation of $p(x|y)$ into two parts. However, it might be beneficial to consider more components for our model, even if their inclusion can not be mathematically justified. For example, in phrase-based models, the translation model is further split into two models, the *lexicon translation model*,

a measure of how good is the translation of a phrase, and the *reordering model*, that tells us how likely it is that the translated phrases are ordered that way.

Log-linear models are models whose logarithm equals a linear combination of a set of feature functions of the model, h_i, that depend on a random variable R.

$$p(R) = \exp \sum_{i=1}^{n} \lambda_i h_i(R) \tag{1.20}$$

A combined model that assigns weights to the different components,

$$p(\mathbf{y}|\mathbf{x}) = p_t(\mathbf{x}|\mathbf{y})^{\lambda_t} p_r(\mathbf{x}|\mathbf{y})^{\lambda_r} p_{LM}(\mathbf{y})^{\lambda_L M} \tag{1.21}$$

is equivalent to a log-linear model defined as:

- $R = (x, y, start, end)$
- $n = 3$
- $h_1 = p_t(\mathbf{x}|\mathbf{y})$
- $h_2 = p_r(\mathbf{x}|\mathbf{y})$
- $h_3 = p_{LM}(\mathbf{y})$

Treating the translation model as a log-linear model opens up the possibility of including additional feature functions that we might find useful for translation. We will now describe each one of the three components that form the basic phrase-based model:

Language model

The language model is independent of the translation model, so the same explanation used in the word-based model applies to the phrase-based model.

Translation model

For a certain segmentation of **x** into I different x_i phrases, the translation model is defined as:

$$p_t(\bar{x}_1^I|\bar{y}_1^I) = \prod_{i=1}^{I} \phi(\bar{x}_i|\bar{y}_i) \tag{1.22}$$

$\phi(x_i|y_i)$ is a phrase translation table that scores the goodness of a translation. This table is estimated in a two step process. Given a corpus, we first extract a series of *correct* phrase pairs for each sentence pair. Once we have obtained all those pairs, we estimate the transition probability for a pair based on the number of times that pair has been extracted, defined as $N(\bar{x}, \bar{y})$, and normalize the result.

$$\phi(\bar{x}|\bar{y}) = \frac{N(\bar{x}, \bar{y})}{\sum_{x'} N(\bar{x}', \bar{y})} \tag{1.23}$$

The concept of what constitutes a correct phrase pair, meaning one that is consistent with a given word alignment is outside the scope of this work. We refer interested readers to chapters 4 and 5 of [24] in order to learn more about word alignments and phrase extraction.

Reordering model

Each phrase in the source sentence is translated into another phrase in the target language, but this does not mean that the target phrases must appear in the same order, as sometimes it is better to alter their order to better convey the meaning of the sentence. The reordering model is in charge of estimating this reordering probability.

$$p_r(\bar{x}_1^I|\bar{y}_1^I) = \prod_{i=1}^{I} d(start_i - end_{i-1} - 1) \tag{1.24}$$

The reordering function d is computed with respect to the distance between the start of phrase i ($start_i$) and the end of the previous phrase (end_{i-1}), computed in the source sentence.

One possible definition for d is that of an exponential decay function, $d(x) = a^{|x|}$, in order to penalize bigger movements.

Neural-Based Models

This section briefly introduces the basic concepts of neural networks. Readers are recommended to consult chapter 6 of [14] in order to gain a more profound understanding.

Neural networks started being applied to MT quite recently, but they are not a new technology. The Perceptron[34], precursor of the neural network, was introduced in 1957, and the backpropagation algorithm [35] that allowed the training of multi-layer neural networks was introduced in the 1980's. Since then, these models have been applied to a variety of Pattern Recognition problems.

In contrast with phrase-based models, NMT models do not decompose $p(\mathbf{y}|\mathbf{x})$ using Bayes' rule, instead they estimate it directly.

The most basic type of neural network is the multilayer perceptron (MLP). The MLP is a feedforward neural network (whose nodes do not form cycles), and is the most commonly used model in almost all types of classification problems. These models are made up of an input layer, a variable number of hidden layers and an output layer. Each layer applies some computation to the inputs of the previous layer, and this result is then used by the next layer. We will now describe the equations governing how an MLP functions.

The first hidden layer receives as input the feature vector $\mathbf{x}$:

$$\mathbf{h}^{(1)} = g^{(1)}(\boldsymbol{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \tag{1.25}$$

For the other layers, the output of a hidden layer i is computed as:

$$\mathbf{h}^{(i)} = g^{(i)}(\boldsymbol{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}) \tag{1.26}$$

where $\boldsymbol{W}^{(i)}$ is the weight matrix of the layer and $\mathbf{b}^{(i)}$ is the bias term of the layer. $g^{(i)}$ is the activation function of the layer, and this is usually a non-linear function that allows the model to learn non-linear relations. The input to the function $g(.)$ is sometimes denoted as $\mathbf{a}$.

The output layer is in charge of applying a transformation to the set of features produced by the hidden layers in order to obtain the final result of the computation. For example, in the case of a classification problem, the output layer produces a probability distribution over all possible classes by applying a softmax function.

$$softmax(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{1.27}$$

$$p(c|x) = softmax(\mathbf{W}^{(out)}\mathbf{h}^{(I)} + \mathbf{b}^{(out)}) \tag{1.28}$$

The set of weight matrices and biases that define the parameters of the model ($\boldsymbol{\theta}$) is iteratively trained by using gradient descent and the backpropagation algorithm [35]. Gradient descent is an iterative technique that performs an optimization with respect to a cost function, $J(\boldsymbol{\theta})$, that measures the performance of the parameters of our model with respect to the training data. On each step, we subtract the gradient of the cost function with respect to the parameters.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{1.29}$$

being α the *learning rate* or *step size* that controls how big is the update we make on each step. Because the cost function is usually defined as computing the average of a loss function over each of the training samples, this computation becomes unfeasible when dealing with big datasets. That is why it is common to use *minibatch* gradient descent instead, where we compute the cost function with regards to a small subset or minibatch of the training data. Conceptually, this can be understood as updating the parameters with a noisy estimation of the gradient.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \epsilon) \tag{1.30}$$

The previous explanation applies to the architecture and training of any neural network, but we will now introduce a special type of neural network that is better suited for MT. Because we are working with sequential data, it is desirable to use a network that is able to model dependencies between words. Recurrent Neural Networks (RNN) are a type of neural networks that are specialized in working with sequential data. These networks compute a state that depends on the state on the previous time step. The equation for the hidden layer of a basic recurrent network at time t is given by:

$$\mathbf{h}_t^{(i)} = g^{(i)}(\boldsymbol{W}^{(i)}\mathbf{h}_t^{(i-1)} + \boldsymbol{U}^{(i)}\mathbf{h}_{t-1}^{(i)} + \mathbf{b}^{(i)}) \tag{1.31}$$

Compared with Equation 1.26, the difference is the addition of weight matrix $\boldsymbol{U}$, that controls the effect of the previous state.

Even so, learning long-term dependencies is very hard, due to the gradients' tendency of either becoming infinitesimally small or exponentially big due to the application of the same weight matrix over multiple time steps. This is know as the gradient vanishing or gradient explosion problem, and is studied in more detail in works such as [18]. In order to fight this problem, the use of special units that are better suited for this task, such as the LSTM unit [19] has become standard. The hidden layer output for these networks depends not only on $\mathbf{h}_t^{(i-1)}$ and $\mathbf{h}_{t-1}^{(i)}$, but also on an internal state $\mathbf{c}_t$ that can store values between each time step.

Having explained the basics of neural networks, we can are now ready to move on to how they are applied to MT in what is known as Neural Machine Translation (NMT). NMT systems are explained in detail in Chapter 3.

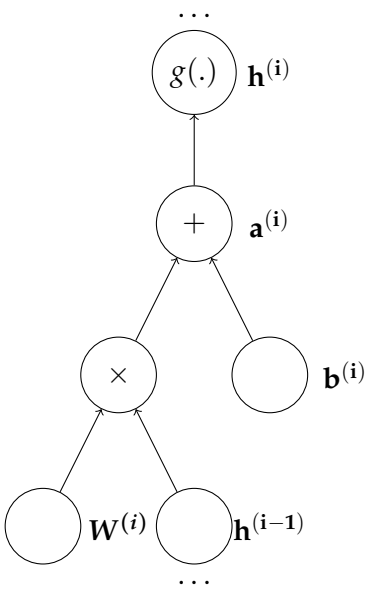


Figure 1.3: Computational graph representation of the operations carried out in one of the hidden layers of a neural network.

Computational graphs

As explained in Section 1.3.3, the weights of a neural network are usually trained by gradient descent using the backpropagation algorithm. This means that for each training step, we need to compute the derivative of the cost function with respect to each variable of our model. Because this is one of the most time-consuming parts of the training procedure, it is interesting to have tools that make this process as easy as possible.

In order to compute these derivatives in an easy and efficient way, libraries such as Tensorflow and MXNet internally define models as a computational graph. A computational graph is a directed graph composed by a series of *variables*, each represented by a node, and operations between those variables. The operations are represented with the help of *edges*, so that a node with incoming edges defines an operation (or function) over the input variables. Nodes usually define simple operations such as the sum or multiplication of its inputs. The combination of different operations, whose input is usually the output of a previous node, allows the definition of a graph that represents a set of mathematical computations and the existing dependencies between them. Additionally, computational graphs offer an explicit representation that helps avoid many kinds of ambiguities that can occur when trying to understand a mathematical process. Figure 1.3 shows an example of one computational graph that represents the computations carried out in the layers of a neural network.

We can take advantage of these graphs in order to facilitate processes such as training machine learning models. The definition of a model as a series of simple operations enables fast parallel computation of the model's output, since the edges define the existing dependencies between operations. The user is therefore freed from the burden of needing to manually define the execution order for its model. Furthermore, if each operation also defines how to compute its derivative with respect to each of its inputs, the backpropagation algorithm can be applied in an straightforward way.

Both libraries offer a graph language that already defines a series of different mathematical operations and their respective derivative computation. This allows us to use any arbitrary model just by defining its model graph according to the language definition of the chosen library.

Evaluation of results

The question of how to best assess the translation quality of MT systems remains open. Manual evaluation, that is, evaluation made by humans about the quality of the translated text, could very well be the best evaluation measure, but it has the disadvantage of needing a human to carry out the task. Carrying out manual evaluations every time we define a new system configuration and translate a large amount of test sentences quickly becomes unfeasible.

This has given rise to search for *automatic evaluation* metrics that are ideally correlated with human judgement. Automatic evaluation is carried out by comparing the output of a system with a *reference translation* produced by a human. The most basic evaluation metric is the *precision*, the ratio between correct output words (shared words between the system output and the reference translation) and the number of words present on the output sentence. The problem with this metric is that it does not penalize short sentences and therefore can be easily fooled [9]. A better evaluation measure that takes the idea of precision into account together with hypothesis length is the Bilingual Evaluation Understudy (BLEU) [30] score. The BLEU score computes a modified precision, p_n, at different n-gram levels. Unlike regular precision, the clipped-precision used by the BLEU metric requires that an n-gram appears the same number of times both in the reference translation and in the candidate translation. If a certain n-gram appears more times in the candidate than in the reference, it will only be counted as correct as many times as it appears in the reference.

$$AveragePrecision(N) = \frac{1}{N}\sum_{n=1}^{N} \log p_n \tag{1.32}$$

This score also includes a Brevity Penalty term that is detrimental if the length of the candidate translation (c) is smaller than the length of the reference (r).

$$BrevityPenalty = \begin{cases} 1 & : if\ c > r \\ \exp(1 - \frac{r}{c}) & : if\ c \leq r \end{cases} \tag{1.33}$$

The usual definition of the BLEU is calculated over the concatenation of all test sentences, and is usually computed up to n-grams of order 4, such that:

$$BLEU(4) = BrevityPenalty * AveragePrecision(4) \tag{1.34}$$

The final result is a value ranging from 0 to 1, higher values are better. The value is usually multiplied by 100 to obtain better readability.

Another common automatic evaluation measure is the Translation Error Rate (TER) [41]. The TER measures the number of edits required to transform the candidate hypothesis into the reference, divided by the number of words in the reference.

$$TER = \frac{\text{Number of required edits}}{\text{Length of the reference}} \tag{1.35}$$

The available type of edits are:

- Insertion of a word
- Deletion of a word

[9] A system that emitted the translation "the" for any given input sentence would achieve an unusually high precision, since "the" is the most common English word. [24]

- Substitution of a single word by another
- Movement of a block of contiguous word to another part of the sentence

The optimal number of edits is approximately computed with a greedy algorithm. Since this is a score that measures how many edits are required, lower scores of TER are better. As in the case of BLEU, the value is usually multiplied by 100 when being reported.

Framework of this work

This work has been made possible thanks to the author's research internship at the Machine Learning and Language Processing (MLLP) research group of Universitat Politècnica de València (UPV). This stay was carried out in the context of the H2020 X5gon project, and was funded by the Spanish Ministry of Education Research Collaboration Grant nr. 2017/32/00005. X5gon is a Horizon2020 project funded by the European Commission. The work explained is this document was carried out during UPV's participation in this project.

X5gon's goal is to develop an open platform that will combine scattered OER sites into an analytics network to improve the learning experience of users.

Out of the 5 types of solutions proposed in the project (cross-site, cross-domain, cross-modal, **cross-language** and cross-cultural), the presence of a cross-language component that aims to provide cross lingual content recommendation, requires the use of efficient translation tools that are able to cope with the massive amounts of OER content available.

Document structure

This document is organized into 6 chapters. The first chapter serves as an introduction to the fields of Pattern Recognition and Machine Translation, how MT systems work and how they are evaluated. The motivation and goals of this work have also been introduced.

Chapter 2 describes the data preprocessing steps that need to be carried out prior to system training, as well as the data collected in order to carry out this work. Chapter 3 introduces the principles behind Neural Machine Translation and the Attention-based RNN architecture. Once those theoretical concepts have been explained, the chapter moves onto describing the different systems that have been trained based on that architecture and a detailed description of the obtained results. Chapter 4 introduces the state-of-the art Transformer architecture and the theoretical concepts behind it. A series of systems trained with this architecture are explained in detail. This chapter ends with a comprehensive description of the use of data filtering for system training and its inclusion for participation in the WMT18 competition. Chapter 5 describes the work carried out in order to integrate the translation systems obtained as output of the previous process into a live production environment. The work finishes with an overview of the different conclusions that have been obtained as a result of this works as well as a summary of the work carried out, available on Chapter 6. It is recommended that the reader follows a sequential reading order, but experienced readers can skip the sections describing the data used and the theoretical models if they are familiarized with the field and go straight into the description of the different trained systems.

CHAPTER 2
Data processing

This chapter introduces the data processing techniques and decisions applied to the available data, in order to obtain a prepared form that allows us to maximize the performance of the developed models. Data processing and preparation is a crucial step in the development of our systems if we want to obtain competitive results. This chapter also describes the datasets themselves, used for training and testing the different models. The chosen datasets are the news translation task of WMT from German to English, and the poliMedia dataset from Spanish to English. Both datasets are bilingual datasets whose training sets contain sentences with their corresponding translation. Models will be trained on this data and then given the test source sentences for their translation into the target language. The translations are then evaluated against the actual translations of the test set.

Data preparation

Preprocessing steps

MT systems are usually limited by the amount of different words that they consider in the training process, also known as vocabulary size. In the case of NMT systems, the size of the final softmax layer depends on the vocabulary size, and the bigger its size, the more computationally and memory demanding becomes the training process of the MT system. Thus, the choices we make when building the vocabulary for our systems will have considerable effects on system performance, and is one of the main areas that need to be considered when building MT systems. All of this means that we want to reduce the number of unnecessary words and only consider those that are useful. This is achieved with a *tokenization* step, that breaks up a text into the individual units (*tokens*) that will be considered by the system. Ideally, we want this step to produce an output that does not hinder a system ability to generate translations while at the same time keeping the number of unique tokens as low as possible. There are many cases where we need to make a decision about what constitutes a word, such as punctuation, compound words, etc.

Consider for example the case of a word car and that word followed by a comma car,. It does not make sense to consider both occurrences as different words, and to repeat the same for every word that appears followed by a comma. Instead, it would be better to consider car as a word and , as another. Figure 2.1 shows an example of tokenization made with the Moses tokenizer.

Deciding what to do with upper-cased text is another preprocessing decision related with vocabulary size. Once again, if we do nothing, we can find that we end up with a

Original text: Just because he's wise, doesn't mean that he's honest.
Tokenized text: Just because he 's wise , doesn 't mean that he 's honest .

Figure 2.1: Example of tokenization with Moses.

vocabulary that contains multiple entries for what turns out to be the same word. For example, any word that appears at the start of a sentence, and is therefore written with its first letter capitalized, can appear with two different spellings. One simple solution to this problem is to convert the entire text to lower-case characters, therefore ensuring that we do not waste vocabulary size due to capitalization differences. This method is very fast and provides a good reduction in the number of tokens, but there exist cases where the capitalization of a word does provide some linguistic information that could be lost otherwise. For example, nouns in German are always capitalized. A *truecasing* model tries to transform text to its appropriate capitalization, by collecting different statistics and building a model that makes a prediction about the appropriate capitalization for each word.

In this work, the poliMedia corpus was converted to lowercase, and a truecasing model was applied to the WMT corpus. The truecasing was carried out with the Moses Truecaser. This model changes words at the start of a sentence to their most common form, as well as any word that would be unknown if not changed.

Data filtering

Data filtering, sometimes called data selection, is an umbrella term for a variety of techniques that share the same goal: Selecting a subset of the data of a corpus in order to train a MT system. Traditionally, the term data selection has been used to put emphasis in extracting the best set of sentences out of a corpus. This can be carried out, for example, in the context of Domain Adaptation, which consists in training a general purpose system and then adapting it to translate data from a specific domain. For example, one could train a general German to English system, and then take that base system and fine-tune it with sentences of the medical domain in order to use it to translate medical documents.

The focus of this approach changes when taking into account that the performance of NMT models depends greatly on both quality and quantity of the training data. Recent works such as [9] and [5], show that NMT systems are very susceptible to any type of noise in the training data. Whereas with data selection we are looking to select the best sentences in order to improve performance, when we talk about data filtering we want to remove those noisy sentences that degrade MT performance.

If we want to utilize some corpus that contains noisy data, it is therefore very important that we carry out some sort of data filtering. Otherwise we could find out that we have achieved the opposite of what we were trying to do: The additional data might degrade the model instead of improving it.

Techniques

One possible approach to data filtering is to use a translation model to score the sentence pairs that form our training corpus. This can be carried out by taking the translation model and computing the translation probability $p(y|x)$ given by the model to every

sentence pair (x, y). This gives us a score for every pair that we can use to rank them and select only those that we consider adequate. The drawback of this approach is the need to build a good enough translation model before being able to carry out the filtering. While running the translation model for each sentence on a small corpus might be feasible, filtering large corpus with this approach takes considerable time, time that could be spent on other endeavours such as training additional systems for an ensemble.

A different approach that is much cheaper computationally is to filter by using language models. This can be carried out by first selecting an in-domain or clean corpus, and training two language models with this data, one trained on the source side of the corpus, and the other trained on the target side of the corpus. Then, for every sentence pair, we obtain a score by computing the likelihood given to the sentence pair (x, y) by the language models and combining them by using a function $f(.)$.

$$score(x, y) = f(p(x), p(y)) \tag{2.1}$$

The scores can then be used to select the appropriate sentences. This approach can not detect sentences whose source and target parts are valid sentences, but not a direct translation of each other. However, it is perfectly able to detect the rest of common data noise, such as sentences in a wrong language, foreign characters or nonsensical sentences, that can be detected independently of the other sentence in the pair. The main advantage of this approach is that is able to carry out an adequate amount of filtering, while being orders of magnitude faster than the translation model approach. The training of these language models is very fast, because it is carried out just by counting n-gram occurrences, and the time required to apply it is even lower, because it consists in looking up the probability of the sentence n-grams.

Byte Pair Encoding

Byte Pair Encoding (BPE) [39] is a technique whose goal is to allow translation with a fixed-vocabulary system in a way that mimics an open-vocabulary. This is done by transforming rare or unknown words into a sequence of known subword units. This technique works by first learning a number of merge operations (each merge operation produces one sub-word unit that will form part of the vocabulary), and then segmenting the words of the input sentences into sequences of subwords.

Merge operations are learnt by segmenting the text into individual characters, and appending a special end of word token </w> after every word. The algorithm performs one iteration per merge operation. On each iteration, we count how many times each pair of symbols appears. The most frequent pair is selected to become a merge operation, and every occurrence of the pair is replaced by the concatenation of both symbols. Once a pair has been merged back into the original word no further operations are applied to it.

Once the desired number of merge operations has been learned, we can now apply BPE to our corpora. The text is once again broken down into individual characters, and then we apply every merge operation in order. Words that have not been merged back into a full word are annotated with the suffix @@ in order to be able to restore the original segmentation. The final result is a text that has approximately the same number of unique tokens than merge operations applied, ensuring that almost no unknown words are given to the system. Figure 2.2 shows an example of BPE segmentation carried out in a sentences of the WMT test set. This technique has shown good performance improvements and its use has become part of the standard data preparation process of any NMT system.

Instead of che@@ wing out his ro@@ ok@@ ies in front of the rest of the team , he encouraged them .

Figure 2.2: Example of BPE segmentation of a text. Notice how rare words "chewing" and "rookies" have been split into subword units, avoiding a potential Out-of-vocabulary problem.

Both datasets were processed by applying 20000 BPE operations before training. This process is carried out over the joint source and target corpus as per the recommendation of [39], and we do not merge operations appearing less than 50 times over the entire corpus. The operation were undone at testing time in order to recover the final translation and compare it with the test sets.

Data collection

MT systems are trained using parallel corpora. A parallel corpus contains a number of sentences and their corresponding translation. That way we have a representation of the sentence both in the source and the target language. This is an example of supervised learning. Unlike monolingual text, that is available in places such as books, articles, and websites, parallel data is a much scarcer resource that can be difficult to obtain for certain language pairs.

It is therefore vital to obtain as much parallel data as possible in order to train a system that achieves good performance. Other important aspects are the data quality and the domain. One source of such parallel data is the Opus[1] project, that provides free access to a series of open source corpora. The Opus website hosts the data for dozens of language pairs, and is a good starting place for obtaining parallel data.

Although the main focus is in obtaining parallel data, monolingual data can also be used in MT systems. In traditional phrase-based MT systems, monolingual data can be used to train the language model component, because a language model is trained for a single language, and therefore we are not limited to training only with the parallel data. The addition of monolingual data different from the parallel corpus improves the performance of language models thanks to the bigger quantity of available sentences. Taking advantage of monolingual resources is also possible in NMT systems. In fact, the decoder of an NMT model can be understood as a language model (this is explained in detail in Section 3.1). A very successful use of monolingual data is the Backtranslation approach of [38], that consists in augmenting the training data with synthetic sentences. The synthetic data is created by producing a translation for each sentence in the monolingual target corpus, using a NMT system trained in the opposite direction. This approach allows us to obtain additional parallel sentences, whose target side is a sentence of a monolingual corpus, and the source side is its automatic translation. Although producing synthetic data is not cheap in terms of computing power, due to the need of training a backwards system and producing the translations, the use of synthetic data has been shown to significantly improve the performance of NMT systems, thanks to the improvement obtained in the language model by the use of more data.

We will now describe the corpora used in this work.

[1] http://opus.nlpl.eu/

Table 2.1: WMT 2017 corpus statistics.

		Average sentence length		# unique words	
	Number of sentences	German	English	German	English
train	5852.4k	23.0	24.2	1804.7k	878.5k
dev(newstest2015)	2.2k	20.7	21.9	9.9k	7.7k
test(newstest2017)	3.0k	20.7	22.0	12.6k	9.5k

WMT2017: German → English

The Workshop on Machine Translation (WMT [2]) is an annual international conference for researchers on the field of MT. Apart from the dissemination of recent developments in this field, the conference also organizes each year a series of competitions to test different aspects of MT. The *news* translation task, that evaluates the ability of systems to translate a series of online news articles, is the task that receives the highest amount of submissions each year and has become the reference task in order to compare and report results of new research.

The training data of the competition is published by the organizers each year as well as the corresponding test data. It is common to use the test data of a previous year as development data, and that has been the approach carried out in this work. The training data chosen is the one available in the 2017 competition, and the development data is the test data of 2015. The test data chosen was the one corresponding to the year 2017 in order to evaluate the results in the same conditions of the participants in the competition.

Table 2.1 shows basic statistics about the WMT corpus. We have computed the number of sentences, the average sentence length and the number of unique words for each language. The training data consists of almost 6 million parallel sentences used to train the model. The sentence length in German is shorter than its English counterpart, in part due to its use of many compound words. This phenomenon appears more clearly when we compare the number of unique words (vocabulary size). As shown on the table, the number of unique German words that appear in the text is more than twice the number of unique English words.

The number of training sentences is several orders of magnitude higher than that used in the dev and test sets, because having at our disposal big amounts of training data greatly benefits the learning of our models, but we can obtain a measure of system performance with a much lower number of sentences.

WMT2018 and the Paracrawl Corpus

The main difference introduced in the 2018 edition of WMT is the addition of the Paracrawl corpus. This corpus presents some differences and unique characteristics compared with the rest of the training corpora of WMT. Parallel corpus are usually produced by taking direct translations of documents, such as government records, that have been published in more than one language, and therefore contain an almost perfectly aligned document structure. The growing need for additional training data has motivated the search for new ways of obtaining parallel data. Paracrawl is a corpus produced by web crawling, searching the Internet for webpages that have versions in different languages, and trying to align both versions in order to obtain sentence pairs. This process is very error-prone, because there is no guarantee that the different document structures contain the same sentence structure or even the same content. As a result, Paracrawl is a corpus that in-

[2] http://www.statmt.org/wmt17

Table 2.2: Paracrawl corpus statistics.

Number of sentences	Average sentence length		# unique words	
	German	English	German	English
35808k	15.7	16.3	6581.0k	4610.3k

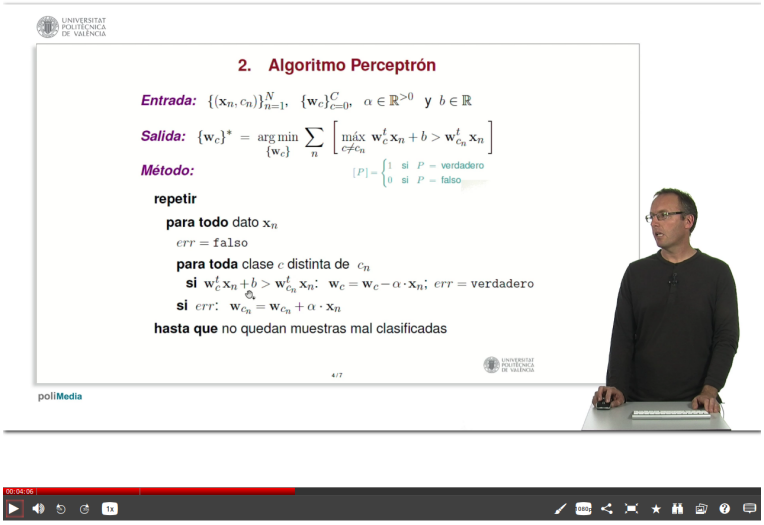


Figure 2.3: Screenshot of a poliMedia video.

cludes a considerable number of noisy sentences, but at the same time, due to its size, also contains many useful sentences that could improve model performance if selected. These characteristics make Paracrawl a prime target for using data filtering techniques in order to extract useful sentence pairs for training, while eliminating poor quality pairs that would otherwise harm model performance.

Table 2.2 shows some basic statistics about the Paracrawl corpus. This corpus contains almost 36M sentence pairs, a staggering amount of sentences. This means that this corpus provides a significant increase in bilingual data compared with the resources previously available for the WMT competitions. Whereas before we had only around 6M sentence pairs available, the Paracrawl corpus contains almost 6 times as many sentences. We can also see how the average sentence length is significantly lower than the one of the WMT17 corpus. The average sentence length for German sentences is 15.7 (it was 23.0 for WMT17), and 16.3 for the English sentences (24.2 for WMT17). This already tells us that there might be a significant difference in the type of sentences included in the two corpora.

poliMedia: Spanish → English

poliMedia[3] is the UPV's repository of educational videos that holds a variety of educational videos recorded by university staff. This platform is commonly used for hosting different kinds of supplemental material for many of the undergraduate and graduate courses taught at UPV. Figure 2.3 shows an example of one of the kind of educational resources that can be found in poliMedia. The *Clase Inversa*[4] project uses this platform for providing students with resources to learn outside the classroom. The MLLP research

[3]https://media.upv.es
[4]http://docenciainversa.blogs.upv.es/el-proyecto/proyecto-clase-inversa-upv-2/

Table 2.3: poliMedia corpus statistics.

		Average sentence length		# unique words	
	Number of sentences	Spanish	English	Spanish	English
train	145.8k	17.1	18.1	68.2k	46.9k
dev	1.4k	27.0	27.6	4.5k	3.7k
test	1.1k	28.2	28.2	4.1k	3.3k

group develops automatic transcription and translation systems with the goal of being able to produce educational videos that are automatically available in different languages than the one they were recorded on.

The repository contains resources in Spanish, Catalan and English, out of which we have chosen to train MT systems from Spanish into English. The corpus consists on a series of parallel sentences obtained from the video transcriptions of the aforementioned educational content hosted in poliMedia. Additionally, the data obtained from the video files is augmented with data collected from other educational resources such as TED talks.

Table 2.3 shows statistics for the poliMedia corpus. This corpus training data consists of around 145 thousand parallel sentences, while the dev and test sets contain a smaller quantity of sentences. In a similar way as shown for the WMT corpus, the average sentence length is lower in Spanish than in English, and the number of unique words is higher in Spanish, but these differences are not as high as in the case of WMT.

This task is specially important for this work because it provides a benchmark for testing the performance of different MT systems when translating educational resources.

CHAPTER 3
Attention-based RNN NMT models

This chapter describes a series of developed systems based on a RNN Attention-based model, and their implementation. We start with a general description of the components that form the basis of any modern NMT system. Once those concepts have been introduced, we describe the theoretical model behind the Attention-based model, how it relates with the previously described approach, and its corresponding architecture.

After that, we introduce the software tools used to implement the aforementioned model. Using those tools, we have trained a series of different systems based on the RNN-Attention approach. We make a detailed description of each of those systems configuration and describe the different architectural decisions, hyperparameters, techniques and methods used in the training process for those systems. The chapter ends with a detailed description of the evaluation of those systems with respect to their quality and efficiency measures.

Overview of NMT models

The basic principles behind neural networks have been introduced in Section 1.3.3. Those concepts are common to all use cases of neural networks, and serve as a starting point for building systems with different applications. This section introduces the specific considerations adopted for building NMT systems.

One of the first specific decisions that need to be made in MT, and in general most applications of Natural Language Processing, is how to consume the input data, which is made up of words, and obtain a numerical representation that can be used by the neural network, because these systems and associated training procedures work with numerical data. The solution adopted in NMT is to first codify the words as *one hot vectors*, a vector of a dimension equal to the number of words in the vocabulary, whose values are all 0 except for the dimension of that word. Because these vectors usually have a very high number of dimensions, the vectors are then multiplied by a word-embedding matrix, $\boldsymbol{E}$, that condenses this representation into a lower number of dimensions. A desirable property for this embedding is that they represent similar words (or words with similar meanings) with similar embeddings. A similar process is applied to the output of the last hidden layer of the network. In order to transform the output of the last hidden layer of the network into a vector of the same shape as the output vocabulary, the hidden representation is multiplied by an output embedding matrix $\boldsymbol{E}_o$. In contrast with other NLP applications, in NMT the embedding matrices are considered parameters of the

model that need to be learned, and are trained with the rest of the model using gradient descent.

Previous attempts at using neural networks for MT, such as [11], used the output of the NMT system as an additional feature for a phrase-based model, following the log-linear framework explained in Section 1.3.2. Those systems played only a small part in a model that combined many different features, most of them usually trained independently of each other. The first stand-alone NMT system that obtained competitive results was the RNN-Attention model [4], whose main contribution was the introduction of an attention mechanism.

On its most basic form, a NMT system has an architecture that contains an encoder neural network, a decoder neural network and, optionally, an attention mechanism, although the last component is mandatory if one wishes to obtain a competitive system.

The encoder component is a recurrent neural network, tasked with producing a representation of the input sentence, that will be then used by the rest of the system for obtaining the translations. This component is usually a recurrent neural network with hidden layers of LSTM[19] or GRU[11] units, and the representation produced by the encoder consists in the hidden state of the encoder neural network. Our hope is that, once trained, this encoder extracts a good representation $\mathbf{c}_i$ of the meaning of the input sentence, also called context vector. The context vector will be then used as the basis for producing the output translation.

The decoder recurrent neural network receives the encoded representation of the input sentences and emits the output words one at a time. This decoder is usually autoregressive, that means that the choice of which word is emitted depends not only on the encoded representation, but also on the previously word emitted by the decoder. This way, the decoder is in fact acting in a similar way to a language model. Both the encoder and the decoder are jointly trained.

Although here we have described the encoder and the decoder as RNN since they were used in the first approach to this problem, that is only one of the available options. The encoder-decoder architecture, or in its general form of sequence-to-sequence models, does not restrict us to use only one specific type of encoder/decoder component. In fact, as we will see later in Section 4.1, there are alternative proposals that use components that differ from a RNN. The encoder-decoder framework simply uses some sort of encoder component, that extracts a representation from the input sentence, and a decoder component that uses that representation in order to produce the output.

RNN-Attention

Having described the basic building blocks of a NMT system, we will now explain in detail the architecture of the RNN-Attention model [4]. The encoder is a Bidirectional RNN (BiRNN) formed by two individual Recurrent Neural Networks, one forward and one backward. The only architectural difference between these two networks is that the forward one reads the input sentence in order (from x_1 to x_J), and the backward network reads it starting from the end (from x_J to x_1). When reading input word x_j, each network emits a representation or encoding for that word. Therefore, for each word, we obtain a forward representation $\overleftarrow{\mathbf{h}_j}$ and a backward representation $\overrightarrow{\mathbf{h}_j}$. The two representations are then concatenated together to obtain the final representation for each word, so that the representation of a word contains information about both preceding and following words, in order to preserve as much meaning as possible.

$$\overrightarrow{\mathbf{h}_j} = f_{enc\_forward}(\overrightarrow{\mathbf{h}}_{j-1}, \mathbf{x}_j) \tag{3.1}$$

$$\overleftarrow{\mathbf{h}_j} = f_{enc\_backward}(\overleftarrow{\mathbf{h}}_{j+1}, \mathbf{x}_j) \tag{3.2}$$

$$\mathbf{h}_j = [\overrightarrow{\mathbf{h}_j}; \overleftarrow{\mathbf{h}_j}] \tag{3.3}$$

The type of function implemented by the encoder depends on the type of RNN unit selected to form the hidden layer, and the same holds true for the hidden layer of the decoder. As previously mentioned, usual choices are either LSTM[19] or GRU [11] units. This information is used by the decoder to output the translation. The decoder is also a RNN that maintains an internal state, $\mathbf{s}_i$, that is updated after each step by a function that depends on the previous state $\mathbf{s}_{i-1}$, the last word emitted y_{i-1} and a context vector $\mathbf{c}_i$ that will be explained later.

$$\mathbf{s}_i = f_{dec}(\mathbf{s}_{i-1}, \mathbf{y}_{i-1}, \mathbf{c}_i) \tag{3.4}$$

Attention Mechanism

Up to now, the components we have seen present a challenge in how to properly encode the information of the source sentence. Because the input to the a neural network must be a fixed-length vector, we are forced to encode all input sentences, no matter their length, using a vector with a fixed size. This also means that the input representation remains constant during the decoding process. The context vector was traditionally considered to be the final state of the encoder network, $\mathbf{h}_T$. It has been shown that the translation quality of these systems quickly degrades when having to produce translations for long sentences.

The introduction of an attention mechanism allows us to feed the decoder network with a potentially different context vector for each time step. This allows the decoder to receive a representation that is more suitable for choosing what is the next word to produce. For example, it would be beneficial to stop receiving information about parts of the input sentence whose meaning has already been translated and provide no further useful information. The attention mechanism allows us to do exactly that by producing the context vector, $\mathbf{c_i}$, fed to the decoder.

An attention function is described by [44] as a function whose arguments are a query, $\mathbf{q}$, and a set of key-value pairs, grouped into matrices $\boldsymbol{K}$ and $\boldsymbol{V}$, and whose output is a weighted sum of the values. The weights for each value are computed by a compatibility function between its key and the query.

$$c_i = \sum_j \alpha(j|i)\mathbf{V}_j \tag{3.5}$$

$$\alpha(j|i) = softmax(attention(\mathbf{q}, \boldsymbol{K}))_j \tag{3.6}$$

In NMT, we compute the context vector as the weighted sum of the different encoder representations at each time step. The generic attention mechanism described in Equations 3.5 and 3.6 can be specified to produce a context vector at a generic step i from a series of encoder representations $h_1, \ldots, h_J$ by using the decoder state as the query and the encoder representations as key-value pairs. Therefore, $\mathbf{q} = \mathbf{s_{i-1}}$, $\mathbf{K_j} = \mathbf{h_j}$, and $\mathbf{V_j} = \mathbf{h_j}$, leaving us with:

$$\mathbf{c}_i = \sum_j \alpha(j|i)\mathbf{h}_j \tag{3.7}$$

$$attention(\mathbf{s}_{i-1}, \mathbf{h}_j) = \mathbf{v}_a^T \, tanh(\boldsymbol{W}_a \mathbf{s}_{i-1} + \boldsymbol{U}_a \mathbf{h}_j) \tag{3.8}$$

$\mathbf{v}_a$, $\boldsymbol{W}_a$ and $\boldsymbol{U}_a$ are the different trainable weights of the attention mechanism.

This allows us to obtain a representation of the input sentence that assigns more weight to the appropriate parts for translating each part of the sentence. This vector is the weighted sum of the different encoder representations at each time step. In fact, these $\alpha(j|i)$ can be interpreted as acting as an alignment. At each time step i, $\alpha(j|i)$ can be understood as the probability that the target word at position i is aligned with the input word at position j.

The compatibility function of Equation 3.8 is known as additive or Bahdanau attention. Later works have proposed different scoring functions, such as the dot-product attention of [28], that computes the scoring function as:

$$attention(\mathbf{s}_{i-1}, \mathbf{h}_j) = \mathbf{s}_{i-1}^T \mathbf{h}_j \tag{3.9}$$

The advantage of the later approach is that the attention mechanism does not contain any parameter and works only by computing the dot product between the query and the key.

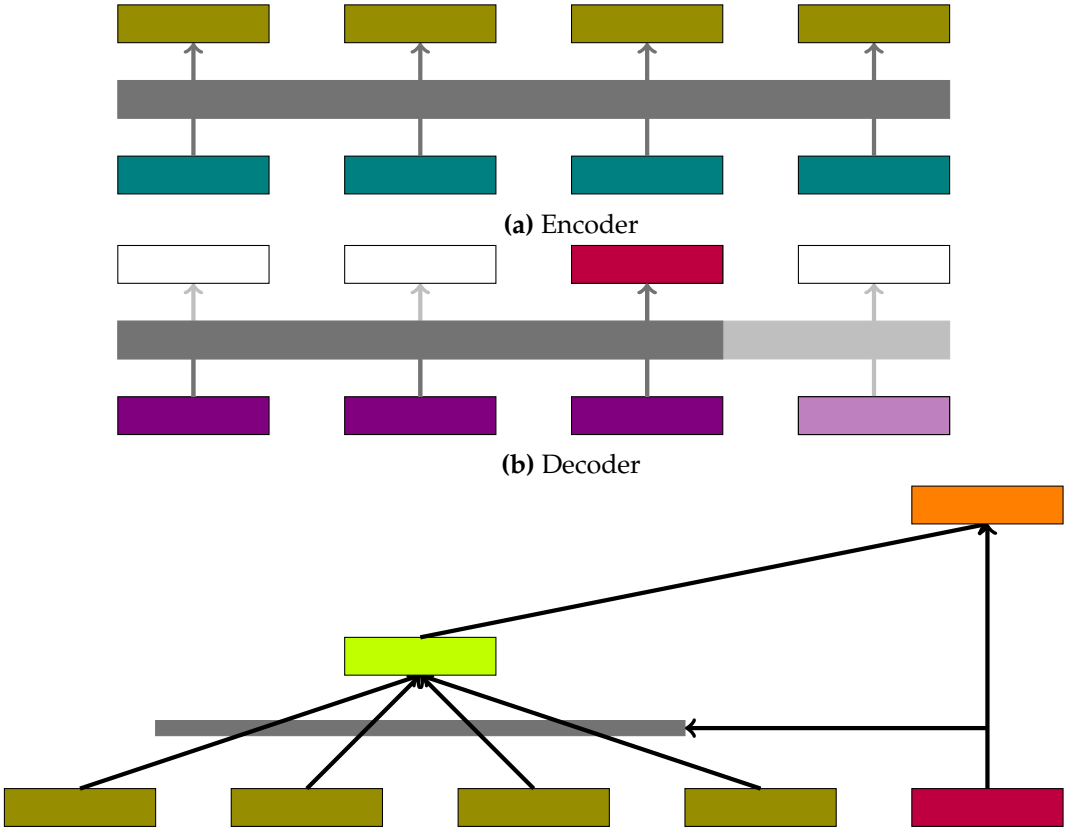
The output layer of the decoder component emits the probabilities for each possible word to be produced. First, a feedforward layer is applied, whose inputs are the context vector, the state of the decoder at the previous time step, and the last emitted word , $\mathbf{y}_{i-1}$, that is supplied to the decoder by applying an additional embedding matrix, $\boldsymbol{E}_{decoder}$, sometimes called target embedding. Then, a softmax function is applied in order to obtain the probability distribution over the target words.

$$p(y_i|\mathbf{s}_i, yi-1, \mathbf{c}_i) = softmax(f_{emiss}(\mathbf{s}_{i-1}, \mathbf{y}_{i-1}, \mathbf{c}_i)) \tag{3.10}$$

The use of RNN components as the elements of the encoder and decoder component, with the addition of the attention mechanism for computing the appropriate context vector, are the basic implementation of the encoder-decoder framework. These techniques, in combination with the processing steps such as BPE explained in Section 2.1, form the basic NMT system. Figure 3.1 illustrates the principles behind the encoder-decoder architecture.

Computer tools

This section describes the software used to train the MT systems presented on this work, Tensorflow-nmt, based in the Tensorflow [1] library, and Sockeye, based on the MXNet [10] library. Both libraries use the concept of computational graphs and GPU acceleration, introduced in Section 1.3.4, to enable fast training of various machine learning models. These general purpose libraries offer the tools to develop any kind of machine learning models, and are nowadays widely used in both research and industry applications. Tensorflow-nmt and Sockeye build on top of their respective libraries, and serve as a more specialized tool that is specifically used for developing NMT systems, due to implementing various theoretical models and improving the process of using them by avoiding the need to implement them from scratch.



(a) Encoder

(b) Decoder

(c) Alignment (Attention)

Figure 3.1: Encoder/decoder architecture.

Tensorflow-nmt

Tensorflow-nmt [27] is a software package together with an in-depth guide for building Neural Machine Translation systems. This toolkit is written on top of Tensorflow and its source code is available on Github. The toolkit allows the training of models that follow the Attention-RNN architecture already introduced in [4], and also has support for models that follow Google's Neural Machine Translation [45] (GNMT) attention architecture.

The toolkit has one entrypoint script *nmt.py* that allows the training of a model or the translation from an already trained model. Trained models are stored on disk using Tensoflow's own checkpoint file format, allowing permanent storage of models, and many different training and inference metrics are logged and can be visualized using Tensorboard, TensorFlow's Visualization Toolkit. This tool also allows the visualization of the computational graph that defines the model, a great help for better understanding how the model works internally and detecting any possible mistake.

The model configuration and training hyperparameters can be specified in *json* files instead of manually passing them through the terminal. The software already come with a set of preset scripts used by the developers that serve as a starting point for training models.

Sockeye

Sockeye [17] is another open-source toolkit for NMT. It is based on the MXNet library, therefore also allowing GPU acceleration. The toolkit can be installed from source or a pip package, and includes a series of scripts for training and translating, as well as a series

of helper scripts for different pre-processing tasks or model combination tasks, such as a tool for checkpoint averaging, a technique that will be described later.

The toolkit implements Attention-RNN, Transformer [44] and Fully Convolutional (ConvSeq2Seq) [13] models. The two last architectures are recent developments in the field of NMT that aim to improve model training time and performance by eliminating the explicit dependencies introduced by the recurrent model.

Apart from the aforementioned architectures, Sockeye also offers a series of different training and inference techniques implemented on top of those models, including the very important feature of model ensembling, which consists in using more than one model at inference time and combining those translations in order to obtain a final one that is better than the individual ones. Sockeye even allows the combination of models that use different architectures.

System description

Having explained all the necessary pre-requisites, we are now ready to describe the different developed systems based on the RNN-Attention model.

The general policy adopted for training all systems is that, after a fixed number of training steps, the model parameters were stored in a checkpoint and the performance was checked against the development set.

This performance check is carried out by measuring the perplexity of the model computed over the sentences of the validation set, previously explained in Section 1.3.1. Once the perplexity stopped improving on the development set, the training was stopped and the model was evaluated with respect to the test set.

We choose as a starting point a RNN-Attention model with 1 bidirectional layer both in the encoder and decoder. The size of this hidden layers and the embedding size is a variable that will be changed in each experiment in order to find out their optimum values. The type of unit selected for the hidden layers was the LSTM unit. The Adam optimizer [23], which is a modified version of Gradient Descent, and the dropout [42] technique were used for training the model. There are a series of training choices and techniques that affect model performance. We will now describe the different changes that are going to be tested:

- **Model Ensembling & checkpoint averaging.** In Machine Learning, ensemble methods aim to improve system performance by combining a series of individual models in order to construct an overall better system by combining the predictions of the individual models.

  The combination of multiple models improves the system results if the errors produced by the models are not strongly correlated between them, smoothing the mistakes produced by one model thanks to the predictions from the rest. While this technique has the capacity to improve performance, it also has some drawbacks. The main one is that, due to having to combine multiple system, the training time required to obtain the system increases linearly with the number of systems. This can be offset if we have multiple GPUs at our disposal in order to enable training multiple systems in parallel. That still leaves the problem of needing to run all the models at inference time.

  Checkpoint averaging is one technique that deals with both problems while still retaining some of the benefits of constructing an ensemble. This is done by selecting some of the checkpoints saved at different times that contain the parameters of

one model, and treating the parameters contained on each of those checkpoints has an as independent model. The checkpoints are then combined in order to obtain the ensemble. This technique, used by the AMU-UEDIN team at WMT16 [22], simulates the construction of the ensemble by selecting the desired checkpoints and computing the average of the parameters they contain. This way, some of the benefits of constructing a checkpoint ensemble are obtained and at the same time we only need to run the averaged model to obtain translations, saving both on time and memory. The simulated ensemble is obtained as:

$$\boldsymbol{\theta}_{ensemble} = \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{\theta}_m \tag{3.11}$$

Tensorflow-nmt does not implement a tool to carry out this averaging, so a tool was developed and integrated into the code of this toolkit in order to support this desired feature. For these experiments, the averaged model was constructed from the 4 checkpoints with the highest validation score, although the developed tool allows the combination of any arbitrary number of checkpoints. In contrast with Tensorflow-nmt, Sockeye does offer a command line tool called *sockeye-average* that allows the user to apply checkpoint averaging without the need to write additional code. As outlined in [3], the big improvements achieved by checkpoint averaging can be an indicator of a suboptimal training schedule. One of the possible solutions to that problem is to reduce the learning rate as the training progresses.

- **Sequence length.** Another change tested against the base model was to increase the maximum length of sentences fed to the system. By default, sentences longer than a certain number of tokens are discarded, in order to speed-up training and filter long sentences that might not make sense or are very hard to learn for the model. Increasing this threshold allows for greater use of the training data. We refer to this modification of the model as *long-sequences*.

- **Learning rate.** The choice of learning rate affects the size of the update applied to the model's parameters at each training step, and has a significant effect in model convergence. Sockeye allows the definition of different learning schedules. One of them, *plateau-reduce*, is of special interest since it reduces the learning rate if the validation score does not increase for a certain number of checkpoints. This enables the model to slow down learning once we have arrived near a good point of the parameter space. This reduction means that the model is updated more slowly, and therefore it is less susceptible to making a bad update and diverging from a good area of the parameter space. This reduction also means that the model is more guaranteed to converge once the learning rate has been reduced many times, because the gradient descent updates will have almost no effect on the parameters of the model. We denote the use of this technique with the label *lr_reduce*.

Results

In this section we present the experimental results achieved in the poliMedia and WMT corpora.

Table 3.1: Results obtained for models trained with the poliMedia corpus.

		Tensorflow test		Sockeye test	
Model RNN-Attention	Parameters	BLEU	TER	BLEU	TER
$d_h = 256$, $d_e = 256$	5.4M	20.1	64.7	20.5	61.4
$d_h = 512$, $d_e = 256$	10.3M	20.3	62.2	20.5	61.2
$d_h = 512$, $d_e = 512$	14.0M	20.4	60.8	20.3	62.5
$d_h = 512$, $d_e = 1024$	21.4M	20.9	60.8	21.0	62.0
$d_h = 1024$, $d_e = 1024$	40.6M	19.4	62.3	20.1	63.7

poliMedia

As previously mentioned, we use a 1-layer bidirectional RNN in both the encoder and the decoder. We begin with the poliMedia corpus due to the differences in the number training samples available, which is much lower than WMT. Thanks to the reduced training data size, it is feasible to try different hyperparameter combinations and to test their performance. The models were trained during 50000 steps, and the development performance was evaluated every 1000 steps. After training stopped, the checkpoint with the best development performance was selected to carry out inference.

We are mainly interested in finding out adequate values for the model's hidden layer dimension and the embedding dimension. Conceptually, a higher model dimension means higher model capacity, and translates into a better training performance. However, our goal is to obtain a model that is able to generalize, something that could be compromised if the model has too much learning capacity and overfits to the training data. We have trained 5 different models by varying the hidden dimension size (1024,512,256) and the embedding dimension (1024,512,256). In a similar way to the checkpoint averaging case, Tensorflow-nmt does not allow to define the embedding dimension independently from the hidden layer dimension, so it was necessary to modify the framework in order to be able to carry out these experiments.

Table 3.1 shows the results of the systems trained with poliMedia. We will start by describing the results achieved with Tensorflow-nmt, available on the second column. The results are very interesting, since they show a clear pattern in the way the system's performance is affected by the changes to the embedding and hidden dimensions. The most simple RNN-Attention with hidden dimension=256 and embedding dimension=256 achieves a result of 20.1 BLEU and 64.7 TER when evaluated with respect to the test data. Then, if we start to increase these dimensions, we begin to notice performance improvements with respect to the baseline model. Increasing the hidden dimension to 512 results in a model that achieves 20.3 BLEU and 62.2 TER, while increasing both the hidden and the embedding dimesion to 512 result in a system with 20.4 BLEU and 60.8. The difference of this system with the baseline one is small in terms of BLEU, 0.3 points of improvement, but this is much more noticeable when looking at the TER metric, that improves in this case by 3.9 points. The best result is obtained with a model that uses hidden dimension=512 and embedding dimension=1024, achieving a result of 20.9 BLEU and 60.8 TER. If we further increase the hidden dimension to 1024, we find that model performance decreases instead of improving, with a BLEU of 19.4 and 62.TER, which represents a decrease of 0.5 BLEU and an increase of 1.5 TER with respect to the best model. This result shows that this model is likely to be overfitting to the training data due to being too complex to be learnt with the amount of data available. This means that the model has started to simply memorize training data, and as a result, has lost some generalization capabilities.

We can conclude that the second-to-last model we tried is in an optimal point with respect to the choice of hidden and embedding dimension. If we increase those dimensions, the model has too much learning capacity and it starts overfitting to the training data, whereas if we decrease them, the model has less learning capacity, and this reduced learning capacity results in worse generalization capacity, due to not having estimated parameter values properly.

The fourth column of Table 3.1 shows the equivalent systems trained with Sockeye. The models trained with Sockeye obtain somewhat similar results to the equivalent models trained with Tensorflow-nmt, and we can notice the same effects of model complexity that we saw when changing the model dimensions of the Tensorflow-nmt models. Model performance increases as we increase the number of parameters of the model, until we arrive to a point where performance degrades if we continue increasing the number of parameters. The peak in performance is reached by using hidden dimension 512 and embedding dimesion 1024. The Sockeye model obtains 21.0 BLEU and 62.0 TER, while the equivalent Tensorflow-nmt implementation obtains 20.9 BLEU and 60.8 TER. We believe these differences can be explained by implementation details and the random nature of the training process that is present in every training run.

Now that we have described the results in terms of quality, we will then look at some of the efficiency results of the training. First and foremost, we are interested in measuring the time it takes to train these systems, since it will be one of the most limiting factors if we wish to try different configurations. Because this training time can be affected by many variables, measuring the amount of tokens processed each second during training might provide a better estimate of model efficiency. We also need to keep in mind the amount of VRAM used by the models, and we might need to reduce the batch size or the model size if our available GPUs do not offer enough memory. Tables 3.2 shows the different resource consumptions of the Tensorflow-nmt and Sockeye models trained with the poliMedia corpus. The base Tensorflow-nmt model with hidden and embedding dimension 256 takes around 3h to train and uses 0.7GB of GPU memory. If we start increasing these dimensions, we can see how a higher number of model parameters is reflected in the amount of memory used by the model, since a lower number of parameters means than fewer activations and gradients need to be computed at each training step. This also affects the time it takes for the models to train. A lower hidden and embedding dimension translates into a lower amount of computation that needs to be carried out at each time step, so we can process a higher number of training samples per second. Usually, smaller models can also be trained faster than large models with more parameters even if the computation time is similar, because the latter need to optimize much more complex systems. With respect to the efficiency measures of the Sockeye RNN-Attention models, shown on the second part of the table, we can see how the Sockeye model trains much faster than the same model implemented in Tensorflow-nmt. In fact for the first model configuration we tried, Sockeye is able to process almost 4 times as many tokens per second than Tensorflow-nmt, altough this difference is reduced when we look at the bigger models of the last two rows of the table. Even in that case, Sockeye is more than twice as fast than Tensorflow-nmt. In terms of memory consumption, we can see how Sockeye does use more memory than the equivalent Tensorflow model. As expected, memory consumption grows as the number of parameters of the model increases. The models described in this and the following chapter have all been trained under the same conditions using a GTX 1080 GPU in order to be able to properly compare their training time.

Table 3.2: Resource consumption for systems trained with the poliMedia corpus.

		Tensorflow-nmt			Sockeye		
RNN-Attention Model	Parameters	RAM	Training time	Tokens/s	RAM	Training time	Tokens/s
$d_h = 256, d_e = 256$	5.4M	0.7 GB	3 h	±6.3K	2.0 GB	1 h	±20.6K
$d_h = 512, d_e = 256$	10.3M	1.3 GB	4 h	±6.5K	2.5 GB	1 h	±16.3K
$d_h = 512, d_e = 512$	14.0M	1.3 GB	4 h	±6.5K	2.6 GB	1 h	±13.8K
$d_h = 512, d_e = 1024$	21.4M	1.5 GB	6 h	±4.1K	2.8 GB	1 h	±11.1K
$d_h = 1024, d_e = 1024$	40.6M	1.8 GB	7 h	±3.4K	3.8 GB	2 h	±7K

Table 3.3: Results obtained for models trained with the WMT corpus.

	Tensorflow-nmt newstest2017		Sockeye newstest2017	
RNN-Attention Model	BLEU	TER	BLEU	TER
Baseline	22.2	66.3	21.8	68.5
+ checkpoint averaging	23.9	63.5	24.5	64.3
+ long-sequences	25.1	61.2	25.3	63.0
+ checkpoint averaging	26.6	59.9	27.0	60.8
+ lr_reduce	-	-	27.5	60.1
+ checkpoint averaging	-	-	27.7	59.9

WMT

We have chosen as a baseline model a RNN-Attention with hidden dimension=1024 and embedding dimension=1024 thanks to the information gained carrying out experiments with the poliMedia corpus. Table 3.3 shows the TER and BLEU scores of the different model configurations. All Tensorflow-nmt models were trained for a minimum of 300k steps and then stopped once the validation score did not improve for 10 checkpoints. One checkpoint was produced every 5000 steps. Similarly, Sockeye models finished training once their valuation score did not improve for 10 consecutive checkpoints. The Tensoflow-nmt baseline model obtains a BLEU score of 22.2 when evaluated with regards to the test set. The creation of a simulated ensemble with the checkpoint averaging technique yields an increase of 1.7 BLEU. The application of the conceptually simple checkpoint averaging has already considerably boosted performance by leveraging the various checkpoints stored while training the model, without the need of training any additional models.

The long-sequences model consists in increasing the sentence maximum length threshold. Previously this threshold was 50, so sentences longer than 50 tokens were discarded. In this experiment, we have increased this amount to 75 tokens. In order to compensate this increase in the number of effective training sentences, we bumped-up the minibatch size from 32 to 50, in order to process a bigger number of sentences per step. These major changes were accompanied by small hyper-parameter changes to match the configuration used by RWTH Aachen at WMT16 [31]. The long-sequences model achieves a higher performance over the baseline model and its averaged version by making a better use of the training data available in the corpus. By not eliminating the longer sequences, we are providing the model with more data that it can use in order to learn to model long-term dependencies, and this can then be used to obtain better translations at inference time. This results in an increase of 2.9 BLEU over the baseline model, and an increase of 1.2 BLEU over its averaged version.

The best result obtained with Tensorflow-nmt consists in producing a checkpoint ensemble out of the long-sequences model, obtaining a BLEU of 26.6 in newstest2017, an improvement of 1.5 over the single model. When compared with the baseline model, this represents an improvement of 4.4 BLEU, a massive performance increase. This achievement was due to a better utilization of both the available training data as well as the whole information produced during the training process, because checkpoint averaging takes advantage of not only the best model parameters of the last checkpoint, but also the parameter vector previously stored in checkpoints that would normally be simply discarded.

The beneficial effect of this improvements with regards to the BLEU score can also be observed on the TER metric. The application of each of them is accompanied by a further decrease in TER, with the averaged long-sequences model obtaining a TER of 59.9 on newstest2017. Compared with the initial model, that obtained a TER of 66.3, we have been able to decrease this metric by 6.4 points.

The third column of Table 3.2 shows the evaluation results obtained by the systems trained with Sockeye. The results show that the different techniques perform in a similar way than when applied to the Tensoflow-nmt models, just as they should be. The baseline model achieves 21.8 BLEU and 68.5 TER, and this result is improved by 2.7 BLEU and 4.2 TER when we apply checkpoint averaging. If we then apply the long-sequences scheme, this new model obtains 25.3 BLEU and 63.0 TER on newstest2017. Applying checkpoint averaging on top of this results in a simulated ensemble that obtains an improvement of 1.7 BLEU on the test set.

Aditionally we have tried a learning rate reduce scheme (lr_reduce), available only in Sockeye. This lr_reduce model applies the plateau-reduce learning rate schedule, with an initial learning rate of 0.001. This value is halved every time model performance does not improve for 3 consecutive checkpoints. When using this lr_reduce schedule, we obtain a performance improvement of 2.2 BLEU on the test set with respect to not using it. This means that we have once again obtained a significant improvement by applying additional changes to the model. The result also confirms the importance of reducing the learning rate once we have arrived at a good area of the parameter space, allowing the optimization procedure to behave better during training, that translates to the observed increase in the test scores both in BLEU and TER.

The application of checkpoint averaging to the lr_reduce model does not show the same improvements that were obtained when applied previously. This technique has improved 0.1 - 0.2 BLEU with respect to the model without averaging, far less than the 1.7 improvement when applied to the baseline model. This result that shows that checkpoint averaging does not translate into a big quality gain when applied to models whose training has been more carefully carried out, is consistent with the hypothesis laid out in [3], that states that the improvements obtained by checkpoint averaging are mostly due to the unstable training regime, and this technique loses its effectiveness if the defects of the training schedule are fixed.

We will now look at resource consumption for the models trained using the WMT corpus. Table 3.4 shows the resources consumed by the models trained using Tensorflow-nmt and Sockeye. At first glance, one can see how the long-sequences model consumes a much higher amount of VRAM, 7.4GB, getting close to the maximum amount of memory available on a GTX 1080, which is 8GB. This is due to 2 facts, the use of longer sequences requires the creation of a bigger computational graph to propagate the gradient, and the fact that this model uses a higher minibatch size (50 when compared with size 32 of the base model) that means a higher number of sentences is being processed in parallel inside the GPU.

Table 3.4: Resource consumption for WMT models.

	Tensorflow-nmt			Sockeye		
RNN-Attention Model	RAM	Training time	Tokens/s	RAM	Training time	Tokens/s
Baseline	4.2 GB	57 h	±3.40K	3.9 GB	11 h	±5.5K
+ long-sequences	7.4 GB	65 h	±4.68K	5.2 GB	16 h	±7K
+ lr_reduce	-	-	-	5 GB	30 h	±7K

The bigger minibatch size allows the long-sequences model to process a significantly higher number of sentences per second, even if they are of a longer length. Although it has a higher throughput, the model does take longer to train simply because it must process a bigger quantity of sentences due to the fact increasing the length threshold means that we are discarding a smaller portion of the corpus. This is reflected in the fact that the long-sequences model takes 8 hours longer to train than the baseline model. The long-sequences model, when trained with Sockeye, finishes training in only 16h, in contrast with the 65h that it took for the same model to train with Tensorflow-nmt. This can be explained by the difference in the amount of tokens processed per second. Where the Tensorflow-nmt model processed 4.68K tokens, the same model implemented in Sockeye is able to process 7K tokens per second. We can see how the lr_reduce schedule has allowed the model to continue training for a much longer amount of time. In fact, the base model trains for 16 hours, while the reduce schedule allows it to train for 30h, almost double. The longer training time translates into the higher scores reported in Table 3.3.

Framework choice moving forward

Looking at the results from previous experiments, we can see significant differences between Tensorflow-nmt and Sockeye, the two frameworks used for training the NMT models. The Sockeye models all train significantly faster than their Tensorflow-nmt counterparts, due to a much higher throughput achieved by Sockeye. For example, comparing the results of the long-sequences RNN-Attention model trained in the WMT corpus available in Table 3.4, the Tensorflow-nmt model is able to precess around 4.68K tokens per second, while the same model implemented in Sockeye is able to process 7K tokens per second, which almost doubles the amount of tokens processed per second. Sockeye has consistently achieved higher training speeds on every model tried, while being able to reproduce the results of the Tensorflow-nmt models. Additionally, Sockeye as a framework offers more options and features than Tensorflow-nmt, such as the checkpoint averaging technique or the option to specify the embedding dimension. This, and the fact that Sockeye implements architectures other than RNN-Attention, as we will see in the following section, have made us choose this framework over Tensorflow-nmt, and therefore all further experiments will be carried out using the Sockeye framework.

CHAPTER 4
Transformer Self-Attention NMT models

This chapter describes a series of systems trained following the Transformer architecture. We describe this new model that improves the previously best results obtained with RNN-Attention and then move onto carrying out experiments with this architecture. These models were trained by leveraging the experience gained training the previous models. The Transformer architecture is implemented in the Sockeye toolkit, and we will use it to train the different models presented in this chapter.

Transformer

The Transformer [44] architecture is a recently introduced architecture that replaces recurrent layers by a new type of layer, Self-Attention layers, as well as a series of architectural changes in both the encoder and decoder components. This model achieves significant improvements in both speed and quality of translations, and is currently considered the state of the art.

This section will give a general overview of the model and the proposed improvements. Due to the model's complexity and the wide variety of introduced changes, readers should refer to the original article to learn about details of the implementation.

Both the encoder and the decoder are composed of a series of layer blocks stacked on top of each other. Each of these blocks is made up of a series of sub-layers. A sub-layer implements a function, such as a neural network hidden layer, jointly with Residual Connections [16] and Layer Normalization [2]. We will now describe each of those techniques, starting with the main contribution of the Transformer model, the introduction of a new way of computing attention.

Generalization of the Attention mechanism

The Attention layers of the Transformer architecture perform multiple Scaled Dot-Product Attention by using Multi-Head Attention. We have previously introduced Dot-Product Attention in Equation 3.9. Scaled attention introduces a scaling term that depends on the dimensions of the key, d_k. Remember that in traditional attention we use $\mathbf{q} = \mathbf{s_{i-1}}$, $\mathbf{K_j} = \mathbf{h_j}$, and $\mathbf{V_j} = \mathbf{h_j}$.

$$attention(\mathbf{q}, \mathbf{K}_i) = \frac{\mathbf{q}^T \mathbf{K}_i}{\sqrt{d_k}} \tag{4.1}$$

If we wish to compute attention with multiple queries, the queries can be packed in a matrix Q, and the computation may be expressed solely on terms of matrix multiplication. This means that the entire Attention process can be computed with the following function:

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{4.2}$$

Up to now, the Attention mechanism has provided us with an answer for each query. Multi-Head attention extends the previous mechanism in order to produce an answer that is the combination of multiple key-query comparisons. Multi-head attention consists in performing several attention operations in parallel and combining the results to obtain the final context vector. Each individual attention operation or head is carried out by applying a linear projection to the query, keys and values, computing attention between them, and then projecting back into a common space.

$$MultiHead(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Concat(head_1, \ldots, head_h)\boldsymbol{W}^o \tag{4.3}$$

$$head_i = Attention(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V) \tag{4.4}$$

The projections are applied by the means of matrices $\boldsymbol{W}^o$,$\boldsymbol{W}_i^Q$,$\boldsymbol{W}_i^K$ and $\boldsymbol{W}_i^V$. These projection matrices are parameters learned during training. An additional difference of this layers with respect to traditional Attention is the way they are applied in the Transformer model, something that we will explain once we describe the architecture.

Layer Normalization

Layer Normalization [2] normalizes the outputs of a neural network layer, so that each neuron behaves as a normal distribution. To normalize a layer, we compute a mean and standard deviation from the activations of every unit in that layer, and then we subtract the mean and divide by the standard deviation. Layer normalization is applied to the output of the layer operations, $\mathbf{a}^{(\mathbf{i})}$, before the activation function $g(.)$ is applied. Thus, we can apply Layer Normalization to layer i as follows:

$$LayerNorm(\mathbf{a}^{(\mathbf{i})}) = \frac{\boldsymbol{\gamma}}{\sigma^{(i)}} \odot (\mathbf{a}^{(\mathbf{i})} - \mu^{(i)}) + \boldsymbol{\beta} \tag{4.5}$$

$$\mu^{(i)} = \frac{1}{H}\sum_{h=1}^{H} a_h^{(i)} \tag{4.6}$$

$$\sigma^{(i)} = \sqrt{\frac{1}{H}\sum_{h=1}^{H}(a_h^{(i)} - \mu^{(i)})^2} \tag{4.7}$$

$\mu^{(i)}$ and $\sigma^{(i)}$ are broadcasted to have the same dimension as $\mathbf{a}^{(\mathbf{i})}$. $\boldsymbol{\gamma}$ is the *gain* and $\boldsymbol{\beta}$ is the *bias*, parameters that are learned during training. They are used so that the layer can output normal distributions that are different from the standard normal distribution.

Residual Connections

Residual Connections [16], sometimes known as skip connections, is the name of a technique where the input of a layer *skips* one or more transformations, and is then added to the output of those transformations. Suppose we have a vector $\mathbf{x}$ and a series of neural

network layers that compute the transformation $\boldsymbol{F}(\mathbf{x})$. The output of that residual block would be computed as $\mathbf{x} + \boldsymbol{F}(\mathbf{x})$. The goal of this technique is to facilitate the optimization process for networks with many layers.

Modeling word position

Attentive readers will have noticed that we have yet to describe a mechanism by means of which we are able to incorporate positional information about word order. Self-attention layers, by themselves, offer no way of knowing word order in a sentence, since they treat all inputs the same way. One simple approach to this problem is to encode the position of the word by means of a one-hot vector p_i, the same way we do with the different words that form the vocabulary, w_i. Given an arbitrary embedding matrix $\boldsymbol{E}$, we obtain the representation by simply concatenating the vectors and multiplying by the embedding matrix.

$$e_n = [w_n; p_n]\boldsymbol{E} \tag{4.8}$$

That formulation is equivalent to having 2 different embedding matrices, one for the word embedding ($\boldsymbol{E}_w$), and one for the positional embedding ($\boldsymbol{E}_i$).

$$e_n = w_n\boldsymbol{E}_n + p_n\boldsymbol{E}_p = WE^{(n)} + PE^{(n)} \tag{4.9}$$

$WE^{(n)}$ represents the word embedding information for word n, and $PE^{(n)}$ represents the positional embedding information for that word. We have not explained how to obtain those positional encodings yet. The straightforward approach is to treat them like the word embeddings and let the network learn them during training. Another option is to use a predefined function that produces these positional embeddings.

In the case of the Transformer model, the positional embeddings are given by two sine and cosine functions, calculated differently for every position *pos* of the sentence, $\forall pos \in [1, N]$. These functions compute a positional embedding vector for each word whose entries are calculated as follows:

$$PE_{2i} = \sin\left(pos\frac{1}{10000^{2i/d_{model}}}\right) \forall i \in [0, d_{model}/2 - 1] \tag{4.10}$$

$$PE_{2i+1} = \cos\left(pos\frac{1}{10000^{2i/d_{model}}}\right) \forall i \in [0, d_{model}/2 - 1] \tag{4.11}$$

Experiments carried out in the original paper show that these fixed positional embeddings do not incur in any performance decrease, and they offer slightly more generalization capacity that learned embeddings if we have to translate sentences that are longer that the ones appearing in the training data.

Model Architecture

In the Transformer architecture, both techniques are combined, and the output of each sub-layer is computed as $LayerNorm(x + Sublayer(x))$. There are two types of sub-layers in the Transformer architecture, Feed Forward layers and the previously described Multi-Head Attention layers. Feed Forward layers are standard neural network layers consisting on weight matrix multiplication followed by an activation function.

Up to now, the described techniques and layers can be applied to any NMT model, but the change in the Transformer architecture is how these layers are applied. The standard Transformer block consists in a Multi-Head attention sub-layer followed by a Feed

Forward sub-layer. The base model configuration includes 6 of these blocks in both encoder and decoder. Instead of feeding one input at a time like a RNN, the entire input sentence is fed to the encoder, and the input sentence representation is computed all at the same time. The self attention layers in the encoder apply Multi-Head attention to the output of the previous layer, using that output as both query and key-value pairs. Therefore, at each layer, the encoder produces a representation for each word, that can incorporate information about any other word in the sentence thanks to the self-attention mechanism. The entire representation can therefore be produced in a single pass.

Additionally, the decoder blocks include a Multi-Head attention sub-layer that attends to the output of the encoder stack, allowing the decoder to access the input sentence representations. In the same way as all other NMT models, the decoder produces one word at a time, conditioned by the previously emitted words. The decoder is fed by the previously emitted words and its Multi-Head attentions sub-layers perform their computations over the output of the previous decoder layer. Decoder blocks contain an additional attention layer that attends to the encoder output. In those sub-layers, the output of the previous decoder layer acts as query, while the encoder output acts as key-value pair. One important change is to modify the decoder self-attention layers so that they can only attend to already emitted words. A graphical overview of this architecture is shown in Figure 4.1. The Transformer Encoder layer is shown in Figure 4.2, and the Encoder layer is shown in Figure 4.3.

Up to now, we have described the changes introduced by the Transformer model to the encoder and decoder components. However, the Transformer paper also introduces some additional considerations used for training the model that are not related to the architecture of the model itself and can be independently considered.

Weight tying

Weight-tying [32] is a proposed technique for improving the performance of neural network language models through manipulation of the different embedding layers, while also providing a significant reduction in number of parameters of the model. As previously explained in Section 3.1, NMT models make use of embedding matrices in order to convert the input text into a numeric format usable for the network. In NMT encoder-decoder systems, we have 3 of such matrices. First we have an encoder embedding matrix, $\boldsymbol{E}_e$, that when applied to the one-hot vectors of the source sentence, produces the representation to be used by the rest of the encoder. This is what has traditionally been considered as a proper embedding matrix, and that means that each of the entries in the matrix can be considered as an embedded representation of its corresponding input word. There exist various works that have evaluated the possibility of substituting this embedding matrix learned during training by some other different word-embedding method, usually one that has been previously computed by a different special purpose model tasked with finding these word embeddings. For example, [33] studies the effects of using different types of pre-trained embeddings.

However, that is not the only embedding matrix used in NMT. Since the decoder has a dependency on y_{t-1}, we also use a decoder embedding matrix, $\boldsymbol{E}_d$. Additionally, the output of the last hidden layer of the decoder must be projected into a space that has the same dimension as the output vocabulary, so that we obtain a probability distribution over all possible words once we apply the softmax function. This is achieved by multiplying by an output embedding matrix, $\boldsymbol{E}_o$. Putting all of that together, we have 3

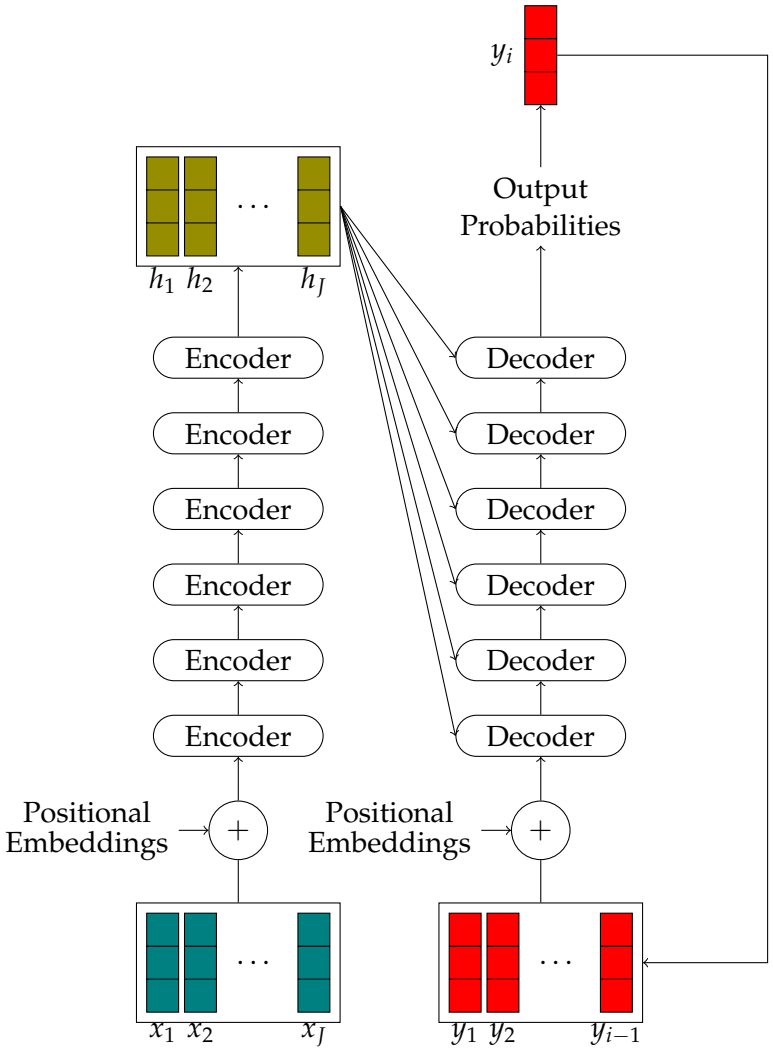


Figure 4.1: Figure of the Transformer architecture.

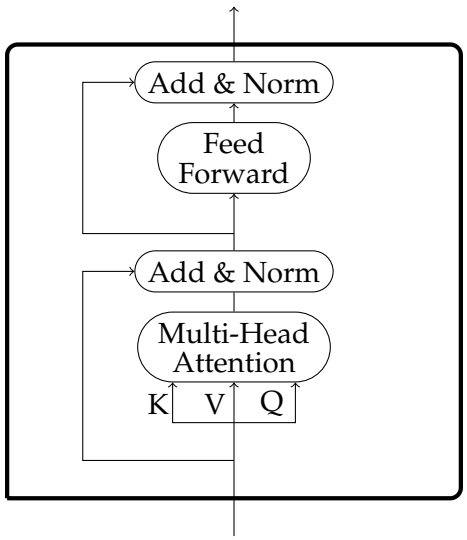


Figure 4.2: Transformer Encoder Block.

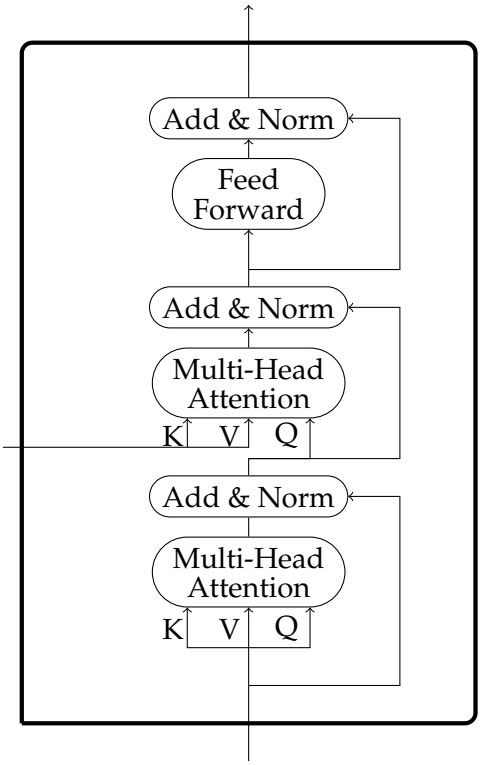


Figure 4.3: Transformer Decoder Block.

embedding matrices that characterize our models. $\boldsymbol{E}_e$, $\boldsymbol{E}_d$, and $\boldsymbol{E}_o$ are the encoder, decoder and output embedding matrices, respectively.[1]

$\boldsymbol{E}_e$ has dimension (Embedding dimension, Input vocabulary size), $\boldsymbol{E}_{dr}$ has dimension (Embedding dimension, Output vocabulary size) and $\boldsymbol{E}_o$ has dimension (Output vocabulary size, Hidden layer size). The authors of the weight tying paper realized that, if we were to have the same input and output vocabulary, that is, the same dimension, the encoder/decoder embeddings could be carried out by a single matrix with the same shape. This consideration is easy to carry out since the units that form our vocabulary are BPE fragments that are usually jointly learn over the source and target corpus, so we can use the same vocabulary for both source and target. Once we have compatible encoder and decoder embeddings, we can see that the transpose of the output embedding, $\boldsymbol{E}_o^T$, also has the same dimension as the other 2 matrices provided that the embedding dimension and the hidden layer size are also equal. Therefore, weight tying consists in "tying" these 3 matrices together so that their job is carried out by a single matrix, by considering that $\boldsymbol{E}_e = \boldsymbol{E}_d = \boldsymbol{E}_o^T$. Experiments have shown that we can carry out this tying without losing performance since the output embedding shares the same properties of the input embedding by representing similar words in a similar way. This technique has two main advantages over using different embeddings. First, it allows the rows of the input embedding to update at every training step, instead of only when their corresponding word appears in the input. This helps the model to train faster. Second, and most importantly, it massively reduces the number of parameters that need to be learned by the model. We can observe this by comparing the size of the hidden matrices compared with the size of the embedding matrices. Whereas it is usual to have a hidden layer dimension of 1024, giving us hidden layer matrices of dimension (1024,1024), it is very common for the vocabulary to contain 20000 words or higher, which would translate into embedding matrices of dimension (1024,20000). It is usual to see models where the majority of pa-

[1] Named as $\boldsymbol{W}$,$\boldsymbol{U}$ and $\boldsymbol{V}$ in the weight-tying paper

rameters are part of the different embedding matrices. If we are able to use only one embedding matrix instead of 3, we achieve a very significant reduction in the number of parameters, with the associated gains in training performance and memory savings. Experiments carried out using weight tying show that the application of this technique can produce models that have 53% fewer parameters than the standard separate embedding approach, while at the same time maintaining or even improving translation quality.

The Transformer model uses this technique in order to obtain the aforementioned advantages in terms of performance and reduction of the number of parameters.

Label smoothing

We have previously explained in Equation 1.29 how we train neural networks by optimizing a cost function $J(\theta)$ that depends on the parameters of the model. We update the weights by using stochastic gradient descent and processing a batch of samples at the same time. For a batch of N samples, with y_b as the label of the sample, and $\hat{y}_n$ as the output of the network for that sample, the cost function is usually defined as the average of a loss function applied to each sample:

$$J(\theta) = \frac{1}{N}\sum_{n=1}^{N} L(y_n, \hat{y}_n) = \frac{1}{N}\sum_{n=1}^{N} L(y_n, f(x_n;\theta)) \tag{4.12}$$

The choice of loss function is therefore one of the decisions that need to be made when training a model by gradient descent. In classification problems, the most common choice of loss function is the cross-entropy function, that computes a measure of dissimilarity between two probability distributions, p and q. If the distributions are discrete, this can be computed as:

$$H(p,q) = \sum_{c} p(c)log(q(c)) \tag{4.13}$$

When working in classification problems, we have already established that we wish to obtain a system that outputs a probability distribution. Therefore, the distribution emitted by the network will take the place of the q distribution used to compute the cross-entropy. The label, or assigned class to the sample, can also be considered as a probability distribution. This is usually done by setting the probability of the label to 1, so $p(y) = 1$, and the rest of the possible values will have probability 0 such that $p(z) = 0, \forall z \neq y$. Having obtained those probability distributions, the cross-entropy can be applied as the loss function in the following way:

$$L(y,\hat{y}) = -\sum_{c} p_{label}(y = c|x)log(p_{network}(\hat{y} = c|x)) \tag{4.14}$$

We will now look at the specific case of MT. For a single training sample y with length I, we will denote the i-th token of the target sentence as y_i. The loss is the computed as:

$$L(y,\hat{y}) = -\sum_{i=1}^{I}\sum_{c} p_{label}(y_i = c|x)log(p_{network}(\hat{y}_i = c|x)) \tag{4.15}$$

An eager reader will have noticed that, due to the way we have defined the probability distribution given by the label, all the probability mass is given to a single value, and when computing the cross-entropy, the term of the label distribution can be ignored in all but one of the possible output values. The computation can be simplified to:

$$L(y,\hat{y}) = log(p_{network}(\hat{y} = y|x)) \tag{4.16}$$

While the previous simplification does make sense for some of the general applications of pattern recognition, it is not quite clear that this is the best possible approach for MT. Let us look at the way the label probability distribution is constructed. Say that, in the target sequence, the speaker describes something as "beautiful". Is it right to assume that such word is the only correct translation? What about other synonyms such as "gorgeous"?

The label smoothing [43] technique aims to improve model generalization capacity by reducing the confidence that the model has to assign to the training predictions. This technique consists in smoothing the one-hot label distribution. This works by setting aside a certain quantity of probability mass for the wrong-labels, and to redistribute this probability somehow.

So far the label probability distribution was computed as:

$$p_{label}(y = c|x) = \delta_{y,c} \tag{4.17}$$

Label smoothing introduces the following modification, where we discount some ϵ probability mass, and distribute it over all possible label values C by means of the uniform probability distribution.

$$p'_{label}(y = c|x) = (1 - \epsilon)\delta_{y,c} + \frac{\epsilon}{C} \tag{4.18}$$

Apart from helping the model to avoid being over-confident on its predictions due to being overfitted to the training data, this can have additional beneficial effects during the decoding step. We believe that the smoothing distribution helps the model during decoding by performing a less strict pruning of partial hypothesis, which can result in a better overall translation after the decoding process finishes.

Put together, all those changes result in significant improvements in both speed and quality that we hope to reproduce in the following experimental section.

System description

The Transformer architecture that was described in Section 4.1 currently achieves the best results in MT benchmarks. In order to try to replicate those performance improvements, and compare this model with the previous RNN-Attention architecture, we are now going to train a series of systems using this configuration. Sockeye implements this novel architecture, and we have used it to train a Transformer model that closely follows the parameters laid out in the original paper. We are first going to train a model that uses the same configuration as the Transformer Base model, but without label smoothing and weight tying. Then, we will add back those techniques and train another system. This will allow us to compare the performance of the Transformer model architecture without taking into consideration those 2 techniques that could also be applied to other models such as RNN-Attention.

Data filtering

After all the previous experiments, there is still one avenue of improving performance that we have yet to study. We have previously introduced the need for data filtering in Section 2.1.2, and we have described the Paracrawl corpus included in the WMT18 corpus and the problematic sentences it contains in Section 2.2.1. We will now look at the task of using data filtering in order to take advantage of the Paracrawl corpus to further

Table 4.1: Results obtained for Transformer models trained with the poliMedia corpus.

	test	
Transformer Model	BLEU	TER
layers=2	23.8	56.3
layers=4	23.8	56.1
layers=6	23.9	56.2

improve the performance of our German to English models. Additionally, we are going to combine this technique with additional techniques and configurations that can take advantage of the extra data we plan to introduce by using data filtering.

Results

This section describes and discusses the results obtained with the different models trained following the Transformer architecture in the poliMedia and WMT corpora. We will also discuss the effects of the data filtering carried out for the Paracrawl WMT corpus.

poliMedia

With respect to the poliMedia corpus, we begin by training a model that uses the complete Transformer architecture. We hope that this model will be able to improve the results obtained by the previous RNN-Attention models. By taking into account what we have learned when performing the previous experiments, we are also going to train additional models that have a lower number of layers, due to the aforementioned overfitting effect that appeared previously in the poliMedia corpus. The Transformer base model's encoder/decoder components are composed of 6 layers each, and we are going to train some different configurations by training a model that uses 4 layers in both components, and one that uses 2 layers.

Table 4.1 shows the results of the Transformer architecture when trained with the poliMedia corpus. When compared with the best result obtained with RNN-Attention models, available in Table 3.1, we can observe how the Transformer model obtains significantly better results than its RNN counterpart. The first Transformer configuration that we tried, with 6 layers, achieves 23.9 BLEU and 56.2 TER, compared with the 20.9 BLEU and 60.8 TER obtained by the RNN-Attention model. This represents an improvement of 3 points of BLEU and 4.6 points of TER, a very significant improvement over the previous model. The results obtained when training this architecture with 2 or 4 layers do not show any significant differences with the 6 layer model, so we can assume that, with this model and this corpus, there does not seem to be any advantage in reducing the number of layers of the model. We will therefore also use 6 layers when training systems for the WMT corpus.

Even though the 3 models have different amount of layers, they all perform in a similar way in terms of resource usage, something that we have already seen when looking at the qualitative results. All models take up around 6.4GB of memory, process around 4.8K tokens per second, and take between 7 and 8 hours to finish training. If the amount of layers of the model does not significantly affect computation time, that shows how there exists a speed bottleneck in the softmax of the output layer, something that is true to a greater or a lesser extent in all NMT models.

Table 4.2: Results obtained for Transformer models trained with the WMT corpus.

	newstest2017	
Model Transformer	BLEU	TER
- label smoothing & w.tying	30.5	56.5
Complete model	32.2	54.7
+ checkpoint averaging	32.3	54.7

Table 4.3: Resource consumption for the WMT task.

Model Transformer	VRAM consumption	Training time	Tokens/s
- label smoothing & w.tying	6.5 GB	49h	±8.5K
Complete model	7.6GB	140h	±4.8K

WMT

Table 4.2 shows the result of the different Transformer model configurations. The Transformer reduced model, without label smoothing and weight tying, obtains a BLEU score of 30.5 and a TER of 56.5, and is the model that has obtained the highest score in both the dev and test sets with respect to all previous models, with an improvement of 2.8 BLEU over the previous best model. The only other difference with the original paper was the learning rate schedule, we used the *lr_reduce* as with the previous RNN-Attention models. These results reflect its current status as the state of the art architecture for MT, due to the different quality and speed improvements laid out in [44]. The application of checkpoint averaging does not yield any improvement over the base model, further confirming the previous findings. The complete Transfomer model obtains 32.2 BLEU and 54.7 TER. This represents an additional improvement of 1.7 BLEU and 1.8 TER over the model that does not use the two aforementioned techniques. These results highlight that, although the Transformer architecture obtains a massive performance boost over the RNN-Attention architecture, some of this difference is not only because of the architecture, instead the improvement is obtained by the combination of the Transformer architecture with the label smoothing and weight tying techniques.

Table 4.3 shows the statistics concerning the training of the models. The reduced Transformer model consumes 6.5GB of memory, and takes around 49h to train. This represents an increase of around 16h with respect to the best RNN-Attention model, but brings with it a non-negligible performance improvement. The complete Transformer takes much longer to train, 140h, in part due to label smoothing, since we are telling the model to be unsure about its predictions, so we are effectively reducing the learning rate and making the training harder for the model. Once again, we find that even though the training time has significantly increased, we have obtained as a result a much better system.

The complete model continues training for more time because it is able to continuously obtain better scores in the validation sets, where as other models stop earlier once the validation performance starts to decrease.

After obtaining that system as a baseline, we now want to start applying data filtering to the corpus and to observe its effects. For carrying out this process, we have chosen the dual language model approach described in Section 2.1.2. We trained a language model both for the source and the target side, using as in-domain data newstest2014 from the WMT competition. The language models were then used them to obtain a score for each sentence pair. We carried out this scoring by computing the perplexity assigned by the language models to each part of the sentence, and then computing the geometric

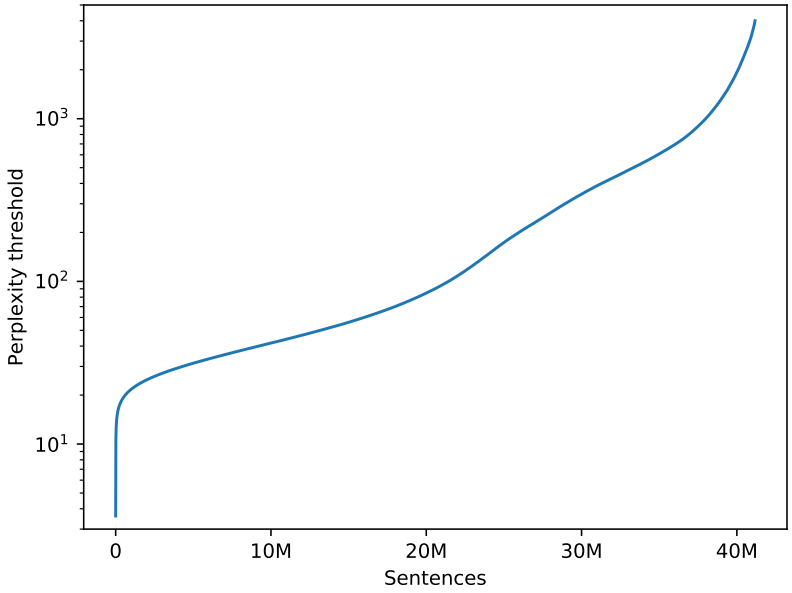


Figure 4.4: Number of sentences with perplexity under a given threshold.

mean. This means that lower scores are better, representing sentences that are assigned a high probability by the model, in contrast with sentences with higher scores that are considered very unlikely. We train the source (German) and target (English) language models with this data, and then apply it to all the sentence pairs that form the training data, including Paracrawl. The type of language model used is a 9-gram character model. This means that this model works at character-level instead of word-level. We believe that this is enough to detect the language of a sentence, while at the same time keeping the number of parameters low compared to what would happen with a 9-gram word model.

After obtaining this score, we order the sentences from lower to higher perplexity. Due to having trained the language models with the newstest2014 data, we can consider that this filtering step is applying both data filtering and domain data selection at the same time. Figure 4.4 shows the perplexity of the ordered sentence pairs. Using this information, we can calculate a threshold in order to carry out data filtering. With that threshold, we can simply keep sentence pairs whose score is lower than the threshold, while discarding pairs that have higher scores. We have calculated different thresholds in order to obtain 4 data subsets that contain 5M, 7.5M, 10M and 15M sentence pairs respectively. We then train 4 different models using those subsets, in order to find out which is the best filtering threshold, so that we can better take advantage of the training data.

The detailed results of this data filtering are shown in Table 4.4. We have chosen as baseline a system trained without the Paracrawl corpus. This model is trained using around 5.8M sentence pairs and obtains 32.0 BLEU and 54.8 TER. We are now going to compare its results with the data filtering models. The one trained with the 5M subset obtains 31.4 BLEU and 55.5 TER, 0.6 points of BLEU less than the no-Paracrawl counterpart. This could indicate that the filtering step is still allowing some noisy sentences to go through, therefore obtaining a somewhat worse result than not using Paracrawl. However, once we start using bigger filtered subsets, we start to see better results. The 7.5M subset already obtains 33.7, an improvement of 1.7 BLEU over the reference system,

and the 10M systems achieves 34.5 BLEU and 52.9 TER, a improvement of 2.5 BLEU and 2.3 TER over the reference system. This shows that, although the filter might allow some noise sentences through, this is offset by the increase in the number of training sentences that allows the model to learn how to translate better. The system trained with 15M obtains 34.3 BLEU and 52.7, very similar to the 10M system, but we selected the 10M subset over this one, because the larger subset does not show any significant improvements while at the same time being trained using 5M additional sentences that do not convert into translation improvements. In order to better understand the importance of data filtered, we also trained a system that does not apply this technique by using the entire Paracrawl corpus jointly with the rest of the WMT corpora as training data. The results obtained by this model that uses all available data for training are 21.3 BLEU and 70.2 TER. Compared with the baseline model trained without using the Paracrawl data that obtained 32.0 BLEU and 54.8 TER, we notice a massive performance loss of 10.7 BLEU points experienced by this non-filtered model. The results are, in fact, even worse than the ones obtained by the first RNN-Attention models that we tried. Such a hefty quality difference highlights the absolute need for data filtering if we are going to use noisy corpora such as Paracrawl.

Knowing this information, we have used the 10M subset in order to produce synthetic data following the method explained in Section 2.2. Instead of training a German to English system, we have used the data in order to train a system that translates in the opposite direction, English to German. This system was trained using the exact same hyperparameters and architecture as the other models, the only change being the translation direction. We can then use that system for obtaining backtranslations. In order to obtain the artificial source-side for the synthetic data, we first need to select a set of monolingual sentences in order to translate them into the source language. Following the strategy laid out in [37] we have randomly sampled 20M sentences from the English News Crawl 2017 corpus in order to carry out this task. This corpus contains thousands of online news articles crawled from the web, and we have selected it because it belongs to the same domain than the news translation we are interested in. Otherwise, if we had selected data from a different domain, we could find that there is no performance improvement, or even worse, that the out-of-domain synthetic sentences end up diminishing performance. This monolingual data was pre-processed using the same steps used for the rest of the bilingual data. The translations were carried in an adequate amount of time by grouping them into subtasks and executing them in parallel in a computer cluster.

Once we have obtained the backtranslations, we are then ready to train the final systems. In order to train these systems, we combined the 20M synthetic sentences with the original 10M pairs selected by the LM filtering. The original 10M pairs were oversampled by a factor of 2, obtaining a final corpus formed by 20M synthetic sentences and 20M bilingual sentece pairs (2*10M+20M). Additionally, we have processed the data using 40K BPE operations instead of 20K, in order to obtain a bigger vocabulary size that might synergize better with the language model improvements that we wish to obtain by using synthetic data. We trained 4 systems using this configuration, but each of them using a different random seed, in order to obtain runs that are independent from each other. Then, once the systems finished training, we were able to produce translations by making an ensemble that combined the predictions of the 4 systems.

Recall that we had used the Transformer model trained with 10M sentences in order to produce a synthetic corpus of 20M sentences. These backtranslations were used to train the definitive Transformer models. A single one of these systems obtains 35.9 BLEU and 51.2 TER. This represents an improvement of over 1.4 points of BLEU and 1.7 points of TER over the 10M Transformer model. This sizable increase in model per-

Table 4.4: Results obtained for different amounts of filtered sentences, WMT18 task.

	newstest2017	
Model Transformer	BLEU	TER
Baseline (5.8M, no Paracrawl)	32.0	54.8
Data filtering (5M)	31.4	55.5
Data filtering (7.5M)	33.7	56.5
Data filtering (10M)	34.5	52.9
+ Synthetic data (2*10M+20M)	35.9	51.2
+ Ensemble (x4)	**36.2**	**51.0**
Data filtering (15M)	34.3	52.7

Table 4.5: WMT18 news track results.

	newstest2018
System	BLEU
RWTH Trans. ensemble	48.4
UCambridge NMT-SMT	48.0
NTT Trans.	46.8
JHU RNN ensemble+R2L	45.3
MLLP Trans. ensemble	45.1
Ubiqus-NMT single	44.1
UEdin Trans. ensemble	43.9
LMU-München	40.9
NanjingU Trans.	38.3

formance is due to the combination of a higher vocabulary size and the inclusion of the synthetic data. Both changes serve to improve the language model of the NMT model, and as a result we are able to obtain better quality translations. An ensemble of 4 independent models trained with the previous configuration obtains 36.2 BLEU and 51.0 TER, an improvement of 0.3 and 0.2 respectively. Although not as significant as some of the previous improvements, we can still see how the inclusion of additional models can be leveraged in order to obtain extra quality improvements over a single model, due to the error correcting effect of the ensemble.

As a result of the efforts carried out in this work in collaboration with other MLLP members, this section has described how we obtained a competitive German-English system that we entered in the news translation task of WMT18, obtaining competitive results. The detailed system description, results and findings will be published in an paper titled "The MLLP-Universitat Politècnica de València German-English Machine Translation System for WMT 2018" [20], to be presented at the WMT 2018 Conference. This section has provided an ample description of the work that was used for the competition, but interested readers can refer to the system paper in order to have additional details available at their disposal. Table 4.5 shows the results obtained by all the primary systems submitted to the WMT18 competition. Our system, tagged as **MLLP** Trans. ensemble obtained 45.1 BLEU. This is a very competitive result when compared with the systems submittied by other participants, and is a result that is not far from the one achieved by the winner of the competition, a system submitted by RWTH Aachen.

CHAPTER 5

System integration into a transcription and translation platform

We have obtained a series of translation systems as an output of all the previous work described in this document. Amongst all of them, we are specially interested in the German to English translation system developed for WMT 2018. This system achieves state-of-the art results on a real task thanks to the inclusion of the advanced techniques described in previous chapters.

Now, we are interested in integrating that system into a production environment where it can be used to provide quality translations to real users. Our goal is to integrate it into the *Transcription and Translation Platform* (TTP). This cloud-based transcription and translation service has been developed by the MLLP research group, based on the *transLectures-UPV Platform* (TLP) software.

The TLP software is released under an open-source license and is the backbone behind TTP. TTP[1] is an active service with many users, and currently offers audio transcription services for 10 different languages, and translation services between 16 language pairs. An example of the TTP dashboard with a collection of transcribed and translated videos is shown in Figure 5.1. TTP is used to provide the transcriptions and translation of the poliMedia repository. This platform is what provides the automatically generated Spanish, English and Catalan subtitles that are shown with these videos.

We will now describe the TLP software and its architecture. We will first present a general overview of the whole TLP pipeline, and then we will focus our efforts in the translation generation module related to our work as well as describing the technical decisions taken in order to integrate our translation system.

TLP

TLP is a software for integrating transcription and translation services into media repositories. Broadly speaking, TLP provides a set of APIs that are available in order to provide services to media resources stored in a database. Users can request transcription or translation tasks for some of those media resources. Once received, these requests are processed by TLP in order to carry out the appropriate steps to complete the task depending on what was specifically requested. The required transcription and translation

[1] https://ttp.mllp.upv.es

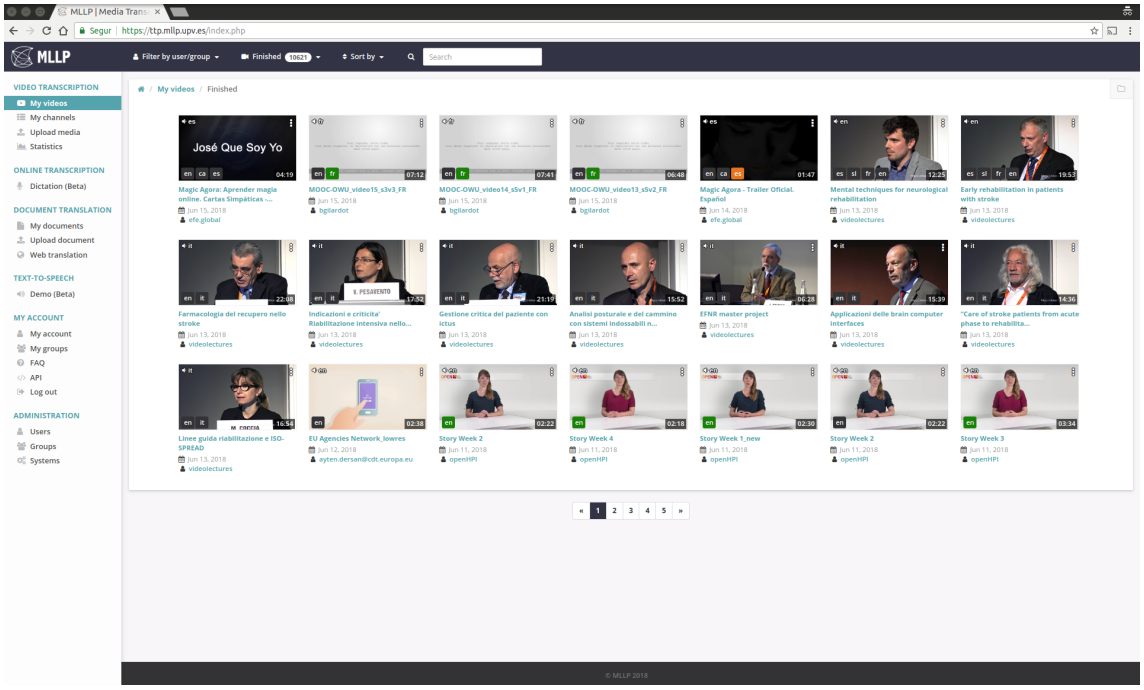


Figure 5.1: Example of some of the videos uploaded to TTP.

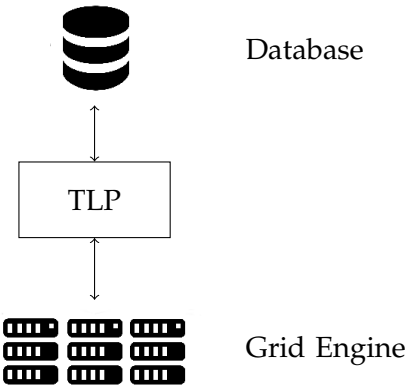


Figure 5.2: General TLP architecture.

tasks can be carried out efficiently on a computer cluster by means of a job scheduler that can be used to allocate the different tasks using a grid engine or job management system. This allows a high-degree of parallelization and a shorter time to completion for user task even when under heavy load. Once a job finishes, the Grid Engine notifies TLP and the output of the task is processed and stored in the database so as to make it available to the user. This architectural overview that we have described can be seen in Figure 5.2.

We are now going to describe what is the workflow used by TLP. Internally, the TLP software is organized in terms of modules, each of them tasked with carrying out an specific task such as translation. Once a media resource has been stored in the database and is submitted for that resource, TLP starts the workflow process for producing the appropriate output. The database contents are first processed in order to extract the required media files and their metadata. Then, the different transcription and translations services are run depending on the user request. Figure 5.3 illustrates the workflow followed by TLP. The workflow shown in the figure represents the complete process supported by TLP, but for each request only the modules that correspond to the tasks required by the user will be run. If the user requests an audio transcription for a video, the transcription

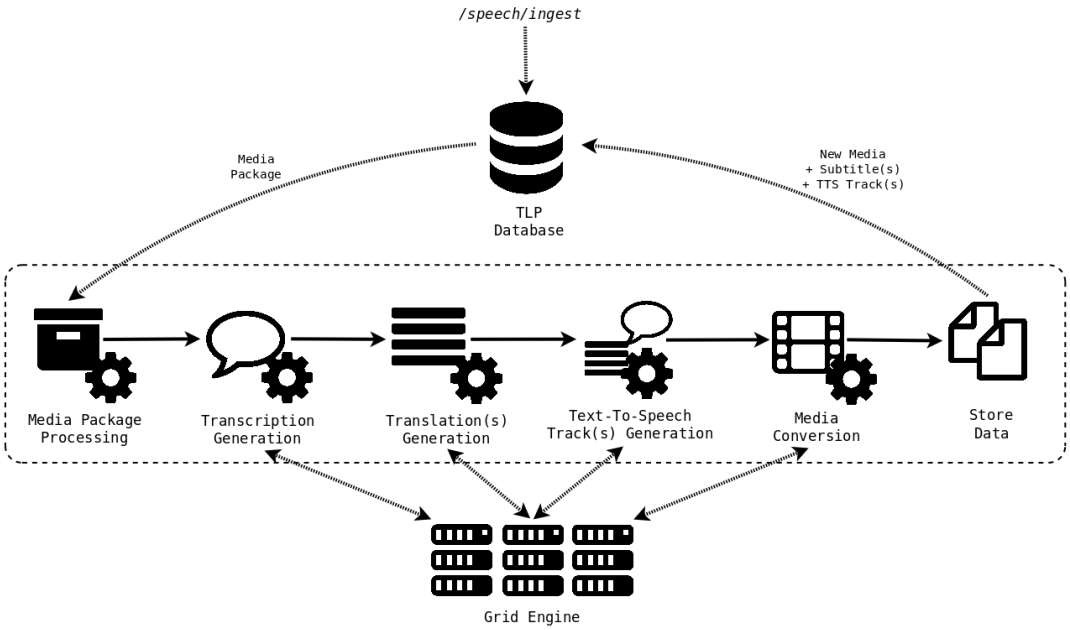


Figure 5.3: TLP Workflow.

module of the video's language is activated and produces a textual form of the spoken audio. This transcription can be further used if the user has asked for translation of the text into a different language, by using the output of the transcription module as the input of the translation module. This way we can obtain subtitles in many different languages for a video, a process that allows us to obtain an automatic multilingual version of an educational video, producing versions that are available on different languages from the ones it was recorded on. After obtaining the translations, it is also possible to use a Text-To-Speech module to produce audio files. As long as this modular structure is respected, one can carry out improvements and changes to individual components while maintaining a fully-functional system. Following this scheme, the integration of our German-English system will be carried out by defining a new translation module that can be selected when a translation involving that language pair is requested.

Translation generation

Now that we have an overview of the TLP software, we can focus our attention in the developed translation module for German-English. When talking about a trained model, we are actually referring to a computational graph that defines the model architecture and the different weights that define the parameters of the model. Translation is carried out by executing the computational graph jointly with a decoding algorithm. Our module will carry out this process by running Sockeye with the trained weights obtained as a result of training.

The process of integrating a translation system into a live production environment introduces additional difficulties apart from those inherent in adapting any piece of software into an already existing system. While we were previously concerned with the time it takes to train the system, we now are mostly interested in the translation speed achieved by the system. In order to process significant amount of sentences, changes in translation speed can greatly affect our service's ability to offer translations when facing big workloads. Due to this, we have selected for integration the single model version instead of the ensemble. Although the ensemble offers some translation improvements

with respect to the base model, it also means that we need the predictions of 4 models instead of 1 when carrying out inference, so we are using significantly higher amounts of memory while at the same time slowing translation speed, for only a small increase in translation quality. This is one of the many examples that illustrate the difference between research and industry. Research is interested in expanding human knowledge and moving the field forward. When carrying out experiments, it is often assumed that they are being carried out in perfect conditions. Even though a technique has a higher resource consumption than another, it might still be worth it if we obtain improvements by using it. However, when looking to adapt research developments into the real world, it is necessary to look at the actual conditions and business needs that need to be fulfilled by our system in order to choose which decision to make, because we will often find out that the specific set of requirements for an application might make many approaches unfeasible.

With respect to the data processing aspects, previously we thought of this process as a simple series of steps to apply to the data before starting training our systems. Once applied, they could pretty much be forgotten since training is mostly independent of those details. However, if we wish to later translate new data, we have to make sure that we have applied the same preprocessing steps that were used on the training data. Additionally, it might be necessary to apply some postprocessing to the resulting translations, for example to recover the original word segmentation if we have used BPE. In order to carry out this, we have developed preprocessing and postprocessing scripts containing the appropriate steps. The preprocessing script is applied to the input data before it is fed to the translation model itself, and the postprocessing script is applied to the system's translations before they are returned to the system.

Performance evaluation

As mentioned above, translation speed is paramount for our service, so we want to find out the optimum value for the parameters used during the decoding process. Because this process is going to be carried out in a computing cluster equipped with GPUs, sentences are going to be processed in batches in a similar way than during training, in order to take advantage of the parallelization capabilities of GPUs. The decoding process makes this decision harder than simply picking the maximum batch size supported by our GPU, because we need to balance the possible speed-up of using higher batch sizes with the associated increase in VRAM consumption. In fact, a lower batch size might be better if it allows us to parallelize some translation task by splitting it in multiple jobs. In order to study the effects of different batch sizes, we have carried out a series of experiments in order to measure system performance for batch sizes ranging from 1 to 4.

Table 5.1 shows the peak memory consumption and the time spent per sentence, measured for different batch sizes. If we translate each sentence separately, the model consumes 1.7GB of memory and processes one sentence every 0.64 seconds. Processing 2 sentences per batch yields a decrease of 0.21s in the time it takes to process a sentence, and a batch size of 3 an additional 0.08s decrease per sentence. While the amount of memory consumption roughly increases by 0.9GB for each additional sentence present in the batch, we can observe how the speed-up effect of increasing the batch size is not linear, in fact there exists almost no difference in speed between using batch size 3 or batch size 4.

The computing cluster where this system is deployed consists in a series of GTX960, GTX1080 and GTX1080Ti GPUs. These devices offer 4GB, 8GB and 11GB of VRAM memory, respectively. Based on the results of the previous experiments, we have selected a

Table 5.1: Resource consumption and system speed with respect to different batch sizes.

Batch Size	Max VMEM	Sec/Sent
1	1.7 GB	0.64
2	2.6 GB	0.43
3	3.5 GB	0.35
4	4.4 GB	0.34

batch size of 2 for our system. This allows us to translate sentences with a significant speed gain with respect to processing sentences in isolation, while keeping the memory consumption low enough in order to allow us to launch multiple translation jobs on the same GPU.

Due to the aforementioned speed concerns, we have elected to use a single model trained with synthetic data instead of the whole ensemble. This single model obtains 35.9 BLEU and 51.2 TER when evaluated with newstest2017, whereas the ensemble obtains 36.2 BLEU and 51.0 TER. This means that the single model's performance is worse by 0.3 BLEU and 0.2 TER when compared with the ensemble. This is a necessary trade-off because it allows us to run 1 model instead of 4, greatly reducing the time it takes to produce translations and the memory consumed by the model.

In order to compare the overall improvement achieved with this new production system, we are going to compare it with the previous production system, a phrase-based model similar to the ones described in Section 1.3.2. That system was trained by using the WMT 2016 data and an additional 200M monolingual sentences. When evaluated with newstest2016, the phrase-based model obtains 30.1 BLEU, and the Transformer production system obtains 41.9 BLEU. In addition to this quality improvement, the Transformer model takes less time per translated sentence, 0.43 seconds/sentence, compared with the 2.4 seconds/sentence of the phrase-based model. The combination of various NMT breakthrough has allowed us to obtain a model that beats the previous production system by more than 10 BLEU, a very substantial difference that has considerable effects in the way sentences are translated by this new system, all while achieving a considerable speed-up in terms of translation speed.

The meaning of this BLEU difference can be understood better by looking at some of the sentences translated by these models. We have sampled a series of sentences from newstest2017 in order to illustrate the differences in translation quality between these systems. Table 5.2 shows the comparison between the phrase-based production system, the Transformer baseline model and the Transformer production system. Looking at the sentences shown in the Table, one can see how for this particular set of sentences, the phrase-based system's translations are not very good and often include German words in the English translation. In contrast, the Transformer model, and specially the production version, are able to preserve the semantic meaning of the sentence much better. We also show some example from the poliMedia corpus in Table 5.3, comparing the performance of RNN-Attention with Transformer.

Table 5.2: Translation samples comparing the performance of SMT models, WMT corpus.

Source	Er sagte, dass ihn ein weiterer Stiefsohn, der in der Nähe wohnt, gewarnt hätte.
Reference	he said another stepson who lives nearby alerted him.
PB-production	he said that it will provide a further stiefsohn, who lives in valiasr, have been warned.
Transformer-baseline	he said that another boy who lives near him would have warned him.
Transformer-production	he said another stepson who lives nearby would have warned him.

Source	Russlands Putin feuert Stabschef Sergei Ivanov
Reference	russia's putin sacks chief of staff sergei ivanov
PB-production	russlands stabschef sergei ivanov putin fires
Transformer-baseline	russia's putin fighters sergei ivanov
Transformer-production	russia's putin fires chief of staff sergei ivanov

Source	Clintons Kampagne hat Steuererklärung zurückgehend bis zum Jahr 2007 veröffentlicht
Reference	clinton's campaign has released tax returns going back to 2007.
PB-production	clintons campaign has steuererklärung down until the year published in 2007.
Transformer-baseline	clinton's campaign published tax returns back to 2007.
Transformer-production	clinton's campaign has released tax returns back to 2007.

Table 5.3: Translation samples comparing the performance of SMT models, poliMedia corpus.

Source	y aquí sin embargo se ha hecho con una elución con gradiente.
Reference	and here, nevertheless, it has been done with a gradient elution.
RNN-Attention	and yet, it's done with a fupe with grassroots.
Transformer	and here, though, it has been done with a gradient elution.

Source	bien, las estadísticas son muchas veces las grandes mentiras.
Reference	well, statistics are sometimes the biggest lies.
RNN-Attention	well, the statistics are often a lot of big lies.
Transformer	well, statistics are often the big lies.

CHAPTER 6
Conclusions

This chapter summarizes the tasks carried out during this work as well as the conclusions and lessons we have obtained as a result of the previous process.

Chapter 1 has highlighted the importance of MT, and serves as an introduction to the theoretical basics of this field. We have described the 3 different approaches to MT: word-based models, phrase-based models and NMT. Neural networks have also been described in order to give the reader the tools required to understand the rest of the work.

In Chapter 2 we have described the importance of the whole data collection and processing pipeline, first by explaining the need for different data processing techniques, and how we have applied them to our work. The characteristics of the corpus used in this work require a specific answer to the problem of noisy data. Further experiments show the vital importance of data filtering, because otherwise we find ourselves with a significant drop in translation quality. We have overcome this problem by developing a data filtering technique based on language models, that achieves a balance between filtering capacity and resource consumption.

Chapter 3 presents a description of the ins and outs of NMT. We first have described the general encoder-decoder theoretical framework used by most NMT models, and then we have started describing the RNN-Attention model architecture as well as the computer tools that implement this architecture. Out of all the different models trained using this architecture, we want to highlight the effects of the checkpoint averaging technique. Our results have served us to validate the fact that the improvements achieved with this technique are due to a sub-par training procedure, as described in other previous research articles.

In Chapter 4 we have introduced the novel Transformer architecture and its components. This architecture has been used to train MT systems that obtain better performance than all previous ones.

We combined the experience gained from training Transformer models with data filtering techniques in order to develop our best translation system. This German to English system, an ensemble of Transformer models, was trained using the data of the WMT 2018 competition. We submitted this system to the news-translation task of the WMT competition, where we obtained competitive results that are not far from the winners of the competition. The detailed approach and techniques used for training this system will be published in a system paper that will be presented at the WMT conference.

In Chapter 5 we have described the integration of the German to English system into the TTP live production environment that is used to provide transcriptions and translations for the poliMedia repository. We wish to highlight how the end result of this work

is a not a mere research prototype, but a fully-fledged state-of-the-art translation system integrated into a real production environment. The final system is currently live at TTP and is ready to provide translations for any potential user.

In terms of future work, we have identified one clear avenue of improvement with respect to data filtering. The system developed for this work represents a first approach to a problem that only recently began to gain traction. We believe there is a chance to significantly improve the performance of this technique by developing systems that are specifically developed for data-filtering. Currently a common data filtering technique is to score sentence pairs with a translation model such as the RNN-Attention models we have described in this work. These models have been trained in order to produce a sentence that is a correct translation of the input. However, data filtering only requires that we are provided with a score that tells us if a sentence pair is good or not. This means that traditional translation systems might have an excessive number of parameters if we want to use them for a relatively simpler task such as data filtering. Building upon [26], we want to study the feasibility of a data-filtering model based on sentence embeddings. This specific model consists in two neural networks, each of them producing an embedding of one of the sentences in the pair. Then, the embeddings can be compared using a similarity measure in order to obtain an estimate of how related those sentences are. This approach would allow us to achieve speed and resource usage improvements with respect to a general purpose model.

Another future avenue is to develop, train and integrate into TTP additional translation systems for language pairs other than German-English. We have already seen how the Transformer architecture obtains significant improvements in Spanish-English translation. Our next immediate step is to develop a production ready Spanish-English system based on this architecture by training a system using additional bilingual corpora, in order to supplement the data of the poliMedia corpus and improve performance. Additionally, we also want to develop production systems for the other language pairs of the X5gon project.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[3] Parnia Bahar, Tamer Alkhouli, Jan-Thorsten Peter, Christopher Jan-Steffen Brix, and Hermann Ney. Empirical investigation of optimization algorithms in neural machine translation. In *Conference of the European Association for Machine Translation*, pages 13–26, Prague, Czech Republic, June 2017.

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

[5] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. *CoRR*, abs/1711.02173, 2017.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[7] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

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

[9] Boxing Chen, Roland Kuhn, George Foster, Colin Cherry, and Fei Huang. Bilingual methods for adaptive training data selection for machine translation. In *The Twelfth Conference of The Association for Machine Translation in the Americas*, pages 93–106, 2016.

[10] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics, 2014.

[12] 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)*, pages 1–38, 1977.

[13] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[15] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.

[17] Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. Sockeye: A toolkit for neural machine translation. *CoRR*, abs/1712.05690, 2017.

[18] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

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

[20] Javier Iranzo-Sánchez, Pau Baquero, Gonçal Garcés, Jorge Civera, and Alfons Juan. The MLLP-Universitat Politècnica de València German-English Machine Translation System for WMT 2018. In *Proceedings of the Third Conference on Machine Translation*. To appear.

[21] Frederick Jelinek. *Statistical methods for speech recognition*. MIT press, 1997.

[22] Marcin Junczys-Dowmunt, Tomasz Dwojak, and Rico Sennrich. The amu-uedin submission to the wmt16 news translation task: Attention-based nmt models as feature functions in phrase-based smt. In *Proceedings of the First Conference on Machine Translation*, pages 319–325, Berlin, Germany, August 2016. Association for Computational Linguistics.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[24] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[25] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics, 2003.

[26] Lajanugen Logeswaran and Honglak Lee. An efficient framework for learning sentence representations. In *International Conference on Learning Representations*, 2018.

[27] Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*, 2017.

[28] Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[30] 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.

[31] Jan-Thorsten Peter, Andreas Guta, Tamer Alkhouli, Parnia Bahar, Jan Rosendahl, Nick Rossenbach, Miguel Graça, and Hermann Ney. The rwth aachen university english-german and german-english machine translation system for wmt 2017. In *Proceedings of the Second Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 358–365, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[32] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *EACL (2)*, pages 157–163. Association for Computational Linguistics, 2017.

[33] Ye Qi, Devendra Sachan, Matthieu Felix, Sarguna Padmanabhan, and Graham Neubig. When and why are pre-trained word embeddings useful for neural machine translation? In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 529–535. Association for Computational Linguistics, 2018.

[34] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

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

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[37] Rico Sennrich, Alexandra Birch, Anna Currey, Ulrich Germann, Barry Haddow, Kenneth Heafield, Antonio Valerio Miceli Barone, and Philip Williams. The university of edinburgh's neural mt systems for wmt17. In *Proceedings of the Second Conference on Machine Translation, Volume 2: Shared Task Papers*, pages 389–399, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[38] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[39] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[40] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

[41] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, volume 200, 2006.

[42] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[43] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826. IEEE Computer Society, 2016.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[45] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.