UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

MASTER'S THESIS

Online Multilingual Neural Machine Translation

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Javier Iranzo Sánchez

Supervised by:
Dr. Jorge Civera Saiz
Dr. Alfons Juan Ciscar

Experimental director:
Dr. Adrià Giménez Pastor

July 13, 2019

Contents

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. The techniques presented in this work have been applied in order to develop MT systems for a variety of language pairs in an online context, that allows the possibility of providing live translations.

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.

1.1 Motivation

Due to recent technological developments, 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

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

[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.

1.2 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.

Formally, given a sentence in one language, $\mathbf{x} = x_1, x_2 \ldots, x_J$, the goal is to find the best translation, that is, the $\mathbf{y} = y_1, y_2 \ldots, y_I$ that maximizes $p(\mathbf{y}|\mathbf{x})$.[6]

This probability is learned from parallel corpora, collections of text that contain sentences in one language paired with their translations into another language. 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 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[7] 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. The parallel data 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:

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{y})p(\mathbf{y})}{p(\mathbf{x})} \tag{1.1}$$

[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/

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

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

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

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* [57], $p(\mathbf{y})$ 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 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:

$$\hat{\mathbf{y}} = \arg\max_{y} p(\mathbf{y})p(\mathbf{x}|\mathbf{y}) \tag{1.2}$$

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 [20] or beam-search [28] 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.

1.2.1 Word-Based Models

Translation model

Word-based models [8, 9] 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:

$$p(\mathbf{x}|\mathbf{y}) = \sum_{\mathbf{a}} p(\mathbf{x}, \mathbf{a}|\mathbf{y}) \tag{1.3}$$

$\mathbf{a}$ 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:

$$p(\mathbf{x}, \mathbf{a}|\mathbf{y}) = \prod_{j=1}^{J} p(x_j, a_j | x_1^{j-1}, a_1^{j-1}, \mathbf{y}) = \prod_{j=1}^{J} p(a_j | x_1^{j-1}, a_1^{j-1}, \mathbf{y}) p(x_j | x_1^{j-1}, a_1^{j}, \mathbf{y}) \tag{1.4}$$

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:

$$p(a_j | x_1^{j-1}, a_1^{j-1}, \mathbf{y}) := \frac{1}{I+1} \tag{1.5}$$

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.

$$p(x_j | x_1^{j-1}, a_1^{j}, \mathbf{y}) := p(x_j | y_{a_j}) \tag{1.6}$$

The previous two assumptions allow us to obtain a simplified computation for Equation 1.4:

$$p(\mathbf{x}, \mathbf{a}|\mathbf{y}) = \frac{1}{I+1} \prod_{j=1}^{J} p(x_j | y_{a_j}) \tag{1.7}$$

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:

$$p(u|v) = \frac{N(u,v)}{\sum_{u'} N(u',v)} \tag{1.8}$$

where $N(u,v)$ 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:

$$N(u,v) = \sum_{j=1}^{J} \delta(x_j = u)\delta(y_{a_j} = v) \tag{1.9}$$

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) [15], 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:

$$p(\mathbf{x}|\mathbf{y}) = \frac{1}{(I+1)^J} \prod_{j=1}^{J} \sum_{i=1}^{I} p(x_j|y_i) \tag{1.10}$$

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.

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.

$$p_{LM}(y_0^I) = \prod_{i=1}^{I} p(y_i|y_0^{i-1}) \tag{1.11}$$

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:

$$p(y_i|y_0^{i-1}) \simeq p(y_i|y_{i-(n-1)}^{i-1}) \tag{1.12}$$

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.

$$p(y_i|y_{i-(n-1)}^{i-1}) = \frac{N(y_{i-(n-1)}, \ldots, y_{i-1}, y_i)}{\sum_v N(y_{i-(n-1)}, \ldots, y_{i-1}, v)} \tag{1.13}$$

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

$$PP = 2^{-\frac{1}{N}\log p(y_1, y_2, \ldots, y_n)} \tag{1.14}$$

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.

src: $\boxed{\text{28}}\ \boxed{\text{@-@ jähriger}}\ \boxed{\text{Koch}}\ \boxed{\text{in San}}\ \boxed{\text{Francisco}}\ \boxed{\text{Mall}}\ \boxed{\text{tot}}\ \boxed{\text{aufgefunden}}$

tgt: $\boxed{\text{the 28}}\ \boxed{\text{@-@ year @-@ old}}\ \boxed{\text{Koch}}\ \boxed{\text{in San}}\ \boxed{\text{Francisco}}\ \boxed{\text{mall}}\ \boxed{\text{found}}\ \boxed{\text{dead .}}$

Figure 1.1: Example of the phrase segmentation used by a SMT model when translating sentence from the WMT corpus.

1.2.2 Phrase-Based Models

Phrase-based models [37] 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 1.1 shows the phrase segmentation selected by a phrase-based model built using the Moses toolkit [35], 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.

$$p(R) = \exp \sum_{i=1}^{n} \lambda_i h_i(R) \tag{1.15}$$

A combined model that assigns weights to the different components,

$$p(\mathbf{y}|\mathbf{x}) = p_t(\mathbf{x}|\mathbf{y})^{\lambda_t} p_r(\mathbf{x}|\mathbf{y})^{\lambda_r} p_{LM}(\mathbf{y})^{\lambda_{LM}} \tag{1.16}$$

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 $\mathbf{x}$ into I different x_i phrases, the translation model is defined as:

$$p_t(\bar{x}_1^I|\bar{y}_1^I) = \prod_{i=1}^{I} \phi(\bar{x}_i|\bar{y}_i) \tag{1.17}$$

$\phi(x_i|y_i)$ 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.

$$\phi(\bar{x}|\bar{y}) = \frac{N(\bar{x}, \bar{y})}{\sum_{x'} N(\bar{x'}, \bar{y})} \tag{1.18}$$

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 [34] 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.

$$p_r(\bar{x}_1^I|\bar{y}_1^I) = \prod_{i=1}^{I} d(start_i - end_{i-1} - 1) \tag{1.19}$$

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.

1.2.3 Neural-Based Models

This section briefly introduces the basic concepts of neural networks. Readers are recommended to consult chapter 6 of [19] 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[51], precursor of the neural network, was introduced in 1957, and the backpropagation algorithm [52] 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, Neural Machine Translation (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 outputs 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 $\mathbf{x}$:

$$\mathbf{h}^{(1)} = g^{(1)}(\boldsymbol{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \tag{1.20}$$

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

$$\mathbf{h}^{(i)} = g^{(i)}(\boldsymbol{W}^{(i)}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}) \tag{1.21}$$

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.

$$softmax(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{1.22}$$

$$p(c|x) = softmax(\boldsymbol{W}^{(out)}\mathbf{h}^{(I)} + \mathbf{b}^{(out)}) \tag{1.23}$$

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 [52]. 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.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{1.24}$$

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.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha(\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \epsilon) \tag{1.25}$$

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:

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

Compared with Equation (1.21), 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 [23]. In order to fight this problem, the use of special units that are better suited for this task, such as the LSTM unit [24] 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 2.1.

1.2.4 Computational graphs

As explained in Section 1.2.3, 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.

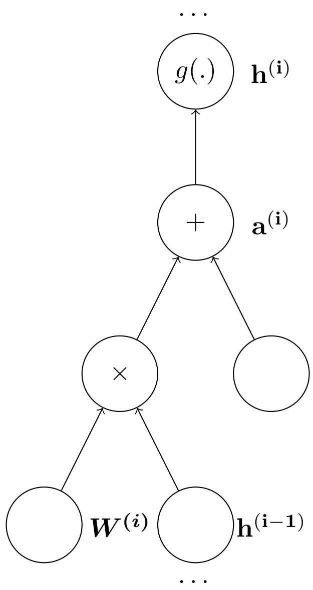


Figure 1.2: Computational graph representation of the operations carried out in one of the hidden layers of a neural network.

In order to compute these derivatives in an easy and efficient way, libraries such as PyTorch 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 1.2 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 (ML) 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 PyTorch and MXNet 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.

1.2.5 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) [44] 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.

$$AveragePrecision(N) = \frac{1}{N}\sum_{n=1}^{N} \log p_n \tag{1.27}$$

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).

$$BrevityPenalty = \begin{cases} 1 & : if\ c > r \\ \exp(1 - \frac{r}{c}) & : if\ c \leq r \end{cases} \tag{1.28}$$

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:

$$BLEU(4) = BrevityPenalty * AveragePrecision(4) \tag{1.29}$$

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.

[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. [34]

Another common automatic evaluation measure is the *Translation Error Rate* (TER) [58]. 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.

$$TER = \frac{\text{Number of required edits}}{\text{Length of the reference}} \tag{1.30}$$

The available type of edits are:

- Insertion of a word
- Deletion of a word
- 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.

Both measures should be evaluated with tokenized reference and hypothesis (See Section 3.1). Ideally, we want to use an agreed-upon tokenization, in order to be able to measure them consistently across works carried out by different authors. This can be achieved by using a tool such as SacreBLEU [48], that carries out a standard tokenization without user intervention. In case this is not possible (for example, because the detokenized version has not been made public), we have to rely on tools that use a user-supplied tokenization such as `multi-bleu` from Moses.

1.3 Framework of this work

This work has been made possible thanks to author's research position at the Machine Learning and Language Processing (MLLP) research group of Universitat Politècnica de València (UPV). The research leading to these results has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement no. 761758 (X5gon). The work explained in this document was carried out during UPV's participation in this project.

1.4 Document structure

This document is organized into 7 chapters. The first chapter serves as an introduction to the field of Machine Translation, how MT systems work and how they are evaluated. The motivation of this work has also been introduced.

Chapter 2 introduces the principles behind Neural Machine Translation. We describe the current NMT architectures, Attention-based RNN and Transformer, as well as outlining some practical issues for training NMT models. Chapter 3 describes the experiments performed for the WMT18 competition, using the previously described architectures. These experiments have been used as a starting point point for the rest of experiments described in this Master's Thesis. Chapter 4 introduces the X5gon H2020 project and describes the data, techniques and architectures that have been used in order to develop MT systems for language pairs that are relevant to the project. Chapter 5 and 6 describe the work carried out for two translation tasks of the WMT19 competition. They describe the systems developed for the Similar Language Translation Task and for the Online News Translation Task, respectively. Chapter 7 describes the work carried out in order to integrate the translation systems obtained as output of the previous process into a live production environment. Additionally, the case of implementing online NMT systems, as well as the challenges that this brings, is studied in detail. The work finishes with an overview of the different conclusions that have been obtained as a result of this project as well as a summary of the work carried out, available on Chapter 8.

CHAPTER 2

NEURAL MACHINE TRANSLATION

This chapter offers a detailed description of Neural Machine Translation and its current architectures. This is done in order to describe the theoretical background required for understanding the rest of the work. We will describe two of the most relevant architectures: Attention-RNN and Transformer, as well as the practical issues faced and computer tools used for training NMT systems.

2.1 Attention-based RNN NMT models

This section describes the RNN Attention-based model. 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.

2.1.1 Overview of NMT models

The basic principles behind neural networks have been introduced in Section 1.2.3. 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 code 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 [13], used the output of the NMT system as an additional feature for a phrase-based model, following the log-linear framework explained in Section 1.2.2. 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[24] or GRU[13] 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 2.2, 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.

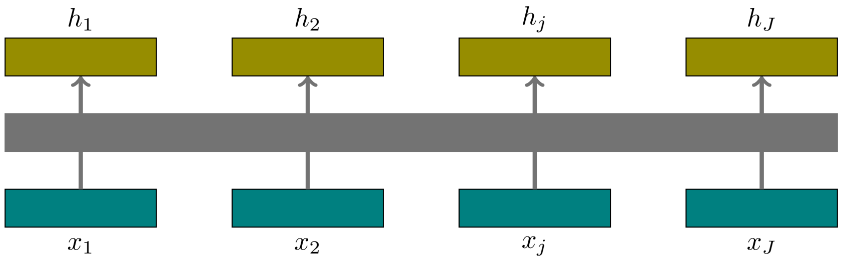


Figure 2.1: Encoder

2.1.2 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]. Figure 2.1 shows a representation of the encoder component.

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.

$$\overrightarrow{\mathbf{h}}_j = f_{enc\_forward}(\overrightarrow{\mathbf{h}}_{j-1}, \mathbf{x}_j) \tag{2.1}$$

$$\overleftarrow{\mathbf{h}}_j = f_{enc\_backward}(\overleftarrow{\mathbf{h}}_{j+1}, \mathbf{x}_j) \tag{2.2}$$

$$\mathbf{h}_j = [\overrightarrow{\mathbf{h}}_j; \overleftarrow{\mathbf{h}}_j] \tag{2.3}$$

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[24] or GRU [13] units.

This information produced by the encoder will be then used by the decoder to output the translation. Figure 2.2 shows the structure of the decoder component.

The decoder is a unidirectional 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.

$$\mathbf{s}_i = f_{dec}(\mathbf{s}_{i-1}, \mathbf{y}_{i-1}, \mathbf{c}_i) \tag{2.4}$$

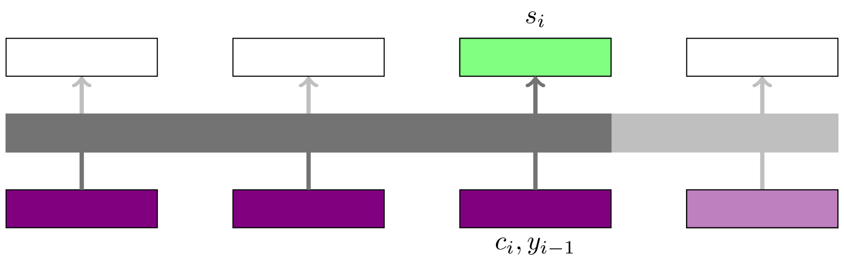


Figure 2.2: Decoder

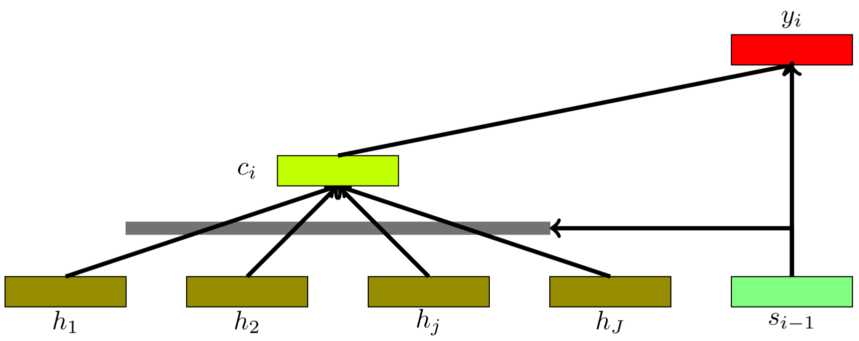


Figure 2.3: Alignment (Attention)

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 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. Figure 2.3 illustrates how the context vector $\mathbf{c_i}$ is produced.

An attention function is described by [62] 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.

$$c_i = \sum_j \alpha(j|i)\mathbf{V}_j \tag{2.5}$$

$$\alpha(j|i) = softmax(attention(\mathbf{q}, \boldsymbol{K}))_j \tag{2.6}$$

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 (2.5) and (2.6) 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:

$$\mathbf{c}_i = \sum_j \alpha(j|i)\mathbf{h}_j \tag{2.7}$$

$$attention(\mathbf{s}_{i-1}, \mathbf{h}_j) = \mathbf{v}_a^T tanh(\boldsymbol{W}_a \mathbf{s}_{i-1} + \boldsymbol{U}_a \mathbf{h}_j) \tag{2.8}$$

where $\mathbf{v}_a$, $\boldsymbol{W}_a$ 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 (2.8) is known as additive or Bahdanau attention. Later works have proposed different scoring functions, such as the dot-product attention of [41], that computes the scoring function as:

$$attention(\mathbf{s}_{i-1}, \mathbf{h}_j) = \mathbf{s}_{i-1}^T \mathbf{h}_j \tag{2.9}$$

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.

$$p(y_i|\mathbf{s}_i, yi-1, \mathbf{c}_i) = softmax(f_{emiss}(\mathbf{s}_{i-1}, \mathbf{y}_{i-1}, \mathbf{c}_i)) \tag{2.10}$$

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 3.1, form the basic NMT system.

2.2 Transformer Self-Attention NMT models

The Transformer [62] 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 [21] and Layer Normalization [1]. 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.

2.2.1 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 (2.9). Scaled attention introduces a scaling term that depends on the dimensions of the key, d_k. Remember that in conventional attention we use $\mathbf{q} = \mathbf{s_{i-1}}$, $\mathbf{K_j} = \mathbf{h_j}$, and $\mathbf{V_j} = \mathbf{h_j}$.

$$attention(\mathbf{q}, \mathbf{K}_i) = \frac{\mathbf{q}^T \mathbf{K}_i}{\sqrt{d_k}} \tag{2.11}$$

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 in terms of matrix multiplication. This means that the entire Attention process is computed as:

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{2.12}$$

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:

$$MultiHead(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Concat(head_1, \ldots, head_h)\boldsymbol{W}^o \tag{2.13}$$

$$head_i = Attention(\boldsymbol{Q}\boldsymbol{W}_{\boldsymbol{i}}^{\boldsymbol{Q}}, \boldsymbol{K}\boldsymbol{W}_{\boldsymbol{i}}^{\boldsymbol{K}}, \boldsymbol{V}\boldsymbol{W}_{\boldsymbol{i}}^{\boldsymbol{V}}) \tag{2.14}$$

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

2.2.2 Layer Normalization

Layer Normalization [1] 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, $\mathbf{a^{(i)}}$, before the activation function $g(.)$ is applied. Thus, we can apply Layer Normalization to layer i as follows:

$$LayerNorm(\mathbf{a^{(i)}}) = \frac{\boldsymbol{\gamma}}{\sigma^{(i)}} \odot (\mathbf{a^{(i)}} - \mu^{(i)}) + \boldsymbol{\beta} \tag{2.15}$$

$$\mu^{(i)} = \frac{1}{H}\sum_{h=1}^{H} a_h^{(i)} \tag{2.16}$$

$$\sigma^{(i)} = \sqrt{\frac{1}{H}\sum_{h=1}^{H}(a_h^{(i)} - \mu^{(i)})^2} \tag{2.17}$$

where $\mu^{(i)}$ and $\sigma^{(i)}$ are broadcasted to have the same dimension as $\mathbf{a^{(i)}}$. $\boldsymbol{\gamma}$ 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.

2.2.3 Residual Connections

Residual Connections [21], 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.

2.2.4 Modelling 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.

$$e_n = [w_n; p_n]\boldsymbol{E} \tag{2.18}$$

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

$$e_n = w_n\boldsymbol{E}_n + p_n\boldsymbol{E}_p = WE^{(n)} + PE^{(n)} \tag{2.19}$$

where $WE^{(n)}$ 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:

$$PE_{2i} = \sin\left(pos\frac{1}{10000^{2i/d_{model}}}\right) \forall i \in [0, d_{model}/2 - 1] \tag{2.20}$$

$$PE_{2i+1} = \cos\left(pos\frac{1}{10000^{2i/d_{model}}}\right) \forall i \in [0, d_{model}/2 - 1] \tag{2.21}$$

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.

2.2.5 Model Architecture

In the Transformer architecture, residual connections and layer normalization 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.

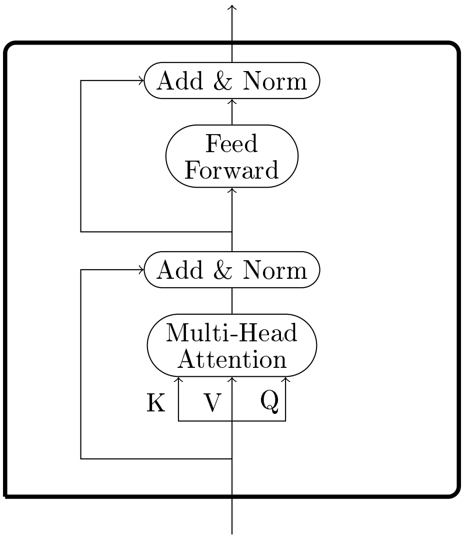


Figure 2.4: Transformer Encoder Block.

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. Figure 2.4 shows a Transformer Encoder block.

The standard Transformer block consists in a Multi-Head attention sub-layer followed by a Feed Forward sub-layer. 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.

Figure 2.5 shows a Transformer Decoder block. Compared with the Encoder blocks, the decoder blocks include an additional 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.

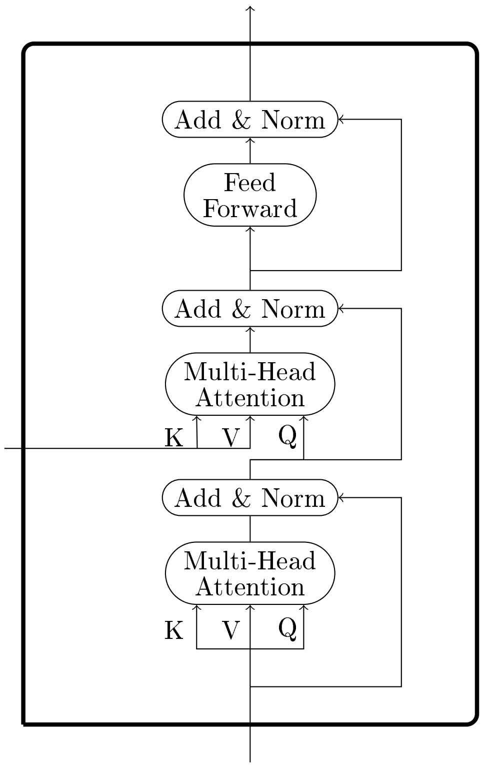


Figure 2.5: Transformer Decoder Block.

A graphical overview of the Transformer architecture is shown in Figure 2.6. It is usual to choose one of the following configurations, Transformer Base or Big, when building NMT systems.

- **Transformer Base**: 6 encoder/decoder blocks, embedding dimension 512, hidden layer size of 2048 and 8 attention heads.
- **Transformer Big**: 6 encoder/decoder blocks, embedding dimension 1024, hidden layer size of 4096 and 16 attention heads.

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.

2.2.6 Weight tying

Weight-tying [49] is a proposed technique for improving the performance of neural network language models through manipulation of the different embedding layers,

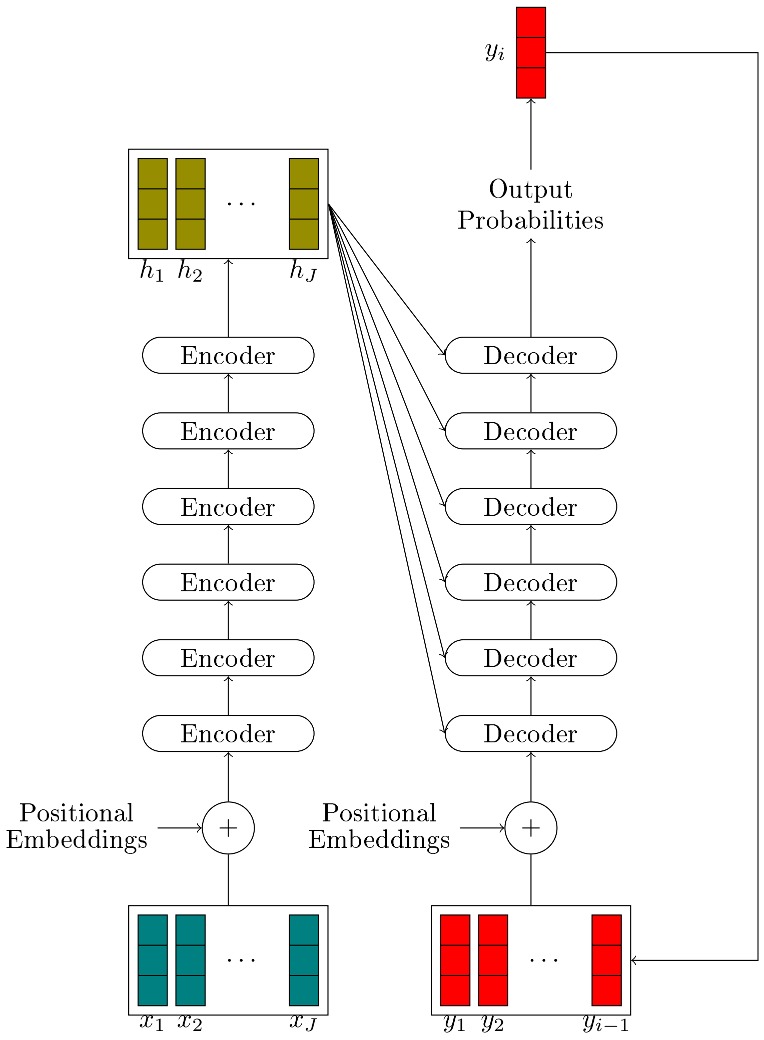


Figure 2.6: Figure of the Transformer architecture.

while also providing a significant reduction in number of parameters of the model. As previously explained in Section 2.1.1, 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 usually 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 are authors 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, [50] 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 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 [49] 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. 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 parameters 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.

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

2.2.7 Label smoothing

We have previously explained in Equation (1.24) 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:

$$J(\theta) = \frac{1}{N}\sum_{n=1}^{N} L(y_n, \hat{y}_n) = \frac{1}{N}\sum_{n=1}^{N} L(y_n, f(x_n;\theta)) \tag{2.22}$$

The choice of the 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:

$$H(p,q) = \sum_{c} p(c)log(q(c)) \tag{2.23}$$

When working with 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:

$$L(y,\hat{y}) = -\sum_{c} p_{label}(y = c|x)\log(p_{network}(\hat{y} = c|x)) \tag{2.24}$$

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:

$$L(y,\hat{y}) = -\sum_{i=1}^{I}\sum_{c} p_{label}(y_i = c|x)\log(p_{network}(\hat{y}_i = c|x)) \tag{2.25}$$

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:

$$L(y,\hat{y}) = \log(p_{network}(\hat{y} = y|x)) \tag{2.26}$$

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 [61] 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 re-distribute this probability somehow.

So far the label probability distribution was computed as:

$$p_{label}(y = c|x) = \delta_{y,c} \tag{2.27}$$

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.

$$p'_{label}(y = c|x) = (1 - \epsilon)\delta_{y,c} + \frac{\epsilon}{C} \tag{2.28}$$

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.

2.3 Practical issues

Apart from the architectures themselves, there are many experimental decisions and techniques that need to be taken into account in order to train state of the art systems. These range from hyperparameters values to be used during training, to specific techniques for data pre-processing and data generation. We will describe some of the most important issues in this section:

- **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 systems, 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 [31], 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:

$$\boldsymbol{\theta}_{ensemble} = \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{\theta}_m \tag{2.29}$$

As outlined in [2], 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.** By default, sentences longer than a certain number of tokens are discarded, in order to speed-up training and to filter out 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 and better translation quality for longer sentences, but might also significantly slow-down training, increase memory footprint and make convergence more difficult.

- **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. There are many learning rate schedules that are used in the literature. 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*. Other schedules modify the learning rate depending on the current training step. For example, the inverse square root schedule [62], as its name suggests, updates the learning rate to be proportional to the inverse square root of the step number.

- **Batch size.** Recall from Equation (1.25) that minibatch gradient descent updates the model's paramteters using a noisy estimation of the gradient. The amount of noise depends on the batch size. The closer the batch size is to the number of training samples, the better estimation of the gradient, and therefore a smaller amount of noise is added. A smaller batch size might require more time for training, but the additional noise might be useful in order to improve model performance and prevent overfitting. However, too much noise might cause problems optimizing the model. As such, the right batch size will depend both on the model architecture as well as the training data used. It is therefore important to carry out an adequate exploration of possible values of this hyperparamteter for each experiment that we wish to carry out.

- **Synthetic data (Backtranslations).** Although the main focus in MT lies on obtaining additional 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 2.1.1). A very successful use of monolingual data is the Backtranslation approach of [55], 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.

- **Byte-pair-encoding (BPE).** Byte Pair Encoding (BPE) [56] 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 subword 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 $\boxed{\text{</w>}}$ after every word. The

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

Figure 2.7: 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.

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 2.7 shows an example of BPE segmentation carried out in a sentence 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.

- **Fine-tuning.** NMT models perform best when trained with data from the domain of the test data. However, most available parallel corpora belong to institutional documents or internet-crawled content domains, so it is common to find situations where there is a domain mismatch between train and test data. In such cases, small amounts of in-domain data can be used to improve system performance by carrying out an additional training step, often referred to as the fine-tuning step, using the in-domain data after the main training finishes. This technique has been used to adapt models trained with general domain corpora to specific domains with only small amounts of in-domain data [40, 55].

2.4 Computer tools

This section describes the software used to train the MT systems presented on this work, Sockeye, based on the MXNet [12] library, and fairseq, based on the PyTorch [45] library. Both libraries use the concept of computational graphs and GPU acceleration, introduced in Section 1.2.4, 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. fairseq 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.

Sockeye [22] is an open-source toolkit for NMT, based on the MXNet library. 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. The toolkit implements Attention-RNN, Transformer [62] and Fully Convolutional (ConvSeq2Seq) [18] models. 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. Sockeye even allows the ensembling of models that use different architectures.

fairseq [42] is another open-source NMT toolkit, based on the Pytorch library. Apart from the Attention-RNN, Transformer and ConvSeq2Seq, fairseq also implements the recently proposed LightConv and DynamicConv architectures [63]. Much work has been put into improving the efficiency of the training process, and the toolkit implements techniques such as Half-precision training that significantly reduce model training times [43]. However, compared with Sockeye, the documentation is less detailed, and overall it offers a user experience that is less polished than Sockeye.

2.5 Conclusions

This chapter has described the necessary concepts for the development of the current state-of-the-art NMT systems, by introducing the relevant techniques and architectures (Attention RNN and Transformer) as well as describing the computer tools used to train NMT models. The necessary techniques and practical issues faced when trying to build a state-of-the-art NMT system have also been discussed.

CHAPTER 3

TRANSLATION SYSTEMS FOR WMT18: NEWS TASK

This chapter summarizes the previous work carried out as part of the author's B.S. Thesis [25] for the WMT18 competition, parts of which have been published in [26]. The participation in this task is described in order to put into context the experiments carried out for this Master's Thesis. A series of different systems based on the RNN-Attention and Transformer architectures have been trained for the WMT German→ English task.

The chapter is divided into 2 parts. First, we will introduce the data acquisition and data processing steps required for training any MT system. Then, the experiments carried out with RNN Attention and Transformer will be described.

3.1 Data processing and adquisition

We will first begin by describing 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. We will then move into describing the dataset itself, that has been used for the experiments described in this chapter.

3.1.1 Data preparation

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

Original text: $\boxed{\text{Just}}$ $\boxed{\text{because}}$ $\boxed{\text{he's}}$ $\boxed{\text{wise,}}$ $\boxed{\text{doesn't}}$ $\boxed{\text{mean}}$ $\boxed{\text{that}}$ $\boxed{\text{he's}}$ $\boxed{\text{honest.}}$
Tokenized text: $\boxed{\text{Just}}$ $\boxed{\text{because}}$ $\boxed{\text{he}}$ $\boxed{\text{\&apos;s}}$ $\boxed{\text{wise}}$ $\boxed{\text{,}}$ $\boxed{\text{doesn}}$ $\boxed{\text{\&apos;t}}$ $\boxed{\text{mean}}$ $\boxed{\text{that}}$ $\boxed{\text{he}}$ $\boxed{\text{\&apos;s}}$ $\boxed{\text{honest}}$ $\boxed{\text{.}}$

Figure 3.1: Example of tokenization with Moses.

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 $\boxed{\text{car}}$ and that word followed by a comma $\boxed{\text{car,}}$. 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 $\boxed{\text{car}}$ as a word and $\boxed{\text{,}}$ as another. Figure 3.1 shows an example of tokenization carried out with the Moses tokenizer.

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

3.1.2 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 finetune 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 [11] and [6], 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 $f(.)$.

$$score(x, y) = f(p_s(x), p_t(y)) \tag{3.1}$$

This approach uses language models for estimating the quality of a sentence pair, under the assumption that a low-perplexity sentence is more likely to be an adequate sentence for training. The score (perplexity) of a sentence pair is the geometric mean of the likelihoods, $\sqrt{p_s(x) \cdot p_t(y)}$. We select sentence pairs with the lowest score. 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.

3.1.3 Data collection: WMT2017 German → English

The Workshop on Machine Translation (WMT [1]) 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 3.1 shows basic statistics about the WMT corpus. We have computed the number of sentences, number of words 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 around twice the number of unique English words.

Table 3.1: Statistics of the WMT German-English dataset.

Corpus	Sentences(K)	Words(M)		Vocabulary(K)	
		De	En	De	En
news-commentary-v13	284.2	6.4	6.2	302.8	182.5
europarl-v7	1920.2	44.6	47.9	649.0	304.8
commoncrawl	2399.1	47.0	51.4	2733.7	1718.9
rapid2016	1329.0	22.1	23.0	674.8	387.7
WMT18: paracrawl(v1)	36351.6	450.7	478.8	14054.0	10353.1
dev(newstest2015)	2.2	0.0	0.0	9.9	7.8
test(newstest2017)	3.0	0.1	0.1	12.6	9.2

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

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 includes 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.

The basic statistics of the Paracrawl corpus are included in Table 3.1. 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.

3.2 RNN-Attention systems

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

System description

The WMT dataset was 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 [56], 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. 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 1.2.1. Once the perplexity stopped improving on the development set, the training was stopped and the model was evaluated with respect to the test set. The models finished training once their valuation score did not improve for 10 consecutive checkpoints.

We choose as a starting point a RNN-Attention model with 1 bidirectional layer both in the encoder and decoder, with hidden dimension=1024 and embedding dimension=1024. The type of unit selected for the hidden layers was the LSTM unit. The Adam optimizer [33], which is a modified version of Gradient Descent, and the dropout [59] technique were used for training the model.

Results

In this section we present the experimental results achieved in the WMT17 corpora. Table 3.2 shows the TER and BLEU scores of the different model configurations. 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. 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 [46]. 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 3.5 BLEU over the baseline model, and an increase of 0.8 BLEU over its averaged version. Applying checkpoint averaging on top of this results in a simulated ensemble that obtains an improvement of 1.7 BLEU on the test set.

Additionally we have tried a learning rate reduce scheme (lr_reduce). 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

Table 3.2: Results obtained for models trained with the WMT corpus.

	newstest2017	
RNN-Attention Model	BLEU	TER
Baseline	21.8	68.5
+ checkpoint averaging	24.5	64.3
+ long-sequences	25.3	63.0
+ checkpoint averaging	27.0	60.8
+ lr_reduce	27.5	60.1
+ checkpoint averaging	27.7	59.9

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 [2], 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.

Table 3.3 shows the resources consumed by the models trained using the WMT corpus. At first glance, one can see how the long-sequences model consumes a much higher amount of VRAM. 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. 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 5 hours longer to train than the baseline model. 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 3.2.

3.3 Transformer systems

The Transformer architecture that was described in Section 2.2 currently achieves the best results in MT benchmarks. In order to try to replicate those performance

Table 3.3: Resource consumption for WMT models.

RNN-Attention Model	RAM	Sockeye Training time	Tokens/s
Baseline	3.9 GB	11 h	±5.5K
+ long-sequences	5.2 GB	16 h	±7K
+ lr_reduce	5 GB	30 h	±7K

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.

Results

Table 3.4 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 [62]. 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 3.5 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

Table 3.4: 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 3.5: 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

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.

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 3.1.2, and we have described the Paracrawl corpus included in the WMT18 corpus and the problematic sentences it contains in Section 3.1.3. We will now look at the task of using data filtering in order to take advantage of the Paracrawl corpus to further 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.

Using the previous systems as baselines, we now want to start applying data filtering to the WMT18 Paracrawl corpus and to observe its effects. For carrying out this process, we have chosen the dual language model approach described in Section 3.1.2. We trained a language model both for the source and the target side, using as in-domain data newstest2014 from the WMT competition. The models were built using the SRI Language Modelling Toolkit [60]. 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 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

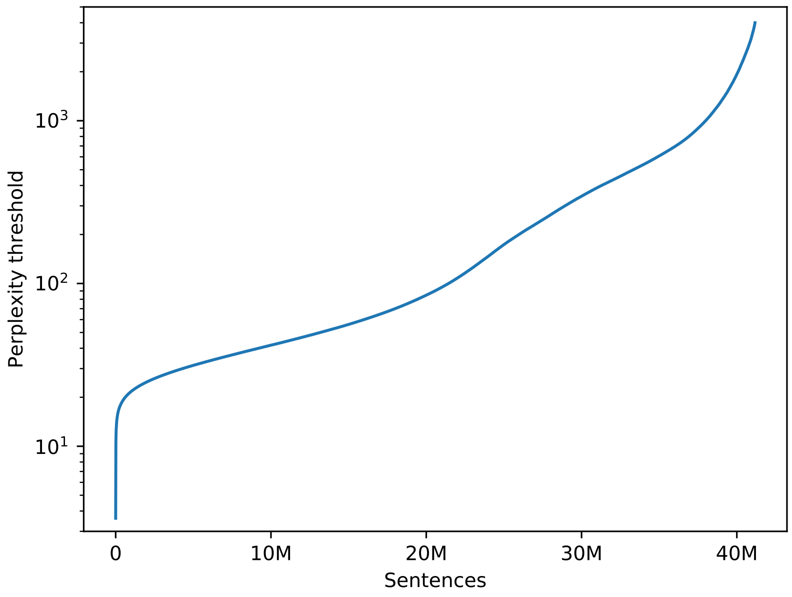


Figure 3.2: Number of sentences with perplexity under a given threshold.

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 3.2 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 3.6. 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 3.1.2. 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 [54] 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

Table 3.6: 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

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 performance 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 are published in an paper titled "The MLLP-Universitat Politècnica de València German-English Machine Translation System for WMT 2018" [26], that was 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 3.7 shows the results obtained by all the primary systems submitted to the WMT18 competition. Our system, tagged

Table 3.7: WMT18 news track results.

System	newstest2018 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

as **MLLP** Trans. ensemble obtained 45.1 BLEU. This is a very competitive result when compared with the systems submitted 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.

3.4 Conclusions

In this chapter, we have carried out experiments using the WMT corpus. The use of data filtering techniques has enabled us to obtain competitive results in the WMT18 News Translation task. These results, as well as the lessons learned, will be used as a starting point in order to carry out the work that is described in the following chapters.

Chapter 4

Translation systems for X5gon

This chapter will describe the work developed for the X5gon project. We will begin by describing the project itself, and we will then move into describing the translation systems developed for the language pairs of the X5gon project, as well as the data used to train them.

4.1 The X5gon 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. The name X5gon comes from the 5 types on solutions offered to OER sites (cross-site, cross-domain, cross-modal, **cross-language** and cross-cultural). Out of the 5 types of solutions proposed in the project, 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. Combined with Automatic Speech Recognition (ASR) technology, MT can be used in order to provide multilingual OER content automatically from their monolingual versions. Apart from the obvious gains in usability, accessibility and target audience that are obtained by having translated versions of OER, automatic translation can be used in order to improve other X5gon services such as the recommender engine that powers the cross-site and cross-lingual components. Therefore, the development of accurate and efficient MT systems is a crucial element for ensuring X5gon's success.

4.2 Translation of OER

This section describes the different MT systems developed for the X5gon project during Y2.

4.2.1 Experimental setup

Our systems architecture is based on the Transformer model described in Section 2.2. In order to train our systems, we have the two previously described configurations, Transformer Base and Transformer Big. The Big configuration has been shown to achieve better results, but it requires more data to properly estimate its parameters, and is harder to train. We have also experimented with training systems with more than 1 GPU.

This section describes the work carried out for the development of NMT systems for the following language pairs of the X5gon project: Italian-English, English-Italian, German-English, English-German, French-English, English-French, Spanish-English and English-Spanish.

For each of those language pairs, we first describe the resources used to train the MT systems. Secondly, we describe the characteristics of the MT systems. Finally, we describe the evaluation process for the developed systems.

4.2.2 German-English

We chose the data published for the News Translation Shared Task of the WMT 2018 competition in order to train the German-English systems. The data has been described in Section 3.1.3.

Following the setup of Section 3.3, the training data was filtered in order to extract 10M sentences from Paracrawl. We also used the 20M backtranslations in order to increase the amount of available training data. We compare the results of the Transformer Base model trained for WMT18 with a second Transformer model that follows the same configuration than the previous system, but trained using 3 GPU, and therefore it uses a batch size that is 3 times bigger than the previous one. Additionally, this second system was trained with longer sentences (maximum sentence length of 100, compared with 75 of the previous one).

In order to evaluate our systems, we have elected to use a set of standard sets from the news translation task of the WMT competition, using newstest2015 as the development set, and newstest2017 as the test set. Table 4.1 shows the results obtained by the German-English MT systems.

Table 4.1: Evaluation results of the German-English MT systems.

System	dev BLEU	test BLEU
Transformer Base	34.3	35.9
Transformer Base, 3 GPU	35.3	36.9

The system trained with 3 GPU obtains an improvement of 1.0 BLEU both in the dev and test sets. This improving can be attributed almost entirely to the increase in batch size. The only other difference between the two systems is the maximum sentence length, and there are almost no sentences in the test set whose length is longer than 75 tokens, so we believe that the effect of this setting will be quite minor.

4.2.3 English-German

The resources chosen to build the English-German system are the same as those of the German-English MT system, as the language pair involved is the same. The data has been described in Section 3.1.3, and we have followed the same setup in this language pair, by selecting 10M sentences using filtering.

We present the results of the two systems developed for English-German, a Transformer Base and a Transformer Base model trained with 3 GPU. The second system is trained with a maximum sentence length of 100. Additionally, the second model was trained by augmenting the 10M parallel sentences with an additional 18M backtranslations. The backtranslations were generated with the **Transformer Base** German-English model of Section 4.2.2.

Table 4.2 shows the results obtained by the English-German MT systems on the news translation WMT dataset.

Table 4.2: Evaluation results of the English-German MT systems.

System	dev BLEU	test BLEU
Transformer Base	29.1	27.4
Transformer Base, 3 GPU + Backtrans.	31.1	29.4

We see how the 3 GPU model is able to obtains improvements of 2.0 BLEU in both dev and test sets, following a similar pattern than the German-English system of Section 4.2.2. The combination of a higher batch size with the additional 18M backtranslations makes it hard to isolate the individual contribution of each change to the overall improvement, but based on previous experience, it is very likely that both changes have significantly contributed to the improvement.

4.2.4 Spanish-English

We will now describe the data used for the Spanish-English language pair. We have 3 distinct type of corpora. The first consists in 6M pseudo in-domain data for the OER domain. This is the data that was used to train our previous phrase-based SMT systems. We also have a series of general domain corpora (commoncrawl, EUbookshop, EU-TT2, eutv and un) as well as a small in-domain corpora from the poliMedia repository. Table 4.3 shows statistics of the different corpora.

We present the results for two Spanish-English systems, a Transformer Base model, and a Transformer Big model trained with 3 GPU. The first system was trained using the 6M pseudo in-domain data. The second system has been trained using all the available data from the general-domain and in-domain corpora, and using a 3 GPU machine.

The Spanish-English systems have also been evaluated using a set of standard test sets from the news translation shared task of the WMT competition, as test sets are also available for this language pair. In this case, we use newstest2012 as development set and newstest2013 as test set. Table 4.4 shows the results of the Spanish-English models.

Table 4.3: Statistics of the data sets used to train the Spanish-English MT systems.

Corpus	Sentences(K)	Words(M)		Vocabulary(K)	
		Es	En	Es	En
pseudo in-domain data	6005.7	144.1	133.4	820.7	756.2
commoncrawl	1845.3	43.5	40.8	1555.2	1371.5
EUbookshop.en-es	5215.5	136.8	121.0	2203.1	2052.7
EU-TT2	1039.9	23.0	21.2	223.7	202.6
eutv	180.5	1.8	1.9	70.5	56.9
un	11196.9	366.1	320.1	668.2	651.7
PM	150.0	2.3	2.4	122.5	88.2

Table 4.4: Evaluation results of the Spanish-English MT systems.

System	dev BLEU	test BLEU
Transformer Base	27.2	25.1
Transformer Big, 3 GPU, Full dataset	34.7	32.3

The Transformer Base model obtains 27.2 BLEU in the dev set and 25.1 BLEU in the test set. The Transformer Big model obtains 34.7 BLEU in the dev set and 32.3 BLEU in the test set, which represents an improvement of 7.5 BLEU and 7.2 BLEU, respectively. When comparing this with the results of Section 4.2.5, it is likely that the big improvement in BLEU is thanks to the additional data, and not to the change from Base to Big model.

4.2.5 English-Spanish

The resources chosen to build the English-Spanish system are the same as those of the Spanish-English MT system, as the language pair involved is the same. The details are shown in Table 4.3.

We present the results for two Transformer Base models. In the same way as the previous case, the first Transformer Base model is trained using a single GPU. The system was trained using the 6M pseudo in-domain data. The second system has been trained using all the available data from the general-domain and in-domain corpora, and using a 3 GPU system. Table 4.5 shows the results obtained by the English-Spanish MT systems.

Table 4.5: Evaluation results of the English-Spanish MT systems.

System	dev BLEU	test BLEU
Transformer Base	26.6	25.1
Transformer Base, 3 GPU, Full dataset	35.0	32.2

Following the trend of the Spanish-English systems of Section 4.2.4, the first Transformer Base obtains 26.6 BLEU in the dev set and 25.1 BLEU in the test set, whereas the 3 GPU model obtains 35.0 and 32.2 BLEU, respectively. This represents an improvement of 8.4 and 7.0 BLEU, respectively.

4.2.6 English-French

The data used for training English-French systems is the WMT14 News Translation Shared Task English-French data. This is a well known dataset that is frequently used in order to compare results in the literature, so selecting it allows us to measure our progress compared with other teams. This datasets contains two medium sized corpora (europarl and commoncrawl) as well as two significantly larger corpora, undoc, which is a collection of UN documents, and the Gigaword corpus, a collection of news text data with more than 20M parallel sentence pairs. Table 4.6 shows the statistics of the WMT14 dataset.

Table 4.6: Statistics of the data sets used to train the English-French MT systems.

Corpus	Sentences(K)	Words(M)		Vocabulary(K)	
		En	Fr	En	Fr
commoncrawl	3244.2	70.7	76.7	1918.2	2081.8
europarl	2007.7	50.3	52.5	311.9	417.8
giga	22520.4	575.8	672.2	7029.5	6899.5
news-commentary	183.8	4.0	4.7	146.4	175.9
undoc	12886.8	316.5	354.2	2079.8	2548.7

We have trained 2 models for this language pair, a Transformer Base and a Transformer Big model using 3 GPUs. All models had a maximum sequence length of 100 tokens.

Following the setup of the WMT14 competition, we have used newstest2013 as the dev set, and newstest2014 as the test set. The results obtained by the English-French MT models are shown in Table 4.7.

Table 4.7: Evaluation results of the English-French MT systems.

System	dev BLEU	test BLEU
Transformer Base	30.9	35.2
Transformer Big, 3 GPU	33.6	37.9

The Transformer Base obtains 30.9 BLEU in the dev set, and 35.2 BLEU in the test set, whereas the Big model obtains 33.6 and 37.9 BLEU, respectively. This represents an increase of 2.7 BLEU in both the dev and the test sets.

4.2.7 French-English

The resources chosen to build the French-English system are the same as those of the English-French MT system, as the language pair involved is the same. The details are shown in Table 4.6.

We have trained a Transformer Base as well as a Transformer Big model, this one trained using 3 GPUs. The models were trained with a maximum sequence length of 75.

In a similar way to the English-French case, we used newstest2013 as dev set, and newstest2014 as test set. Table 4.8 shows the results obtained by the French-English systems.

Table 4.8: Evaluation results of the French-English MT systems.

System	dev BLEU	test BLEU
Transformer Base	33.1	36.8
Transformer Base, 3 GPU	33.0	36.8

In this case, both systems show similar performance, with 33.1 and 36.8 BLEU in the dev and test sets. This result is different from other language pairs, where an increase in batch size also meant an improvement in translation quality. Further experiments using the Transformer Big configuration as well as bigger batches could be a way of improving results.

4.2.8 English-Italian

For the English-Italian systems, we have collected a series of public datasets from a variety of domains such as: medical (EMEA) and institutional documents (ECB,Europarl and JRC-Aqcquis), book translations (EUbookshop) and Wikipedia. The statistics of these datasets are shown in Table 4.9.

Table 4.9: Statistics of the data sets used to train the English-Italian MT systems.

Corpus	Sentences(K)	Words(M)		Vocabulary(K)	
		En	It	En	It
ECB	193.0	5.5	5.8	62.2	77.7
EMEA	1081.1	12.1	13.4	130.3	153.7
EUbookshop	6490.0	144.6	147.4	2332.6	2515.8
Europarl	1944.9	50.7	49.0	380.1	492.3
JRC-Acquis	811.0	15.5	15.4	217.6	248.4
Wikipedia	957.0	20.6	19.2	1530.0	1520.1

We have trained two Transformer Base models, one trained with 1 GPU and the other with 3 GPU, with a maximum sequence length of 100.

The WMT competition has not been held for the English-Italian pair. As such, we must look elsewhere to find reliable test sets. In this case, we have used the tests sets from IWSLT17 [10], another international MT competition. We used the provided dev and test sets for English-Italian. Table 4.10 shows the results obtained by the English-Italian MT systems.

Table 4.10: Evaluation results of the English-Italian MT systems.

System	dev BLEU	test BLEU
Transformer Base	21.4	21.4
Transformer Base, 3 GPU	23.7	23.3

The 1 GPU Transformer Base models obtains 21.4 BLEU in both the dev and the test set. The 3 GPU Transformer obtains an improvement of 2.3 and 1.9 BLEU, respectively. As the only difference between these two models is the batch size, this results prove that the choice of batch size is critical for Transformer models.

4.2.9 Italian-English

The resources chosen to build the Italian-English system are the same as those of the English-Italian MT system, as the language pair involved is the same. The details are shown in Table 4.9.

Following the English-Italian setup, we train both a Transformer Base with 1 GPU and another one with 3 GPU. Table 4.11 shows the results obtained by the Italian-English MT systems.

Table 4.11: Evaluation results of the Italian-English MT systems.

System	dev BLEU	test BLEU
Transformer Base	25.1	25.4
Transformer Base, 3 GPU	25.1	25.9

The Transformer Base achieves 25.1 BLEU in the dev set, and 25.4 BLEU in the test set, and the 3 GPU version improves 0.5 BLEU in the test set. Although not as significant as in the English-Italian case, we also observe performance differences due to different batch sizes.

4.3 Conclusions

We have described the development of MT systems for the X5gon project. Using the Transformer NMT architecture, we have obtained excellent results in all translation pairs, without requiring any specific model adaptation to each pair, showing the flexibility and power of modern NMT approaches. These results are not without caveats. When using the Transformer Big architecture, we have observed that the results are very dependent on the batch size. In fact, there are cases where a model trained using the Base configuration obtains better results than the Big counterpart. We must carry out additional experiments in order to explore the effects of the batch size on system performance.

Chapter 5

Translation systems for WMT19: Similar Task

5.1 Introduction

In this chapter we describe the NMT systems developed for the Related Languages Translation Shared Task of the *ACL 2019 Fourth Conference on Machine Translation* (WMT19). For this task, we participated in both directions of the Portuguese $\leftrightarrow$ Spanish language pair. This chapter introduces a novel NMT model that is currently being developed. We report results for this approach and compare them with models based on the well-performing Transformer NMT architecture. A domain adapted version of this latter system achieves the best results out of all submitted systems on both directions of the shared task.

The chapter is organized as follows. Section 5.2 describes the architecture and settings of the novel 2D RNN model. Section 5.3 describes our baseline systems and the results obtained. Section 5.4 reports the results obtained by means of the fine-tuning technique. Section 5.5 reports comparative results with respect to the systems submitted by the other competition participants. Section 5.6 outlines our conclusions for this shared task.

5.2 2D Alternating RNN

In this section, we will describe the general architecture of the 2D alternating RNN model. The 2D alternating RNN is a novel translation architecture in development by the MLLP group. This architecture approaches the machine translation problem with a two-dimensional view, much in the same manner as [32, 3] and [17]. This view is based on the premise that translation is fundamentally a two-dimensional problem, where each word of the target sentence can be explained in some way by all the words in the source sentence. Two-dimensional translation models define the distribution $p(e_i|f_0^J, e_0^{i-1})$ by jointly encoding the source sentence (f_0^J) and the target history

(e_0^{i-1}), whereas the usual translation models encode them separately, in separate components usually called "encoder" and "decoder".

The proposed architecture is depicted in Figure 5.1. It defines a two-dimensional translation model by leveraging already known recurrent cells, such as LSTMs or GRU, without any further modification.

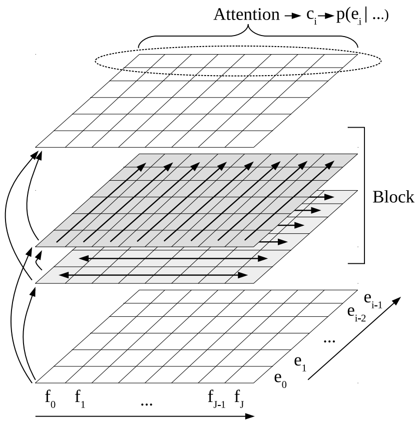

Attention → c_i → $p(e_i | ...)$
Block
e_{i-1}
e_{i-2}
...
e_1
e_0
f_0 f_1 ... f_{J-1} f_J

Figure 5.1: The 2D alternating RNN architecture. White grids on the top and bottom represent the input/output of a block. Arrows in grey grids represent the RNNs, while the arrows on the left depict how the layers are interconnected. Arrows on the bottom and bottom right indicate the source and target dimensions.

As many other translation models, we have a context vector which is projected to vocabulary size and a softmax (σ) is applied to obtain the probability distribution of the next word at timestep i:

$$p(e_i = x | f_0^J, e_0^{i-1}) = \sigma(Wc_i)_x \tag{5.1}$$

To explain how this context vector is drawn from a two-dimensional processing style, we need to define a grid with two dimensions: one for the source, and one for the target. From this point, we will define a layer-like structure called block, where each block of the model has such a grid as the input, and another one as the output.

The first grid that serves as input to this two-dimensional architecture has each cell s_{ij}^0 containing the concatenation of the source embedding in position j and the target embedding in position $i-1$:

$$s_{ij}^0 = \begin{bmatrix} f_j \\ e_{i-1} \end{bmatrix} \tag{5.2}$$

Each block of the model has two recurrent cells: one along the source dimension and another one along the target dimension. They process each row or column independently of one another. The horizontal cell is bidirectional and receives the grid s^l as its input:

$$h^l_{ij} = \begin{bmatrix} \mathrm{RNN_{h1}}(h^l_{i,j-1}, s^{l-1}_{ij}) \\ \mathrm{RNN_{h2}}(h^l_{i,j+1}, s^{l-1}_{ij}) \end{bmatrix} \tag{5.3}$$

The vertical cell receives the concatenation of h^l and s^l:

$$k^l_{ij} = \mathrm{RNN_k}(k^l_{i-1,j}, [s^{l-1}_{ij}; h^l_{ij}]) \tag{5.4}$$

The output of the block is the concatenation of the output of both cells:

$$s^l_{ij} = \begin{bmatrix} h^l_{ij} \\ k^l_{ij} \end{bmatrix} \tag{5.5}$$

From the output of the last block, s^L, we generate a context vector as follows:

$$c_i = \mathrm{Attention}([\mathrm{s^L_{i0}}, \ldots, \mathrm{s^L_{iJ}}]) \tag{5.6}$$

The Attention function extracts a single vector from a set of vectors leveraging an attention mechanism. That is, it scores the vectors according to a learned linear scoring function, which is followed by a softmax to extract scores; and with those scores it performs a weighted sum to obtain a context vector.

5.3 Baseline systems

This section describes training corpora as well as the baseline model architectures and configurations adopted to train our NMT systems. As said in Section 5.1, two different model architectures were trained: the Transformer architecture and our proposed 2D alternating RNN architecture. BLEU scores were computed with the `multi-bleu` utility from Moses.

5.3.1 Corpus description and data preparation

The training data is made up of the JCR, Europarl, news-commentary and wikititles corpora. Table 5.1 shows the number of sentences, number of words and vocabulary size of each corpus. The provided development data was split equally in two disjoint sets, and one was used as development set and the other as test set.

The data was processed using the standard Moses pipeline, specifically, punctuation normalization, tokenization and truecasing. Then, we applied 32K BPE operations. We included in the vocabulary only those tokens occurring at least 10 times in the training data.

5.3.2 Transformer baseline models

For the Transformer models, we used the “Base” configuration (512 model size, 2048 feed-forward size), trained on one GPU. The batch size was 4000 tokens, and we carried out gradient accumulation by temporarily storing gradients and updating the weights every 4 batches. This setup allowed us to train models using an effective batch

Table 5.1: Statistics of the data sets used to train the Spanish $\leftrightarrow$ Portuguese MT systems.

Corpus	Sent.(K)	Words(M)		Vocab.(K)	
		Es	Pt	Es	Pt
JCR	1650	42	40	264	264
Europarl	1812	53	52	177	156
news	48	1	1	49	47
wikititles	621	1	1	292	295
Total	4131	98	96	623	604

size of 16000 tokens. We used dropout with 0.1 probability of dropping, and label smoothing where we distribute 0.1 of the probability among the target vocabulary. We stored a checkpoint every 10000 updates, and for inference we used the average of the last 8 checkpoints.

We used the Adam optimizer [33] with $\beta_1 = 0.9$, $\beta_2 = 0.98$. The learning rate was updated following an inverse square-root schedule, with an initial learning rate of $5 \cdot 10^{-4}$ and 4000 warm-up updates.

The models were built using the fairseq toolkit.

5.3.3 2D alternating RNN baseline model

For the 2D alternating RNN models, we used GRU as the recurrent cell, 256 for the embedding size and 128 as the number of units of each layer of the block. The model consisted of a single block. The batch size was 20 sentences, with a maximum length of 75 subword units.

We used the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$. The learning rate was initialized at 10^{-3} and kept constant, but halved after 3 checkpoints without improving the development perplexity. A checkpoint was saved every 5000 updates. The model was built using our own toolkit. Due to time constraints, the 2D alternating model was only trained for the Portuguese $\rightarrow$ Spanish direction.

5.3.4 Results

Table 5.2 shows the evaluation results for the Portuguese$\rightarrow$Spanish systems, and Table 5.3 shows the evaluation results for our Spanish$\rightarrow$Portuguese Transformer system. For the Portuguese $\rightarrow$ Spanish direction, the Transformer model obtains 57.4 BLEU in the test set, and 51.9 in the hidden test set of the competition. The 2D alternating model achieves 55.1 and 49.7 BLEU, respectively. These results show how, even though it is in early stages of development, the 2D alternating RNN model is able to obtain competitive results for this task that are not very far from those obtained by the state-of-the-art Transformer architectures. It is worth noting that this has been achieved with a model that has significantly less parameters than its Transformer counterpart.

Table 5.2: Baseline BLEU scores on the Portuguese → Spanish task.

	BLEU	
System	test	test-hidden
Transformer	57.4	51.9
2D altern. RNN	55.1	49.7

Table 5.3: Baseline BLEU scores on the Spanish → Portuguese task.

	BLEU	
System	test	test-hidden
Transformer	51.2	45.5

5.4 Fine-tuning

Section 2.3 has highlighted the importance of carrying out fine-tuning in case there exits a domain mismatch between train and test data. In order to empirically test if this is one of such cases, we have trained two language models, one using only the presumably out-of-domain data (the train corpora from Table 5.1), and one using only the in-domain development data. The models were 4-gram language models trained using the SRI Language Modelling Toolkit. We then computed the perplexity of the test set using these two language models. The model that was trained with the out-of-domain data obtains a perplexity of 298.0, whereas the model that used the in-domain data obtains a perplexity of 81.9. This result shows that there is in a fact a domain mismatch between the train and test data, which supports the idea of carrying out fine-tuning.

We applied this to both translation directions, using the first part of the development data as in-domain training data, and the second part as a new dev set. One checkpoint was stored after every fine-tuning epoch, and we monitored model performance on the new dev set in order to stop fine-tuning once the BLEU results started decreasing. For the Transformer models, we used the same learning rate as when training stopped, while for the 2D alternating models we used 10^{-3}.

Tables 5.4 and 5.5 compare the BLEU scores achieved by the fine-tuned systems with that of the baseline non fine-tuned ones on the Portuguese→Spanish and Spanish→Portuguese tasks, respectively.

Table 5.4 shows that for this particular task, fine-tuning is a key step for achieving very substantial performance gains: in the Portuguese→Spanish task, we obtained a 14.1 BLEU improvement in the test set and a 14.7 BLEU improvement in the hidden test set for the Transformer model. The 2D alternating RNN obtained a 8.9 BLEU improvement thanks to fine-tuning. This also applies to the Spanish→Portuguese task, shown in Table 5.5: we obtained a 18.1 BLEU improvement in the test set, and a 19.2 BLEU improvement in the hidden test set after applying fine-tuning.

In order to understand the impact and behaviour of the fine-tuning process, we have analyzed the model's performance as a function of the number of fine-tuning

Table 5.4: Comparative BLEU scores of the Transformer and 2D alternating RNN models on the Portuguese → Spanish task.

System	BLEU test	BLEU test-hidden
Transformer	57.4	51.9
+ fine-tuned	72.4	66.6
2D altern. RNN	55.1	49.7
+ fine-tuned	64.0	-

Table 5.5: Comparative BLEU scores of the Transformer model on the Spanish → Portuguese task.

System	BLEU test	BLEU test-hidden
Transformer	51.2	45.5
+ fine-tuned	70.7	64.7

epochs. Figure 5.2 shows the impact of the fine-tuning step for the Transformer and 2D alternating RNN models on the Portuguese → Spanish task, while Figure 5.3 shows the results of the fine-tuning step applied to the Transformer model on the Spanish → Portuguese task. In both language pairs, the first epochs are the most beneficial for system performance, and additional fine-tuning epochs bring diminishing returns until the BLEU curve flattens.

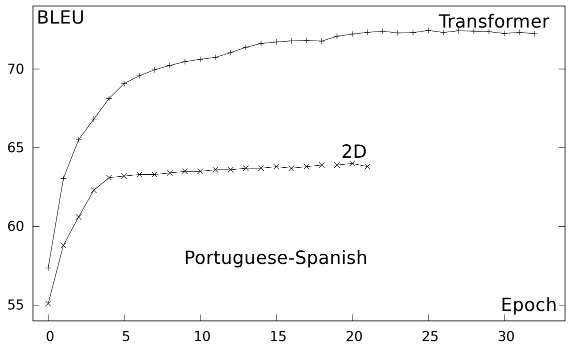


Figure 5.2: BLEU scores as a function of the number of fine-tuning epochs on the Transformer and 2D alternating RNN models for the Portuguese→Spanish task.

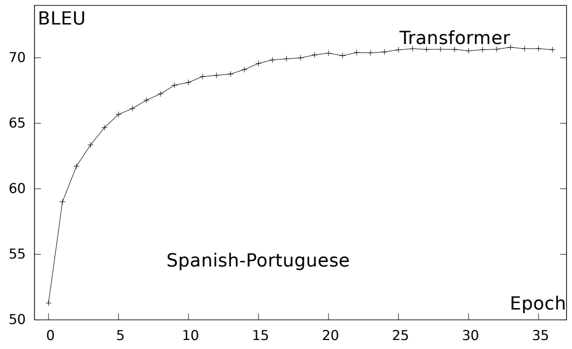


Figure 5.3: BLEU scores as a function of the number of fine-tuning epochs on the Transformer model for the Spanish→Portuguese task.

Table 5.6: Primary submission results of the Portuguese → Spanish shared task in the hidden test set.

Team	BLEU	TER
MLLP	**66.6**	**19.7**
NICT	59.9	25.3
U. Helsinki	58.4	25.3
Kyoto U.	56.9	26.9
BSC	54.8	29.8
UBC-NLP	52.3	32.9

5.5 Comparative results

We now move on to the results for the primary submissions of all participants in the Shared Task. We chose to send our fine-tuned Transfomer systems as primary submissions to both tasks after reviewing the results on the provided test set (Section 5.4). The submission was made with the checkpoint that achieved the best performance on the fine-tuning dev data. Table 5.6 shows the results of the Portuguese→Spanish task, while Table 5.7 shows the results of the Spanish→Portuguese task; both in BLEU and TER [58].

In both tasks, our system outperformed all other participants by a significant margin. In the Portuguese→Spanish task, our submission outperforms the next best system by 6.7 BLEU and 5.6 TER. In a similar manner, our submission to the Spanish → Portuguese task improves the results of the second-best submission by 2.6 BLEU and 2.2 TER points. We attribute our success to the domain adaptation carried out by means of the fine-tuning technique. We have been able to apply this technique by using part of the competition's development data as in-domain training data.

Table 5.7: Primary submission results of the Spanish → Portuguese shared task in the hidden test set.

Team	BLEU	TER
MLLP	**64.7**	**20.8**
UPC-TALP	62.1	23.0
NICT	53.3	29.1
U. Helsinki	52.0	29.4
UBC-NLP	46.1	36.0
BSC	44.0	37.5

5.6 Conclusions

We have taken on the similar language task with the same approaches that we found useful for other kinds of translation tasks. NMT models, specifically the Transformer architecture, fare well in this task without making any specific adaptation to the similar-language setting. In fact, we achieved the best results among the participants using a general domain-adaptation approach.

For this particular task, the use of in-domain data to carry out fine-tuning has allowed us to obtain remarkable results that significantly outperform the next best systems in both Portuguese→Spanish and Spanish→Portuguese. We believe these results are explained by the domain difference between training and test data, and are unrelated to the similarity between Spanish and Portuguese.

We have introduced the 2D alternating RNN model, a novel NMT architecture, that has been tested in the Portuguese→Spanish task. With small embedding and hidden unit sizes and a shallow architecture, we achieved similar performance to the Transformer model, although the difference between them increases after applying fine-tuning.

In terms of future work, we plan to fully develop the 2D alternating RNN model in order to support larger embedding and hidden unit sizes as well as deeper architectures using more regularization. All these improvements should allow us to increase the already good results achieved by this model.

Chapter 6

Translation systems for WMT19: News Task

6.1 Introduction

In this chapter we describe the NMT systems for the News Translation Shared Task of the *ACL 2019 Fourth Conference on Machine Translation* (WMT19). For this year's edition, we participated in both directions of the German $\leftrightarrow$ English and German $\leftrightarrow$ French language pairs, using the Transformer architecture. Following the lessons learned from WMT18, we have continued working on data filtering, and we have experimented with additional synthetic data techniques and bigger neural network architectures trained with multi-GPU machines.

This chapter is organized as follows. Section 6.2 describes the data processing steps (including data filtering and synthetic data generation) carried out prior to system training. Section 6.3 describes the architecture and settings used for our NMT models, and the different experiments and evaluations performed are detailed in Section 6.4. Our conclusions for this shared task are outlined in Section 6.5.

6.2 Data preparation

Data preprocessing, corpus filtering and data augmentation are described in the following sections.

6.2.1 Corpus preprocessing

Following the configuration described in Section 5.3.1, the data was processed using the standard Moses pipeline (normalization, tokenization and truecasing). Additionally, we applied 40K BPE operations, and excluded from the vocabulary all subwords that did not appear at least 10 times in the training data. BPE operations are learned before adding the data extracted using corpus filtering, described in Section 6.2.2. Sentences longer than 100 subwords were excluded from the training data.

6.2.2 Corpus filtering

As was discussed in Section 3.1.2, the addition of the ParaCrawl corpus to the WMT shared tasks has placed an increasing importance in filtering and data selection techniques in order to take advantage of this additional data. This is highlighted by the fact that a majority of participating systems in the WMT18 News Translation Task [7] apply filtering techniques to ParaCrawl. Additionally, the experiments carried out for our 2018 submission [26], described in Section 3.3, show that using a noisy corpus such as ParaCrawl without filtering can result in a worse performance compared with a baseline system that simply excludes the noisy corpus from the training data.

We have compared two different approaches to corpus filtering:

- **LM-based filtering**: This is the approach we used for our WMT18 submission, and has been described in detail in Section 3.1.2.

- **Dual Conditional Cross-Entropy filtering** [30]: This approach computes the sentence pair score by means of a product of a series of partial scores.

$$f(x,y) = \prod_i f_i(x,y) \tag{6.1}$$

  We have used the same configuration sent for the WMT18-filtering task, which uses 3 partial scores: a language identification score (*lang*), a dual conditional cross-entropy score (*adq*), and a cross-entropy difference score (*dom*) with a cut-off value of 0.25. The full details of each of these partial scores is given in [30]. The translation models for the *adq* score are Transformer Base models trained with the Europarl portion of WMT19. In terms of the data for the *dom* score, we randomly sampled 1M sentences from NewsCrawl 2016 as in-domain data, and 1M sentences from ParaCrawl as out-of-domain data.

6.3 System description

This section describes the configuration and decisions adopted for training our NMT systems. We will first begin by describing the details that are common to all systems, and we will then move on to specific details for each of the considered translation directions.

Our models follow the Transformer architecture [62], and are configured based on the Transformer Base and Transformer Big settings.

The Transformer Base models are trained with a batch size of 3000 tokens per GPU, whereas the Transformer Big models use a batch size of 2300 tokens per GPU. We store a checkpoint every 10 000 updates, and inference is carried out by averaging the last 8 checkpoints.

We used the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$. The learning rate was updated following an inverse square-root schedule, with an initial learning rate of 0.0005, and 4000 warm-up updates. All models use 0.1 label smoothing and 0.1 dropout, with the exception of the German $\leftrightarrow$ French models, that use 0.3 dropout due to having less training data.

The systems from our WMT18 submission (described in described in Section 3.3) and this year's baseline systems were built using the Sockeye toolkit. The rest of the systems were built using the fairseq toolkit, in order to train using Half Precision and gradient accumulation like in [43].

We have produced backtranslations (See Section 2.3) for all the language pairs we have participated in, using the following configuration:

- German $\rightarrow$ English: We have used 20M sentences from our WMT18 submission [26], and an additional 24M sentences generated using a system with the same configuration as WMT18, but trained with 3 GPUs instead of 1. The monolingual sentences were randomly sampled from News Crawl 2017.
- English $\rightarrow$ German: We have generated 18M sentences using our German $\rightarrow$ English system submitted to WMT18, with monolingual sentences randomly sampled from News Crawl 2017.
- German $\rightarrow$ French: We have generated 10M synthetic sentences, using the reverse direction baseline system described in Section 6.3. The monolingual sentences were sampled from News Crawl 2015-2018.
- French $\rightarrow$ German: We have generated 18M synthetic sentences, using the reverse direction baseline system described in Section 6.3. The monolingual sentences were sampled from News Crawl 2017.

Prior to selecting sentences, we filtered out from the German News Crawl 2017 all sentences that were written in a language different from German, using the `langid` tool [39]. When combining bilingual and synthetic data, the original bilingual data was upsampled in order to achieve a 1:1 ratio.

6.4 Experimental evaluation

This section describes the experiments and evaluation carried out for each of the language directions, with special emphasis placed in the German $\leftrightarrow$ English systems.

For the German $\leftrightarrow$ English systems, we have used newstest2017 as dev set, and newstest2018 as test set. Additionally, we report results on this year's test set, newstest 2019. For the German $\leftrightarrow$ French systems, we splitted in half the supplied euelections dev set into two sets, dev 1 and dev 2, and used the former as dev set and the latter as test set. We also report the results obtained in the official test set newstest2019. We report BLEU scores computed using SacreBLEU.

As the use of fine-tuning achieved significant improvements in the WMT19 Similar Task, we have also elected to carry out fine-tuning in this task. For the finetuning step,

Table 6.1: Filtering techniques comparison for WMT18 German → English

	BLEU		
Filtering technique	nt2015	nt2017	nt2018
LM-based	31.2	32.3	40.5
Cross-Entropy	32.2	33.3	41.3

we set the learning rate to the value that was being used when training finished, and we reduced the checkpoint interval in order to store a checkpoint every 20 updates. Finetuning continues as long as the performance does not decrease in the appropriate dev set. For the German ↔ English systems, we follow the setup of [53], and use test sets from previous years (newstest08-16) as training data for the finetuning step. Since this is the first time the German ↔ French language pair is included in WMT, we do not have available test sets from previous editions, so we resort to using the dev1 set as training data, and stop finetuning when performance drops in dev2.

6.4.1 German → English

In order to test the performance of the two described filtering techniques, we trained a Transformer Base model with the WMT18 German → English bilingual data (excluding ParaCrawl), as well as 10M sentence pairs extracted from ParaCrawl using each technique. The models were trained on a single GPU with a batch size of 3000 tokens, with the same hyperparameters as in [26]. The results are shown in Table 6.1.

The Cross-Entropy model obtains better results than the LM model, with an improvement of 1.0 BLEU in newstest2017, and an improvement of 0.8 BLEU in newstest2018. This is consistent with the fact that the Cross-Entropy filtering was the winning submission to the WMT18 Shared Task on Parallel Corpus Filtering [36]. As a result, we have elected to use the Cross-Entropy filtering method for filtering the different versions of the ParaCrawl corpus present in all language pairs.

Table 6.2 shows the results obtained by our systems trained for the German → English direction. As baselines, we take our WMT18 system, trained with 1 GPU (this is the configuration that was used for our WMT18 submission), and the same setup trained with 3 GPUs. The increase in effective batch size from 3000 to 9000 tokens results in an improvement of 1.7 BLEU in newstest2018 and 2.0 BLEU in newstest2019 without any other change in hyperparameters.

We began our WMT19 experiments by building a system following the Transformer Big architecture, trained in a 4-GPU machine and using the 20M backtranslations produced for WMT18 as well as 10M filtered sentences from ParaCrawl. This results in an increase of 0.3 BLEU in newstest2018 and 0.6 BLEU in newstest2019. We then applied gradient accumulation by setting the Update Frequency (UF) to 2. Under this setting, the model's weights are updated every two steps (this simulates a batch size equivalent to training on 8 GPUs). This model obtains a significant improvement in the dev (+0.7 BLEU), and test sets (+1.4 BLEU), however the performance decreases by 0.7 BLEU when evaluating on newstest2019. We have found

Table 6.2: Evaluation results of German → English systems

System	GPUs	BLEU newstest2018	BLEU newstest2019
WMT18 (Transformer Base)	1	44.2	35.6
WMT18 (Transformer Base)	3	45.9	37.6
Transformer Big, 20M backtrans	4	46.2	38.3
+ UF=2	4	47.6	37.7
+ finetuned	4	47.8	39.4
+ 24M backtrans, noise	4	48.0	40.2
+ finetuned	4	47.9	40.1

no explanation for this phenomenon. Finetuning on the news in-domain data results improves all previous results, resulting in 47.8 BLEU in newstest2018 and 39.4 BLEU in newstest2019.

For our final submission, we trained a system with noisy backtranslations, following the work of [16]. We used the previous 20M backtranslations and appended an additional 24M generated with the system in row 2 of Table 6.2. We added noise to the source side of the synthetic sentence pairs using the technique described by [38]. Following the setup of [16], bilingual data was not upsampled, resulting in a ratio of around 1:3 original to synthetic sentences. The system trained with noisy backtranslation obtains 48.0 BLEU in newstest2018 and 40.2 BLEU in newstest2019. An additional finetuning step , however, obtains a decrease of 0.1 BLEU in both newstest2018 and newstest2019.

We observe that, in the case of the noisy system, finetuning seems to obtain mixed results, in contrast with other trained systems and language directions (see Sections 6.4.2, 6.4.3 and 6.4.4 for the other WMT19 News systems, Section 5.4 for the WMT19 Similar systems), where finetuning achieves a performance increase in all cases. We theorize this could be due to the fact that the system was first trained with a ratio that included 3 times as many noisy sentences as clean data, but the finetuning was carried out only with clean data, without any added noise.

6.4.2 English → German

Table 6.3 shows the results obtained by our systems trained for the English → German direction. We began with a baseline system trained using our WMT18 configuration and data, plus an additional 18M backtranslations. This system obtains 45.2 BLEU in newstest2018 and 39.3 BLEU in newstest2019. For our WMT19 submission, we trained a Transformer Big model, using the WMT19 data (including 10M filtered sentences from ParaCrawl), as well as the already mentioned 18M backtranslations. This system was trained with 2 GPUs and an Update Frequency of 2, giving us an effective batch size equivalent to 4 GPUs. This system obtains an improvement of 0.4 BLEU in newstest2018 and 0.1 BLEU in newstest2019 over the baseline. Increasing the number of GPUs from 2 to 4 shows no significant differences in either newstest2018

Table 6.3: Evaluation results of English → German systems

System	GPUs	BLEU newstest2018	BLEU newstest2019
WMT18 (Transformer Base), 18M backtrans	3	45.2	39.3
Transformer Big, 18M backtrans, UF=2	2	45.6	39.4
+ GPU=4	4	45.7	39.4
+ finetuned	4	48.1	41.7

Table 6.4: Evaluation results of German → French systems

System	GPUs	BLEU dev2	BLEU nt2019
WMT19 - {ParaCrawl}	1	31.1	32.1
Transformer Big, UF=2	2	33.3	34.4
+ finetuning	2	33.5	34.5

or newstest2019. Our final submission was generated after applying a finetuning step to the previous configuration. This finetuning resulted in an increase of 2.4 BLEU in newstest2018 and 2.3 BLEU in newstest2019 when compared with the non-finetuned model.

6.4.3 German → French

Table 6.4 shows the results obtained by our systems trained for the German → French direction. Our baseline system is a Transformer Base model trained with all the WMT19 data excluding ParaCrawl. This system obtains 31.3 BLEU in dev2 and 32.1 BLEU in newstest2019. We then moved on to training a Transformer Big model, adding 1M sentences filtered from ParaCrawl, and 10M backtranslations generated with the French → German baseline system. This system was trained with 2 GPUs and an Update Frequency of 2. This results in an increase of 2.2 BLEU in dev2 and 2.3 BLEU in newstest2019. An additional finetuning step, carried out using the dev1 data, results in an increase of 0.2 BLEU in dev2 and 0.1 BLEU in newstest2019, and constituted our submission to the competition.

6.4.4 French → German

Table 6.5 shows the results obtained by our systems trained for the French → German direction. The approach and configurations for this language directions mirror those of the German → French direction (Section 6.4.3). We began with a baseline Transformer Base model, that obtains 22.8 BLEU in dev2 and 25.7 BLEU in newstest2019. The Transformer Big model obtains an improvement of 2.1 BLEU in dev2 and 1.2 BLEU in newstest2019, and the finetuning step results in an additional increase of 0.5 BLEU in dev2 and 0.6 BLEU in newstest2019.

Table 6.5: Evaluation results of French → German systems

		BLEU	
System	GPUs	dev2	nt2019
WMT19 - {ParaCrawl}	1	22.8	25.7
Transformer Big, UF=2	2	24.9	26.9
+ finetuning	2	25.4	27.5

6.5 Conclusions

The experiments carried for WMT19 have allowed us to explore one of the missing pieces of our WMT18 submission, which is the interaction between the Transformer architecture and different batch sizes. The results show that the performance of models following the Transformer architecture is highly dependent on the batch size used to train the model, requiring multiple GPUs or gradient accumulation in order to fully take advantage of this architecture. This result is consistent with other works such as [47].

As future work, we would like to look further into using massive amounts of synthetic data jointly with noise, as our experiments this year have not provided conclusive results. Overall, the finetuning steps looks like an effective way of obtaining translation improvements, at the expense of only a small amount of computation. This domain adaptation step can be carried out as long as we have some amount of in-domain data available. More work needs to be carried out to explore the interaction between finetuning and adding noise to the data. Another avenue for improvement is to look into the optimal amount of filtered data to extract from ParaCrawl, as well as the upsampling ratio to mix bilingual and synthetic data. These aspects were not explored in our WMT19 submission due to time constraints.

Chapter 7

System Integration and Online MT

This chapter describes the development and integration of MT systems, with a special emphasis in the online MT case. We will first make a distinction between offline and online systems. An offline system begins processing a request once it has all the input data available. Usually, the requests sent to a system are organized and processed in such a way that we maximize throughput (for example, through the use of batch decoding) and minimize expenses of computational resources. Additionally, offline systems might offer other features that require additional time before the content can be returned to the user. In contrast, online systems refer to systems that start processing the data as soon as it arrives, and it is understood that such systems must work in a real-time fashion, with negligible delays that allow for a live usage of the system.[1]

7.1 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. Now, we are interested in integrating those systems 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[2] is an active service with many users, and currently offers audio transcription services for 10 different languages, and translation services between 16

[1]In fact, online systems are also sometimes called real-time or live systems, without much distinction among the 3 terms.

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

language pairs. An example of the TTP dashboard with a collection of transcribed and translated videos is shown in Figure 7.1. 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.

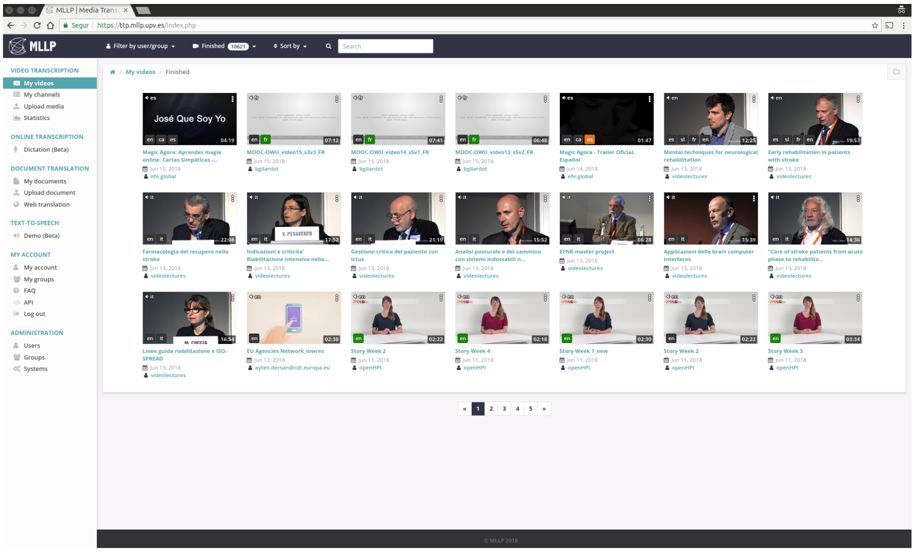


Figure 7.1: Example of some of the videos uploaded to TTP.

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.

7.1.1 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 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 parallelism 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

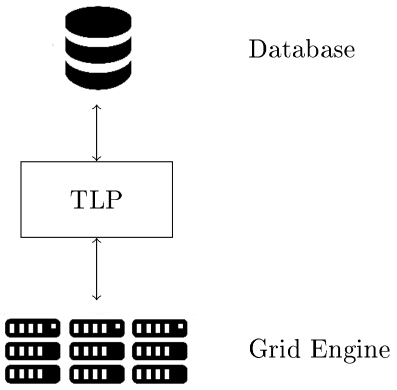


Figure 7.2: General TLP architecture.

overview that we have described can be seen in Figure 7.2.

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 7.3 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 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 MT systems will be carried out by defining new translation modules that can be selected when a translation involving that language pair is requested.

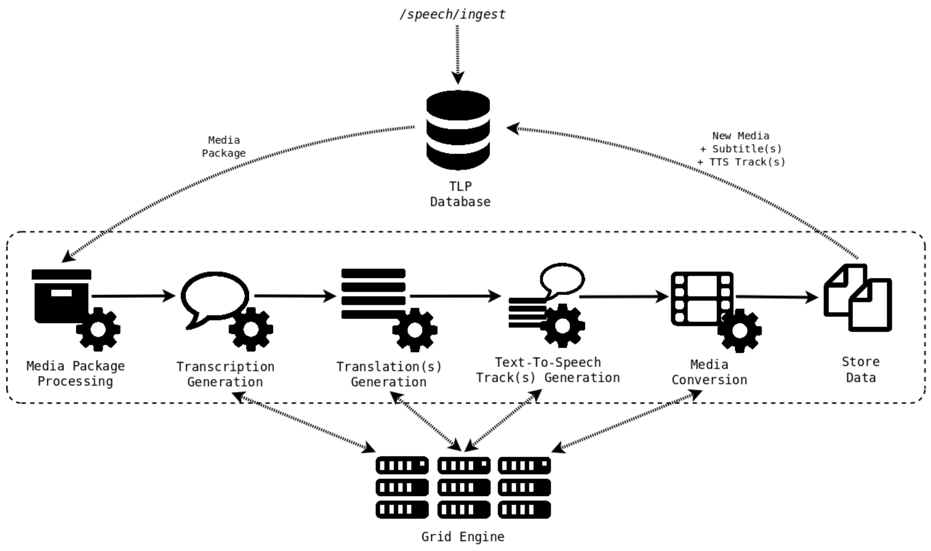


Figure 7.3: TLP Workflow.

7.1.2 Translation generation

Now that we have an overview of the TLP software, we can focus our attention in the developed translation modules. 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. 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 optimal 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 be pretty much 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 by the system.

7.2 Development of Online MT systems

7.2.1 Motivation

We have described the integration process carried out to include the developed systems in the TTP environment. As such, the previously described schema is optimized for maximizing throughout instead of single-sentence translation latency. While the offline approach is adequate for many tasks that do not have strict time requirements, there exists cases where an online system is needed.

The MLLP research group develops MT, ASR and Text-to-Spech (TTS) systems with the goal of being able to produce educational content that is automatically available in different languages than the one they were recorded on. Recently, the MLLP group has made advances in one-pass decoding for ASR [29], allowing the development of online ASR systems of exceptional quality. As such, the development of online MT systems and integration with the ASR step, would allow for real-time multilingual subtitling of any arbitrary video stream.

We will now described the steps carried out in order to build an online MT system.

7.2.2 Architecture description

In the same way as the previously described offline case, when deploying a system in an online way, we are concerned with the technical challenges involved in integrating our model into a service that fulfils the requirements of our use case in terms of efficiency, reliability, etc. There is significant research interest in coming up with specific models that exhibit features that make them more suitable for online deployments. However, when carrying out this deployment, the underlying model and its computational graph and weights do not change. In fact, for this section, we are going to use the same systems that were developed in the previous chapters, but they will be deployed in an online way.

Setting up the service as an HTTP REST endpoint provides maximum flexibility, as any potential service that needs to call the online system can be easily adapted to send the inference input in an HTTP POST request, and then receive the system's output in the response. This is a popular approach in the ML community, and a series of different model serving tools have been developed following this model. Some of the most popular ones are Tensorflow-serving[3], MXNET-model-server[4] and Clipper[5]. As the underlying framework used by Sockeye is MXNET, we have elected to use MXNET-model-server for our deployment.

Using a model server has a series of advantages. Fist, it controls all the tasks related to the web server itself, allowing us to focus on the models. On top of that, these tools tend to offer specific features tailored for deployment of ML models, such as automatic batching for requests made in a short interval of time. If one machine is not enough to serve our needs, this approach can be extended by launching multiple instances of the model server and using a load balancer to distribute incoming requests. This enables deployments in both a private cluster and a public cloud environment.

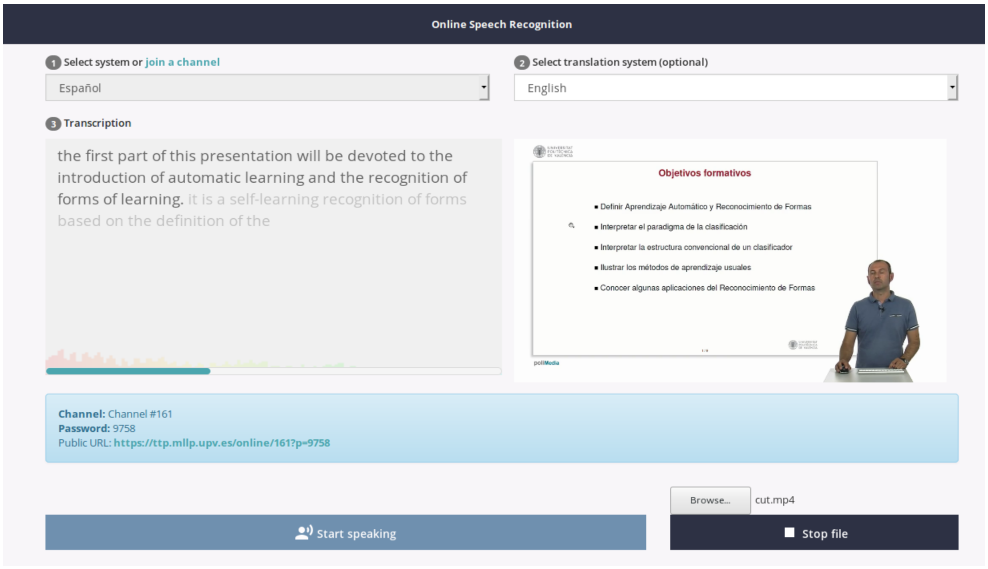
In the case of MXNET-model-server, before using a model, we first need to package it. For each model, we define a service, which is simply a piece of code that receives the data from a request, carries out the appropriate computations using the model, and returns the response. The service jointly with the ML model (weights) and any additional files that might be required for inference, are all then packed into a compressed file using the `model-archiver` tool. The code for serving Sockeye models is built upon one of the provided examples of the MXNET-model-server repository [6].

Once the model-server is started, different models can be started and stopped through a management API. Each model is assigned a different HTTP endpoint, which allows us to serve multiple models in a single model-server instance. You can also modify the number of workers per each model.

The developed online MT service has then been integrated into an online speech translation service of TTP, that combines both ASR and MT in order to carry out automatic multilingual subtitling of videos. Following a cascade model, one of the online ASR systems developed by the MLLP research group [14] receives an audio stream and outputs a text transcription. The ASR system itself incorporates a segmentation model that divides the live transcript into segments. Every time a segment is emitted, it is then sent to the appropriate online MT system, that returns a translation in the desired language. The returned text is then rendered to be used as subtitles for the video.

Figure 7.4 shows a screenshot of the application. The user first selects the language of the input audio for the ASR step, and then, optionally, can select the target language for which they want to obtain the subtitles. As for the audio itself, the user can either upload a pre-recorded video, or can directly speak through their device's microphone. Once the system begins transcribing, a channel is created in order to return the results. Provided they know the password, other users can join the channel as well and receive the subtitles. This enables subtitling for public events and seminars, where the speaker's voice is captured by a microphone, and the attendees can, by joining a channel, follow the event by reading the subtitles written in their

[3]https://github.com/tensorflow/serving
[4]https://github.com/awslabs/mxnet-model-server
[5]http://clipper.ai/
[6]https://github.com/awslabs/mxnet-model-server/tree/master/examples/sockeye_translate


Online Speech Recognition
1 Select system or join a channel
Español
2 Select translation system (optional)
English
3 Transcription
Browse...
Start speaking
Stop file

the first part of this presentation will be devoted to the introduction of automatic learning and the recognition of forms of learning. it is a self-learning recognition of forms based on the definition of the

Objetivos formativos

- Definir Aprendizaje Automático y Reconocimiento de Formas
- Interpretar el paradigma de la clasificación
- Interpretar la estructura convencional de un clasificador
- Ilustrar los métodos de aprendizaje usuales
- Conocer algunas aplicaciones del Reconocimiento de Formas

poliMedia

Channel: Channel #161
Password: 9758
Public URL: https://ttp.mllp.upv.es/online/161?p=9758

cut.mp4

Figure 7.4: TTP online speech translation service. The screenshot shows how the system generates English subtitles for a Spanish video.

own language.

7.2.3 Evaluation

We will now measure the performance of the online MT system. One key setting that significantly affects decoding speed is the beam size used during beam search [7], since the bigger this parameter is, the more hypothesis kept in memory and tested at each step of the decoding process. However, a bigger beam size might allow the model to find a better translation that might otherwise have been pruned. It is therefore desirable to find a balance between quality and speed that fulfils the requirements of our applications. In order to study the effects of these parameters, we have translated 3000 sentences belonging to the WMT14 test set, using the Transformer 3 GPU English $\rightarrow$ French model described in Section 4.2.6. Table 7.1 shows the effect that the beam size setting has on the quality of the generated translations and the speed of the model.

The results show that there is an impact in translation quality and especially speed depending on the selected beam size. The best BLEU score, 37.9, is achieved with a beam size of 8. However, this is almost 3 times slower than using a beam size of 1 (greedy search), in exchange for an increase of 0.8 BLEU. According to the needs of our application, we have selected a beam size of 1, as we believe that a small decrease in translation quality is worth it in exchange for a much lower latency. For

[7]In NMT literature, the search algorithm is usually referred to as beam search, but the actual technique that is implemented during the search process is histogram pruning, not beam search.

Table 7.1: Effects of beam size in translation quality and speed, measured on the WMT14 En-Fr test set, GPU batch decoding.

Beam size	BLEU	Total time(s)
1	37.1	129
2	37.6	177
4	37.8	201
8	37.9	355
16	37.8	606

Table 7.2: Translation time and average latency depending on the number of worker processes, measured for the translation of 100 simultaneous sentences.

	GPU		CPU	
# workers	Total time(s)	Avg. latency(s)	Total time(s)	Avg. latency(s)
1	41	3.18	359	27.69
2	24	1.85	185	14.32
4	17	1.32	105	8.24
8	19	1.48	83	6.50

other applications that do not have such strict time requirements, a beam size of 2 or 4 could also be interesting, in order to offer better translations with acceptable time delays.

Once we have selected the beam size, we move into evaluating the performance of the online system under severe workloads. For this test, the online system will receive 100 simultaneous requests (each containing 1 sentence to be translated), and we will measure the total time required to process them as well as the average latency time of the requests. In order to carry out these tests, a machine with an i7-3820 CPU and a GTX1080 GPU was used. We report results depending on the number of workers that have been used and inference device. The results are shown in Table 7.2.

Using a single GPU worker, translating 100 sentences takes 41.6s with an average latency of 3.18s/sentence. The time taken by the CPU worker is much higher, 359.0s and an average latency of 27.69s/sentence. The difference is reduced as we increase the number of workers. Even so, at the best possible configuration, the GPU worker has an average latency of 1.32s/sentence compared with the 6.50s/sentence of the CPU version. Using the consumer-level hardware that we have available, one can see that GPU inference is the clear winner. The performance of GPU inference decreases if we increase the number of workers from 4 to 8, as all workers have to compete for the resources of the single GPU.

The results shown on the table represent a worst-case scenario where the model-server instance receives a spike of requests, which will inevitably mean that some delays will appear. We will now measure the number of requests that can be serviced without significant latency. The latency for a single sentence is 0.8s for the GPU case and 4.8s for the CPU case. Assuming requests that are equally spaced in time, the maximum amount of sentences that can be translated per hour by a single-worker instance without additional latency would be 4500 sentences for the GPU case and 750 for the CPU case.

7.3 Conclusions

In this chapter we have studied the development of online MT systems. We have described the need for online MT as well as the unique challenges that need to be addressed in order to build efficient systems. We have described the integration process of an offline MT system into the TTP production environment. We have then moved into the specifics of online MT, and how it differs from the offline case. After describing the chosen architecture, we have carried out a series of experiments in order to measure the efficiency and performance of the developed system. As a result of this work, we have obtained a general software solution for online MT, that can be used in order to deploy NMT models in an online way. This allows us to use the MT models that have been described in the rest of the work. The online MT pipeline has been integrated into a speech translation service for TTP, that is able to deliver quality real-time multilingual subtitles.

CHAPTER 8

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.

Chapter 2 and 3 have described previous work carried out prior to this Master's Thesis, in order to set-up the context of the experiments outlined in this document. Chapter 2 introduces the Attention RNN and Transformer model architectures that form the basis of the current state-of-the-art MT systems as well as practical NMT issues. Chapter 3 reports our participation in the German $\rightarrow$ English news translation task of WMT18 as well as the data filtering and processing techniques used.

In Chapter 4 we have described the MT systems developed for the language pairs of the X5gon project. We have described the X5gon project itself as well as illustrating the need for MT systems. Then, for each language pair, we have described the collected data, experimental setups and evaluation results of the systems developed for that language pair. The results have shown the adaptability of NMT architectures, as we have developed systems across a variety of language pairs without requiring specific changes to the model. At the same time, the results have also shown that a lack of computational resources might prevent us from fully leveraging the power of the Transformer architecture.

Chapter 5 describes our participation in the WMT19 Similar Language Translation Shared Task, where we have submitted systems for the Portuguese $\leftrightarrow$ Spanish language pair. We have compared the results of the Transformer and the 2D alternating RNN model, a novel NMT architecture. The results shown that the 2D RNN is not far from the Transformer model when the base models are evaluated. Due to a domain mismatch between train and test data, we have then applied a fine-tuning step that obtains massive improvements in translation quality. The use of fine-tuning has allowed us to win both translation directions of the shared task.

Chapter 6 describes our participation in the WMT19 News Translation Shared

Task, where we have submitted systems for the German ↔ English and French ↔ German language pairs. In order to filter out the noisy data present in the corpora, two filtering techniques have been introduced and compared. Through the use of the backtranslation technique, we were able to generate synthetic data from monolingual text. The use of multi-GPU machines and gradient accumulation has allowed us to obtain quality systems that obtain competitive results when evaluated with the competition's test sets. The results confirm our previous concerns about Transformer model performance when using a batch size that is not big enough.

In Chapter 7 we have described the process of development and integration of online MT systems. We have first described the integration of offline MT systems, and we have then moved into the specific challenges of online MT. We have described the architecture and design decisions required to achieve this goal, and we have carried out experiments in order to measure the performance of the developed solution. The online MT systems have been integrated into an online speech translation service for TTP that is able to provide multilingual subtitles.

8.1 Future work

In terms of future work, our first goal is to continue working on developing systems for the X5gon project. Apart from developing systems for new language pairs that might be of relevance to the project, we also plan to revisit already existing language pairs, by leveraging our findings from both the Similar and the News WMT19 Translation tasks. This includes generating additional synthetic data as well as using bigger batch sizes and taking advantage of multi-GPU machines for all language pairs. In addition, significant increases in translation quality could be obtained by carrying out domain adaptation.

In this work, the problem of speech translation has been tackled from a cascade approach, where the ASR and MT systems are independent. In order to obtain improvements in this area, there are many avenues to explore, from adding ASR-like noise in order to improve robustness of MT systems, to specific MT models that are able to leverage information from the ASR decoding process, such as receiving the list of n-best transcriptions. Instead of a cascade approach, we can also use end-to-end systems, that handle both transcription and translation in a unified way. The generation of new Speech Translation corpora would also help to improve results by allowing us to use more training data and to learn speech translation as an end-to-end task.

8.2 Contributions

The work included in this Thesis has resulted in the following scientific publications:

- Pau Baquero-Arnal, Javier Iranzo-Sánchez, Jorge Civera, and Alfons Juan. The MLLP-UPV Spanish-Portuguese and Portuguese-Spanish machine translation systems for WMT19 similar language translation task. In *Proceedings of the Fourth Conference on Machine Translation: Shared Task Papers*, 2019

- Javier Iranzo-Sánchez, Gonçal V. Garcés Díaz-Munío, Jorge Civera, and Alfons Juan. The MLLP-UPV supervised machine translation systems for WMT19 news translation task. In *Proceedings of the Fourth Conference on Machine Translation: Shared Task Papers*, 2019

- Javier Iranzo-Sánchez, Pau Baquero-Arnal, Gonçal V. Garcés Díaz-Munío, Adrià Martínez-Villaronga, Jorge Civera, and Alfons Juan. The MLLP-UPV German-English machine translation system for WMT18. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 418–424, Belgium, Brussels, October 2018. Association for Computational Linguistics

Apart from the previously listed publications, some passages have also been quoted verbatim from the author's B.S. Thesis [25].

Bibliography

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

[2] 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.

[3] Parnia Bahar, Christopher Brix, and Hermann Ney. Towards two-dimensional sequence to sequence model in neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3009–3015, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

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

[5] Pau Baquero-Arnal, Javier Iranzo-Sánchez, Jorge Civera, and Alfons Juan. The MLLP-UPV Spanish-Portuguese and Portuguese-Spanish machine translation systems for WMT19 similar language translation task. In *Proceedings of the Fourth Conference on Machine Translation: Shared Task Papers*, 2019.

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

[7] Ondrej Bojar, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Philipp Koehn, and Christof Monz. Findings of the 2018 conference on machine translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 272–303, 2018.

[8] 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.

[9] 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.

[10] Mauro Cettolo, Marcello Federico, Luisa Bentivogli, Niehues Jan, Stüker Sebastian, Sudoh Katsuitho, Yoshino Koichiro, and Federmann Christian. Overview of the iwslt 2017 evaluation campaign. In *International Workshop on Spoken Language Translation*, pages 2–14, 2017.

[11] 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.

[12] 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.

[13] 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.

[14] MA Del-Agua, Adrià Giménez, Nicolás Serrano, Jesús Andrés-Ferrer, Jorge Civera, Alberto Sanchís, and Alfons Juan. The translectures-upv toolkit. In *Advances in Speech and Language Technologies for Iberian Languages*, pages 269–278. Springer, 2014.

[15] 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.

[16] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 489–500, 2018.

[17] Maha Elbayad, Laurent Besacier, and Jakob Verbeek. Pervasive attention: 2D convolutional neural networks for sequence-to-sequence prediction. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 97–107, Brussels, Belgium, October 2018. Association for Computational Linguistics.

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

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

[20] 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.

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

[22] 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.

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

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

[25] Javier Iranzo-Sánchez. A comparative study of neural machine translation frameworks for the automatic translation of open data resources. B.S. Thesis, Universitat Politècnica de València, 2018.

[26] Javier Iranzo-Sánchez, Pau Baquero-Arnal, Gonçal V. Garcés Díaz-Munío, Adrià Martínez-Villaronga, Jorge Civera, and Alfons Juan. The MLLP-UPV German-English machine translation system for WMT18. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 418–424, Belgium, Brussels, October 2018. Association for Computational Linguistics.

[27] Javier Iranzo-Sánchez, Gonçal V. Garcés Díaz-Munío, Jorge Civera, and Alfons Juan. The MLLP-UPV supervised machine translation systems for WMT19 news translation task. In *Proceedings of the Fourth Conference on Machine Translation: Shared Task Papers*, 2019.

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

[29] Javier Jorge, Adrià Giménez, Javier Iranzo-Sánchez, Jorge Civera, Albert Sanchis, and Alfons Juan. Real-time one-pass decoder for speech recognition using lstm language models. In *Proc. of the 20th Annual Conference of the ISCA (Interspeech 2019)*, 2019.

[30] Marcin Junczys-Dowmunt. Dual conditional cross-entropy filtering of noisy parallel corpora. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 888–895, 2018.

[31] 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.

[32] Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. *arXiv preprint*, 2015.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, San Diego, California, USA, 2015.

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

[35] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*, 2007.

[36] Philipp Koehn, Huda Khayrallah, Kenneth Heafield, and Mikel L. Forcada. Findings of the WMT 2018 shared task on parallel corpus filtering. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 726–739, 2018.

[37] 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.

[38] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[39] Marco Lui and Timothy Baldwin. langid.py: An off-the-shelf language identification tool. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the System Demonstrations, July 10, 2012, Jeju Island, Korea*, pages 25–30, 2012.

[40] Minh-Thang Luong and Christopher D. Manning. Stanford neural machine translation systems for spoken language domain. In *International Workshop on Spoken Language Translation*, 2015.

[41] 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.

[42] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*, 2019.

[43] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 1–9, 2018.

[44] 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.

[45] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[46] 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.

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

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

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

[50] 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.

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

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

[53] Julian Schamper, Jan Rosendahl, Parnia Bahar, Yunsu Kim, Arne Nix, and Hermann Ney. The RWTH aachen university supervised machine translation systems for WMT 2018. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers, WMT 2018, Belgium, Brussels, October 31 - November 1, 2018*, pages 496–503, 2018.

[54] 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.

[55] 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.

[56] 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.

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

[58] 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.

[59] 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.

[60] Andreas Stolcke, Jing Zheng, Wen Wang, and Victor Abrash. SRILM at Sixteen: Update and Outlook. In *Proceedings of the 2011 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Waikoloa, Hawaii, USA, 2011.

[61] 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.

[62] 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.

[63] Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019.

List of Figures

List of Tables