UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

# Character-based Neural Machine Translation

**MASTER'S THESIS**

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

*Author:* Antonio Manuel Larriba Flor

*Tutor:* Francisco Casacuberta Nolla
Álvaro Peris Abril

Course 2016-2017

# Resum

Les paraules fora de vocabulari continuen sent un problema per resoldre en els sistemes de traducció actuals. L'objectiu d'aquest treball es tractar de solucionar el problema d'un vocabulari limitat emprant nivells de traducció més menuts que les paraules (e.g: caràcters). Es busca aprofitar característiques com la composició de paraules, propietats morfològiques y semàntiques per a tractar de traduir paraules desconegudes mitjançant sub-unitats sí conegudes. D'aquesta manera el model de traducció no només és capaç de bregar amb paraules desconegudes sinó que també seria possible traduir a paraules no vistes durant la fase d'entrenament.

**Paraules clau:** xarxes neuronals, traducció automàtica, subparaules

# Resumen

Las palabras fuera de vocabulario siguen siendo un problema por resolver en los sistemas de traducción actuales. El objetivo de este trabajo es tratar de solucionar el problema de un vocabulario limitado empleando niveles de traducción más pequeños que las palabras (e.g: caracteres). Se busca aprovechar características como la composición de palabras, propiedades morfológicas y semánticas para tratar de traducir palabras desconocidas a partir de sub-unidades sí conocidas. De este modo el modelo de traducción no sólo es capaz de lidiar con palabras desconocidas si no que también sería posible traducir generando nuevas palabras no vistas durante la fase de entrenamiento.

**Palabras clave:** redes neuronales, traducción automática, subpalabras

# Abstract

Out of vocabulary words are still an open problem in translation systems. The aim of this work is trying to solve the problem of a limited vocabulary using translation levels smaller than words (e.g: characters). We want to take advantage of characteristics like word compounding, morphological and semantical properties to try to translate unknown words from sub-word units already known. This way the translation model is not only able to deal with unseen words but also it will be capable of generating new words not seen during the training phase.

**Key words:** neural networks, machine translation, subword units

# Contents

# List of Figures

# CHAPTER 1
# Introduction

As humans we are social by nature. The tool used to socialize is the language, however we live in a multi-lingual world which makes interaction between speakers of different languages difficult.

From this necessity, and due to the magnitude of the task, machine translation was born. This refers to the use, partial or complete of computers to carry out translations. During the last decades the statistical models have dominated this field. These models rely on the use of huge bilingual corpora to be trained. In the last years neural networks have established as the state-of-the-art of machine translation replacing other classical approximations.

Despite of the advantages of neural networks, out-of-vocabulary words are still a problem in modern translation systems. Most systems have a limited vocabulary size because of efficiency reasons. Not all words can be seen during training phase, specially if those words are numbers, names, compound words, loanwords and other rare words. These out-of-vocabulary or unseen words represent an issue, the translation system does not know how to deal with them since it did not saw them during the training phase. To solve this problem we may approach the translation using the sub-words units (e.g: characters). We do this because we think we will be able to translate unseen words by encoding them via smaller units.

By doing this, we aim to get an open vocabulary system in which, we will try to translate rare words by breaking them into smaller and known units. Indeed, depending on how the system is implemented it could be also able to generate new words. For example, in a translation system from English to Spanish the model might not know the word “parasol” but it may be possible to generate it by the concatenation of “para” and “sol”.

If done right we will get a better system, capable of dealing with unseen words and generating new words.

## Objectives

The prime aim of this work is to carry out translations by using neural networks using sub-word units to handle out-of-vocabulary words. This can be split in smaller objectives:

1. Study different approaches to work with characters and other sub-word units.
2. Implement and combine those approaches into the classical neural machine translation architecture.
3. Analyze how those approaches affect out-of-vocabulary words.
4. Compare results with word-level translation systems.

## Structure

In this first chapter, we have presented the issue this work addresses and the objectives we aim to fulfill. Chapter 2 introduces formally the concept of statistical machine translation. It reviews its most classical approaches and explains the core concepts that will be used along this work.

Chapter 3 will be dedicated to explaining the neural machine translation. It will explain how different is with regard to the previous statistical approaches, its most common architectures, and all the steps carried out to produce a translation.

Chapter 4 presents different techniques and algorithms for working at sub-word level using neural machine translation. These approaches will be tested, evaluated and compared against well-known datasets in Chapter 5. Finally Chapter 6 will present the conclusions extracted from this Master thesis.

# CHAPTER 2
# Statistical Machine Translation

This chapter and the next one are a gentle introduction to machine translation. We will review the literature, starting from the basics, to explain simple models and different statistical approaches, to finally show which of them constitutes the state of the art on this field.

## Fundamental Equation of Machine Translation

Machine translation (MT) refers to the techniques computers use to translate between two different languages. Given a sentence $f_1^J \equiv f_1, f_2, ..., f_J$ in the source language $\mathcal{F}$ we want to translate it into a sentence $e_1^I \equiv e_1, e_2, ..., e_I$ in the target language $\mathcal{E}$.

Our work is to build the best models to get high quality translations. Different approaches have existed on how to define those models. Statistical machine translation (SMT) has been the state-of-the-art for the last decades. SMT engages the translation problem, as its name suggests, from a statistical framework. Given the source sentence $f_1^J$ it tries to obtain the most probable translation $\hat{e}_1^I$ from all the possible sentences in the target language $\mathcal{E}$.

$$\hat{e}_1^{\hat{I}} = \arg\max_{e_1^I, I} P(e_1^I | f_1^J) \tag{2.1}$$

The problem is that we do not know all the possible sentences for a given language. And even if we knew, the cost of searching would be excessive. We can apply Bayes' theorem in order to break Equation (2.1) into two separate and more affordable expressions.

$$\hat{e}_1^{\hat{I}} = \arg\max_{e_1^I, I} P(e_1^I) P(f_1^J | e_1^I) \tag{2.2}$$

Equation (2.2), is broken apart into two different probability distributions. $P(e_1^I)$[1] is modeled through the language model. It should give high probabilities

[1] $P(\cdot)$ is used for the real probability distribution while $p(\cdot)$ is used for models and approximations.

to common and well-constructed sentences in the target language. On the other hand, $P(f_1^J|e_1^I)$ is named as the translation probability. It is estimated through the translation model which estimates how good a translation is. Ideally, it should give large probabilities to a semantically equivalent translation even if it is not syntactically correct (the language model is responsible of that kind of evaluation). Equation (2.2) is known as the basic equation of MT (Brown et al., 1993) and implies the following challenges:

1. Estimate the Language model (LM) probability distribution $p(e_1^I)$.
2. Estimate the Translation model (TM) probability distribution $p(f_1^J|e_1^I)$.
3. Find a proper and efficient search method to find the sentence $\hat{e}_1^I$ which maximizes the product of both models probabilities.

These points will be treated in more detail in the following sections. But first, we have to face a discouraging fact, we will never know the real probability distributions. We have to train our models, which are defined by a set of parameters $\theta$, with huge bilingual corpora to approximate those unknown distributions. The goal is to automatically detect bilingual correlations and extract linguistic information to determine the values of those parameters in our models. The aforementioned set, $\theta$, describes the probability density function related to our model $P(\cdot|\theta)$. Let $\mathcal{X} = x_1, x_2, ..., x_N$ be a set of observations. Therefore, the *log-likelihood* of $\mathcal{X}$ given $\theta$ is computed as:

$$\mathcal{L}(\theta, \mathcal{X}) = logp(x_1, x_2, ..., x_N|\theta) = \sum_i^N logp(x_i|\theta) \tag{2.3}$$

We usually work with the logarithm of probabilities rather than the probabilities because of two reasons. For computers summation is cheaper than multiplication and working in the logarithmic space provides numerical stability and computational robustness against really small values.

To achieve the parameter estimation we are looking for, we employ *Maximum Likelihood Estimation* (MLE) over the given set of observations. MLE searches the subset of parameters $\hat{\theta}$, over the hyperspace of possible parameters $\Theta$, that maximizes the log-probability[2] $\mathcal{L}(\theta, \mathcal{X})$.

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \mathcal{L}(\theta, \mathcal{X}) \tag{2.4}$$

Each part of Equation (2.2) has a different set of parameters, thus we have two set of equations, one for the language model and the other one for the translation model. Equation (2.4) is instantiated for each model and its corresponding observations:

$$\hat{\theta}_{LM} = \arg\max_{\theta \in \Theta_{LM}} \sum_i^I logp(e_i|\theta) \tag{2.5}$$

[2]Indeed, as the logarithm is a monotonically increasing function it does not matter if we maximize the *likelihood* or the *log-likelihood*. We will achieve the same result.

$$\hat{\theta}_{TM} = \arg\max_{\theta \in \Theta_{TM}} \sum_{j}^{J} log p(f_j | e_1^I, \theta) \tag{2.6}$$

Next sections treat in more detail each one of the models and different approaches to its modeling and estimation.

# Language Modeling

As previously stated, the aim of the language model is to compute the probability associated to a given sentence $P(e_1^I)$ in the corresponding language $\mathcal{E}$. In other words, to capture language regularities to determine how well-constructed, how natural a sentence is in its language. Let $e_1^I$ be a sentence composed by $I$ words $(e_1, e_2, ..., e_I)$. By applying the chain rule we can decompose its associated probability mass as follows:

$$P(e_1^I) = P(e_1)P(e_2|e_1)...P(e_I|e_1^{I-1}) = \prod_{i=1}^{I} P(e_i|e_1^{i-1}) \tag{2.7}$$

The cost of this computation is intractable. As the length of the sentence increases, our possibilities to compute it decrease; since each new word depends on the whole sequence of previous words $e_1^{i-1}$. The full computation of this expression is discarded in favor of n-gram models.

N-gram models are the result of applying Markov's assumption to this problem. We assume that future events will depend only in recent facts, thus the next word $e_i$ will only depend on the $n-1$ preceding words. The number $n$ determines the order of the n-gram model. The probability of the sentence can be re-written as follows:

$$P(e_1^I) \approx \prod_{i=1}^{I} p(e_i|e_{i-n+1}^{i-1}) \tag{2.8}$$

Now, the estimation of $p(e_i|e_{i-n+1}^{i-1})$ can be computed using MLE ($p_{ML}$), which results in simply counting and normalizing.

$$p_{ML}(e_i|e_{i-n+1}^{i-1}) = \frac{c(e_{i-n+1}^{i-1})}{\sum_{e_i} c(e_{i-n+1}^{i})} \tag{2.9}$$

c(·) counts how many times that particular n-gram appears in the training data.

## Handling Unknown N-grams: Smoothing

N-grams are a simple and effective way to model the language probability distribution. Its parameter estimation is also pretty straightforward, nonetheless n-grams still have a problem. If one of the n-grams conforming the sentence is

unknown the assigned probability will be null[3] (recall that as appears in Equation (2.8) final probability is a product). Obviously we do not want this displeasing behavior in our model, to solve this low-frequency problem we appeal to different smoothing techniques. The term smoothing refers to different means of refining the probability distribution by decreasing high frequencies and increasing the lower ones. Not only helps to avoid null frequencies but it also has been demonstrated to help the overall system performance (Chen and Goodman, 1996).

Different approaches exist (Goodman, 2001), however the most widely used is the so-called Kneser-Ney discount (KND) (Kneser and Ney, 1995). KND uses interpolation and back-off to estimate the probabilities of unseen events. By back-off, we refer to appeal to a lower-order distribution when the actual one is null (e.g: When a tri-gram model has no counts for a sequence of words we recall the bi-gram distribution to interpolate it). A probability mass D ($0 < D < 1$) is subtracted for each non-zero count. This constitutes the modified high-order n-gram distribution, lower-order models are modified accordingly the higher model counts. By doing this, we can see low-order n-gram distributions as a smoothed version of high-order distributions. The aim is to reduce null counts and to improve system effectiveness. To do so they proposed the following equation,

$$p_{KN}(e_i|e_{i-n+1}^{i-1}) = \begin{cases} \frac{max\{c(e_{i-n+1}^{i-1})-D,0\}}{\sum_{e_i} e_{i-n+1}^{i}}, & \text{if } c(e_{i-n+1}^{i-1}) > 0 \\ \\ \gamma(e_{i-n+1}^{i-1}) p_{KN}(e_i|e_{i-n+2}^{i-1}), & \text{if } c(e_{i-n+1}^{i-1}) = 0 \end{cases} \tag{2.10}$$

where $\gamma(e_{i-n+1})$ is a normalization factor used to ensure probabilities sum to unity. Using this smoothing technique not only affects n-grams with null frequency, but also the rest are modified. This makes KND context-dependent since modifications on lower-order models are affected by higher-order models. Hence, KND is more accurate and different with respect other absolute discount methods.

## Translation Modeling

The goal of the translation model is to estimate $P(f_1^J|e_1^I)$ from Equation (2.2), to determine if a sentence in the target language $\mathcal{E}$ is a good translation, with similar semantic meaning, of the original sentence in the source language $\mathcal{F}$.

### Word-based Translation

One of the first attempts for building a translation model was a simple word-alignment model. Word is the basic unit of translation and those words are reorganized (aligned) and translated to craft the final translation. The term align-

[3] This is a common problem given the enormous number of possible events. Usually much larger than the data available to train (data sparsity problem).

ment (Brown et al., 1993) describes the correspondence between a sentence $f_1^J \equiv f_1, f_2, ..., f_J$ in the source language $\mathcal{F}$ and a sentence $e_1^I \equiv e_1, e_2, ..., e_I$ in the target language $\mathcal{E}$ at a word level. Equation (2.11) represents the possible alignments.

$$a \subseteq 1...J \times 1...I \tag{2.11}$$

where $a_j = i$ if the *jth* word of the source sentence $(f_j)$ is aligned with the *ith* word from the target sentence $(e_i)$. A special position (position 0) exists to represent that a word in the source is not aligned with any word in the target. If we denote the whole set of possible alignments as $\mathcal{A}(f_1^J, e_1^I)$ we can reformulate translation model equation as follows:

$$P(f_1^J|e_1^I) = \sum_{a_i^J \subset \mathcal{A}(f_1^J, e_1^I)} P(f_1^J, a_i^J|e_1^I) \tag{2.12}$$

The alignment is a hidden variable and we must sum over all possible values to obtain the translation probability. Given the source sentence length J and the target sentence length I there exist J*I possible connections, this is $2^{J*I}$ possible alignments in $\mathcal{A}(f_1^J, e_1^I)$. We can represent alignments with lines linking the related words and its positions as represented in Figure 2.1.

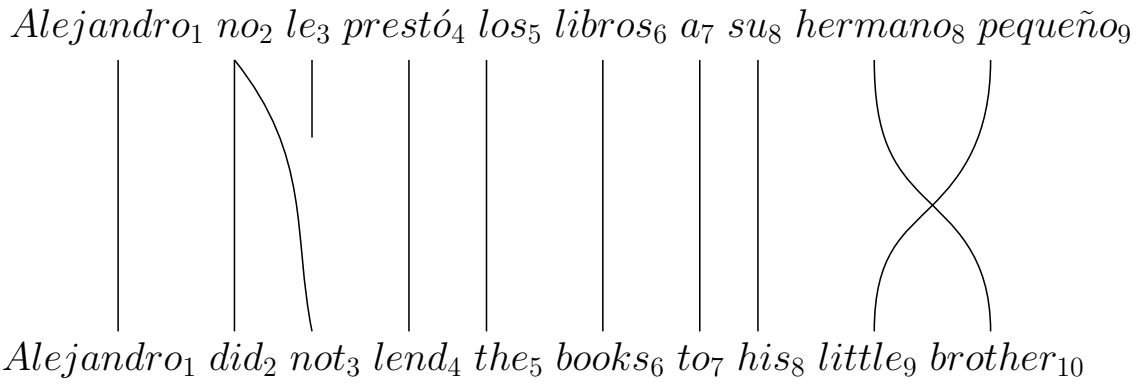



**Figure 2.1:** Word alignment example. Notice $a_3 = 0$, meaning that word *le* is not aligned with any word. It is linked to the special position 0.

### IBM Models

Summing over each element of $\mathcal{A}(f_1^J, e_1^I)$ possible alignments without further studying it is not a good idea. In 1993, the *IBM T.J. Watson Research Center* developed five models (Brown et al., 1993) based on the concept of alignments. All of this models are based on the following derivation, not an approximation, of Equation (2.12):

$$P(f_1^J, a_i^J|e_1^I) = P(J|e_1^I) \prod_{j=1}^{J} P(a_j|a_1^{j-1}, f_1^{j-1}, j, e_1^I) P(f_j|a_1^j, f_1^{j-1}, j, e_1^I) \tag{2.13}$$

This equation implies that regardless of the form of $P(f_1^J, a_i^J|e_1^I)$, it can always be expressed as a product of terms. Now we can analyze Equation (2.13) as a sequential way to generate $f_1^J$ following this steps:

1. We first decide the length $J$ of the source language $\mathcal{F}$ sentence given $e_1^I$. This is done by the length distribution $P(J|e_1^I)$

2. We choose where to connect the first word of sentence $f_1^J$ given the knowledge we have about previous history $(a_1^{j-1}, f_1^{j-1})$, the target sentence $(e_1^I)$ and the length of the source sentence $(J)$. The alignment distribution $P(a_j|a_1^{j-1}, f_1^{j-1}, j, e_1^I)$ handles this term.

3. Finally, we decide which word $f_j$ generate according to the target sentence $(e_1^I)$, the length of the source sentence $(J)$ and the alignment $(a_1^j, f_1^{j-1})$ computed in the previous step. The translation distribution $P(f_j|a_1^j, f_1^{j-1}, j, e_1^I)$ takes care of this point.

4. Iterate over points 2 and 3 until the desired length is reached and we have a fully-aligned sentence.

In this way we make decisions about how to build the target string, based on previous knowledge, at every iteration step. All five aforementioned models share this steps, they just differ on how to estimate step 2, concretely how to compute the alignments probabilities $P(a_j|a_1^{j-1}, f_1^{j-1}, j, e_1^I)$.

The parameter estimation of all models is carried out by a modified version for incomplete data of the Expectation-Maximization (EM) algorithm (Dempster et al., 1977).

Until here we covered the basics of word-based translation. Unfortunately they are very limited since only single words alignments are considered. Contextual information and complex structures are lost. To solve this problem multi-word alignment models were proposed.

## Phrase-Based Models

To tackle word-based systems limitations phrase-based translation models were proposed. The basic unit of translation is now the *phrase* which is defined as a set of one or more consecutive words in the source or target sentence. Now the alignments are done between phrases. Given a source and target sentence $f_1^J, e_1^I$, they can be bi-segmented in K phrases with $1 < K < min(I, J)$, represented as $\tilde{f}_1^K, \tilde{e}_1^K$.

Phrase-based (PB) models use statistical dictionaries of phrase pairs which are automatically learnt from the training corpus. Bilingual phrases are defined (Zens et al., 2002) as a pair of $m$ source words and $n$ target words. Some restrictions are imposed:

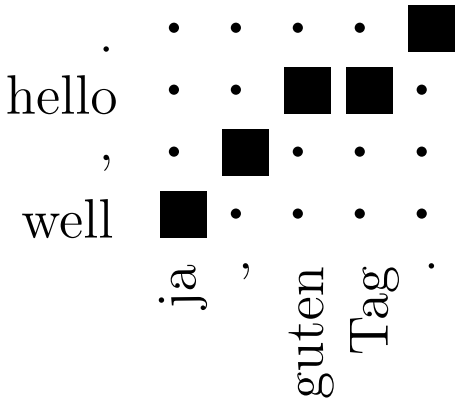
1. Words must be consecutive.

2. They must be consistent with the word matrix alignment. This means that the m source words are aligned only to the n target words and vice versa. No multiple different relations for the same words are allowed.

In Figure 2.2 we can see how the phrases are aligned (2.2a) and how the phrase pairs will be formed (2.2b). We must bear into account that three new hidden variables must be included to deal with bilingual phrases:

1. $\mu$ determines how to segment the source sentence in $K$ phrases ($\tilde{f}_1^K$).
2. $\gamma$ establishes how to segment the target sentence in $K$ phrases ($\tilde{e}_1^K$).
3. $\alpha$ regulates the phrase alignments.

EThe estimation is done through a HMM approximation. Restrictions and the maximum approximation might be used to speed up computations.




(a)

| source phrase | target phrase |
|---|---|
| ja | well |
| ja, | well, |
| ja, guten Tag | well, hello |
| ja, guten Tag. | well, hello. |
| , | , |
| , guten Tag | , hello |
| , guten Tag. | ,hello. |
| guten Tag | hello |
| guten Tag. | hello. |
| . | . |

(b)

**Figure 2.2:** Phrase based alignment and phrase pairs example. Extracted from (Zens et al., 2002)

## Phrase-Based SMT. Log-Linear Model

This model tackles Equation (2.1) from a discriminative point of view. It searches for a target sentence with the maximum posterior probability. Many SMT systems rely on this approach, the log-linear mode has been the prevailing approach of SMT for a long time. It consists in a set of features $h_m(f_1^J, e_1^I)$ weighted by a factor $\lambda_m$:

$$\begin{aligned} \hat{e}_1^I &= \arg\max_{e_1^I} P(e_1^I|f_1^J) \\ &= \arg\max_{e_1^I} \frac{exp(\sum_{m=1}^M \lambda_m h_m(f_1^J, e_1^I))}{\sum_{e_1^{I'}} exp(\sum_{m=1}^M \lambda_m h_m(f_1^J, e_1^{I'}))} \\ &= \arg\max_{e_1^I} \sum_{m=1}^M \lambda_m h_m(f_1^J, e_1^I) \end{aligned} \tag{2.14}$$

From step 2 to 3 in Equation (2.14) we can remove the denominator because it is a constant and we are interested just on the *argmax*. Also we eliminate the *exp* since maximizing $e^x$ and $x$ is equivalent.

An interesting property about this approach is that we can define as many features and as diverse as we want. The following features are the most common:

1. Target language model.
2. The phrase-based model and inverse phrase-based model. From source to target and vice versa.
3. A reordering model.
4. A target word penalty and phrase penalty. These are used to avoid the natural tendency of statistical model of generating short sentences and short phrases.

For estimating the weights ($\lambda_m$), different optimization algorithms exist: MERT (Och, 2003), MIRA (Hasler et al., 2011) or PRO (Hopkins and May, 2011). They usually minimize an error function over a validation set to get the weights.

## Assesment

MT evaluation is still an open problem. We must differentiate between human and automatic evaluation. On the one hand, human evaluation is expensive in time and resources, but it usually provides good estimations over the translations quality. But in addition to the cost, we must also bear into account the subjective nature of humans, two judges may diverge in their evaluation. On the other hand, we have automatic evaluation, faster, cheaper and deterministic but it does not work as well as human evaluation. However, for efficiency reasons, automatic evaluation is widely used.

The main problem with automatic evaluation is to prove that a given metric is related with our human judgment. That a higher score directly implies better translations and vice versa. Nowadays we do not know how to prove it, that is why evaluation is still an open problem. Next we present the most common metrics.

#### BLEU

BLEU (BiLingual Evaluation Understudy) (Papineni et al., 2002) compares the translation generated by a MT system (hypotheses) with a human supervised translation (reference). It is based on the n-gram concept explained before, it counts the number of candidate words from the hypotheses that appear in the reference and later divides this count by the number of words in the hypotheses. To avoid MT systems to generate too many suitable n-grams, the modified n-gram precision is presented to clip the counts of an n-gram:

$$C_{clip} = min(c(w), max\_ref\_c(w)) \tag{2.15}$$

where, $c(w)$ is the count of the n-gram w in the hypotheses and max_ref_$c(w)$ is the maximum count of $w$ in the reference sentence. BLEU is computed on a document by computing it over each sentence:

1. Compute n-gram matches sentence by sentence.

2. Sum over all possible candidates. If we are working with multi-references we must compute n-gram matches for each one.

3. Normalize:

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{w \in C} C_{clip}(w)}{\sum_{C' \in \{Candidates\}} \sum_{w' \in C'} C_{clip}(w')} \tag{2.16}$$

In addition, to control sentence length a brevity penalty (BP) is introduced. Translations longer than the reference are already penalized by the modified n-gram precision, for dealing with shorter translations BP is applied as follows:

$$BP = \begin{cases} 1, & \text{if } t > r \\ e^{1-\frac{r}{t}} & \text{if } t \leq r \end{cases} \tag{2.17}$$

BLEU is computed as a weighted geometric mean of the different n-gram orders employed. Each n-gram has a weight $w_n$ such that $\sum_{n=1}^{N} w_n = 1$. Usually weights are set to $w_n = \frac{1}{N}$ and the maximum order is fixed to $N = 4$. These values are taken from (Papineni et al., 2002). Finally BLEU is computed as:

$$BLEU = BP * exp(\sum_{n=1}^{N} w_n log p_n) \tag{2.18}$$

### TER

TER (Translation Edit Rate) (Snover et al., 2006) is defined as the maximum number of edits needed to change the hypotheses sentence to perfectly match the reference. Possible edits include insertion, deletion, substitution and reordering words. TER is computed as follows:

$$TER = \frac{\text{\#of edits}}{\text{\#of reference words}} \tag{2.19}$$

All edits have equal cost. To compute number of edits a dynamic programming algorithm is used.

### Other metrics

Many other metrics are used, we briefly present three of the most used. METEOR (Banerjee and Lavie, 2005) is a language metric based on alignments which takes advantage of language resources for evaluation. It applies stemming and synonymy to the matching process. A possible drawback is that there is no language resources available for all language pairs.

NIST (Doddington, 2002) is a modified version of BLEU proposed by the NIST (National Institute is Standards and Technology)[4]. WER (Word Error Rate) is

[4] https://www.nist.gov/

based on Levensthein distance, it works by measuring the number of edits at word level.

# CHAPTER 3
# Neural Machine Translation

In the last years a new approach in MT, within the statistical framework, has emerged to constitute itself as the state of the art in automatic translation. This is the neural machine translation (NMT), where the translation process is carried out by artificial neural networks or simply neural networks (NN).

Neural networks are models whose basic unit is the neuron, which performs a linear combination of its inputs and later applies a non-linear function to the output. This simple unit is used to build complex networks which have a wide range of applications.

In Chapter 2, we broke apart Equation (2.1) because we were not able to estimate that probability directly. NN are able to model that probability straight. No need to employ different models. In addition the whole translation system (also referred as the network) can be trained jointly.

NN are not new, they were previously applied, in a simpler way, to MT (Castano et al., 1997) but until the recent years the problems that were dragging this approach were still unresolved: problems propagating the gradients through the network and a high computational cost.

Points 1 is explained in detail in the following sections. Regarding point 3, conventional central processing units (CPUs) have been substituted by graphic processing units (GPUs) specialized in floating-point arithmetic. The use of GPUs has speed up training by a few orders of magnitude, because they allow a higher degree of parallelization and lots of toolkits and libraries have proliferated to ease its use.

## Word Representation

Different choices exist on how to represent words in an NMT system, however the de facto standard is to use a continuous representation. Words are represented as low-dimensional continuous vectors of features, also called word-embeddings. Instead of working with discrete vectors of size equal to vocabulary |V| (e.g: 15000) we work with continuous vectors of lower size $m$ (e.g: 300). Word embeddings are stored in a matrix $\mathbf{E} \in \mathbb{R}^{|V| x m}$ which can be indexed by words, being $|V|$ the size of the vocabulary and $m$ the embedding size. Typically, words are linearly projected to the word-embedding space.

Other discrete word representation approaches became obsolete for the reasons that Bengio et al. (2003) explain in his language model proposal: they do not take into account the similarity between words and they describe a really disperse space (the course of dimensionality). On the other hand it seems logical to think that similar words (at lexical or semantic level) should have some similar continuous vector of features.

To estimate words probability in the continuous space Bengio et al. (2003) propose the following approach. First, words are mapped to a real vector $\mathbf{v} \in \mathbb{R}^m$. Second, to obtain the probabilities associated to each word and its context a function $g$ maps an input sequence of those vectors to a conditional probability distribution over the words in the vocabulary. To simultaneously train word feature vectors and the parameters of probability function $g$, they employ a feed-forward NN as shown in Figure 3.1.

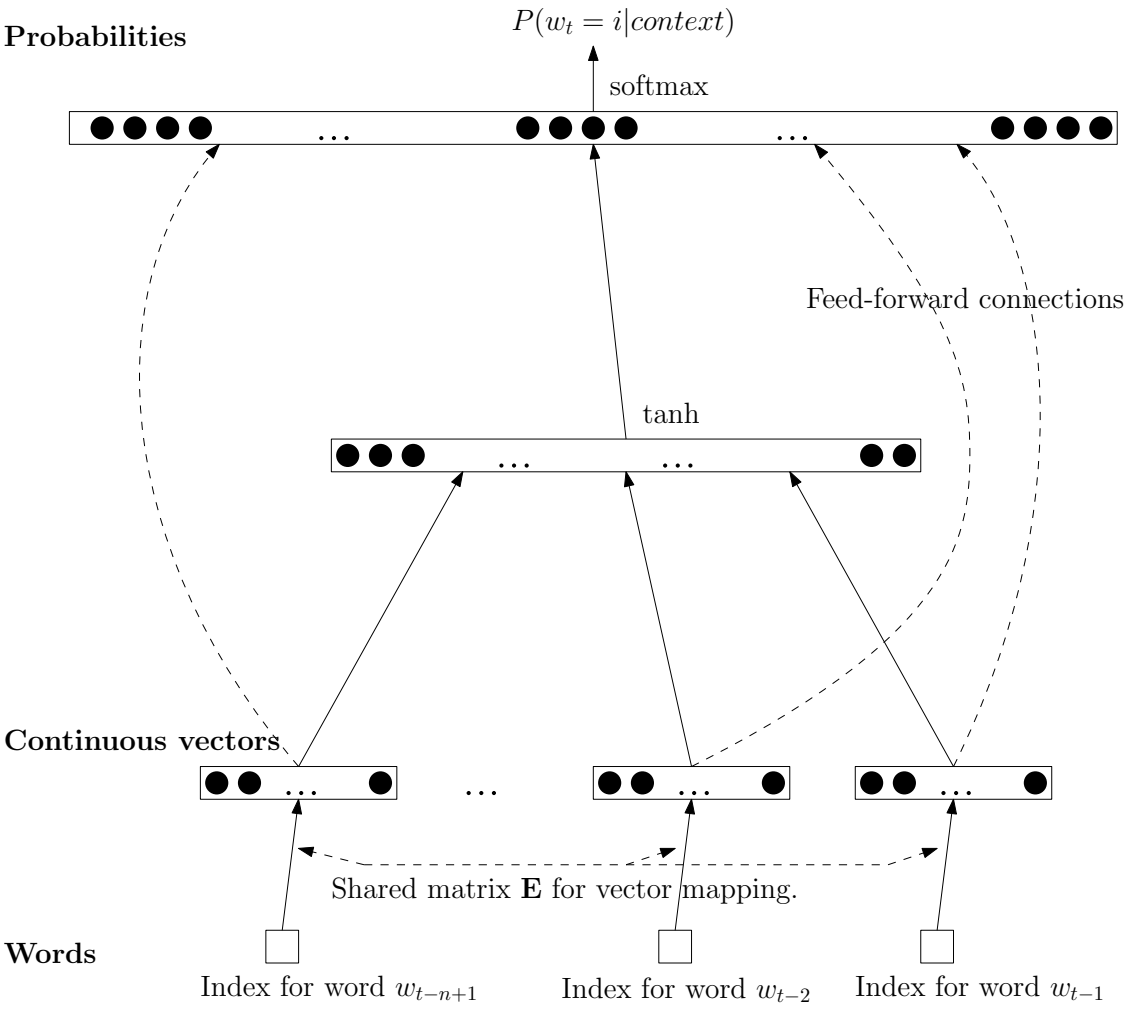



**Figure 3.1:** Neural architecture for word embedding. Image extracted from (Bengio et al., 2003)

Mikolov et al. (2010) used recurrent neural networks with the same objective and came up with a new version of the language model: a recurrent neural network language model (RNNLM) which improved previous results. Over this language model a new approach for obtaining word representations was developed (Mikolov et al., 2013a). If we are not interested in the language model and

its probabilities, if we are just concerned about the word-embeddings we can speed up computations and yet obtain really good results. Some enhancements were added later to improve overall system performance (Mikolov et al., 2013c). Some of them were based on using a simplified version of the noise contrastive estimation technique as a substitution of the softmax of the output layer. The softmax layer is used to obtain probabilities and constitutes a significant part of the computational work. The softmax layer applies the softmax function:

$$\sigma(z_k) = \frac{exp(z_k)}{\sum_{k'=1}^{K} exp(z_{k'})} \tag{3.1}$$

to its inputs and returns a vector in which all the elements sum up to one.

Words-embeddings have proved useful in many areas other of natural language processing: NMT, PB, sentiment analysis and even for exploiting language similarities (Mikolov et al., 2013b).

## Recurrent Neural Networks

Recurrent neural networks (RNN) are a class of NN adequate for processing sequential data. They are widely used in translation tasks since they often can process sequences of variable length (e.g: sentences). As its name suggest it has an internal recurrent connection between its states. This internal state $\mathbf{h}$ allows the network to model a discrete-temporal behavior. Given a sequence of input vectors $\mathbf{x}_1^T \equiv \mathbf{x}_1, ..., \mathbf{x}_T$ a RNN will output a sequence $\mathbf{y}_1^T \equiv \mathbf{y}_1, ..., \mathbf{y}_T$, computed as:

$$\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{3.2}$$

$$\mathbf{y}_t = f_o(\mathbf{h}_t) \tag{3.3}$$

where $f_h$ and $f_o$ are hidden and output activation functions respectively. Different configurations of $f_o$, $f_h$ and $\mathbf{h}$ result in different RNN like Jordan networks (Jordan, 1997) or Elman networks (Elman, 1990). Figure 3.2 shows an Elman RNN unfolded in time.

Parameter estimation of these models is done using stochastic gradient descent (Robbins and Monro, 1951), using back-propagation through time (BPTT) (Werbos, 1990) which tries to minimize a loss function under some optimality criteria. BPTT algorithm is applied to the unfolded version of the RNN. It uses the general back-propagation method to each state of the RNN. Back-propagation modifies the network parameters according to the loss gradient with regard to the weights. The problem is that applying the algorithm at each step is expensive and causes problems propagating gradients trough too many states. To mitigate the overhead of the algorithm, samples are accumulated and the back-propagation algorithm is applied in parallel. We accumulate the loss for $b$ samples (batch size) and then proceed to modify the parameters. A possible solution for propagating the gradients is presented in the next sections.

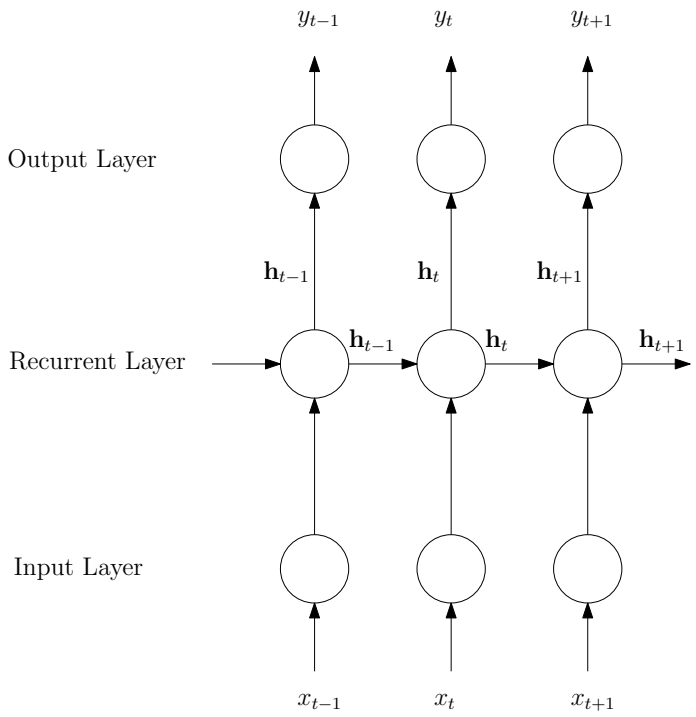



**Figure 3.2:** Elman architecture unfolded through time.

Typically, the loss function used is cross-entropy between system's output and the references. For models in which there exists a relation among the system output and the hidden state **h** we can use, in training phase, teacher forcing (Williams and Zipser, 1989), a procedure which takes advantage of knowing the references. At training phase teacher forcing feeds the correct output $y_t$ (from the reference) to the next hidden state $\mathbf{h}_{t+1}$ trying to speed up the convergence process.

## Bidirectional Recurrent Neural Networks

RNN are used in MT due to its variable input length and its ability to pass information across an undefined number of sequential steps. However they have a drawback, they process the sequential input only in one direction, usually left-to-right. In order to capture the future dependencies Bidirectional Recurrent Neural Networks (BRNN) were proposed (Schuster and Paliwal, 1997). The idea is to have two independent RNN, one to process the input from left-to-right (or from past to future if we want to see it as a time relation) and a second one to process it from right-to-left. Later the output of both networks is combined (sum or concatenation are the most usual choices) to obtain a single output like as in a regular RNN. Figure 3.3 shows the structure of a BRNN to illustrate the process.

As there is no interaction between each RNN both can be trained using the same algorithms explained before. Symbol $\oplus$ represents a union operator. Hidden states and output are now computed as:

$$\mathbf{h}_t^f = f_h(\mathbf{h}_{t-1}^f, \mathbf{x}_t) \tag{3.4}$$

$$\mathbf{h}_t^b = f_h(\mathbf{h}_{t+1}^b, \mathbf{x}_t) \tag{3.5}$$

$$\mathbf{y}_t = f_o(\mathbf{h}_t^f \oplus \mathbf{h}_t^b) \tag{3.6}$$

where notation is shared with previous equations.

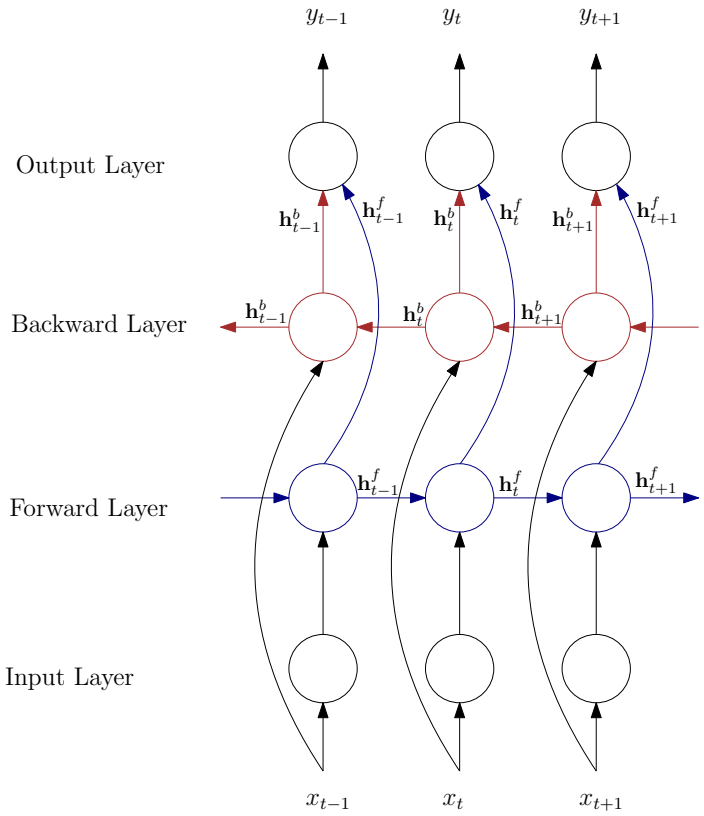



**Figure 3.3:** BRNN architecture unfolded through time. Forward components are colored in blue and backward ones in red.

## Long Term Dependencies: LSTM & GRU

As previously explained we employ RNNs because of its ability to model sequential relations through time, but it has been demonstrated that classical RNNs have problems to model long term dependencies (Bengio et al., 1994). As longer the dependency is, more difficult is to capture due to the so-called vanishing gradient or the exploding gradient effect.

1. **Vanishing gradient effect:** When the network produces an output, it gets the gradient of the loss with regard to the weights that should be back-propagated along its steps. At each step we back-propagate, the gradient gets smaller since we are multiplying numbers smaller than one. As the gradient is applied to deep layers of the network, is so small that it has no effect on the parameters that should be updated.

2. **Exploding gradient effect:** The opposite effect, if we continuously multiply numbers larger than one we will get numbers too big that will affect too much the parameters of the network. It produces numerical instability.

To deal with the exploding gradient problem, we clip the gradient value to a limit. To solve the vanishing gradient problem, new activation functions, which control the flow of information, were proposed. They ensure the derivative of the recurrent function is close to one and avoid the gradient from being vanished. They have become so popular that when talking about RNNs it usually implies the use of one of these units. We present the two most popular options:

### LSTM: Long Short-Term Memory

LSTMs were proposed to address those issues (Hochreiter and Schmidhuber, 1996), a later modification of LSTMs introduced the forget gate (Gers et al., 2000) and became the de facto standard.
LSTMs share the recurrent structure of a regular RNN but each simple cell is substituted by a number of gates which have the ability to add or remove information from the cell state. Cell state $C_t$ acts like a conveyor belt between different neurons and allows a easy flow of the information. Figure 3.4 illustrates the structure of a LSTM cell. Different gates exist, each one with a specific purpose, and they are all composed by a point-wise sigmoid function $\tau$, a point-wise multiplication $\odot$ operation and the addition of a bias vector. Sigmoid is applied to get a value between 0 and 1 representing how much information flows.

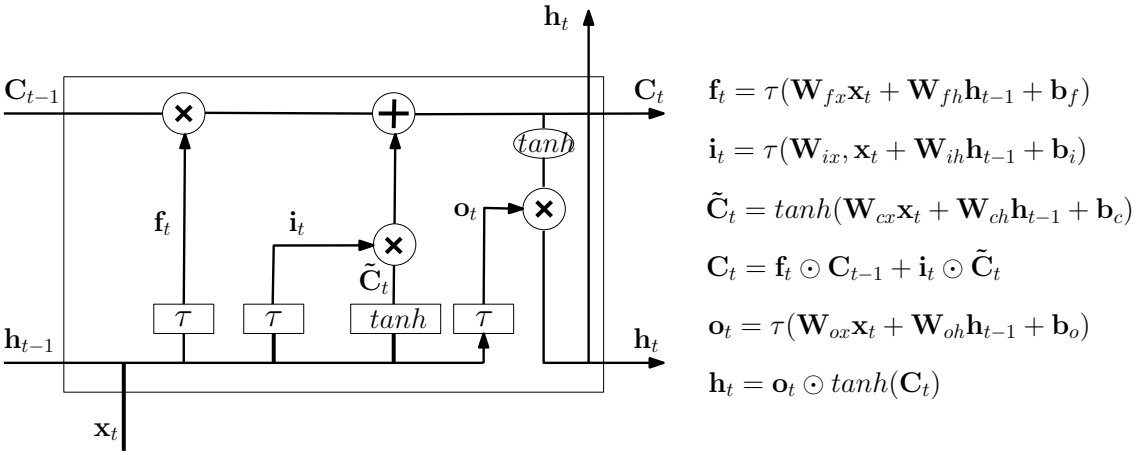



**Figure 3.4:** LSTM architecture representing its gates and computations.

1. **Forget gate ($\mathbf{f}_t$):** This gate decides how much information of the previous cell state $\mathbf{C}_{t-1}$ is going to be forgotten. To make this call the gate uses previous hidden state $\mathbf{h}_{t-1}$ an the input $\mathbf{x}_t$.

2. **Input gate ($\mathbf{i}_t$):** It regulates which part of the input $\mathbf{h}_{t-1}$ passes, how much of the old context affects the new.

3. **Memory gate ($C_t$):** It generates the new memory jointly with the input gate. It decides what new information is going to be stored in the cell state by applying the hyperbolic tangent function.

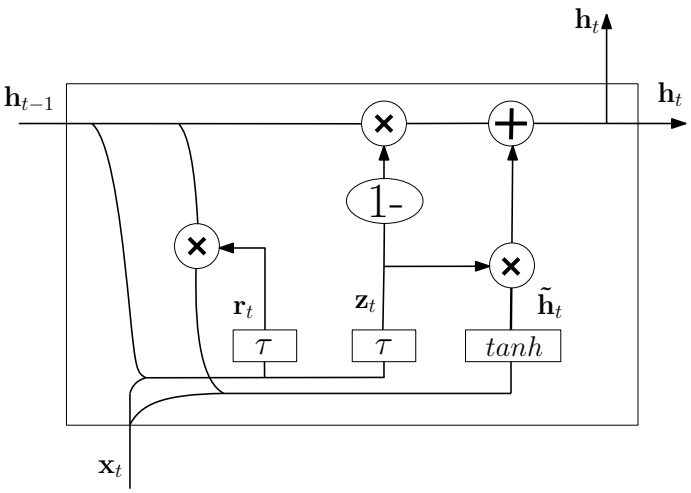
4. **Output gate ($o_t$):** Finally we compute the output of the cell. It is based on the cell state but it is a filtered version.

### GRU: Gated Recurrent Units

LSTMs provide an effective solution to the vanishing gradient problem but they are also rather complicated. Gated recurrent units were proposed (Cho et al., 2014) as an effective simplification (Chung et al., 2014). It combines the forget and the input gate into a single update gate $\mathbf{z}_j$, it also merges the cell and the hidden state. Due to these simplifications it requires less time and computational power to be trained. Figure 3.5 shows the structure of a GRU cell.

1. **Reset gate ($\mathbf{r}_t$):** It decides if the previous hidden state $\mathbf{h}_{t-1}$ is ignored.

2. **Update gate ($\mathbf{z}_t$):** It selects whether the hidden state is updated with a new hidden state $\tilde{\mathbf{h}}$.




$$\mathbf{z}_t = \tau(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\mathbf{h}_{t-1} + \mathbf{b}_z)$$

$$\mathbf{r}_t = \tau(\mathbf{U}_r\mathbf{x}_t + \mathbf{W}_r\mathbf{h}_{t-1} + \mathbf{b}_r)$$

$$\tilde{\mathbf{h}}_t = tanh(\mathbf{U}_h\mathbf{x}_t + \mathbf{W}_h(\mathbf{h}_{t-1} \odot \mathbf{r}_t))$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t + \mathbf{z}_t \odot \mathbf{h}_{t-1}$$

**Figure 3.5:** GRU architecture representing its gates and computations.

## Encoder-Decoder Architecture

The encoder-decoder is the most common framework in NMT. It is based on a simple concept: first, we encode the source sentence into a real-valued vector and second, that information is decoded into a target sentence. Different approximations exist on how to implement the encoder and the decoder, but the same idea underlies, Figure 3.6 illustrates the concept. Two similar models were proposed independently by (Sutskever et al., 2014) and (Cho et al., 2014), the differences between them are architectural. Later (Bahdanau et al., 2014) extended this model

by allowing the decoder to dynamically search in the source sentence while decoding a translation. This system is described in detail in next sections.

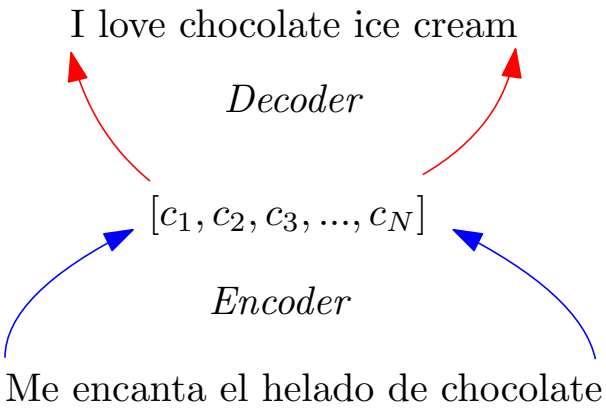



**Figure 3.6:** Encoder-Decoder architecture.

The encoder-decoder system is trained to maximize the conditional log-likelihood over a set of bilingual phrases $\mathcal{X} = \{(x_1, y_1), ..., (x_N, y_N)\}$. We are looking for the optimal set of parameters that maximizes that probability over the training set:

$$\hat{\theta} = \arg\max_{\Theta} \frac{1}{N} \sum_{n=1}^{N} log p_{\Theta}(y_n|x_n) \tag{3.7}$$

## Encoder

The encoder is a RNN in charge of processing the source sentence $f_1, f_2, ..., f_J$ and encode it in a context vector $\mathbf{c}$. The encoder receives as input the sequence of word embeddings of $f_1^J$, which are jointly trained with the network. After processing them, when the sequence has been read, the context vector represents kind of a synopsis of the original sentence. The context vector $\mathbf{c}$ was originally the last hidden state of the encoder $\mathbf{c} = \mathbf{h}_J$ (Sutskever et al., 2014), but a better approach is presented in the next section.

Different options exist on how to implement the encoder, however (Bahdanau et al., 2014) decided to use a bidirectional RNN (explained in Section 3.2.1) to capture sequence relations in both directions. The hidden states computations of the BRNN encoder are the same as the presented in the aforementioned section with the exception of Equation (3.6), where $\oplus$ is instantiated as a concatenation operator and no output function is applied.

$$\mathbf{h}_j = \mathbf{h}_j^f \oplus \mathbf{h}_j^b \tag{3.8}$$

For the encoder, in Equations (3.4) and (3.5) the activation function $f_h$ is replaced by a LSTM or GRU (explained in Section 3.2.2) to deal with long phrases.

Forward layer processes the sequence from left to right and the backward layer the other way around. Finally the hidden states are computed by concate-

nating the output of the forward and backward layer. This way for each word $f_j$ an annotation $\mathbf{h}_j$ is computed.

## Decoder

The decoder is in charge of generating the output sentence $e_i^I$ taking into account the context vector $\mathbf{c}$ and the previously generated words. If we model it using a RNN we can express the conditional probability as:

$$p(e_i|e_1^{i-1}, \mathbf{c}) = \mathbf{e}_i^\top g(\mathbf{E}(e_{i-1}), \mathbf{s}_i, \mathbf{c}) \tag{3.9}$$

where $\mathbf{s}_i$ is the hidden state of the decoder RNN, $\mathbf{c}$ is the context vector from the encoder, $\mathbf{e}_i^\top$ is the one-hot representation of the word $e_i$ and $\mathbf{E}(e_{i-1})$ is the word embedding of the target word $e_{i-1}$. As in the encoder, we work with word embeddings. The non-linear function $g$ applied by the decoder RNN is

$$g(\mathbf{E}(e_{i-1}), \mathbf{s}_i, \mathbf{c}) = \sigma(\mathbf{U} * tanh(\mathbf{W}\mathbf{s}_i + \mathbf{V}\mathbf{E}(e_{i-1}) + \mathbf{Z}\mathbf{c})) \tag{3.10}$$

where $\mathbf{U} \in \mathbb{R}^{|V|xl}$, $\mathbf{W} \in \mathbb{R}^{lxn}$, $\mathbf{V} \in \mathbb{R}^{lxm}$ and $\mathbf{Z} \in \mathbb{R}^{lxk}$ are the weight matrices. Being $|V|$ the target language vocabulary size, $l$ the size of the dimension on which we want to project these vectors, $n$ the size of the decoder hidden state $\mathbf{s}_i$, $m$ the word-embedding size and $k$ the size of the context vector $\mathbf{c}$.

The hidden state of the decoder $\mathbf{s}_i$ is computed as follows:

$$\mathbf{s}_i = \phi(\mathbf{X}\mathbf{s}_{i-1} + \mathbf{C}\mathbf{c} + \mathbf{Y}\mathbf{E}(e_{i-1})) \tag{3.11}$$

where $\mathbf{X} \in \mathbb{R}^{nxn}$, $\mathbf{C} \in \mathbb{R}^{nxk}$ and $\mathbf{Y} \in \mathbb{R}^{nxm}$ are the weight matrices and $\phi$ is a non linear activation function. As in the encoder $\phi$ is usually a LSTM or GRU to deal with long-term dependencies.

The problem is the sentence is represented by a fixed size vector $\mathbf{c}$. When working with long sentences this might not be enough to capture all the information in the sentence (Cho et al., 2014). To deal with this issue Bahdanau et al. (2014) proposed a new approach: employ a variable length context vector $\mathbf{c}_i$ for each target word $e_i$. In this way the length of the vector adapts to the sentence length enabling a better modeling of its information. Equation (3.9) is rewritten as:

$$p(e_i|e_1^{i-1}, \mathbf{c}_i) = g(\mathbf{E}(e_{i-1}), \mathbf{s}_i, \mathbf{c}_i) \tag{3.12}$$

The variable context vector $\mathbf{c}_i$ is computed as a weighted sum of the sequence of encoder's output:

$$\mathbf{c}_i = \sum_{j=1}^{J} \alpha_{ij}\mathbf{h}_j \tag{3.13}$$

where $\alpha_{ij}$ is the weight for each annotation $h_j$. These weights are computed as:

$$\alpha_{ij} = \frac{exp(r_{ij})}{\sum_{k=1}^{J} exp(r_{ik})} \tag{3.14}$$

where $r_{ij} = a(\mathbf{s}_{i-1}, \mathbf{h}_j)$ is an alignment model which scores how well the inputs around position j and the output at position i match.

### Soft Alignment

There are different choices on how to implement the alignment model, but since it has to score $JxI$ possible alignments, we are interested in a fast and lightweight system. Bahdanau et al. (2014) implemented it by using a simple multilayer perceptron (MLP) relying on the annotation of the source sentence $\mathbf{h}_j$ and previous RNN decoder state $\mathbf{s}_{i-1}$:

$$a(\mathbf{s}_{i-1}, \mathbf{h}_j) = \mathbf{v}_a^\top tanh(\mathbf{U}_a \mathbf{h}_j + \mathbf{W}_a \mathbf{s}_{i-1}) \tag{3.15}$$

where $\mathbf{U}_a \in \mathbb{R}^{px2k}$, $\mathbf{W}_a \in \mathbb{R}^{pxn}$ and $\mathbf{v}_a \in \mathbb{R}^p$ are the weight matrices of the MLP. Being $p$ the size of the MLP. Figure 3.7 shows the structure of the NMT system with the attention model.

## Decoding Phase

The ultimate goal of the NMT system is to generate translations. Given the source sentence $f_i^J$, the translation $e_1^I$ will be the sentence with maximum posterior probability:

$$\hat{e}_1^I = \arg\max_{I, e_1^I} p(e_1^I | f_1^J, e_1^{I-1}) \tag{3.16}$$

The search space is all the possible sentences in the target language $\mathcal{E}$. The optimal solution is untractable so we must use sub-optimal solutions to generate translations in an acceptable time. Typically, the strategy is *beam search* an approximation that considers the $\mathcal{B}$ best hypotheses at each time step. Possible partial hypotheses are added at each phase but pruning keeps the number of best hypotheses always equal to $\mathcal{B}$. This process continues until the end-of-sequence symbol is generated.

## Ensembling

A simple way to improve NMT systems performance is to combine various NNs outputs. Different networks have different strengths and weaknesses, thus its correct combination may lead to a synergy of those strengths. Different techniques exist on how to combine the NN, here we present one of the simplest ones. During the search phase, at each time step, we sum the output vectors of the $N$ NMT systems, these vectors are the output of the softmax layer and contain the probabilities for all the words in the output vocabulary. We select the word with

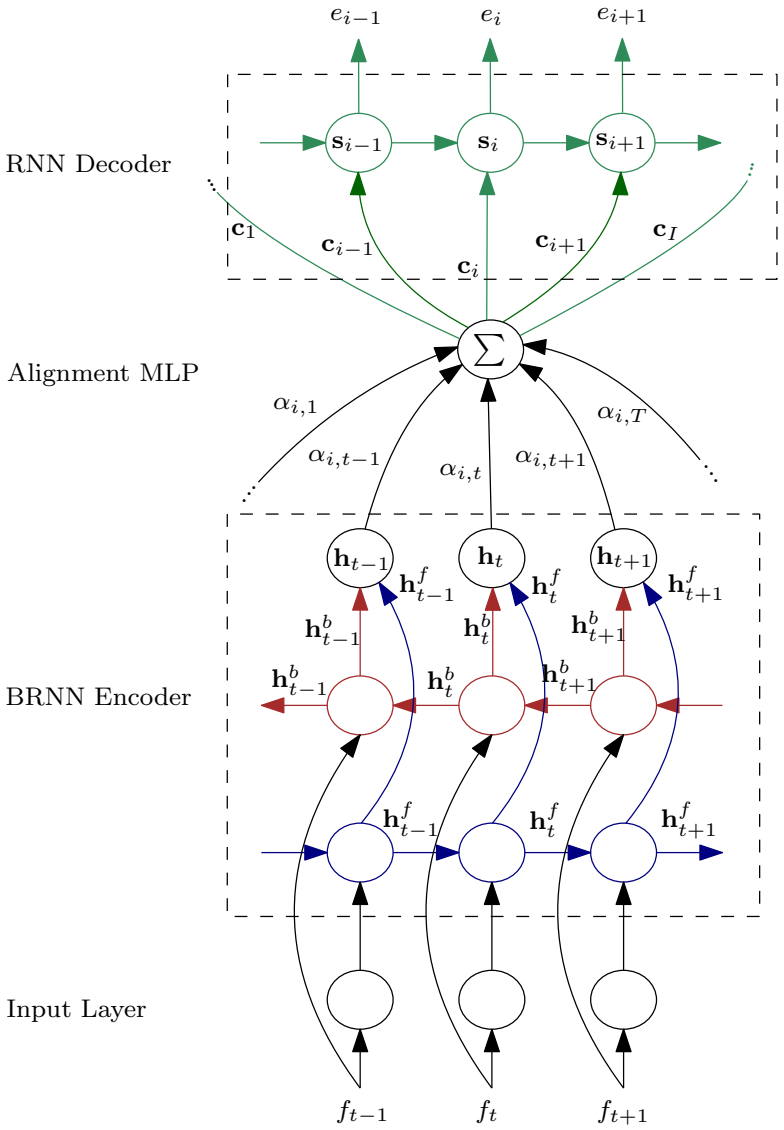



**Figure 3.7:** Encoder-decoder architecture with alignment model.

the highest probability $e_i$. from the resulting vector. Therefore, the final sentence probability can be computed as:

$$p(e_1^I|f_1^J) = \frac{1}{N}\sum_{i=1}^{N} p_i(e_1^I|f_1^J) \tag{3.17}$$

In order to combine different NN, two requisites must be fulfilled: they must share the same vocabulary and they must apply the same factorization to the output layer. There is some empirical component related to which models select and how many of them. However ensembling constitutes a simple and cheap way to obtain better translations.

# CHAPTER 4
# NMT: Sub-word translation

In Chapter 3 the most standard NMT techniques were reviewed. However when using characters and other sub-word units many different approaches exist. After a brief introduction of the topic we will explain the core ideas used in the implementations discussed in Chapter 5.

## Why Character Translation?

The problem with word-level translation is, that we do not know of a prefect word segmentation algorithm for a given language, although extensive research exist for some languages (Creutz and Lagus, 2007) (Finnish), (Zhao et al., 2010) (Chinese). We end up using suboptimal approaches for preprocessing and tokenizing the datasets. This results in having "fly", "flies", "flew", "flying" as different entries in the vocabulary and treated as if they were completely different words which results in:

1. MT systems have difficulties when generalizing into new words since it ignores the word structure.
2. Problems modeling morphological variants efficiently.

These issues can be addressed to a certain extent by using characters and/or other sub-word units. Various works in the literature have studied this issue in different ways: using a hierarchical NMT approach (Ling et al., 2015), a character level decoder (Chung et al., 2016), a character level encoder (Costa-Jussà and Fonollosa, 2016) and even a NMT system which takes advance of Chinese character fonts (Costa-Jussà et al., 2017). Different NMT approximations might be classified as:

1. **Sub-word-level Encoder:** It applies character or sub-word treatment to the source language $\mathcal{F}$, the encoder is able to interpret unseen words.
2. **Sub-word-level Decoder:** It applies character or sub-word treatment to the target language $\mathcal{E}$, the decoder is able to generate new words.
3. **Complete Sub-word-level:** Both, the source and target language are handled at character or sub-word level, the system is able to interpret unseen

words and generate new words. The closest approximation to an open vocabulary system.

## Byte Pair Encoding

A popular technique for working with sub-word units is Byte Pair Encoding (BPE). BPE deals with rare and unknown words by encoding them as a sequence of sub-word units. This is based on the intuition that some word classes are translatable via smaller units: compounds, loanwords or cognates.

BPE was used by Sennrich et al. (2016) with one goal: to propose a simple and effective way to achieve open vocabulary. To do so they employed a data compression algorithm on words: BPE (Gage, 1994). BPE is an iterative algorithm which at each time step takes the two most frequent symbols and merges them, this way the algorithm learns the substitution rules from the datasets separated in characters. Pairs that cross word boundaries are not considered. BPE is a statistical approach, this results in frequent character n-grams (or words) merged into a single symbol. On the other hand, rare n-grams result in short symbols or single characters. Figure 4.1 shows an example of possible results. Here we present a simple sequence of steps to apply BPE over a dataset:

1. Text is tokenized in characters.
2. Set desired number of merges. The only hyper-parameter of the algorithm.
3. Search for the most frequent pair of symbols.
4. Merge the pair as a new single symbol.
5. Go to step 3 until the number of merge operations is reached.

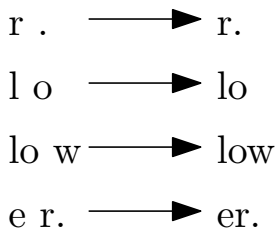



**Figure 4.1:** BPE merge operations learned from dictionary {'low', 'lowest', 'newer', 'wider'}. Example borrowed from Sennrich et al. (2016)

BPE can be learnt separately for each language or jointly. In this work we use joint BPE because of two reasons: it has reported, in general, better results and if they were equal words in the source and target datasets we ensure they would be splitted in the same way (Sennrich et al., 2016). The problem is to determine how many merge operations are necessary to obtain good results. Too few merges result in a small vocabulary and long sentences (it slows the translation process) that might not be powerful enough to capture language richness, on the other hand too many merges might result in almost the original vocabulary providing no improvement.

# Convolutional Encoder

Another choice is to use a convolutional encoder. The main idea is to replace the traditional word-embedding with a new embedding based on the information we can gather from characters. These new embeddings are obtained by applying some convolutional filters, once the new word embeddings are computed the translation works as a classical NMT system. Word-embeddings are no longer a simple projection to the continuous space. The original idea of applying these filters was proposed by Kim et al. (2016) to create a character-aware language model, later it was used in an NMT system (Costa-Jussà and Fonollosa, 2016). Before detailing this process, we briefly introduce two mandatory concepts: convolutional neural networks and highway layers.

## Convolutional Neural Networks

Convolutions are a mathematical operator that involve two functions and produce a third one. It can be seen as sliding one function on top of another, multiplying and adding. This concept has been applied to NN, resulting in the so-called convolutional neural networks (CNN) (LeCun et al., 1998). CNNs have proved useful in many applications such as image classification (Krizhevsky et al., 2017), sentence classification (Kim, 2014) or sentence modeling (Kalchbrenner et al., 2014) among others.

CNNs alternate the use of convolutional and pooling layers to obtain a higher level representation of the input containing sensitive information. Convolutional layers apply the same neuron to all the dimensions of the input. The neuron acts like a convolutional operator, also called kernel, over the input as illustrates Figure 4.2. The kernel slides over the input, new values are obtained multiplying the kernel and the input.

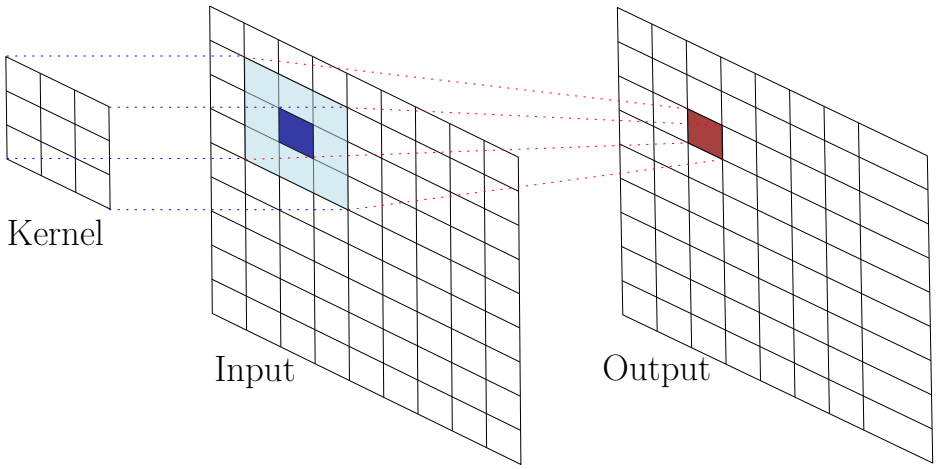



**Figure 4.2:** Example of convolutional operator applied by a convolutional layer.

The pooling operator is used to reduce the size of the inputs. This has three advantages: it reduces the computational cost, it deals with multi-scale and allows to capture higher level features. It acts like a window sliding over the input and

taking the maximum (or average) value. The size of the window decides the output size. Figure 4.3 illustrates the effect of a 2x2 pooling window over a two dimensional input.

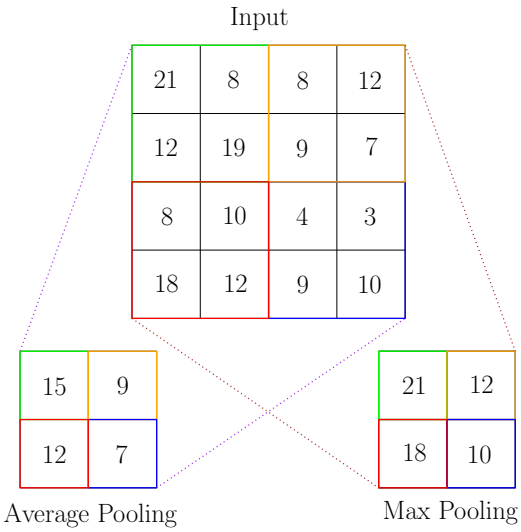



**Figure 4.3:** Example of pooling operator.

Convolutions can be generalized to other dimensions. In the case of the convolutional encoder explained in Section 4.3 we work with one-dimensional convolutions. The same principles apply.

## Highway Networks

Highway networks were proposed (Srivastava et al., 2015) with the aim of reducing the difficulty of training very deep neural networks. They allow the flow of unimpeded information across several layers, by doing so they avoid the information vanishing across the network.

Ordinary layers in a NN apply a linear combination to the input $\mathbf{x}$ (parametrized by the weight matrix $\mathbf{W}_F$), after a non-linear function $F$ is applied.

$$\mathbf{y} = F(\mathbf{x}, \mathbf{W}_F) = \phi(\mathbf{W}_F\mathbf{x} + \mathbf{b}_F) \tag{4.1}$$

A highway layer adds two additional gates, the *transform gate* $T$ and the *carry gate* $C$ . They express how much information is transformed and how much is just carried respectively. Usually, for the sake of simplicity $C = 1 - T$.

$$\mathbf{y} = F(\mathbf{x}, \mathbf{W}_H) * T(\mathbf{x}, \mathbf{W}_T) + \mathbf{x} * (1 - T(\mathbf{x}, \mathbf{W}_T)) \tag{4.2}$$

This way the layer regulates $T$ values, smoothly varying its behavior between an ordinary layer and a dummy layer that simply passes the input.

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W}_T) = 0 \\ \\ F(\mathbf{x}, \mathbf{W}_H), & \text{if } T(\mathbf{x}, \mathbf{W}_T) = 1 \end{cases} \tag{4.3}$$

## Constructing convolutional encoders for NMT

After introducing the concepts of CNN and Highway layers, we present the approach followed in (Costa-Jussà and Fonollosa, 2016) to construct the word embeddings. Figure 4.4 illustrates the following process.

1. Text is tokenized in characters and embeddings are now computed over characters. Let $d$ be the embedding size of the characters.

2. Sequences of character embeddings are grouped for each word. We obtain, for each word, a sequence of character embeddings of size:

   $$dx|w|$$

   Being $|w|$ the length of the word.

3. We apply $h$ one-dimensional convolutions of length $l$ to that sequence. Resulting in $k$ convolutional filters applied.

   $$k = \sum_{l,h} lh$$

   For each different length $l$, we get a matrix of filters of size

   $$hx(|w| - l_i + 1)$$

   The length $l$ of the filters and the number $h$ of them are hyper-parameters of the model.

4. We apply max pooling over time for each one of the filter matrices. This results in a vector of size $h$ for each length $l$.

5. The concatenation of these vectors constitutes the word-embedding. The size of the word-embedding is determined by $k$.

6. Usually two Highway Layers are applied to the previous vector since it has proved to improve system overall performance.

In Kim et al. (2016) fifteen is proposed as the embedding size for the characters. For the convolutional filters they use seven filters of variable length, from one to seven. Number of filters varies depending on the dataset size.

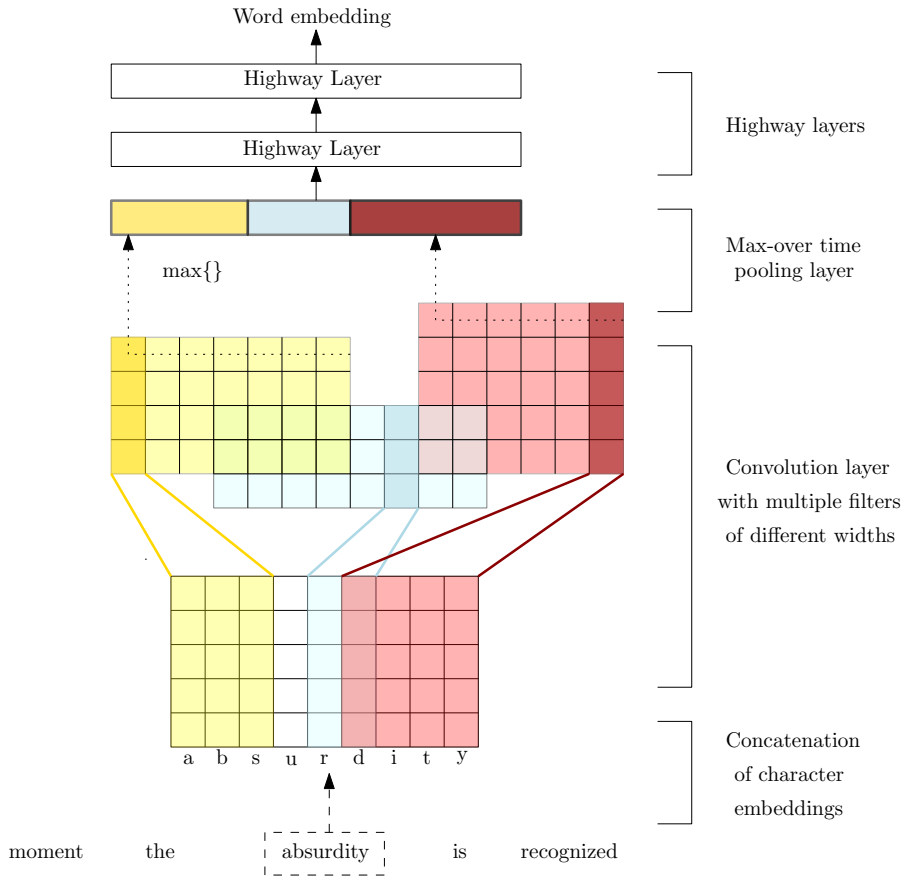



**Figure 4.4:** Process to obtain the word-embedding from characters through convolutions. Note that in the above example we have twelve filters, three filters of width two (blue), four filters of width three (yellow), and five filters of width four (red). Figure adapted from Kim et al. (2016)

# CHAPTER 5
# Implementation, experiments and results

In this chapter we discuss and compare the MT implementations based on the concepts previously explained.

## Implementation details

For the implementation of phrase-based MT systems we used the popular toolkit Moses[1] (Koehn et al., 2007). For the NMT systems we used Keras[2], a high level API written in Python, on top of Theano[3] (Bastien et al., 2012), a Python library for efficiently training and evaluating neural networks. A toolkit was used to work with Keras, (Peris, 2017). The neural models were trained on a GeForce GTX TITAN X.

## NMT Systems

Each model was based on the NMT encoder-decoder architecture with attention mechanism proposed by Bahdanau et al. (2014). For the encoder-decoder we chose GRU as activation function for the RNNs of the encoder and the decoder. The size of the hidden state of each GRU and the word embedding size of the source and target language is 256 for Xerox datasets and 350 for the Eu corpus. These parameters are extracted from (Peris et al., 2017) different hyper-parameters were tested. In the case of working with single characters the word embedding was halved. For the special case of the convolutional encoder the word-embedding was determined by the number of convolutional filters applied, this is 700 for the small set of parameters and 1100 for the large one.

The models have been trained using the Adam algorithm (Kingma and Ba, 2014) with learning rate of $10^{-3}$. The algorithm and the learning rate selection were the result after some previous experimentation process, where other algorithms

[1] http://www.statmt.org/moses/
[2] https://keras.io/
[3] http://www.deeplearning.net/software/theano/

and learning rates were tested. During the training phase Gaussian noise (Zur et al., 2009) and batch normalization (Ioffe and Szegedy, 2015) was applied for preventing the NN from over-fitting. We used 32 as batch size and 6 for the beam size of all the models. The training was early stopped if after 20 epochs for Xerox or 10 for Eu the BLEU did not improved. BLEU was computed over the development set at the end of each epoch. When the models encountered an unseen token we simply substituted it for a special unknown symbol.

Regarding the sentence loading, the convolutional encoder had some peculiarities. In standard NMT systems sentences were loaded in a batch as vectors of indexes. Each index represented a word that was later replaced by its corresponding word-embeddding. Now, since the embedding is computed at character level, sentences are represented by a bi-dimensional matrix of indexes. One row for each word, one column for each character. We apply padding to align the indexes on the matrix.

One drawback of the convolutional encoder is the addition of a set of extra hyper-parameters. The embedding size for the characters, the number and size of convolutional filters and the padding of the sentence. For the embedding size, we chose 15, for the size of the filters we used 7 filters of size from 1 to 7. These values are extracted from the literature. Regarding the padding of the sentence, after doing some tests, we decided to center the words and the characters since it provided the best results. We allowed a maximum of 30 words per sentence and 15 characters per word. Longer words and sentences are truncated. Finally, for the number of the convolutional filters we tried two sets of hyper-parameters, the small one: {25, 50, 75, 100, 125, 150, 175} and the large one: {50, 100, 150, 200, 200, 200, 200}. Both extracted from the original proposal (Kim et al., 2016)

## Datasets

To test the experimentation process the MT systems were evaluated against two tasks: *EU* and *Xerox*. The corpora were tokenized and randomly selected partitions for training, developing and testing phase were extracted. We tried as many language pairs as possible and specially attention was dedicated to German since it is a fusional language. This is a language where morphemes of different types are joined to create new words. It seems logical to think that sub-word units will be more advantageous in these kind of languages. For this reason, when working with a language pair containing German both translations directions were tested.

Now we present the datasets and their statistics. From this information we can infer how difficult a task is: from the length of the sentence, the data available to train or observing how many words from the source language correspond to words in the target language. The information is presented at the three levels of tokenization we use in this work: word level, character level and BPE level. The only hyper-parameter of the BPE algorithm is the number of merge operations. To be exhaustive in the experimentation we tried three different sets of parameters. Since Xerox datasets are smaller than Eu datasets each one has a different number of merge operations. For Xerox we tried 2000, 4000 and 8000 merges, analogously for Eu we tried 10000, 13000 and 20000 merges. BPE does not to-

kenize symmetrically the datasets, the target development and test set are not tokenized because we want the references to be at word level. For this reason, in language pairs where just one translation direction is tested (Xerox En-Es and Xerox En-Fr) some data in their BPE table appears at word levels.

## Xerox

Xerox dataset (Barrachina et al., 2009) consists in the translation of sentences extracted from the user manual of Xerox printers. Next we present the different language pairs and their statistics.

| | | Train | | Development | | Test | |
|---|---|---|---|---|---|---|---|
| | | Word | Char | Word | Char | Word | Char |
| | \|S\| | 55.6k | | 1k | | 1.1k | |
| English | \|T\| | 665k | 3.5M | 14.4k | 77.1k | 8.3k | 47.7k |
| | \|V\| | 14k | 99 | 1.6k | 78 | 1.8k | 75 |
| | \|OOV\| | - | - | 61 | 0 | 206 | 0 |
| | \|CPT\| | 4.3 | 1 | 4.3 | 1 | 4.7 | 1 |
| Spanish | \|T\| | 750k | 4.2M | 16k | 91.6k | 10.1k | 60.5k |
| | \|V\| | 16.7k | 111 | 1.7k | 89 | 1.9k | 87 |
| | \|OOV\| | - | - | 97 | 0 | 225 | 0 |
| | \|CPT\| | 4.6 | 1 | 4.6 | 1 | 4.8 | 1 |

**Table 5.1:** Xerox corpus English-Spanish statistics at word and character level. Number of sentences |S|, tokens |T|, vocabulary size |V|, out-of-vocabulary words |OOV| and characters per token |CPT| for each partition. k stands for thousands and M for millions.

Table 5.1 shows the statistical analysis of the Xerox-EnEs dataset, both at word and character level. We can appreciate how tokenizing into characters affects the sentences: enlarges the number of tokens, reduces the vocabulary and solves the problem of out-of vocabulary words. In this task, Spanish has longer sentences, longer words and a richer vocabulary. However it also presents more OOV. This may indicate that the English-Spanish translation direction is harder than the opposite one.

Table 5.2 presents the analysis of the Xerox-EnEs dataset when tokenized using BPE. These results are showed for three different levels of merge operations. Since we are just translating from English to Spanish, the development and test sets information of Spanish is at word-level. However the OOV are computed at BPE level always. We can appreciate how more merges result in a bigger vocabulary, shorter sentences, longer symbols and less tokens per sentence. It has also the potential effect of augmenting OOV, but as can be seen BPE reduces OOV to neglectable levels.

| | | | Train | Development | Test |
|---|---|---|---|---|---|
| | | \|S\| | 55.6k | 1k | 1.1k |
| English | 2k | \|T\| | 1M | 20.5k | 14.8k |
| | | \|V\| | 1.5k | 1.1k | 1.1k |
| | | \|OOV\| | - | 0 | 2 |
| | | \|CPT\| | 3.5 | 3.6 | 3.5 |
| | 4k | \|T\| | 855.3k | 17.4k | 12.3k |
| | | \|V\| | 2.8k | 1.6k | 1.5k |
| | | \|OOV\| | - | 0 | 2 |
| | | \|CPT\| | 3.8 | 3.9 | 3.8 |
| | 8k | \|T\| | 747.7k | 15.6k | 10.3k |
| | | \|V\| | 5k | 1.8k | 1.8k |
| | | \|OOV\| | - | 0 | 2 |
| | | \|CPT\| | 4.1 | 4.6 | 4.1 |
| Spanish | 2k | \|T\| | 1.1M | 16k | 10k |
| | | \|V\| | 1.8k | 1.7k | 1.9k |
| | | \|OOV\| | - | 0 | 0 |
| | | \|CPT\| | 3.7 | 4.6 | 4.8 |
| | 4k | \|T\| | 995.6k | 16k | 10k |
| | | \|V\| | 3.2k | 1.7k | 1.9k |
| | | \|OOV\| | - | 1 | 0 |
| | | \|CPT\| | 3.7 | 4.6 | 4.8 |
| | 8k | \|T\| | 842.4k | 16k | 10k |
| | | \|V\| | 5.7k | 1.7k | 1.9k |
| | | \|OOV\| | - | 3 | 7 |
| | | \|CPT\| | 3.7 | 4.6 | 4.8 |

**Table 5.2:** Xerox corpus English-Spanish statistics at BPE level. Number of sentences |S|, tokens |T|, vocabulary size |V|, out-of-vocabulary words |OOV|, and characters per token |CPT| for each partition. k stands for thousands and M for millions.

| | | Train | | Development | | Test | |
|---|---|---|---|---|---|---|---|
| | | Word | Char | Word | Char | Word | Char |
| | \|S\| | 51.7k | | 994 | | 984 | |
| English | \|T\| | 615.3k | 3.3M | 10.9k | 58.3k | 11.1k | 57.7k |
| | \|V\| | 13.9k | 98 | 1.8k | 78 | 1.7k | 77 |
| | \|OOV\| | - | - | 141 | 0 | 206 | 0 |
| | \|CPT\| | 4.3 | 1 | 4.3 | 1 | 4.1 | 1 |
| French | \|T\| | 676.4k | 4.1M | 11.7k | 71.7k | 11.8k | 70.2k |
| | \|V\| | 15.5k | 114 | 1.8k | 88 | 1.7k | 90 |
| | \|OOV\| | - | - | 190 | 0 | 194 | 0 |
| | \|CPT\| | 4.9 | 1 | 4.9 | 1 | 4.7 | 1 |

**Table 5.3:** Xerox corpus English-French statistics at word and character level. Number of sentences |S|, tokens |T|, vocabulary size |V|, out-of-vocabulary words |OOV| and characters per token |CPT| for each partition. k stands for thousands and M for millions.

In Table 5.3 we can observe the dataset analysis of the Xerox-EnFr dataset. Again, character tokenization solves the problem of OOV. As it happened with Spanish, French seems a richer language, in this dataset, than English. A larger vocabulary, longer sentences and words support this hypothesis. However OOV are pretty similar in the test set.

Table 5.4 shows the effect of BPE tokenization in the Xe-EnFr dataset. BPE seems to reduce OOV to just a few. The same effect we saw on the Xe-EnEs pair applies, more merges make sentences shorter and increase the vocabulary size. Notice how with just 2000 merges we reduce the number of tokens by four, in the case of French, with respect to the character tokenization. However 8000 merges are not enough to reach the word level representation, as expected the number of merges do not have a linear relation with the number of tokens.

| | | | Train | Development | Test |
|---|---|---|---|---|---|
| | | \|S\| | 51.7k | 994 | 984 |
| English | 2k | \|T\| | 936.1k | 17.1k | 18.2k |
| | | \|V\| | 1.6k | 1.1k | 1.1k |
| | | \|OOV\| | - | 0 | 0 |
| | | \|CPT\| | 3.5 | 3.4 | 3.3 |
| | 4k | \|T\| | 785.7k | 14.5k | 15k |
| | | \|V\| | 2.9k | 1.6k | 1.5k |
| | | \|OOV\| | - | 1 | 4 |
| | | \|CPT\| | 3.8 | 3.7 | 3.6 |
| | 8k | \|T\| | 689k | 12.4k | 13k |
| | | \|V\| | 5.3k | 1.9k | 1.8k |
| | | \|OOV\| | - | 2 | 4 |
| | | \|CPT\| | 4.1 | 4 | 3.8 |
| French | 2k | \|T\| | 1M | 11.7k | 11.8k |
| | | \|V\| | 1.9k | 1.8k | 1.7k |
| | | \|OOV\| | - | 0 | 0 |
| | | \|CPT\| | 3.9 | 4.9 | 4.7 |
| | 4k | \|T\| | 856.5k | 11.7k | 11.8k |
| | | \|V\| | 3.3k | 1.8k | 1.7k |
| | | \|OOV\| | - | 1 | 5 |
| | | \|CPT\| | 4.3 | 4.9 | 4.7 |
| | 8k | \|T\| | 755.3k | 11.7k | 11.8k |
| | | \|V\| | 5.9k | 1.8k | 1.7k |
| | | \|OOV\| | - | 9 | 10 |
| | | \|CPT\| | 4.9 | 4.9 | 4.7 |

**Table 5.4:** Xerox corpus English-French statistics at BPE level. Number of sentences |S|, tokens |T|, vocabulary size |V|, out-of-vocabulary words |OOV|, and characters per token |CPT| for each partition. k stands for thousands and M for millions.

| | | Train | | Development | | Test | |
|---|---|---|---|---|---|---|---|
| | | Word | Char | Word | Char | Word | Char |
| | \|S\| | 50.2k | | 964 | | 995 | |
| English | \|T\| | 592.4k | 3.1M | 10.8k | 56.7k | 12.5k | 64.4k |
| | \|V\| | 13.7k | 97 | 1.5k | 78 | 1.8k | 77 |
| | \|OOV\| | - | - | 28 | 0 | 139 | 0 |
| | \|CPT\| | 4.3 | 1 | 4.2 | 1 | 4.1 | 1 |
| German | \|T\| | 531.7k | 3.6M | 10.4k | 73.7k | 11.7k | 76.1k |
| | \|V\| | 25.1k | 108 | 1.7k | 87 | 2.1k | 89 |
| | \|OOV\| | - | - | 146 | 0 | 483 | 0 |
| | \|CPT\| | 5.8 | 1 | 5.9 | 1 | 5.4 | 1 |

**Table 5.5:** Xerox corpus English-Deutsch statistics at word and character level. Number of sentences |S|, tokens |T|, vocabulary size |V|, out-of-vocabulary words |OOV| and characters per token |CPT| for each partition. k stands for thousands and M for millions.

| | | | Train | Development | Test |
|---|---|---|---|---|---|
| | | \|S\| | 50.2k | 964 | 995 |
| English | 2k | \|T\| | 929.5k | 16.3k | 19.6k |
| | | \|V\| | 1.4k | 1k | 1k |
| | | \|OOV\| | - | 0 | 3 |
| | | \|CPT\| | 3.4 | 3.4 | 3.3 |
| | 4k | \|T\| | 779.4k | 14k | 16.5k |
| | | \|V\| | 2.6k | 1.4k | 1.5k |
| | | \|OOV\| | - | 1 | 3 |
| | | \|CPT\| | 3.7 | 3.6 | 3.6 |
| | 8k | \|T\| | 680.2k | 12.2k | 14.5k |
| | | \|V\| | 4.6k | 1.6k | 1.9k |
| | | \|OOV\| | - | 2 | 3 |
| | | \|CPT\| | 4 | 3.9 | 3.8 |
| German | 2k | \|T\| | 996.6k | 17.9k | 21.5k |
| | | \|V\| | 1.9k | 1.1k | 1.3k |
| | | \|OOV\| | - | 0 | 5 |
| | | \|CPT\| | 4 | 4.2 | 3.8 |
| | 4k | \|T\| | 823.1k | 15.2k | 18.2k |
| | | \|V\| | 3.4k | 1.6k | 1.8k |
| | | \|OOV\| | - | 0 | 20 |
| | | \|CPT\| | 4.4 | 4.6 | 4.1 |
| | 8k | \|T\| | 687.5k | 12.8k | 15.4k |
| | | \|V\| | 6.1k | 2k | 2.3k |
| | | \|OOV\| | - | 1 | 39 |
| | | \|CPT\| | 4.9 | 5.1 | 4.5 |

**Table 5.6:** Xerox corpus English-Deutsch & English-Deutsch statistics at BPE level. Number of sentences |S|, tokens |T|, vocabulary size |V|, out-of-vocabulary words |OOV| and characters per token |CPT| for each partition. k stands for thousands.

Table 5.5 represents the different measures we obtained from the Xerox-EnDe and Xerox-DeEn language pairs. Now, both translation directions are tested since the special interest we have on the German language. Despite of being relatively similar on the number of words, German has almost twice the vocabulary than English. It has also longer words and four times the OOV English has. German seems a difficult language to translate. It seems the German to English translation will be easier than the other way around. However, the character tokenization reduces the OVV to zero.

In Table 5.6 we find he results of applying BPE this corpus. As mentioned before, BPE seems to be able to deal with OOV, especially on German. Since now both directions are tested all the results in the table are at BPE level.

## EU

The EU translation task (Barrachina et al., 2009) consists in a selection from the *Bulletin of the European Union*. As we did with Xerox we present next the statistics for the language pairs selected.

| | | Train | | Development | | Test | |
|---|---|---|---|---|---|---|---|
| | | Word | Char | Word | Char | Word | Char |
| | \|S\| | 222.6k | | 400 | | 800 | |
| English | \|T\| | 5.7M | 34.4M | 10.1k | 60.8k | 20.1k | 121k |
| | \|V\| | 51.1k | 143 | 2.5k | 81 | 3.8k | 89 |
| | \|OOV\| | - | - | 33 | 0 | 58 | 0 |
| | \|CPT\| | 4.9 | 1 | 5 | 1 | 5 | 1 |
| German | \|T\| | 5.3M | 38.6M | 9.7k | 70k | 18.3k | 137.1k |
| | \|V\| | 118.6k | 145 | 2.9k | 81 | 4.7k | 88 |
| | \|OOV\| | - | - | 99 | 0 | 182 | 0 |
| | \|CPT\| | 6 | 1 | 6 | 1 | 6.1 | 1 |

**Table 5.7:** Eu corpus English-Deutsch statistics at word and character level. Number of sentences \|S\|, tokens \|T\|, vocabulary size \|V\|, out-of-vocabulary words \|OOV\|, and characters per token \|CPT\| for each partition. k stands for thousands and M for millions.

Table 5.7 shows the statistical analysis to the Eu dataset. Again we translate in both directions, from English to German and from German to English. We can appreciate how much bigger the Eu corpus is compared with Xerox. The training set is four times bigger, the vocabulary size is doubled and sentences are twice longer. However the linguistic properties we saw in Xerox persist, German is more complicated. It has twice the vocabulary size and more OOV. It is significant that with he same characters as English, its vocabulary is twice the size. Compared to Xerox-EnDE and Xerox-DeEn we have more data to train, longer sentences, a bigger vocabulary, a smaller test set and less OOV.

| | | | Train | Development | Test |
|---|---|---|---|---|---|
| | | \|S\| | 222.6k | 400 | 800 |
| English | 10k | \|T\| | 6.8M | 12.1k | 23.9k |
| | | \|V\| | 6.8k | 2.7k | 3.6k |
| | | \|OOV\| | - | 0 | 1 |
| | | \|CPT\| | 4.5 | 4.5 | 4.5 |
| | 13k | \|T\| | 6.5M | 11.6k | 22.9k |
| | | \|V\| | 8.4k | 2.8k | 3.9k |
| | | \|OOV\| | - | 0 | 1 |
| | | \|CPT\| | 4.6 | 4.6 | 4.6 |
| | 20k | \|T\| | 6.2M | 11k | 21.8k |
| | | \|V\| | 11.8k | 2.8k | 4.2k |
| | | \|OOV\| | - | 0 | 1 |
| | | \|CPT\| | 4.7 | 4.7 | 4.7 |
| German | 10k | \|T\| | 7.3M | 13.1k | 25.7k |
| | | \|V\| | 8.4k | 3.4k | 4.5k |
| | | \|OOV\| | - | 0 | 0 |
| | | \|CPT\| | 5 | 5 | 5 |
| | 13k | \|T\| | 6.9M | 12.4k | 24.2k |
| | | \|V\| | 10.6k | 3.5k | 5.1k |
| | | \|OOV\| | - | 1 | 0 |
| | | \|CPT\| | 5.1 | 5.2 | 5.2 |
| | 20k | \|T\| | 6.4M | 11.5k | 22.4k |
| | | \|V\| | 15.9k | 3.6k | 5.6k |
| | | \|OOV\| | - | 1 | 0 |
| | | \|CPT\| | 5.4 | 5.4 | 5.4 |

**Table 5.8:** EU corpus English-Deutsch & English-Deutsch statistics at BPE level. Number of sentences |S|, tokens |T|, vocabulary size |V|, out-of-vocabulary words |OOV| and characters per token |CPT| for each partition. k stands for thousands and M for millions.

Table 5.8 shows the EU corpus English-Deutsch & English-Deutsch statistics at BPE level. Despite of being a larger corpus the effect of BPE is the same. More merges result in shorter sentences and a larger vocabulary. We can appreciate how BPE reduces the OOV to zero. All the results are at BPE level.

After seeing these corpus statistics we can extract some conclusions. Out-of-vocabulary words are a problem, they go from 1% to 23%, with respect vocabulary size, in test sets. And this is in closed sets, if we are talking about real translation domains these numbers rocket up. Second, the tokenization at character level solves the problem of OOV. It is possible, but unlikely, to find an unknown character at test phase. BPE tokenization situates itself as a compromise between character and word tokenization. BPE usually reduces the problem of unknown words to insignificant levels. Third, Eu dataset is not only larger, but it has longer sentences and words than the Xerox dataset. Finally, German is a more complicated language, compared to English, it presents a richer vocabulary and more OOV.

These results support our approach. They prove the character-based NMT as a valid approximation for dealing with out-of-vocabulary words. These results also demonstrate sub-word approaches bring us closer to open-vocabulary systems.

# Experiments

In this section we will describe each one of the experimentation carried out and the obtained results. The results are expressed as a percentage in BLEU and TER (Section 2.5). We evaluated twice the neural models, one sampling with the best epoch and a second one using model ensembling of the best epochs as explained in section 3.3.4.

## Classical Phrase-Based MT with Moses

The phrase-based system was built using Moses with standard parameters: 5-gram language model with Kneser-Ney discount. The alignments are computed using GIZA++ (Och and Ney, 2003) with the IBM model 4 and bidirectional reordering. Weight estimation of the log-linear model is carried out by MERT over a development set.

We use this model as a reference, a baseline we can compare with other implementations to check if we are working in the right direction. The results obtained can be found in Tables 5.9 and 5.10.

| | BLEU | TER |
|---|---|---|
| Xe-EnEs | 60.1 | 26.1 |
| Xe-EnFr | 36.4 | 50.8 |
| Xe-EnDe | 23.4 | 64.0 |
| Xe-DeEn | 32.6 | 50.1 |

**Table 5.9:** Xerox test set results using Moses.

| | BLEU | TER |
|---|---|---|
| Eu-EnDe | 35.2 | 26.1 |
| Eu-DeEn | 40.7 | 47.3 |

**Table 5.10:** EU test set results using Moses.

**Table 5.11:** Results for the Xerox and Eu datasets using Moses

Xerox-EnEs obtains the best results, probably by the similarity between both languages. Xerox-EnFr comes second in the Xerox task probably for the same reason. Notice how in both tasks, the translation direction from German to English obtain better results than English to German. It is easier translate from German that generate German words due to its fusional nature.

## Word Level NMT

This model is the one explained in section 3.3: a bidirectional encoder connected to the decoder through an attention model. It is a classical NMT architecture we used to translate at word level. Again the goal is to have a baseline to compare

sub-word translation approaches. The results can be found in Tables 5.12 and 5.13.

| | Best Model | | Ensembling | |
|---|---|---|---|---|
| | BLEU | TER | BLEU | TER |
| Xe-EnEs | 51.6 | 36.7 | 59.4 | 29.1 |
| Xe-EnFr | 27.5 | 62.7 | 31.8 | 57.2 |
| Xe-EnDe | 18.0 | 74.3 | 20.9 | 68.0 |
| Xe-DeEn | 30.6 | 56.4 | 33.26 | 51.6 |

**Table 5.12:** Xerox test set results using standard NMT.

| | Best Model | | Ensembling | |
|---|---|---|---|---|
| | BLEU | TER | BLEU | TER |
| Eu-EnDe | 28.6 | 60.0 | 31.7 | 56.2 |
| Eu-DeEn | 33.1 | 56.5 | 37.1 | 51.5 |

**Table 5.13:** Eu test set results using standard NMT.

**Table 5.14:** Results for the Xerox and Eu datasets using standard NMT

The results, using ensembling, are comparable to the ones obtained using Moses (Tables 5.9 and 5.10). Maybe slightly worse, but BLEU usually benefits Moses since both, the evaluation metric and the translation system are based on the concept of n-grams. The same patterns persist: Xerox-EnEs obtains the best results and Xerox-EnFr comes second. Again translate into German is trickier than translate from German. The change in the approach to carry out the translations does not to seem to affect these tendencies. Thus, we can infer that they are data and linguistically oriented.

## Character Level NMT

The same model as before, but now it operates at character level. The sentences are tokenized, the systems translates characters and a posterior detokenization is applied to go back to words and compare with the references. It is a simple approach we implemented to get a baseline for characters translation using NMT. It is a true open vocabulary system, assuming no unknown characters, since it is able to deal with unknown words by splitting them in known characters. It is also capable of creating new words by the concatenation of characters. Results are showed in Tables 5.15 and 5.16.

| | Best Model | | Ensembling | |
|---|---|---|---|---|
| | BLEU | TER | BLEU | TER |
| Xe-EnEs | 41.6 | 48.5 | 52.2 | 32.4 |
| Xe-EnFr | 23.3 | 72.9 | 30.5 | 60.5 |
| Xe-EnDe | 12.9 | 73.4 | 13.8 | 70.4 |
| Xe-DeEn | 17.8 | 62.9 | 20.3 | 59.9 |

**Table 5.15:** Xerox test set results using character NMT.

| | Best Model | | Ensembling | |
|---|---|---|---|---|
| | BLEU | TER | BLEU | TER |
| Eu-EnDe | 10.5 | 76.1 | 11.2 | 75.2 |
| Eu-DeEn | 14.4 | 70.7 | 15.4 | 69.2 |

**Table 5.16:** Eu test set results using character NMT.

**Table 5.17:** Results for the Xerox and Eu datasets using character NMT

As it can be appreciated, the results are worse compared with the previous ones. This might be because despite of being open vocabulary the plain unigram representation does not provide enough information, enough context to model, to capture the relationships between words. By tokenizing the sentences into characters, we lengthen them by a factor of four or five. Longer sentences imply longer term dependencies and extra computational work. In general long sentences are more difficult to translate, especially if the context is reduced to characters. We can observe how in the Eu dataset, where the sentences are much longer, the results get much worse. Despite of the limitations of this approach, it achieved acceptable results in some language pairs.

## BPE NMT

Again, the same model is employed. But now the sentences are tokenized using BPE algorithm explained in Section 4.2. The obtained results can be found in Tables 5.18 and 5.19.

| | 2k merges | | | | 4k merges | | | | 8k merges | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BM | | E | | BM | | E | | BM | | E | |
| | B | T | B | T | B | T | B | T | B | T | B | T |
| Xe-EnEs | 54.2 | 32.2 | 61.7 | 26.1 | 53.8 | 33.1 | 61.5 | 26.2 | 34.8 | 52.9 | 45.7 | 41.2 |
| Xe-EnFr | 26.1 | 61.7 | 31.9 | 56.4 | 27.7 | 61.1 | 23.2 | 56.1 | 24.8 | 62.8 | 33.2 | 53.8 |
| Xe-EnDe | 17.8 | 69.2 | 21.7 | 66.3 | 19.5 | 70.7 | 22.3 | 66.1 | 18.6 | 68.7 | 23.5 | 64.5 |
| Xe-DeEn | 24.1 | 60.9 | 29.7 | 52.7 | 26.5 | 60.3 | 32.7 | 52.3 | 26.7 | 59.1 | 32.8 | 51.8 |

**Table 5.18:** Xerox test set results using BPE encoding. BM stands for best model, E for ensembling, B for BLEU and T for TER.

| | 10k merges | | | | 13k merges | | | | 20k merges | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BM | | E | | BM | | E | | BM | | E | |
| | B | T | B | T | B | T | B | T | B | T | B | T |
| Eu-EnDe | 31.7 | 56.4 | 34.9 | 52.7 | 32.6 | 55.2 | 35.3 | 52.6 | 32.8 | 54.8 | 36.0 | 51.9 |
| Eu-DeEn | 37.2 | 49.5 | 40.1 | 46.8 | 37.5 | 49.5 | 41.4 | 45.9 | 37.3 | 49.7 | 41.6 | 45.5 |

**Table 5.19:** Eu test set results using BPE encoding. BM stands for best model, E for ensembling, B for BLEU and T for TER.

Different number of merge operations affect the final result, we can appreciate how augmenting the merge operations seems to improve the system's outcome. From 2k/10k to 4k/13k merges almost every result gets better and the same occurs with the next increment. The overall results are good. Indeed, they are better than the results obtained with Moses (Tables 5.9 and 5.10), word level NMT (Tables 5.12 and 5.13), and character level NMT (Tables 5.15 and 5.16). BPE offers a trade-off between character and word representations. More merges result in more and longer symbols in the vocabulary, but shorter sentences that have shorter dependencies.

We can appreciate that the biggest hyper-parameters (8000 and 20000 merge operations respectively) offer, in general, better results. The exception is Xerox-EnEs,

where the 8000 merges perform worse than others. Xerox-EnEs was the language pair that obtained the best result translating at character level. This might indicate that this particular task benefits from pretty small sub-word units. BPE seems to be a flexible option that offers tokens small enough to be used to construct words but large enough to offer some context and more information that single characters.

## Convolutional Encoder NMT

We tested the model explained in Section 4.3 with the parameters explained in Section 5.2. The results can be seen in Tables 5.20 and 5.21.

| | Small | | | | Large | | | |
|---|---|---|---|---|---|---|---|---|
| | BM | | E | | BM | | E | |
| | B | T | B | T | B | T | B | T |
| Xe-EnEs | 48.2 | 37.1 | 55.9 | 30.8 | 50.3 | 36.1 | 58.0 | 29.4 |
| Xe-EnFr | 23.0 | 66.4 | 26.4 | 63.1 | 25.0 | 66.2 | 27.1 | 61.8 |
| Xe-EnDe | 18.0 | 72.4 | 19.3 | 69.2 | 19.5 | 70.7 | 21.2 | 67.2 |
| Xe-DeEn | 22.8 | 64.7 | 26.2 | 60.2 | 23.8 | 62.3 | 26.5 | 59.1 |

**Table 5.20:** Xerox test set results using the convolutional encoder. BM stands for best model, E for ensembling, B for BLEU and T for TER.

| | Small | | | | Large | | | |
|---|---|---|---|---|---|---|---|---|
| | BM | | E | | BM | | E | |
| | B | T | B | T | B | T | B | T |
| Eu-EnDe | 27.1 | 61.1 | 29.8 | 58.6 | 26.5 | 61.3 | 30.0 | 57.8 |
| Eu-DeEn | 29.2 | 58.0 | 32.7 | 54.4 | 31.0 | 56.0 | 34.0 | 53.0 |

**Table 5.21:** Eu test set results using the convolutional encoder. BM stands for best model, E for ensembling, B for BLEU and T for TER.

The results are better than the ones obtained simple character tokenization (Tables 5.15 and 5.16) but in general, a bit worse than the rest. Sometimes the difference seems meaning and others it is just a few percentage points of BLEU. Having into account the extra parameters this approach introduces a more in depth experimentation should be done to find the best parameters. This way we will conclude if it is an intrinsic problem of the approach or a bad decision on the parameter selection.

## Convolutional Encoder + BPE NMT

We use the convolutional encoder model again, but this time a previous tokenization, just in the source language, using BPE is applied. This results in the word embedding being computed over the BPE encoding instead of single characters.

We tried this approach after seeing the improvement BPE offered over the simple character translation. We used the large set of parameters of the convolutional encoder since it provided better results in the previous section. For the merges of the BPE algorithm we used the one who performed better in Section 5.4.4 for each language pair. The results are showed in Tables 5.22 and 5.23.

| | Best Model | | Ensembling | |
|---|---|---|---|---|
| | BLEU | TER | BLEU | TER |
| Xe-EnEs | 48.6 | 37.5 | 56.6 | 30.8 |
| Xe-EnFr | 22.1 | 68.1 | 27.1 | 60.9 |
| Xe-EnDe | 16.6 | 72.5 | 19.6 | 66.8 |
| Xe-DeEn | 20.2 | 65.1 | 24.9 | 58.9 |

**Table 5.22:** Xerox test set results convolutional encoder and BPE.

| | Best Model | | Ensembling | |
|---|---|---|---|---|
| | BLEU | TER | BLEU | TER |
| Eu-EnDe | 27.3 | 60.3 | 29.6 | 57.9 |
| Eu-DeEn | 29.1 | 57.2 | 32.3 | 54.4 |

**Table 5.23:** Eu test set results using convolutional encoder and BPE.

**Table 5.24:** Results for the Xerox and Eu datasets using a convolutional encoder and BPE

As can be seen the combination of both approaches does not improve the original proposition of the convolutional encoder (results in Tables 5.20 and 5.21). Again a more exhaustive experimentation should be carried out to test if the results vary significantly when trying other parameters. However, since the sub-word units are used to compute a word embedding at word level it seems possible that BPE does not results in such a great improvement as it was for single character translation.

# CHAPTER 6
# Conclusions & Future Work

## Conclusions

In this Master thesis, four NMT systems based on sub-word units were proposed. The objective of these models was to deal with out-of-vocabulary words and improve neural networks resilience against unseen events. They rely on the concept of encoding words via smaller units. This allowed the models to profit from the information we can gather from characters or sub-word units. Therefore, the proposed models can deal with unseen words through smaller units. The models (except for the convolutional encoder) are also able to generate new words at translation phase.

For doing this, four different approaches were implemented: a NMT using single characters, a NMT using BPE tokenization, an NMT using a convolutional encoder for replacing simple look-up word embeddings and finally a NMT which combines the convolutional encoder and BPE. These models have been tested in as many bilingual datasets as possible to demonstrate that they are valid independently of the pair of languages. Given these new approaches have a high number of hyper-parameters to adjust, we tried to make the experimentation process as thorough as possible. The implemented models have been compared against a classical PB system and a standard NMT system, both at word level. We have measured the quality of the translations as well as how the different approaches affect the sentence length, the word length and the out-of-vocabulary words. NMT systems have been evaluated independently and as an ensembling. This combination aims to enhance the strengths while minimizing the weaknesses of the different models.

According to the results BPE outnumbered any other model. PB and standard NMT also performed pretty well, the convolutional encoder and its combination with BPE come close but they performed generally a bit worse. In last place, as expected, we find the NMT translating single characters. Sub-word based NMT constitutes a valid translation approach on its own. Indeed, it can surpass PB systems or classical NMT at word level. It has also been proved its effectiveness against fusional or agglutinative languages, where morpheme combination is common, and against out-of-vocabulary words.

# Future Work

We have presented how different sub-word approaches can be applied to NMT in order to deal with unseen events. However there is still open lines of work that could extend this Master thesis.

## New Datasets

Due to the high computational cost of training neural networks all the models were estimated using a GPU. Despite of this, models took between 1 to 3 days to be trained. For this reason, the selected corpus, Xerox and Eu, were relatively small. It would be interesting to test the model against bigger and more standard corpus like for example the Europarl (Koehn, 2005).

## Improvements & other sub-word approaches

We have presented a few sub-word systems. However, there are many other approaches that would be worth to compare: full char2char translation (Ling et al., 2015), at encoder and decoder level or convolutional encoder-decoder (Gehring et al., 2017) at character level.

Other techniques can be applied to the presented convolutional encoder: the use dilated convolutions (Yu and Koltun, 2015), variable padding convolutions or replacing the standard GRUs with convolutional LSTMs (Xingjian et al., 2015) .

## Combination of character systems with other paradigms

Character-based NMT could be used as a back-off model when a word-level translation system encounters an unknown word (Luong and Manning, 2016). In such cases, the sub-word system can propose a better translation that the heuristics used for out-of-vocabulary words.
Other possible choice would be combining character-based translation into the interactive machine translation (IMT) (Peris et al., 2017) paradigm. IMT is a paradigm that interactuates with the end user to produce translations. The system proposes a possible translation and the user corrects it. The system proposes a new translation according to user correction and the process repeats until the user is satisfied with the translation.

# CHAPTER 7
# Acknowledgements

I would like to thank Francisco Casacuberta for directing this Master thesis and giving me this opportunity. Also I have to thank Álvaro Peris for co-directing this work, for his help and for his marvelous drawings which helped me a lot understanding some concepts.
Thanks to my family and friends for their support.
**Thanks!**.

# Bibliography


Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *Computing Research Repository-arXiv*, abs/1409.0473.

Banerjee, S. and Lavie, A. (2005). Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, volume 29, pages 65–72.

Barrachina, S., Bender, O., Casacuberta, F., Civera, J., Cubel, E., Khadivi, S., Lagarda, A., Ney, H., Tomás, J., Vidal, E., et al. (2009). Statistical approaches to computer-assisted translation. *Computational Linguistics*, 35(1):3–28.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y. (2012). Theano: new features and speed improvements. *Computing Research Repository-arXiv*, abs/1211.5590.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Bengio, Y., Simard, P. Y., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Institute of Electrical and Electronics Engineers Transactions on Neural Networks*, 5(2):157–166.

Brown, P. F., Pietra, S. D., Pietra, V. J. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Castano, M. A., Casacuberta, F., and Vidal, E. (1997). Machine translation using neural networks and finite-state models. *Theoretical and Methodological Issues in Machine Translation*, pages 160–167.

Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 310–318.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734.

Chung, J., Cho, K., and Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *Computing Research Repository-arXiv*, abs/1412.3555.

Costa-Jussà, M. R., Aldón, D., and Fonollosa, J. A. R. (2017). Chinese-spanish neural machine translation enhanced with character and word bitmap fonts. *Machine Translation*, 31(1-2):35–47.

Costa-Jussà, M. R. and Fonollosa, J. A. R. (2016). Character-based neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 2.

Creutz, M. and Lagus, K. (2007). Unsupervised models for morpheme segmentation and morphology learning. *Association for Computing Machinery. Transactions on Speech and Language Processing*, 4(1):3:1–3:34.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Doddington, G. (2002). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, pages 138–145.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

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

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *Computing Research Repository-arXiv*, abs/1705.03122.

Gers, F. A., Schmidhuber, J., and Cummins, F. A. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471.

Goodman, J. T. (2001). A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434.

Hasler, E., Haddow, B., and Koehn, P. (2011). Margin infused relaxed algorithm for moses. *The Prague Bulletin of Mathematical Linguistics*, 96:69–78.

Hochreiter, S. and Schmidhuber, J. (1996). LSTM can solve hard long time lag problems. In *Advances in Neural Information Processing Systems 9*, pages 473–479.

Hopkins, M. and May, J. (2011). Tuning as ranking. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 655–665.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *Proceedings of the Thirtieth Association for the Advancement of Artificial Intelligence. Conference on Artificial Intelligence*, pages 2741–2749.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *Computing Research Repository-arXiv*, abs/1412.6980.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, pages 181–184.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *Machine Translation summit*, volume 5, pages 79–86.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the Association for Computing Machinery*, 60(6):84–90.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the Institute of Electrical and Electronics Engineers*, 86(11):2278–2324.

Ling, W., Trancoso, I., Dyer, C., and Black, A. W. (2015). Character-based neural machine translation. *Computing Research Repository-arXiv*, abs/1511.04586.

Luong, M.-T. and Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models. *Computing Research Repository-arXiv*, abs/1604.00788.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *Computing Research Repository-arXiv*, abs/1301.3781.

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association*, pages 1045–1048.

Mikolov, T., Le, Q. V., and Sutskever, I. (2013b). Exploiting similarities among languages for machine translation. *Computing Research Repository-arXiv*, abs/1309.4168.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013c). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Papineni, K., Roukos, S., Ward, T., and Zhu, W. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Peris, Á. (2017). NMT-Keras. https://github.com/lvapeab/nmt-keras. GitHub repository.

Peris, Á., Domingo, M., and Casacuberta, F. (2017). Interactive neural machine translation. *Computer Speech & Language*, 45:201–220.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *Institute of Electrical and Electronics Engineers Transactions on Signal Processing*, 45(11):2673–2681.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, volume 200.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *Computing Research Repository-arXiv*, abs/1505.00387.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the Institute of Electrical and Electronics Engineers*, 78(10):1550–1560.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810.

Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *Computing Research Repository-arXiv*, abs/1511.07122.

Zens, R., Och, F. J., and Ney, H. (2002). Phrase-based statistical machine translation. In *Advances in Artificial Intelligence, 25th Annual German Conference*, pages 18–32.

Zhao, H., Huang, C., Li, M., and Lu, B. (2010). A unified character-based tagging framework for chinese word segmentation. *Association for Computing Machinery. Transactions on Asian and Low-Resource Language Information Processing*, 9(2):5:1–5:32.

Zur, R. M., Jiang, Y., Pesce, L. L., and Drukker, K. (2009). Noise injection for training artificial neural networks: A comparison with weight decay and early stopping. *Medical Physics*, 36(10):4810–4818.