UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Online Learning in Neural Machine Translation

MASTER THESIS

Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Author: Luis Cebrián Chuliá

Tutor: Francisco Casacuberta Nolla
Álvaro Peris Abril

Course 2016-2017

Resum

La traducció de gran qualitat es troba molt demanada en l'actualitat. Tot i que la traducció automàtica oferix unes prestacions acceptables, en alguns casos no és suficient i és necessària la supervisió humana. Per a facilitar la tasca de traducció de l'humà, els sistemes de traducció automàtica prenen part en aquest procés. Quan una nova oració en el llenguatge origen necessita ser traduïda, esta s'introduïx en el sistema, el qual obté com a eixida una hipòtesi de traducció. Llavors, l'humà corregix aquesta hipòtesi (també conegut com a post-editar) per a obtindre una traducció de major qualitat. Ser capaços de transferir el coneixement que l' humà exhibix quan realitza la tasca de post-edició al sistema de traducció automàtica és una característica desitjable ja que s'ha demostrat que un sistema de traducció mes precís ajuda a augmentar l'eficiència del procés de post-edició.

Pel fet que el procés de post-edició requerix un sistema ja entrenat, les tècniques d'aprenentatge en línia són les adequades per aquesta tasca. En este treball, es proposen tres algoritmes d'aprenentatge en línia aplicats a un traductor automàtic neuronal en un escenari de post-edició. Estos algoritmes es basen en l'aproximació en línia *Passive-Aggressive* en la qual el model s'actualitza després de cada mostra amb l'objectiu de complir un criteri de correcció al mateix temps que manté informació prèvia apresa. L'objectiu és adaptar i refinar un sistema ja entrenat amb noves mostres al vol mentre el procés de post-edició es du a terme (per tant, el temps d'actualització ha de mantenir-se controlat).

A més, estos algoritmes es comparen amb altres ben conegudes variants en línia de l'algoritme de descens per gradient estocàstic. Els resultats mostren una millora en la qualitat de les traduccions després d'aplicar estos algoritmes, reduint així l'esforç humà en el procés de post-edició.

Paraules clau: Aprenentatge en línia, Traducció automàtica neuronal, *Passive-Aggresive*

Resumen

La traducción de gran calidad está muy demandada en la actualidad. A pesar de que la traducción automática ofrece unas prestaciones aceptables, en algunos casos no es suficiente y es necesaria la supervisión humana. Para facilitar la tarea de traducción del humano, los sistemas de traducción automática toman parte en este proceso. Cuando una nueva oración en el idioma origen necesita ser traducida, esta se introduce en el sistema, el cual obtiene como salida una hipótesis de traducción. El humano entonces, corrige esta hipótesis (también conocido como post-editar) para obtener una traducción de mayor calidad. Ser capaz de transferir el conocimiento que el humano exhibe cuando realiza la tarea de post-edición al sistema de traducción automática es una característica deseable puesto que se ha demostrado que un sistema de traducción mas preciso ayuda a aumentar la eficiencia del proceso de post-edición.

Debido a que el proceso de post-edición requiere un sistema ya entrenado, las técnicas de aprendizaje en línea son las adecuadas para esta tarea. En este trabajo, se proponen tres algoritmos de aprendizaje en línea aplicados a un traductor automático neuronal en un escenario de post-edición. Estos algoritmos se basan

en la aproximación en línea *Passive-Aggressive* en la cual el modelo se actualiza después de cada muestra con el objetivo de cumplir un criterio de corrección a la vez que manteniendo información previa aprendida. El objetivo es adaptar y refinar un sistema ya entrenado con nuevas muestras al vuelo mientras el proceso de post-edición se lleva a cabo (por tanto, el tiempo de actualización debe mantenerse bajo control).

Además, estos algoritmos se comparan con otras bien conocidas variantes en línea del algoritmo de descenso por gradiente estocástico. Los resultados muestran una mejora en la calidad de las traducciones después de aplicar estos algoritmos, reduciendo así el esfuerzo humano en el proceso de post-edición.

Palabras clave: Aprendizaje en línea, Traducción automática neuronal, *Passive-Aggresive*

Abstract

High quality translations are in high demand these days. Although machine translation offers acceptable performance, it is not sufficient in some cases and human supervision is required. In order to ease the translation task of the human, machine translation systems take part in this process. When a sentence in the source language needs to be translated, it is fed to the system which outputs a hypothesis translation. The human then, corrects this hypothesis (also known as post-editing) in order to obtain a high quality translation. Being able to transfer the knowledge that a human translator exhibit when post-editing a translation to the machine translation system is a desirable feature, as it has been proven that a more accurate machine translation system helps to increase the efficiency of the post-editing process.

Because the post-editing scenario requires an already trained system, online learning techniques are suited for this task. In this work, three online learning algorithms have been proposed and applied to a neural machine translation system in a post-editing scenario. They rely on the Passive-Aggressive online learning approach in which the model is updated after every sample in order to fulfil a correctness criterion while remembering previously learned information. The goal is to adapt and refine an already trained system with new samples on-the-fly as the post-editing process takes place (hence, the update time must be kept under control).

Moreover, these new algorithms are compared with well-stablished online learning variants of the stochastic gradient descent algorithm. Results show improvements on the translation quality of the system after applying these algorithms, reducing human effort in the post-editing process.

Key words: Online Learning, Neural Machine Translation, Passive-Aggresive

Contents

List of Figures

List of Tables

CHAPTER 1
Motivation

We have seen in modern history how the improvement of transportation and telecommunications infrastructure has had a big impact in the movement of people. Now we are more exposed to other cultures, products and world views than ever before. As technology improves, the access to any kind of information has become something we can't live without. However, different cultures often mean different languages which implies the appearance of obstacles when the need to communicate is mandatory. Thus, the need for a tool that helps us to overcome this obstacle arises. Machine Translation (MT) investigates the use of software to translate text or speech from one language into another. It aims at providing the best possible translation without human assistance.

This research field was born in 1947 when Warren Weaver published a memorandum stating his beliefs about computer's capability to translate one language into another using criptography, logic and linguistic patterns. In the next years a lot of projects emerged in the United States, the Soviet Union and Western Europe but the practical results were disappointing (Madsen, 2009; Hutchins, 2005). This finally led to the publication of a report from a special committee formed by the United States named ALPAC (Automatic Language Processing Advisory Committee) where they stated that there was no future for good-quality/cost-effective MT.

During the 70's there was a new approach to the MT field that relied on the premise that a language is based on a set of grammatical and syntactic rules. This rule-based system needed a robust and carefully designed bilingual dictionary of linguistic information created by human experts. The big effort that this approach required was a big issue and it was then replaced by the corpus-based systems. At the same time, scientists focused on developing tools that would facilitate the translation process rather than replacing human translators, leading to the development of translation memory (TM) and other computer assisted translation (CAT) (Samson, 2005) tools.

Corpus-based systems rely on a parallel corpus of sentences in a source language and its translation in a target language. With enough number of samples, these systems can be trained to infer the translations of new sentences. For this purpose, statistical models are applied to the translation process as these are good at extracting relevant information from a set of (translation) examples.

Training these models can take days or weeks with the added constraint that when new data is available the system needs to be retrained over the whole corpus. Therefore, it is desirable that when new sentences are available we can adapt or refine an already trained model without training it from scratch. This new framework is called online learning (OL), where the model is updated sequentially with new samples.

The OL framework becomes even more important in a post-editing scenario, where high quality translations are required and the translation process is human-assisted (Martínez-Gómez et al., 2012). In this scenario, translations resulting from the model are corrected by humans. Once a sentence is post-edited it can be used as a new training sample to refine our model in order to avoid making the same mistake next time.

Online learning techniques can also be applied in the recently developed neural networks (NN) architechtures for MT. In this work we propose several new algorithms for training neural machine translation (NMT) systems and compare them with other well-known OL algorithms. This document is structured as follows: In Chapter 2 we develop the basis of the statistical machine translation (SMT) framework and review the state-of-the-art systems used before the irruption of the neural approach in the field. In Chapter 3, the NMT approach is presented and its components are explained in depth. Chapter 4 explains how the neural models are trained and the various algorithms to do so. We dive into the online learning framework in Chapter 5 and we present our approach to tackle the problem. In Chapter 6 we describe the experiments carried out, we compare our work with the algorithms that are already stablished describing the results. Finally, we conclude the document with future work and conclusions in Chapter 7.

CHAPTER 2
Introduction to MT

Before diving into the neural approach to MT we need to first revisit the foundations on which it relies. In this chapter we will cover the basis of the statistical framework used to model the language and allow us to perform the task at hand. Next, we will describe the phrase-based models. These were the state-of-the-art in MT before the irruption of NN in the field. Finally, we will conclude the chapter with metrics used to evaluate the performance of MT models. Knowing the quality of translation is a key aspect in MT and not a trivial task that needs to take care of. We will cover the main used metrics in this field.

Statistical machine translation

We start by formulating the translation problem under a statistical point of view. For a given sentence x, in a source language, we want the best translation $\hat{y}$, in a target language. There are many valid translations of x so we will measure the validity of each translation with a probability distribution $Pr(y|x)$ over all possible translations y given the sentence x. Our goal is to find $\hat{y}$ which maximizes the conditional probability (Eq. 2.1) (Brown et al., 1993).

$$\hat{y} = \arg\max_{y} Pr(y|x) \tag{2.1}$$

The conditional probability $Pr(y|x)$ is unknown because that would require to compute the joint probability of all possible pairs (x, y). Hence we will rewrite $Pr(y|x)$ by applying the Bayes' rule:

$$Pr(y|x) = \frac{Pr(y) \cdot Pr(x|y)}{Pr(x)} \tag{2.2}$$

We can remove the denominator since $Pr(x)$ is fixed and it does not depend on y. As a result we obtain the Fundamental Equation of Machine Translation (Brown et al., 1993):

$$\hat{y} = \arg\max_{y} Pr(y) \cdot Pr(x|y) \tag{2.3}$$

Where $Pr(y)$ can be seen as the probability of y being well-formed (it models the constraints every given language has when it comes to building sentences) and $Pr(x|y)$ as the probability of a translation y produced from a source sentence x. The reason why we don't use $Pr(y|x)$ directly is to avoid the possibility of having a translation y that has a high probability but it is ill-formed. By maximizing the product $Pr(y) \cdot Pr(x|y)$ we force the model to search a translation that is both a well-formed sentence and high chances of being the correct one.

Then, Eq. 2.3 breaks the MT into three problems:

- Building a **language model** that correctly estimates $Pr(y)$
- Building a **translation model** that correctly estimates $Pr(x|y)$
- Search for y that maximizes $Pr(y) \cdot Pr(x|y)$

In the next subsections we will review the models used to build the language model and the translation model that correctly estimate $Pr(y)$ and $Pr(x|y)$ respectively.

Language model

Let $y = y_1 y_2 ... y_I$ be a sentence made of m words. We can compute the probability of y as a product of conditional probabilities:

$$Pr(y) = \prod_{i=1}^{I} Pr(y_i|y_1 ... y_{i-1}) \tag{2.4}$$

We can see that there is a lot of distinct probabilities that need to be estimated and this number grows as the length of the sentence increases. To tackle this, an unmanageable amount of data would be required. That is why we must make some assumptions about the probabilities that need to be estimated to simplify the problem. A good approximation is the *n-gram* assumption, which assumes that the probability of a word is dependent of the $n-1$ previous words. With this mindset, the probability of word y_i in sentence y would be approximated as in Eq. 2.5 and the sentence probability woud be the product of all these probabilities (Eq. 2.6).

$$Pr(y_i|y_1 ... y_{i-1}) \approx Pr(y_i|y_{i-n+1} y_{i-1}) \tag{2.5}$$

$$Pr(y) \approx \prod_{i=1}^{I} Pr(y_i|y_{i-n+1} ... y_{i-1}) \tag{2.6}$$

In order to estimate the conditional probability of $Pr(y_i|y_{i-n+1} ... y_{i-1})$ we count the occurrences of this given *n-gram* in the corpus and normalize it accordingly:

$$p(y_i|y_{i-n+1} ... y_{i-1}) = \frac{\#(y_{i-n+1} ... y_i)}{\#(y_{i-n+1} ... y_{i-1})} \tag{2.7}$$

Where $\#(y_{i-n+1}...y_i)$ is the number of occurrences of the *n-gram* in the corpus. The problem with this procedure is that, even though we have simplified the problem, there are still a lot of *n-grams* and not all of them will appear in our training corpus. With the current approach an *n-grams* that is not seen during training will have 0 probability, and the resulting conditional probability of the whole sentence in which it appears will be 0 as well.

This behaviour causes the model to underestimate the probability of *n-grams* that do not appear in the training set and overestimate the probability of those which do. This feature is not a desirable and numerous *smoothing* techniques have been developed to address the issue (Chen and Joshua, 1998; Kneser and Ney, 1995) by distributing some of the probability mass to unseen *n-grams*.

Translation model

A first approach to model the probability $Pr(x|y)$ is to assume that the unit of translation is a word. With this mindset, a given word in the source sentence correspond to some word(s) in the target sentence. Word-based translation relied in the concept of an *alignment* (Brown et al., 1990) between two strings. An alignment is a match between every word of the source sentence with the corresponding word of the target sentence. In Fig. 2.1 we can see an example where a word in the source sentence can be aligned to zero, one or several words in the target sentence.

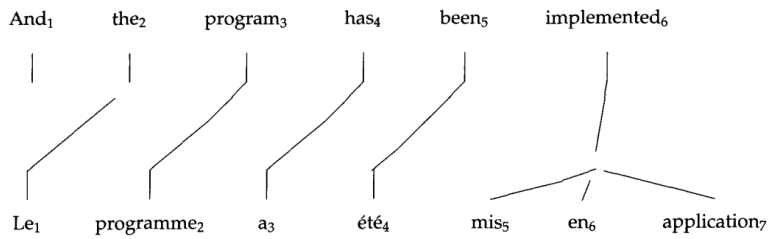


Figure 2.1: Example of a word alignment between a source sentence in english and a target sentence in french. Borrowed from Brown et al. (1993).

We do not know which is the probability distribution of the alignments. Therefore, in order to calculate the probability $p(x|y)$ we introduce the alignment a as a hidden variable, resulting in:

$$p(x|y) = \sum_a p(x, a|y) \tag{2.8}$$

That is, for every possible alignment between x and y we sum the probability of $p(x, a|y)$. Brown et al. (1993) developed the so-called *IBM Models* in order to model $p(x, a|y)$. This resulted in 5 models in each of which different assumptions were made. As a brief summary: Model 1 and 2 assume different alignment distributions; Model 3 includes a *fertility* model which represents the probability that each word in the target sentence generates a given number of source words, allowing the model to translate languages with different fertility; finally, Model

4 and 5 include a distortion model which takes into account the different relative position of words in the source and target sentence.

This word-level approach has the major disadvantage that it can not deal with context (Zens et al., 2002). Sometimes a group of words need to be translated into another group of words and it is not correct if we do it separately. **Phrase-based** models emerged with the goal of dealing with local context.

The unit of translations in phrase-based models are not words but *phrases*. A phrase is just a sequence of words. Phrase-based models consists of a dictionary of bilingual phrases that maps each phrase in the source language with its translation. However, as pointed out in Zens et al. (2002), additional constraints are taken into account when extracting phrases from a corpus. First, the words must be consecutive, and second, they must be consistent with the word alignment matrix, that is, the m source words must be aligned only to the n target words.

So the idea is to segment the given source sentence into phrases, translate each phrase and finally compose the target sentence from these translations (in most cases applying some reordering). If we take look at the word-based system, in order to calculate de probability of $p(x|y)$ we needed to introduce a hidden variable to know the word alignment and calculate for every possible alignment its probability. Here we need to do that for the segmentation of the source and target sentence and also, for the alignment between phrases.

Log-linear model

Current SMT systems are extended with the so-called *log-linear* model. These systems try to directly model $Pr(y|x)$ by combining different feature functions $h_m(x,y)$ along with its correspondent weight λ_m (Eq. 2.9). These weights are learned by minimizing an error function (normally 1-BLEU) over a validation set.

$$p(y|x) = \arg\max_{y} \sum_{m=1}^{M} \lambda_m h_m(x,y) \tag{2.9}$$

Feature functions commonly found are: the target language model; a phrase-based model and inverse phrase-based model; a reordering model; a target word penalty and phrase penalty (Koehn, 2010). The *log-linear* model can be extended with as many feature functions as we wish. The goal is to combine different models with different strengths in order to obtain a more robust system.

Assessment

When it comes to MT, a way to calculate the translation quality of a system is needed for various reasons. First, we need a way to rank systems and second, we want to evaluate incremental changes made to those system in order to know their effect. To do so, we can approach the problem with an automatic tool or with human evaluation. Although the latter provides a more accurate quality

measure the evaluation takes time and it is subject to human preference: a correct translation might be scored differently according to the evaluator. On the other hand, automatic translation yields reusable and cheap evaluation (but not necessarily reliable). Therefore, a variety of automatic tools that try to emulate human evaluation have been developed and it is still an open issue. We will cover in this section the most relevant metrics.

BLEU

The Bilingual Evaluation Understudy or BLEU (Papineni et al., 2002) is a widely used quality measure in the MT field. The idea behind this metric is to measure how close a machine translation is from a profesional human translation. The closer it is, the better. Therefore, a "closeness" metric is developed by using a weighted average of variable-length phrase matches against the reference translations.

The primary task for the BLEU evaluator is to compare the number of matches in terms of the *n-grams* a given hypothesis has with respects to the reference translation. This evaluator uses a modified version of precision called *modified n-gram precision* to avoid the situation where overgeneration of "reasonable" *n*-grams gives a high precision (but improbable) translation. This modified *n*-gram precision clips the count of a given n-gram to the number of occurrences in the reference sentence and then divides them by the number of (unclipped) words in the hypothesis.

The unit that BLEU uses is the sentence but the score is given for a whole document. To extend the modified *n*-gram precision over a set of sentences we need to first, compute the *n*-gram matches sentence by sentence and next, add the clipped *n*-gram counts for all the hypothesis. This is normalized by the number of hypothesis *n*-grams in the test corpus which gives us the modified precision score (Eq. 2.10) p_n for *n*-grams of size n.

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n-gram \in C} Count_{clip}(n-gram)}{\sum_{C' \in \{Candidates\}} \sum_{n-gram \in C'} Count(n-gram)} \tag{2.10}$$

To finalize, a *brevity penalty* (BP) is introduced to penalize candidates that not match in length the reference translation. It is computed as:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{2.11}$$

Where c is the length of the candidate translation and r the effective reference corpus length. Then, to take into account different *n*-grams lengths we just need to combine them by means of a weighted mean, which gives us the final BLEU score:

$$BLEU = BP \cdot exp(\sum_{n=1}^{N} w_n \log p_n) \tag{2.12}$$

According to the experimentation made in (Papineni et al., 2002), N is set to 4, and $w_n = \frac{1}{N}$. The BLEU score ranges from 0 to 1 being 1 the closest a candidate translation can be to the reference (identical in this case). This is the main measure we will use in the experimentation section.

TER

The translation error rate (TER) (Snover et al., 2006) is another measure used to assess translations. It measures the minimum number of edits needed to change a hypothesis so that it exactly matches the reference, normalized by the average length of the reference (if we have more than one). If we have several references the TER score will correspond to the minimum number of edits needed to match the closest reference.

The possible edits include insertion, deletion and substitution of single words as well as shifts of word sequences. A shift is just a movement of contiguous words within the hypothesis. An important thing to note is that all edits, including shifts of any number of words and distance have equal cost. In a post-editing scenario this measure approximates the human effort required to correct a translation produced by a MT system. Because we want to reduce the human effort required, we want to achieve a low TER score.

METEOR

METEOR (Banerjee and Lavie, 2005) was designed to address some of the issues that BLEU introduced, such as: the lack of recall, the use of higher order *n*-grams to model word order or the lack of explicit word-matching between translation and reference. It is based on the unigram matching between the machine-produced translation and the human-produced reference translation.

Given a pair of sentences, METEOR generates an *alignment* between the two strings. In this context, an alignment is a matching between unigrams, such that every unigram in each string maps to zero or one unigram in the other string. In order to generate that alignment the process is divided in two phases, each of which has different *stages*.

In the first phase, based on different criteria, different modules produce unigram mappings between the two strings. The "exact" module produces matches between two unigrams if they are exactly the same. The "porter stem" module produces unigram matches if they are the same after being stemmed. The "WordNet synonymy" maps two strings if they are synonyms. This modules are ordered by priority and a given match will be chosen first if it is possible to form an alignment and if it also has higher priority than other matches produced by lower priority modules. For example, an exact match will be always prefered.

In the second phase, the largest subset of unigram mappings is selected such that the resulting set constitutes an alignment as defined above. If more than one subset constitutes an alignment, this metric selects the one with fewer mapping *crosses*. If both sentences are written one below the other and a line is drawn

between the matching unigrams, a *cross* is produced when these lines intersect with another mapping.

Lastly, in order to obtain the METEOR score, an harmonic mean of precision and recall is computed over the unigrams and penalized if these unigrams are in different order compared to the reference. A high score will mean high quality translations.

CHAPTER 3
Neural machine translation

State-of-the-art MT systems have relied on the phrase-based approach for a long time. However, a new neural approach has emerged, being the first technology that has been able to challenge former systems (Luong and Manning, 2015; Jean et al., 2015). With the use of Graphic Processor Units (GPU), neural machine translation (NMT) has been able to cope with the high computational cost it requires to compete with state-of-the-art phrase-based systems (Bentivogli et al., 2016).

This fact teamed with the improvements made to the encoder-decoder architecture (explained later in this chapter) such as the attention model, or the use of gated recurrent units to cope with context in long sentences, made NMT technology to advance by leaps and bounds. Apart from that, NMT demonstrated its power at the IWSLT[1] 2015 evaluation campaign, where one of these systems outperformed the up-to-then state-of-the-art phrase-based systems on the English-German task, a pair of languages difficult due to the disparity in morphology and syntax.

The next year, Google announced that Google Translate made the leap to NMT because neural networks increase both fluency and accuracy of its translations[2]. Microsoft did the same with Microsoft Translator stating that neural networks better capture the context of full sentences before translating them, providing much higher quality translations[3]. However, despite the reasons these big companies have to shift to NMT we still lack a solid formal background that supports this technology. Things like the activation function choice, the number of units per layer or how many layers to use, are things that need a thorough study to understand them. For now, we can only study NMT systems, by looking at the translations and see what differences them from the ones produced by former approaches (Bentivogli et al., 2016).

In this chapter we will give an overview of the NMT technology and how it can be used to perform the task at hand. First we will see how we can model the language with neural networks. Then we will present the current neural archi-

[1] International Workshop on Spoken Language Translation.

[2] Found in translation: More accurate, fluent sentences in Google Translate.

[3] Microsoft Translator launching Neural Network based translations for all its speech languages.

tecture used to translate. Lastly, we will see how we perform the search task to find the best translation.

Modelling language with neural networks

Being able to represent language with neural networks is not an easy task. First, we need to solve the issue that comes from the words being represented as symbols rather than numbers. For this task, we need a continuous representation (a dense, real-valued vector) that both encapsulates relevant information of such symbols and reduces the impact of the *curse of dimensionality*. This latter term refers to the need of huge number of examples when learning complex functions (such as language). This fact gets worse as the number of variables increases (i.e. the size of the vocabulary) since the model needs to discriminate between a huge number of combinations of such values.

Once we have solved the previous issue we face another: we need to manage the fact that sentences are not of the same length. Not only that, the input and output sentence can be of different length, so we need a way to process information of variable length. Lastly, we need a way to capture contextual information. All these three problems will be addressed in next subsections.

Continuous word representation: word embedding

A continuous representation of a word is a vector of features which characterize the meaning of that word. To understand this, if a human would have to extract these features, he would choose grammatical features such as gender or plurality. With neural networks we let the learning algorithm discover this real-valued features instead. The idea is to map every word in the vocabulary to a low-dimensional continuous-valued vector with the hope that similar words get similar representations in the feature space. The main goal behind it is to allow the model to generalize better to sequences that are not seen during training but whose features are similar to those who have been seen.

Although distributed word representations were first proposed in the early 1980's (Hinton, 1986) and 1990's (Castaño and Casacuberta, 1997), it was not until the 2000's that first approaches using neural networks appeared to model language. These models were studied first in terms of feed-forward networks (Bengio et al., 2003) and later in terms of recurrent neural networks (RNN) (Mikolov et al., 2010, 2011) and yielded good word representation that condensed linguistic regularities in the vector representations (Mikolov et al., 2013b).

One of the first approaches that successfully used neural networks to model language was the one proposed in Bengio et al. (2003). This work projected the words into a real-valued vector. Then, it used a feed-forward network focusing on learning both the word feature vector and the probability distribution function of word sequences in terms of these feature vectors by maximizing the log-likelihood over a training data. The results obtained with this approach demonstrated how distributed representation of words could be teamed up with neural networks to outperform regular n-gram models.

All this work demonstrated outstanding performance in word-prediction but also the need for a more computationally efficient model (specially the ones involving RNN). For this purpose, Mikolov et al. (2013a) proposed a model that could learn word representation much faster with higher quality than previous approaches. It showed the learned relationship between words with simple algebra by simply adding and subtracting vectors for obtaining other words. For example, the word *Rome* could be obtained with: *Paris - France + Italy*.

Dealing with sequences: recurrent neural networks

When humans read a sentence, they understand each word based on their understanding of previous words. As you read, your brain retains information that is used to understand what is coming. Recurrent neural networks model this situation naturally. This is a major advantage since a variety of problems depend on an arbitrary sequence of time-dependent events, such as speech recognition, translation or language modelling.

They take into account past information by means of a cycle between its units. This allows them to have an internal state that holds present and past information. In addition, depending on how these cycles are routed we find different architectures in the literature. An example of this is the Elman (Elman, 1990) and Jordan (Jordan, 1986) networks, also known as "simple recurrent networks" .

The Elman architecture works as follows: given a sequence of vectors $x = \boldsymbol{x}_1 ... \boldsymbol{x}_T$ the network will produce a sequence of outputs $y = \boldsymbol{y}_1 ... \boldsymbol{y}_T$. At each time step (Eq. 3.1) the hidden state at time t, $\boldsymbol{s}_t$ gets calculated based on both, the input at time t, $\boldsymbol{x}_t$ and the hidden state of the previous time step, $\boldsymbol{s}_{t-1}$. Then, the output (Eq. 3.2) gets calculated based on $\boldsymbol{s}_t$.

$$\boldsymbol{s}_t = \sigma_s(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{s}_{t-1}) \tag{3.1}$$

$$\boldsymbol{y}_t = \sigma_y(\boldsymbol{V}\boldsymbol{s}_t) \tag{3.2}$$

In these equations, $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are the weight matrices of the input, recurrent and output connections respectively, σ_s is an activation function such as sigmoid and σ_y is the output activation function, being usually, the softmax function (Eq. 3.18). Alternatively, the Jordan architecture is similar but the recurrent connection is achieved by taking into account the previous output rather than the previous hidden state (Eq. 3.3).

$$\boldsymbol{s}_t = \sigma_s(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{y}_{t-1}) \tag{3.3}$$

However, one of the major flaws these networks have is their dependency on the complete past information. This prevents them from focusing only on recent information or on long-term dependencies, as depicted in (Bengio et al., 1994). When training them, they also feature the so-called vanishing gradient problem (explained in section 4.1) which prevents them from properly learning.

To address this, several new recurrent architectures have been proposed, such as the long short term memory networks (Hochreiter and Schmidhuber, 1997)

known as LSTM (Fig. 3.1), or the gated recurrent units (GRU) (Cho et al., 2014) which have the special trait of mitigating the vanishing gradient problem. Both of these architectures feature parametrized gates that modulate how much input information is allowed to affect the hidden state, how much past information is let through to the next time step and how much information is forgotten. These are trainable parameters that allow the network to do this conveniently.

In LSTM units, the memory cell $\mathbf{m}_t$ (Eq. 3.4) depends on the previous memory cell $\mathbf{m}_{t-1}$ and the new information $\tilde{\mathbf{m}}_t$ (Eq. 3.5) coming from the input of the unit. The amount of information used to update $\mathbf{m}_t$ from both $\mathbf{m}_{t-1}$ and $\tilde{\mathbf{m}}_t$ is modulated by $\mathbf{g}_t$ (Eq. 3.6) and $\mathbf{i}_t$ (Eq. 3.7) which are the vector outputs of the forget gate and input gate whose values range from 0 to 1. By doing an element-wise multiplication $\odot$ they modulate the amount of information that will be used in the update, being 1 completely retain and 0 completely forget. The output of the unit $\mathbf{s}_t$ is modulated by the output gate (Eq. 3.8) getting Eq. 3.9.

$$\mathbf{m}_t = \mathbf{g}_t \odot \mathbf{m}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{m}}_t \tag{3.4}$$

$$\tilde{\mathbf{m}}_t = tanh(\mathbf{W}_Y^M \mathbf{s}_{t-1} + \mathbf{W}_X^M \mathbf{x}_t) \tag{3.5}$$

$$\mathbf{g}_t = \sigma(\mathbf{W}_Y^F \mathbf{s}_{t-1} + \mathbf{W}_X^F \mathbf{x}_t) \tag{3.6}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_Y^I \mathbf{s}_{t-1} + \mathbf{W}_X^I \mathbf{x}_t) \tag{3.7}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_Y^O \mathbf{s}_{t-1} + \mathbf{W}_X^O \mathbf{x}_t) \tag{3.8}$$

$$\mathbf{s}_t = \mathbf{o}_t \odot tanh(\mathbf{m}_t) \tag{3.9}$$

In these equations, $\mathbf{W}_Y^O$, $\mathbf{W}_Y^I$, $\mathbf{W}_Y^F$ and $\mathbf{W}_Y^M$ are the output gate, input gate, forget gate, and memory cell recurrent weight matrices and $\mathbf{W}_X^O$, $\mathbf{W}_X^I$, $\mathbf{W}_X^F$, $\mathbf{W}_X^M$ are the output gate, input gate, forget gate, and memory cell input weight matrices. $\sigma(\cdot)$ and $tanh(\cdot)$ are the sigma and hyperbolic tangent activation functions.

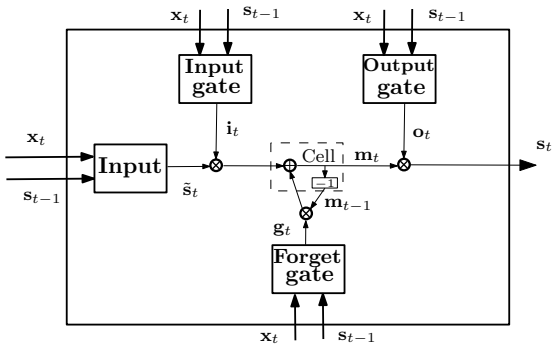


Figure 3.1: LSTM unit

Dealing with context: bidirectional recurrent neural networks

When processing sequences, there are times when future events are useful and provide more information to the network in order to better perform the task at hand. Regular RNN have the limitation that future information can not be reached from the current state.

Bidirectional recurrent neural networks (BRNN) (Schuster and Paliwal, 1997) address this issue allowing future information to be reachable from the current state. They do this by connecting two hidden layers of opposite direction to the same output layer, having then, information of past and future events.

With this architecture we call *forward layer* the one that processes the sequence in the positive time direction (left to right) and *backward layer* the one that processes the sequence in the negative time direction (right to left). At each time step, the output of both layers are combined (e.g. adding them up) before applying the output activation function.

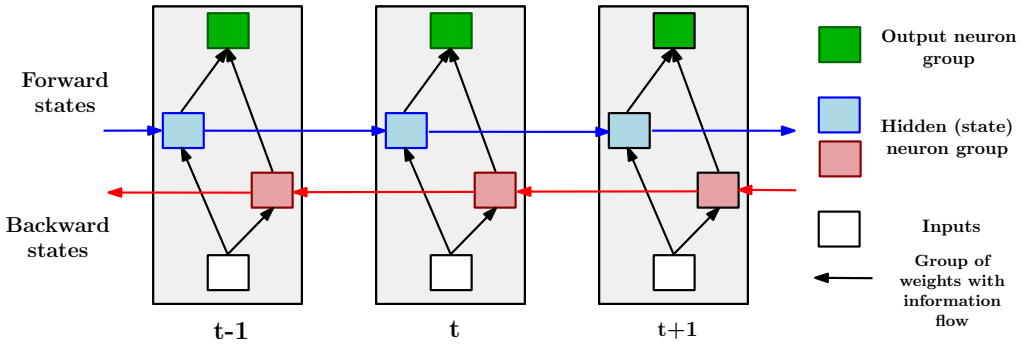


Figure 3.2: Structure of a bidirectional recurrent neural network shown unfolded in time for three time steps. Example borrowed from Schuster and Paliwal (1997).

.

End-to-end translation: encoder-decoder

We have seen that when a RNN processes a sequence of length T it produces an output sequence of the same length. In translation however, source and target sentence can (and usually are) of different lengths. To overcome this, two similar models were proposed by Sutskever et al. (2014) and Cho et al. (2014) which relied on the encoder-decoder approach. This work constitutes a common framework on which later work aims to improve it in different ways.

The encoder-decoder approach consist in a two step process that first, maps the source sentence into a fixed length vector, and second, this vector is decoded to produce the target sentence (possibly of different size). It aims to directly model the conditional translation probability (Eq. 2.1).

The input of the system is a sequence of words $x = x_1 ... x_J$ present in the vocabulary of the source language V_x in a one-hot representation. This means that we have for each word x_j a vocabulary-sized vector $\bar{\mathbf{x}}_\mathbf{j} \in \mathbb{N}^{|V_x|}$ with all elements

set to zero except for the one located at the position of x_j in V_x which is set to one. Each word is projected to a fixed-length real-valued vector:

$$\mathbf{x}_j = \mathbf{E}_s \bar{\mathbf{x}}_j \tag{3.10}$$

where $\mathbf{x}_j \in \mathbb{R}^d$ is the embedding of word x_j, $\mathbf{E}_s \in \mathbb{R}^{d \times |V_x|}$ is the source languaje projection matrix and d the embedding size.

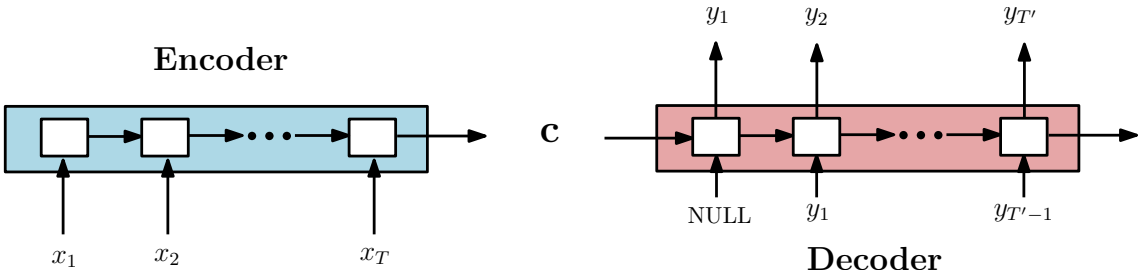


Figure 3.3: Ilustration of the encoder-decoder approach.

The encoder is an RNN that reads each element of the sequence of word embeddings. After reading each element in the positive direction, the hidden state $\mathbf{h}_j^f$ of the RNN is updated (Eq. 3.11). Once the end-of-sentence symbol is reached, the hidden state of the RNN $\mathbf{c}$ (Eq. 3.12), can be seen as a fixed-length summary of the whole input sequence.

$$\mathbf{h}_j^f = f(\mathbf{x}_j, \mathbf{h}_{j-1}^f) \tag{3.11}$$

$$\mathbf{c} = \mathbf{h}_J^f \tag{3.12}$$

where f is a non-linear function.

The decoder is another RNN which is trained to generate the output sequence by predicting the next word y_i given the hidden state $\mathbf{s}_i$, the previous generated word y_{i-1} and $\mathbf{c}$. The hidden state of the decoder at time i is computed as:

$$\boldsymbol{s}_i = f(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}) \tag{3.13}$$

where f is a non-linear activation function (LSTM or GRU) that produce the hidden state vector $\mathbf{s}_i$. The conditional probability for the next word y_i is approximated (Eq. 3.14) assuming that it depends on the previous word (and all previous words that are in $\mathbf{s}_i$ to some extent). In this case, $g(\cdot) \in \mathbb{R}^{|V_y|}$ is the softmax function (Eq. 3.18) that produces a vector of probabilities of size $|V_y|$, which is the size of the vocabulary of the target language. $\bar{\mathbf{y}}_i \in \mathbb{N}^{|V_y|}$ is the one-hot representation of the word y_i. $\mathbf{V} \in \mathbb{R}^{|V_y| \times L}$ is the weight matrix and $\varphi(\cdot)$ is the L-sized output layer of a RNN.

$$Pr(y_i | y_1 ... y_{i-1}, c) \approx \bar{\mathbf{y}}_i' \, g(\mathbf{V} \, \varphi(\boldsymbol{s}_i, \boldsymbol{y}_{i-1}, \boldsymbol{c})) \tag{3.14}$$

It is important to note that, during training, we want the decoder to learn to output the reference sentence. In order to help the model, we use the *teacher forcing* (Pascanu et al., 2013) technique. This technique consist of feeding the model

with the corresponding word in the reference sentence at time step $i-1$ instead of the previously generated word.

Attention model

One of the flaws that encoder-decoder models exhibit is that they have to condense an arbitrary length sentence into a fixed-length vector $\mathbf{c}$. This is a bottleneck when dealing with long sentences, as noticed by Cho et al. (2014), and causes the system to perform poorly in these situations.

In order to solve this, Bahdanau et al. (2014) proposed the so-called *attention model*. The basic idea is to use a different context vector depending on the current decoding stage. With this idea, Equation 3.14 is rewritten as:

$$Pr(y_i|y_1...y_{i-1}, c) \approx \bar{\mathbf{y}}'_i \, g(\mathbf{V} \, \varphi(\boldsymbol{s}_i, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i)) \tag{3.15}$$

where $\boldsymbol{s}_i$ is the RNN hidden state at time i:

$$\boldsymbol{s}_i = f(\boldsymbol{s}_{i-1}, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i) \tag{3.16}$$

We can see that at each time step, we use a different context vector $\boldsymbol{c}_i$. This vector is calculated based on a sequence of *annotations* $(\boldsymbol{h}_1, ..., \boldsymbol{h}_J)$ extracted from the source sentence. These annotations have information that strongly describe the i-th word and surroundings. They are calculated by concatenating the forward and backward hidden states of a BRNN. The annotation $\boldsymbol{h}_j = [\boldsymbol{h}_j^f; \boldsymbol{h}_j^b] \in \mathbb{R}^{2S}$ where $\boldsymbol{h}_j^f$ and $\boldsymbol{h}_j^b$ are the hidden states of the forward and backward layers of the BRNN and S is the size of the hidden state (we assume that they are of the same size). Therefore, $\boldsymbol{c}_i$ is calculated as the weighted sum of the annotations:

$$\boldsymbol{c}_i = \sum_{j=1}^{J} \alpha_{ij} \boldsymbol{h}_j \tag{3.17}$$

where α_{ij} is computed by the softmax function:

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{J} exp(e_{ik})} \tag{3.18}$$

being $e_{ij} = a(\boldsymbol{s}_{i-1}, \boldsymbol{h}_j)$ an *alignment model* which scores how related the inputs at position j and the output at position i are. The alignment model is parametrized with a feed-forward neural network which is trained jointly with the whole system.

Decoding translations: beam search

When decoding the translation, the neural system provides, at each time step, a set of probabilities (with the softmax function) for each word in the vocabulary. Because we are maximizing equation 2.1 and the output probabilities of a given time step depend over the previous chosen word, we cannot just choose the one

with higher probability at each time step because we would be loosing a lot of the (possible better) options. Moreover, we cannot take into account all possible words because that would be extremely costly.

Instead, we use the *beam search* (Sutskever et al., 2014) algorithm to keep, at each time step, a set of the η partial hypothesis with higher probability, being a partial hypothesis the prefix of the given translation. At each time step, all possible new words are added to the prefix but only the η hypothesis with higher probability are kept to face the next time step. This algorithm generates a kind of n-best sentence hypothesis (Chow and Schwartz, 1989).

CHAPTER 4
Training neural networks

We have seen so far how the neural networks perform the task of translation but we have left aside how these networks are trained in order to achieve better translations. The task of training recurrent neural networks or any kind of neural network is reduced to the correct estimation of the weights that connect each neuron to its neighbours (i.e. the weight matrices). In the case of NMT, the word embedding matrices also need to be estimated.

When training supervised neural networks (that is, when we know the expected value of a sample data in advance), the system is fed with samples and produces as output a set of probabilities, one for each class (in the case of a classification task). In translation, each word in the vocabulary is a different class and the network will output a *score* (probability) associated to each word at each time step.

The goal of the training process is that the class of the sample should have the highest probability. Applied to translation, this means that the hypothesis produced with higher probability (that is, the set of classes or words produced) must match as much as possible as the reference sentence provided as the correct translation.

However, this is unlikely to happen unless we modify the weights accordingly. The backpropagation algorithm address this problem by automatically modifying the parameters in order to learn to correctly estimate the proper classes. In the next section we will explain how the backpropagation algorithm works and how it is applied to correctly update the parameters through gradient descent.

The backpropagation algorithm

The backpropagation (BP) algorithm was successfully applied to neural networks during the 1980s (Rumelhart et al., 1988), although the basis of the algorithm were explored during the 1960s. It aims at minimizing an objective function (known as *loss function*) that measures the error produced between the output of the network and the desired scores (in classification, the classes). The algorithm must modify the parameters of the network to properly reduce the error.

In order to know how these weights must be modified, the algorithm must compute a partial derivative, known as *gradient* $\frac{\partial \ell}{\partial w}$ of the loss function ℓ with

respect to any weight of the network. These partial derivatives indicates the amount of increased or decreased error obtained when a slight modification to a parameter is performed. Hence, the problem can be seen as the search of the minimum value for the loss function in a high-dimensional space defined by the weights.

The algorithm has two main phases. In the first phase, called the *forward* step, the input data passes through the network, computing the input value of each neuron in every layer and then applying the activation function. Once we have obtained the output value, denoted as y we can calculate the error produced with respect to the true value, denoted with t. In Fig. 4.1 we can see an example of the backpropagation algorithm in a feed-forward network when optimizing the squared error $E = 0.5(y-t)^2$ in a regression task A common way of measuring the error in a classification task is by means of the *cross-entropy* (Golik et al., 2013) loss function:

$$H(t,p) = -\sum_x t(x)\log(p(x)) \tag{4.1}$$

This function measures the cross-entropy between two distributions, the true distribution t (the target class) and a predicted distribution p (outputted by the network) giving us an idea of how different they are. In the case of NMT, both $t(x)$ and $p(x)$ are a vector where, $t(x)$ is the one-hot encoding of the class and $p(x)$ is the vector of probabilities that comes from the softmax function.

Once the error is calculated, the second phase of the algorithm starts, called the *backward* step. This phase consists on (back) propagating the error through the network (starting from the last layer to the first one) computing the error derivatives with respect to the weights. To achieve this, the *chain rule* must be used in order to compute the derivative of a composed function.

For years, NN and deep NN have been left out because it was not possible to train them due to the *vanishing gradient problem* (Hochreiter et al., 2001), which basically is the progressive shrinking of the gradient that arrives to superficial layers. If the gradient disappears, the network is unable to modify its weights and then, unable to learn. Fortunately, the use of new architectures, together with a great computational power available and the availability of huge amounts of data are the main reasons for which nowadays we are able to train deep networks.

Backpropagation through time

In the case of training RNN, the BP algorithm is extended with the so-called back-propagation through time (BPTT) algorithm. When computing the gradient of a weight in a feed forward network, a weight only contributes once to the error produced. In RNN however, the recurrent weights are applied several times at different time steps. Therefore, in order to calculate the gradient of recurrent weights we must unfold the RNN by duplicating the weights spatially for an arbitrary number of time steps. Once unfolded, we calculate the error update of each weight as if it were a feed forward network. After all error updates have been obtained, weights are folded back adding up to one big change for each un-

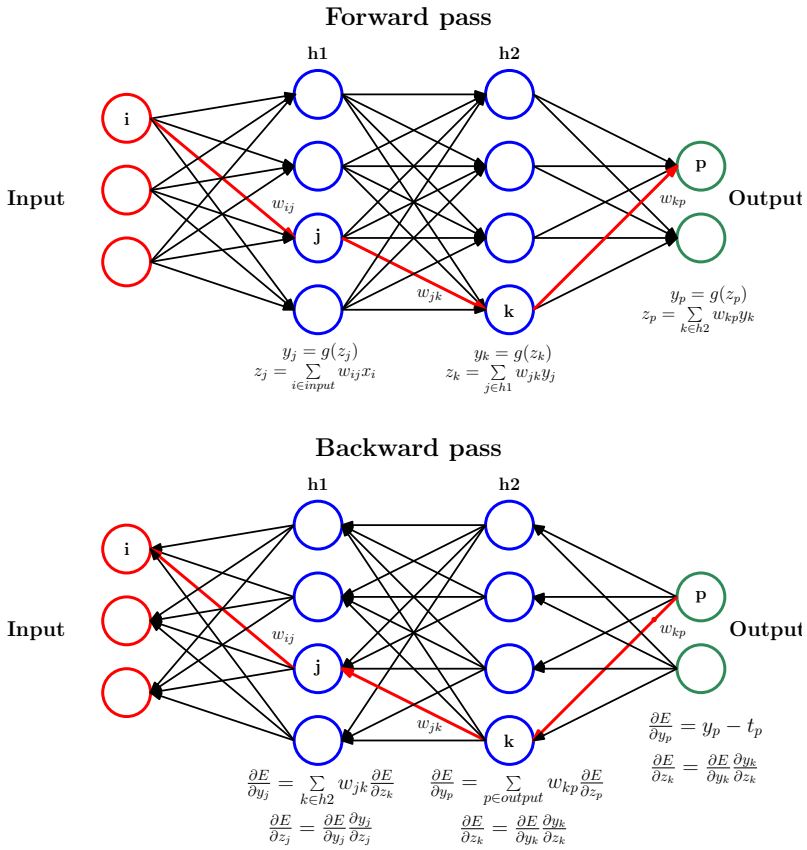


Figure 4.1: Ilustration of the backpropagation algorithm. Forward and backward pass. Example extracted from LeCun et al. (2015)

folded weights. A complete description of this algorithm can be found in (Guo, 2013).

Stochastic gradient descent

With the BP algorithm we are able to know the direction in which the error increases or decreases with a small variation of each parameter. In order to update the parameters of the network to a new state where the error is minimized the stochastic gradient descent (SGD) procedure is used (Bottou, 1991). This method computes the average gradient of a small set of samples from the training data and updates the weights accordingly. The term *batch* is used when referring to this small portion of samples. Also, the *epoch* term is used when all the samples of the training set are seen once. It is common that the network is trained for several epochs until reaching a convergence criterion.

In order to train the network with SGD we need to iteratively update the parameters of the network with the gradient of the loss function with regard to the weights. We do this by updating them in the opposite direction of the gradients in order to minimize the error function. A learning rate ρ is used to determine the step size of the update to reach a (local) minimum. In practice, its value is usually

$\rho < 1$ and it has to be adjusted manually. Expressed mathematically, the update of a weight vector $\mathbf{\Theta} \in \mathbb{R}^d$ is performed as:

$$\mathbf{\Theta}_{t+1} = \mathbf{\Theta}_t - \rho \cdot \nabla \ell(\mathbf{\Theta}_t; x_{(i:i+n)}; y_{(i:i+n)}) \tag{4.2}$$

where $\nabla \ell(\mathbf{\Theta}_t; x_{(i:i+n)}; y_{(i:i+n)})$ is the gradient of the loss function ℓ with respect to the parameters $\mathbf{\Theta}$ and the batch (of size n) formed by the samples ranging from (x_i, y_i) to (x_{i+n}, y_{i+n}). By updating the parameters in batches, the algorithm leads to a smooth convergence because it reduces the high variance of the gradients of a single sample (Bishop, 2006).

However, the vanilla SGD presented above introduce a series of challenges that new variations of the algorithm aim to address. First, choosing a proper learning rate can be difficult. A too small one will mean a really slow convergence and a too large one will lead the algorithm to oscillate. Also, the use of the same learning rate for all parameter might no be desirable when features have different frequencies. In that case, we might want to make larger update for infrequent features. Lastly, techniques such as the learning rate annealing that reduce the learning rate when the algorithm starts to converge need to be planned in advance and are not self-adaptive.

All these challenges are what motivate the emergence of new variations of the vanilla SGD that try to tackle them. In next subsections we will describe the most common ones. For the sake of clarity, we will omit $x_{(i:i+n)}; y_{(i:i+n)}$ from the formulas. Another aspect to note is that all operations involving vectors produce vectors as result (for example the square root or the power operation).

SGD with momentum

This variation sees the iterative minimization from a physic point of view. In this case, the minimization is like a ball rolling down the slope of the error function's surface. To prevent the ball from oscillating around surfaces more steeply in one dimension than another (as noticed in Qian (1999)) it follows the relevant direction by remembering the main direction it was following previously. The use of momentum helps the ball to accelerate the minimization in the relevant direction and attenuates the oscillations.

To achieve this, it adds a fraction γ of the update vector of the past time step to the current one. The update rule is:

$$\boldsymbol{v}_t = \gamma \cdot \boldsymbol{v}_{t-1} + \rho \cdot \nabla \ell(\mathbf{\Theta}_t) \tag{4.3}$$

$$\mathbf{\Theta}_{t+1} = \mathbf{\Theta}_t - \boldsymbol{v}_t \tag{4.4}$$

where, in practice, the momentum term γ is set close to 1 (e.g. 0.9 or similar).

Adagrad

Adagrad (Duchi et al., 2011) is a gradient-based algorithm that adapts the learning rate to every parameter, performing larger updates for infrequent parameters

and smaller updates for very frequent parameters. Pennington et al. (2014) found it useful when training word embeddings as infrequent words required larger updates that those seen numerous times. The Adagrad update rule is:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \frac{\rho}{\sqrt{\boldsymbol{G}_t + \epsilon}} \odot \nabla \ell(\boldsymbol{\Theta}_t) \tag{4.5}$$

where $\boldsymbol{G}_t$ is a vector $\mathbb{R}^d$ whose element i is the sum of the squares of the gradients with regards to $\boldsymbol{\Theta}_i$ up to time step t and ϵ is a small number ($1e$-8) placed for numerical stability (i.e. avoid dividing by zero). The $\odot$ symbol denotes an element-wise vector multiplication.

The main benefit of Adagrad is that it smooths the influence of the learning rate. The main weakness, however, is the accumulator $\boldsymbol{G}$, which grows during training (as all the terms added are positive) and it eventually makes the learning rate become so small that the algorithm stops learning.

Adadelta

Adadelta (Zeiler, 2012) aims to address the main issue of Adagrad in which the learning rate decreases monotonically. It achieves this by limiting the number of past squared gradients accounted in $\boldsymbol{G}$ to a fixed size w by defining $\boldsymbol{G'}$ as a recursive decaying average of all past gradients. $\boldsymbol{G'}$ at time t is defined as a fraction γ of the previous $\boldsymbol{G'}$ and the current gradient:

$$\boldsymbol{G'}_t = \gamma \cdot \boldsymbol{G'}_{t-1} + (1 - \gamma) \cdot \nabla \ell(\boldsymbol{\Theta}_t) \tag{4.6}$$

Now, we replace $\boldsymbol{G}'_t$ in the update rule of the Adagrad algorithm, being:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \rho \frac{1}{\sqrt{\boldsymbol{G'}_t + \epsilon}} \odot \nabla \ell(\boldsymbol{\Theta}_t) \tag{4.7}$$

then, they adjust the updates so they have the same units as the parameters by defining an exponentially decaying average of the squared parameter updates:

$$\boldsymbol{\mu}_t = \gamma \cdot \boldsymbol{\mu}_{t-1} + (1 - \gamma) \cdot \Delta \boldsymbol{\Theta}_t^2 \tag{4.8}$$

where $\Delta \boldsymbol{\Theta}_t$ is defined as:

$$\Delta \boldsymbol{\Theta}_t = \frac{\sqrt{\boldsymbol{\mu}_{t-1} + \epsilon}}{\sqrt{\boldsymbol{G'}_t + \epsilon}} \odot \nabla \ell(\boldsymbol{\Theta}_t) \tag{4.9}$$

leaving the update rule of Adadelta as:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \rho \Delta \boldsymbol{\Theta}_t \tag{4.10}$$

Adam

Adam (Kingma and Ba, 2014) is another adaptive method that computes different learning rates for each parameters. As Adadelta, Adam stores an exponentially

decaying average of past squared gradients v_t and an exponentially decaying average of past gradients m_t:

$$\boldsymbol{m}_t = \beta_1 \cdot \boldsymbol{m}_{t-1} + (1 - \beta_1) \cdot \nabla \ell(\boldsymbol{\Theta}_t) \tag{4.11}$$

$$\boldsymbol{v}_t = \beta_2 \cdot \boldsymbol{v}_{t-1} + (1 - \beta_2) \cdot \nabla \ell(\boldsymbol{\Theta}_t)^2 \tag{4.12}$$

The authors noticed that at early stages, the algorithm is biased towards zero. To counteract this they correct the above expressions as follows:

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t} \tag{4.13}$$

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t} \tag{4.14}$$

leaving the update rule as:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \frac{\rho}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon} \odot \hat{\boldsymbol{m}}_t \tag{4.15}$$

The default values the authors propose are 0.9 for β_1 and 0.999 for β_2.

CHAPTER 5
Online learning

Although computational resources have evolved to the point where it is relatively cheap to train a Pattern Recognition (PR) system, it is still a time-consuming process that takes days or even weeks to complete. As more and more data is available this problem is aggravated, specially on systems that interact with changing environments or that require a quick response at the same time they learn. In these cases, being able to adapt our system without training it from scratch becomes mandatory.

In this chapter we will present the online learning framework which will serve to introduce our work, emphasizing its application to a post-editing scenario in MT. Next we will explain the *Passive-Aggressive* (PA) online learning algorithm. It has the main goal of avoiding the catastrophic interference (McCloskey and Cohen, 1989) problem that threatens neural network systems in which they tend to forget previously learned information upon learning new information. Finally, we will propose three new PA-based algorithms for the task of NMT.

Online learning framework

PR systems have achieved an acceptable performance in very complex tasks that involves structured output and ambiguity such as machine translation or image description. Although these systems can reach a high performance on data similar to the one used to train them, their performance rapidly drops when the task is slightly different. In addition, getting enough manually annotated data of a specific domain in order to train a whole system might not be possible. Therefore, it is useful to train with generic out-of-domain data and then tune the system with domain-specific data. Because a complete retraining of the system might be infeasible, online learning techniques (Blum, 1998) are chosen for this task.

These techniques are particularly useful in the computer assisted translation (CAT) and interactive machine translation (IMT) paradigms, where human translators work jointly with machine translators in order to efficiently obtain high quality translations. In a post-editing scenario, the MT system translates a sentence and propose this as a hypothesis to a human translator which then corrects the sentence (if needed). Once a sentence is corrected what we have is a new pair of sentences that can be used to train the system. By using these new train-

ing samples to train our model, we progressively make the post-editing process more efficient. First, by making the system learn from its own errors and second, by easing the work of the human translator as it will have to correct less errors. Prior work in this area has also shown that human translators become more productive as the MT quality improves (Tatsumi, 2009).

The application of such techniques has been studied thoroughly (Martínez-Gómez et al., 2012; Lavie, 2014; Martínez-Gómez et al., 2011) in classical phrase-based SMT systems. Results show that significant improvements can be achieved by adapting the system by means of online learning in terms of the effort the human would need to correct the hypothesis provided. However, there has been no published work (to the best of our knowledge) related to the application of these techniques in NMT systems. This works aims at providing new algorithms to apply online learning to these systems.

Passive-Aggressive online learning

A variety of online learning methods have been proposed in literature (Lu et al., 2016). A classical OL method is the Perceptron algorithm (Rosenblatt, 1958) which updates the model by adding a misclassified example (parametrized with a factor) to the current parameters. From this work, a lot of new online learning algorithms have been developed based on the maximum margin criterion (Crammer and Singer, 2003; Gentile, 2001; Kivinen et al., 2004; Li and Long, 2000) which tries to separate the classes as much as possible. One important technique that falls in this category is the Passive-Aggressive (PA) online learning method (Crammer et al., 2006). This has been proved as a very successful and popular online learning technique for solving many real-world applications.

The idea behind the PA algorithm is very simple. When a new sample arrives it is classified and a loss is obtained, measuring the degree to which the prediction is wrong. The idea now is to find a new set of parameters (weights) that make the classifier to correctly classify the sample (aggressiveness) but by remaining relatively close to the previous set of parameters (passiveness). We will outline the PA approach in a binary classification problem as explained in Crammer et al. (2006).

In a binary classification problem, each sample $\boldsymbol{x}_t \in \mathbb{R}^d$ has a unique label $y_t \in \{+1, -1\}$ associated. We assume that the classification function is based on a vector of weights $\boldsymbol{\Theta} \in \mathbb{R}^d$ which take the form of $sign(\boldsymbol{\Theta} \cdot \boldsymbol{x})$. The magnitude $|\boldsymbol{\Theta} \cdot \boldsymbol{x}|$ is interpreted as the degree of confidence in the prediction. We refer to the term $y_t(\boldsymbol{\Theta}_t \cdot \boldsymbol{x}_t)$ as the margin calculated at time t. Whenever the margin is positive, the sample has been classified correctly. However, we want the classifier to achieve a correct classification with some margin (in Crammer et al. (2006) this margin is set to 1). Then we define the loss ℓ that suffers the classifier whenever the samples is classified incorrectly:

$$\ell(\boldsymbol{\Theta}; (\boldsymbol{x}, y)) = \begin{cases} 0 & \text{if } y(\boldsymbol{\Theta} \cdot \boldsymbol{x}) \leq 1 \\ 1 - y(\boldsymbol{\Theta} \cdot \boldsymbol{x}) & \text{otherwise} \end{cases} \tag{5.1}$$

In order to progressively learn the weight vector $\boldsymbol{\Theta}$, the algorithm must update it after every sample. At time t, the new weight vector $\boldsymbol{\Theta}_{t+1}$ is calculated by solving this constrained optimization:

$$\boldsymbol{\Theta}_{t+1} = \arg\min_{\boldsymbol{\Theta}\in\mathbb{R}^d} \frac{1}{2}\|\boldsymbol{\Theta} - \boldsymbol{\Theta}_t\|^2 \qquad \text{s.t.} \qquad \ell(\boldsymbol{\Theta}; (\boldsymbol{x}_t, y_t)) = 0 \tag{5.2}$$

In this algorithm, if the loss is 0, the optimal solution is $\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t$. On the other hand, when the loss is greater than 0, the algorithm forces $\boldsymbol{\Theta}_{t+1}$ to satisfy the constraint $\ell(\boldsymbol{\Theta}_{t+1}; (\boldsymbol{x}_t, y_t)) = 0$ while being as close as possible to the previous weight vector to preserve knowledge of previous samples. If we derive Eq. 5.2, the update rule for the weight vector is:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t + \tau_t y_t \boldsymbol{x}_t \quad \text{where} \quad \tau_t = \frac{\ell(\boldsymbol{\Theta}_t; (\boldsymbol{x}_t, y_t))}{\|\boldsymbol{x}_t\|^2} \tag{5.3}$$

However, this update rule is too aggressive and it might result in updating the model in such a way that in order to meet the constraint, it produces a lot of errors in later samples because the model was changed drastically. Therefore, a gentler update strategy is required. To do this, the authors introduce a non-negative slack ξ variable into the optimization problem defined in Eq. 5.2:

$$\boldsymbol{\Theta}_{t+1} = \arg\min_{\boldsymbol{\Theta}\in\mathbb{R}^d} \frac{1}{2}\|\boldsymbol{\Theta} - \boldsymbol{\Theta}_t\|^2 + C\xi \quad \text{s.t.} \quad \ell(\boldsymbol{\Theta}; \boldsymbol{x}_t, y_t)) \leq \xi \quad \text{and} \quad \xi \geq 0 \tag{5.4}$$

The parameter C controls the influence of the slack variable ξ. The authors coin the C variable as the aggressiveness parameter of the algorithm. The update rule in this case is the same $\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t + \tau_t y_t \boldsymbol{x}_t$ but τ_t changes to:

$$\tau_t = min\{C, \frac{\ell(\boldsymbol{\Theta}_t; (\boldsymbol{x}_t, y_t))}{\|\boldsymbol{x}_t\|^2}\} \tag{5.5}$$

PA online learning applied to NMT

In this section we propose a new version of PA-based SGD. The development of this works is inspired by the application of the MIRA (a PA-based) algorithm (Crammer and Singer, 2003) used in traditional SMT to tune the weights of the log-linear model. Also, the work develop in Martínez-Gómez et al. (2012) to the task of online adaptation gives us a feel of what can be achieved when adapting a MT system with OL. As any other PA technique, our algorithm aims to perform the minimum modification to the model parameters while fulfilling a correctness criterion. In our case, we use the loss function of the neural model for efficiency reasons (but we could have used another loss function such as BLEU).

Let h_t be the hypothesis generated by the NMT system (using the parameters $\mathbf{\Theta}_t$ at time t) of the source sentence x_t. We consider that $\mathbf{\Theta}_t$ is incorrect if the model assigns a lower probability to the target reference sentence y_t than to h_t:

$$p_{\mathbf{\Theta}_t}(y_t|x_t) < p_{\mathbf{\Theta}_t}(h_t|x_t) \tag{5.6}$$

When that happens, we want to search for a $\widehat{\mathbf{\Theta}}$ such that $\widehat{\mathbf{\Theta}}$ is close to $\mathbf{\Theta}_t$ and $p_{\mathbf{\Theta}_t}(y_t|x_t) > p_{\mathbf{\Theta}_t}(h_t|x_t)$. This is expressed with the loss function ℓ:

$$\ell(\widehat{\mathbf{\Theta}}, x_t, y_t, h_t) = \log p_{\widehat{\mathbf{\Theta}}}(h_t|x_t) - \log p_{\widehat{\mathbf{\Theta}}}(h_t|x_t) \tag{5.7}$$

With this loss function we are measuring how large is the gap between the probability given to the hypothesis and the probability given to the reference. By optimizing this function we reduce this gap with the idea that progressively, the hypothesis will be closer (i.e. similar) to the reference sentence. With the goal of optimizing this function, three variations of PA-based algorithms are developed.

Passive-Aggressive via subgradient techniques

In order to find $\widehat{\mathbf{\Theta}}$, the problem can be formulated as:

$$\widehat{\mathbf{\Theta}} = \arg\min_{\mathbf{\Theta}} \frac{1}{2}\|\mathbf{\Theta} - \mathbf{\Theta}_t\|^2 + C\xi \quad \text{s.t.} \quad \ell(\widehat{\mathbf{\Theta}}, x_t, y_t, h_t) \leq \xi \quad \text{and} \quad \xi \geq 0 \tag{5.8}$$

being C the parameter that controls the aggressiveness of the algorithm and ξ a slack variable (as discussed in Section 5.2). From this equation we have $\xi \geq \max(0, \ell(\widehat{\mathbf{\Theta}}, x_t, y_t, h_t))$ and then, we define F_t as the function to optimize:

$$F_t(\mathbf{\Theta}, x_t, y_t, h_t) = \frac{1}{2}\|\mathbf{\Theta} - \mathbf{\Theta}_t\|^2 + C\max(0, \ell(\widehat{\mathbf{\Theta}}, x_t, y_t, h_t)) \tag{5.9}$$

aiming to find the set of parameters that minimize this function, i.e.:

$$\widehat{\mathbf{\Theta}} = \arg\min_{\mathbf{\Theta}} F_t(\mathbf{\Theta}, x_t, y_t, h_t) \tag{5.10}$$

Because we are optimizing a function (F_t) with discontinuity points (because the $\max(\cdot)$ function has no derivative at 0) we have to use a subgradient method (Shor et al., 2003). This results in the derivative $\partial_\Theta F_t(\Theta, x_t, y_t, h_t)$ being:

$$\partial_\Theta F_t(\mathbf{\Theta}, x_t, y_t, h_t) = \begin{cases} \mathbf{\Theta} - \mathbf{\Theta}_t - C\nabla\ell(\mathbf{\Theta}, x_t, y_t, h_t) & \ell(\mathbf{\Theta}, x_t, y_t, h_t) < 0 \\ \mathbf{\Theta} - \mathbf{\Theta}_t & \ell(\mathbf{\Theta}, x_t, y_t, h_t) > 0 \\ [\mathbf{\Theta} - \mathbf{\Theta}_t, \mathbf{\Theta} - \mathbf{\Theta}_t - C\nabla\ell(\mathbf{\Theta}, x_t, y_t, h_t)] & \ell(\mathbf{\Theta}, x_t, y_t, h_t) = 0 \end{cases} \tag{5.11}$$

where, on the discontinuity point ($\ell(\boldsymbol{\Theta}, x_t, y_t, h_t) = 0$), we choose a value in the interval $[\boldsymbol{\Theta} - \boldsymbol{\Theta}_t, \boldsymbol{\Theta} - \boldsymbol{\Theta}_t - C\nabla\ell(\boldsymbol{\Theta}, x_t, y_t, h_t)]$. We assume $\boldsymbol{\Theta} - \boldsymbol{\Theta}_t$ when $\ell(\boldsymbol{\Theta}, x_t, y_t, h_t) = 0$. Finally, we perform the update as:

$$\boldsymbol{\Theta}^{k+1} = \boldsymbol{\Theta}^k - \rho \cdot \partial_\Theta F_t(\boldsymbol{\Theta}, x_t, y_t, h_t)|_{\boldsymbol{\Theta}^k} \tag{5.12}$$

where ρ is the learning rate, $\partial_\Theta F_t$ is the subgradient of F_t with respect to $\boldsymbol{\Theta}$. We initialize $\boldsymbol{\Theta}^{k=0} = \boldsymbol{\Theta}_t$ and the update is performed k times per sample. We denote this PA via subgradient technique update rule as PAS.

Because we may want to achieve a higher probability of the reference sentence but with some margin m: $p_{\boldsymbol{\Theta}}(y_t|x_t) + m > p_{\boldsymbol{\Theta}}(h_t|x_t)$ we can rewrite the subgradient $\partial_\Theta F_t$ with the discontinuity point being m rather than zero (as defined in Eq. 5.11).

Passive-Aggressive via projected subgradient techniques

An extension of the PAS method is the projected PA subgradient method (PPAS) in which the optimization problem is reformulated (Boyd et al., 2003). We define $G_t(\boldsymbol{\Theta}, x_t, y_t, h_t)$ as $\max(0, \ell(\widehat{\boldsymbol{\Theta}}, x_t, y_t, h_t))$. Then, Eq. 5.8 can be rewritten as:

$$\widehat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\Theta}} G_t(\boldsymbol{\Theta}, x_t, y_t, h_t) \quad \text{s.t.} \quad \|\boldsymbol{\Theta} - \boldsymbol{\Theta}_t\|^2 \leq C \tag{5.13}$$

In this case, $\partial_\Theta G_t(\boldsymbol{\Theta}, x_t, y_t, h_t)$ is defined as:

$$\partial_\Theta G_t(\boldsymbol{\Theta}, x_t, y_t, h_t) = \begin{cases} \nabla\ell(\boldsymbol{\Theta}, x_t, y_t, h_t) & \ell(\boldsymbol{\Theta}, x_t, y_t, h_t) < 0 \\ 0 & \ell(\boldsymbol{\Theta}, x_t, y_t, h_t) > 0 \\ [0, \nabla\ell(\boldsymbol{\Theta}, x_t, y_t, h_t)] & \ell(\boldsymbol{\Theta}, x_t, y_t, h_t) = 0 \end{cases} \tag{5.14}$$

and the update rule is performed in a two step process. First, we calculate the intermediate weight update $\bar{\boldsymbol{\Theta}}^{k+1}$ as:

$$\bar{\boldsymbol{\Theta}}^{k+1} = \boldsymbol{\Theta}^k - \rho\partial_\Theta G_t(\boldsymbol{\Theta}, x_t, y_t, h_t)|_{\boldsymbol{\Theta}^k} \tag{5.15}$$

and second, we apply the projection operator, being $\boldsymbol{\Theta}^{k+1}$:

$$\boldsymbol{\Theta}^{k+1} = \frac{\bar{\boldsymbol{\Theta}}^{k+1} - \boldsymbol{\Theta}_t}{\|\bar{\boldsymbol{\Theta}}^{k+1} - \boldsymbol{\Theta}_t\|} C + \boldsymbol{\Theta}_t \tag{5.16}$$

As in the previous case, we initialize $\boldsymbol{\Theta}^{k=0} = \boldsymbol{\Theta}_t$.

Passive-Aggressive via SGD with regularization

We can also optimize $\ell(\boldsymbol{\Theta}, x_t, y_t, h_t)$ by means of SGD but with a regularization term, inspired by the PA principle of updating the model with new parameters

that are close to the current ones. We will refer to this algorithm as PAR. With this in mind, we can define the minimization problem as:

$$\widehat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\Theta}}(\ell(\boldsymbol{\Theta}, x_t, y_t, h_t) + \frac{C}{2}\|\boldsymbol{\Theta} - \boldsymbol{\Theta}_t\|^2) \tag{5.17}$$

Because the minimization has no discontinuity points we don't need to apply a subgradient method. The update rule then, results in:

$$\boldsymbol{\Theta}^{k+1} = \boldsymbol{\Theta}^k - \rho \cdot (\nabla \ell(\boldsymbol{\Theta}, x_t, y_t, h_t)|_{\boldsymbol{\Theta}^k} + C(\boldsymbol{\Theta}^k - \boldsymbol{\Theta}_t)) \tag{5.18}$$

As usual, $\boldsymbol{\Theta}^{k=0} = \boldsymbol{\Theta}_t$. We observe that when $k = 1$ the update rule is plain SGD with no regularization term:

$$\boldsymbol{\Theta}^{k+1} = \boldsymbol{\Theta}^k - \rho \cdot (\nabla \ell(\boldsymbol{\Theta}, x_t, y_t, h_t)|_{\boldsymbol{\Theta}^k} + \cancelto{0}{C(\boldsymbol{\Theta}^k - \boldsymbol{\Theta}_t)}) \tag{5.19}$$

in that case, we introduce a small Gaussian noise $\boldsymbol{\nu}$ to the model parameters in order to avoid reaching local minima:

$$\boldsymbol{\Theta}^{k+1} = \boldsymbol{\Theta}^k - \rho \cdot (\nabla \ell(\boldsymbol{\Theta}, x_t, y_t, h_t)|_{\boldsymbol{\Theta}^k} + C\boldsymbol{\nu}) \tag{5.20}$$

CHAPTER 6
Experiments and results

After the proposal of three PA-based algorithms in Section 5.3, we will proceed to its testing and comparison with other well-known algorithms (those described in Section 4.2). In this section we will describe the experimental setup used to conduct the experimentation. First, we will describe the task in which we want to evaluate our algorithms. Then we will describe the NMT system used and the tools needed to work with it. After describing the corpora used, we will compare and discuss the results obtained.

Experimental framework

Before presenting the results, we will present the task at hand and describe the tools used to develop the mentioned algorithms and the corpora used to assess them.

Task description

In order to test these algorithms we will simulate (otherwise it would be too costly) a post-editing scenario. We will simulate this scenario in three different corpus or tasks (detailed in next subsections). First, we will train three neural models using for each one the training partition of its corresponding dataset. Once trained, these models will constitute the baseline models for each task.

Then, we will refine these models with their respective test or development set of the corpus in which they were trained. For a given source sentence, the system produces its translation (the system's hypothesis). This translation is stored for later evaluation. Then we take the post-edited sentence (in this case, the reference sentence) and update the system with it. This is repeated until the full set of sentences are translated. We evaluate the stored hypotheses and compare them with the ones produced by the baseline system. Our hope is to see an improvement in the quality of these translation. With this procedure we want to see up to what extent the OL algorithm can refine the baseline system.

In order to measure the impact of each algorithm in the post-editing effort reduction we use the TER metric. In order to assess the quality of translations we will use the BLEU and METEOR metrics.

Software

We have developed our work using the NMT-Keras toolkit[1] (Peris, 2017). This (still in progress) software provides tools for the NMT task, such as: beam search decoding, support for GRU and LSTM networks, unknown word replacement, use of pretrained word embedding vectors an more.

This software is based on a Python library called Keras[2] (Chollet et al., 2015). Keras offers an abstraction layer on top of Theano[3] (Theano Development Team, 2016) (a library for defining and evaluating mathematical expressions involving multi-dimensional arrays) and eases the task of building and training different neural network architectures. It has implemented utilities to save and load neural network models, monitor the training process and use predefined NN architectures.

All the algorithms listed in Section 5.3 have been implemented in Keras and integrated in the NMT-Keras toolkit.

Corpora

In this work, we test all the algorithm in three different corpus: Xerox, Emea and TED. We use the standard partitions and the English-French language pair for all experiments. The text is kept true case and it is tokenized using the script from Moses [4]. In the next subsections we will describe these corpus.

Xerox

The Xerox corpus (Barrachina et al., 2009) consists in translations of user manuals for Xerox printers. Table 6.1 shows all information regarding the partitions for the English-French language pair.

		Training	Development	Test
Sentences		52k	984	994
Vocabulary size	English	14k	1.8k	1.7k
	French	15.5k	1.9k	1.8k
Words	English	615k	10.9k	11.1k
	French	676k	11.7k	11.8k

Table 6.1: Statistics of the Xerox corpus. In the table are collected the number of sentences and vocabulary size of each partition and language. k stands for thousands

[1] https://github.com/lvapeab/nmt-keras/
[2] https://keras.io/
[3] http://deeplearning.net/software/theano/
[4] http://www.statmt.org/moses/

Emea

The Emea parallel corpus (Tiedemann, 2009) is made out of documents from the European Medicine Agency. Table 6.2 shows all information regarding the partitions for the English-French language pair.

		Training	Development	Test
Sentences		319k	500	1k
Vocabulary size	English	52k	2.8k	4.5k
	French	59.3k	2.9k	4.5k
Words	English	4.2M	10.3k	21.4k
	French	4.8M	12.3k	25.7k

Table 6.2: Statistics of the Emea corpus. In the table are collected the number of sentences and vocabulary size of each partition and language. k stands for thousands and M for millions

TED

The TED corpus (Federico et al., 2011) gathers transcribed TED talks. Table 6.3 shows all information regarding the partitions for the English-French language pair.

		Training	Development	Test
Sentences		159k	887	1.7k
Vocabulary size	English	46.7k	3.4k	4.9k
	French	58.2k	3.9k	4.9k
Words	English	2.17M	21.3k	33.5k
	French	2.3M	21.8k	35.7k

Table 6.3: Statistics of the TED corpus. In the table are collected the number of sentences and vocabulary size of each partition and language. k stands for thousands

NMT system

For each corpus, we train a neural model over the training partition. Such model (similar to the one depicted in Bahdanau et al. (2014)) consists in an encoder-decoder LSTM network equipped with the attention mechanism. We made use of single-layered LSTM due to time limitations. The size of the hidden state of each LSTM, the word embedding size and the attention mechanism layer is 512 (our choice is based on the experimentations carried out in Britz et al. (2017)). The baseline systems have been trained with the Adadelta algorithm with the default parameters. During training, Gaussian noise (Graves, 2011) and layer normalization (Ba et al., 2016) are applied as regularization methods. We also early stopped the training if the BLEU on the development set did not improve in 100,000 updates. The size of the beam is 6 in all experiments.

Due to the high number of hyperparameters all the proposed online learning algorithms have, a wide grid search has been carried out in one of the corpus (Emea). After discarding values that ruin the system, a narrower grid search with promising values has been carried out in the rest of the corpus to find the best configuration. These hyperparameters include the C constant, the margin m, the number of iterations per sample k and the learning rate ρ.

Results

Because these are new algorithms, it is interesting to see how tuning different hyperparameters can affect their performance. After finding the best configuration for each algorithm, a comparison between these algorithms and SGD-based algorithms will be carried out.

Hyperparameter configuration

In order to find the best possible configuration, a grid search for each algorithm has been performed. Although this grid search has been done in all the corpus for all algorithms in order to avoid any corpus-dependent side-effect, a first wider grid search has been carried out on the Emea corpus by using the development set. With this grid search we have spotted ill configurations that ruin the system and we have narrowed the search for later corpus. For completeness purposes, we have provided this values of the grid search (Table 6.4) along with the best configuration for each algorithm (Table 6.5). This is to give a brief overview about in which range each parameter fits each algorithm.

One of our findings with this search is that, for example, the optimum value of C, the aggressiveness parameter, is greatly separated from one algorithm to another. While the PAS algorithm find its best value around 0.5, the PPAS algorithm find its best value around 0.01, both with the same learning rate. This suggests that the PPAS update rule is more aggressive than the one found in the PAS algorithm, as it needs a much lower value of C.

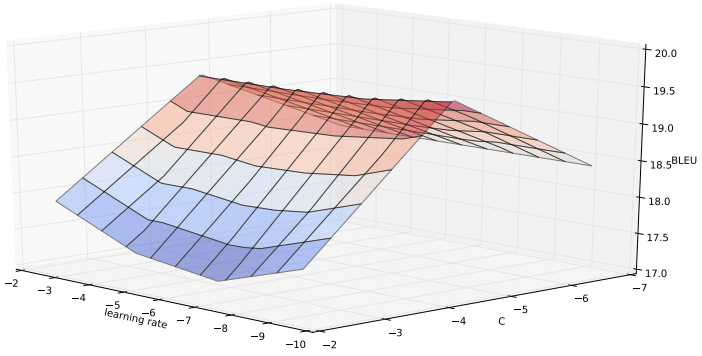
Another behaviour observed is that the correct performance of these algorithms is tightly related to the correct choice of both ρ and C. This is illustrated in Fig. 6.1 where, if we fix both the k and m parameter the model performance in terms of BLEU greatly decreases when ρ and C are not chosen adequately. On the other hand, if we fix ρ and C and we vary k and m, the performance is roughly the same.

ρ	$\{10^{-1}, 10^{-2}, 10^{-3}\}$
C	$\{1.5, 1, 0.5, 10^{-1}, 10^{-2}, 10^{-3}\}$
m	$\{0, 0.1, 0.3, 0.5, 1, 1.5\}$
k	$\{1, 2, 3\}$

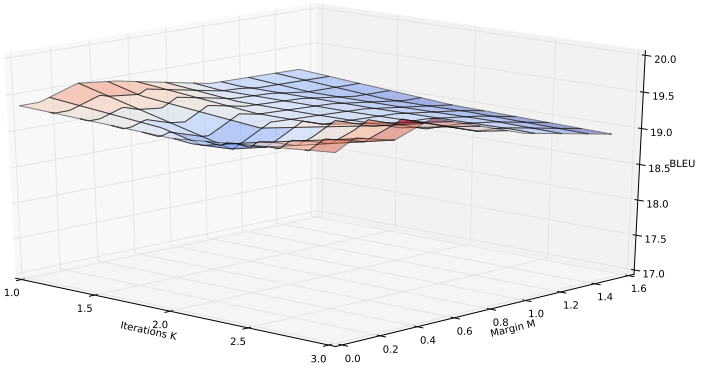
Table 6.4: Grid search values on the Emea development set

	PAS	PPAS	PAR
ρ	10^{-2}	10^{-2}	10^{-2}
C	0.5	10^{-2}	10^{-1}
m	10^{-1}	0.5	-
k	2	3	1

Table 6.5: Best configuration for each algorithm on the Emea development set



(a) k and m fixed with 1 and 0.1 respectively. Both learning rate and C are in log scale



(b) ρ and C fixed with 0.01 in both cases

Figure 6.1: Performance (in terms of BLEU) obtained by the PPAS algorithm on the Emea development set when pair-evaluating the influence of ρ-C and k-m parameters. Figure a) shows great influence on the performance when varying ρ and C while Figure b) shows little variance when changing k and m.

Comparison between OL algorithms

Once we know how our algorithms behave, we need to test them against SGD-based algorithms. In order to perform a fair comparison, all the PA-based algorithms will iterate once (i.e. $k = 1$) per sample. The margin m is set to 0 as it has been shown in the previous section to have little impact in terms of performance. The Gaussian noise used in the PAR algorithm has 0.01 standard deviation. In Table 6.6 we provide the different parameter values used in the different algorithms. The rest of hyperparameters have been left to its default value.

Now we will pass on to compare the different OL algorithms. First we will compare them by just looking at the end result (after the whole training) in terms

	SGD	Adagrad	Adadelta	Adam	PAS	PPAS	PAR
ρ	10^{-3}	10^{-4}	10^{-1}	10^{-4}	10^{-2}	10^{-2}	10^{-2}
C	–	–	–	–	10^{-1}	10^{-2}	10^{-1}

Table 6.6: Algorithm hyperparameters. ρ refers to the learning rate and C to the aggressiveness of PA algorithms.

of BLEU and TER and then we will see how the system performance evolves (in terms of BLEU) as the training process takes place.

In Fig. 6.2 is depicted the improvement of the quality of translations (in terms of BLEU) accomplished after the OL training process takes place. Generally, we obtain better results with adaptive algorithms such as Adam or Adadelta. Plain SGD also achieves a comparable performance (albeit not being adaptive). In the case of the PA-based algorithm their performance is not as high as its competitors, but we can see that they also increase the translation quality of the system. Although in all three tasks we improve the baseline system by several points, the results in the TED task of all algorithms are not as high. Moreover, we have had to reduce (by a factor of 10) the learning rate of several algorithms and the aggressiveness of the PA algorithms in this task because they worsened the baseline system. We can see a taste of this in the PPAS algorithm whose aggressiveness makes it score below the baseline.

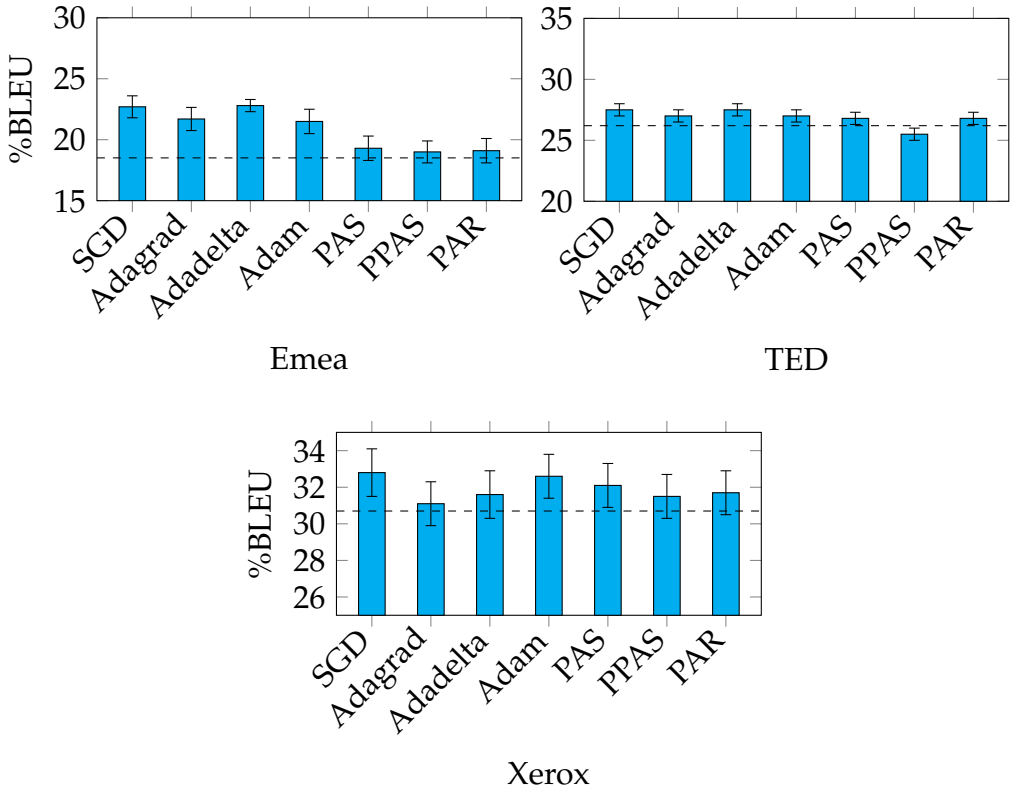


Figure 6.2: Effect of tuning NMT systems (in terms of BLEU) trained on different corpus with their respective test set. The higher the BLEU score, the better. We show 95% confidence intervals, obtained by means of bootstrap resampling (Koehn, 2004)

In terms of human effort reduction, we also observe an improvement (reduction of TER) but only in two of the tasks (Fig. 6.3). In TED we find that, although the quality of translation has slightly improved, in terms of TER we obtain a slightly different picture. This might be due to the fact that first, BLEU and TER are not completely correlated, and second, the improvement accomplished in terms of BLEU is sufficiently small so that the variations in terms of TER can fluctuate.

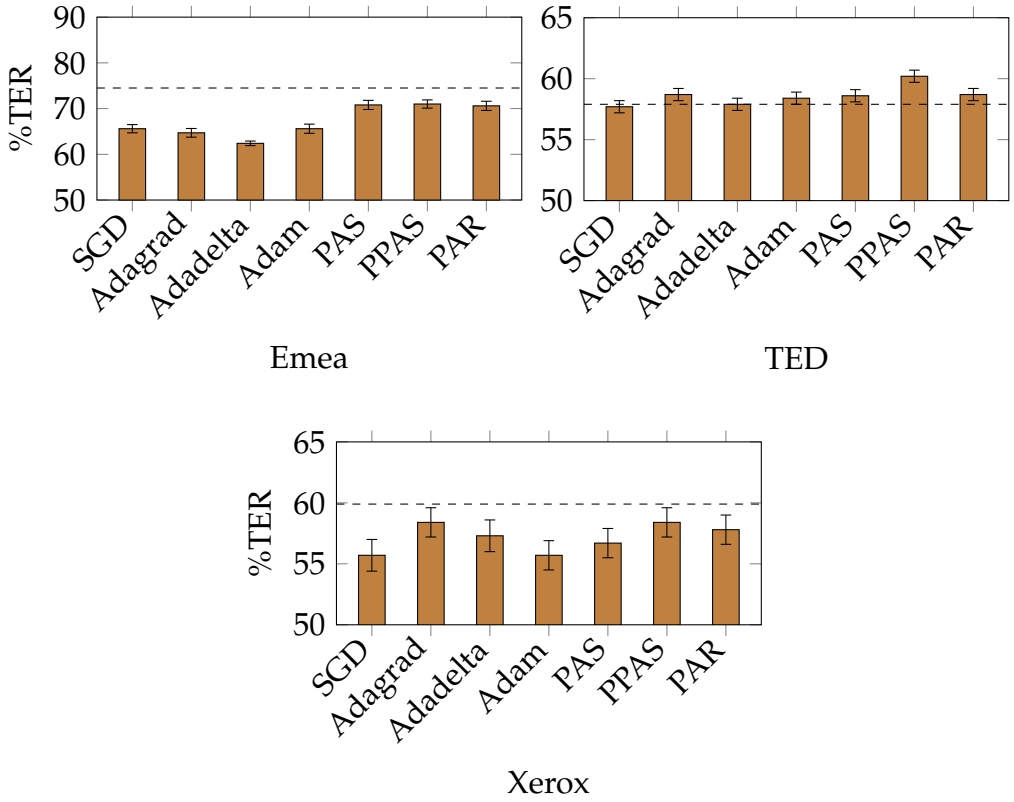


Figure 6.3: Effect of tuning NMT systems (in terms of TER) trained on different corpus with their respective test set. The lower the TER score, the better. We show 95% confidence intervals, obtained by means of bootstrap resampling (Koehn, 2004)

After looking at the end picture it is interesting to see the evolution of the different learning algorithms. Fig. 6.4 shows the BLEU difference between the baseline and online system. We show the average BLEU up to the n-th sentence. As we can see, all algorithm suffer an erratic behaviour at early stages but they stabilize later on. In the case of adaptive algorithms this might be due to the fact that they rely on an accumulator of the past gradients in order to perform the weight update. Thus, at early stages, the accumulated gradients of a few samples can have undesired results. Nevertheless, at mid to latter stages all algorithms stabilize, offering a consistent improvement over the baseline system.

For completeness purposes, we have provided all the results obtained in our experimentation in Table 6.7 in terms of BLEU, METEOR and TER of all three tasks.

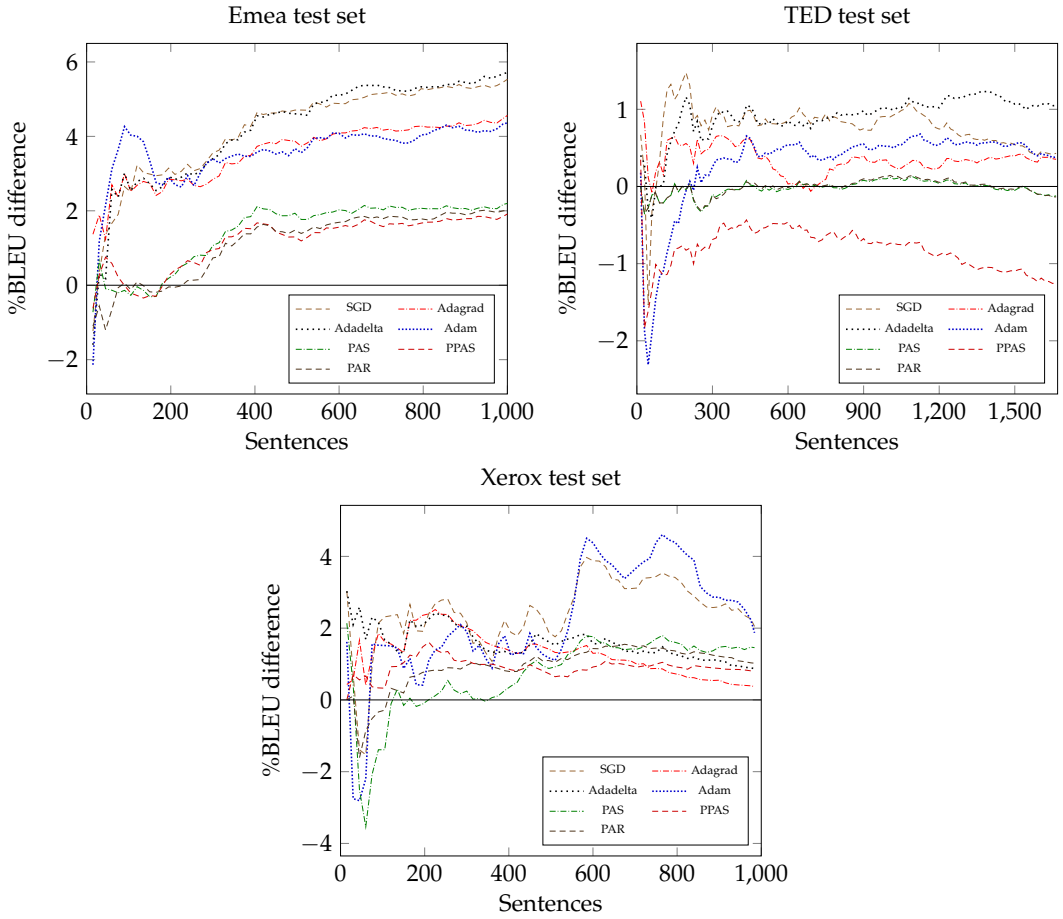


Figure 6.4: Evolution of a system trained on Emea, TED and Xerox and tuned with their respective test set. We show the BLEU difference (averaged up to the n-th sentence) with respect to the baseline system.

	Algorithm	Adam	Adadelta	SGD	Adagrad	PAS	PPAS	PAR	Baseline
Xerox	BLEU	**32.6 ± 1.2**	31.6 ± 1.3	**32.8 ± 1.3**	31.1 ± 1.2	**32.1 ± 1.2**	31.5 ± 1.2	31.7 ± 1.2	30.7 ± 1.2
	METEOR	45.9 ± 1.1	45.6 ± 1.1	46.6 ± 1.2	45.3 ± 1.1	46.6 ± 1.1	46.0 ± 1.1	46.4 ± 1.0	45.2 ± 1.1
	TER	**55.7 ± 1.1**	57.3 ± 1.1	**55.7 ± 1.2**	58.4 ± 1.1	**56.7 ± 1.1**	58.4 ± 1.1	57.8 ± 1.1	59.9 ± 1.1
	Algorithm	Adam	Adadelta	SGD	Adagrad	PAS	PPAS	PAR	Baseline
Emea	BLEU	**21.5 ± 0.5**	**22.8 ± 0.5**	**22.7 ± 0.5**	**21.7 ± 0.5**	**19.3 ± 0.5**	**19.0 ± 0.5**	**19.1 ± 0.5**	17.1 ± 0.4
	METEOR	**38.3 ± 0.5**	**40.2 ± 0.5**	**40.0 ± 0.5**	**39.2 ± 0.5**	**38.0 ± 0.5**	**37.9 ± 0.5**	**37.7 ± 0.5**	35.5 ± 0.5
	TER	**65.6 ± 0.9**	**62.4 ± 0.6**	**63.7 ± 0.8**	**64.7 ± 0.7**	**70.8 ± 0.8**	**71.0 ± 0.7**	**70.6 ± 0.8**	74.5 ± 0.9
	Algorithm	Adam	Adadelta	SGD	Adagrad	PAS	PPAS	PAR	Baseline
TED	BLEU	27.0 ± 0.5	**27.5 ± 0.5**	**27.5 ± 0.5**	27.0 ± 0.5	26.8 ± 0.5	25.5 ± 0.5	26.8 ± 0.5	26.2 ± 0.5
	METEOR	47.2 ± 0.4	48.0 ± 0.5	47.7 ± 0.4	47.0 ± 0.4	46.7 ± 0.4	45.6 ± 0.4	46.7 ± 0.4	47.2 ± 0.5
	TER	58.4 ± 0.6	57.9 ± 0.6	57.7 ± 0.6	58.7 ± 0.6	58.6 ± 0.6	60.2 ± 0.6	58.7 ± 0.6	57.9 ± 0.6

Table 6.7: Results of the experimentation in all tasks. Bold results indicate a significant improvement over the baseline system.

CHAPTER 7
Future work and conclusions

Before ending this document, we add some lines of research this work could follow in the future. In addition, we will expose our conclusions after the work developed in this document as well as some final notes.

Future work

We have shown how OL strategies can be applied to NMT in order to refine or adapt the system to changing environments. They provide a way of training dynamic systems as soon as the data is available, which in many situations, is very beneficial to keep a competitive performance. We have approached the online learning scenario from a post-editing view point by relying on the PA strategy. This opens two main lines of research in which this approach could be extended or applied to a different domain:

Using other loss functions

One of the main issues with NMT is the independence of the function to optimize with respect to the evaluation metric. An NMT system tries to optimize a loss function but when it is evaluated, another function (such as BLEU or TER) is used. This creates a gap between training and evaluation which in some cases might affect translation quality (Shen et al., 2016). In classical SMT, as well as in NMT, this issue has been addressed by optimizing directly the metric function by means of the minimum risk training method (Och, 2003; Chiang, 2012; Shen et al., 2016).

With this work, we set the basis for using BLEU or other non-differentiable loss functions, since some of the algorithms developed such as PAS and PPAS deal with non-differentiable functions. A natural next step to take is to directly optimize the evaluation metric, integrating it into the online learning framework while keeping the update time at a reasonable level.

Inclusion of OL strategies in IMT

The IMT scenario is tightly related with the post-editing scenario. IMT tries to collaborate with the human on the translation task with the goal of minimizing human effort. Often, the user only needs to point out the position in the sentence in which it is wrongly translated and the system will output an alternative suffix starting from that point. IMT is an active field and recent work has studied its application on NMT systems (Knowles and Koehn, 2016; Peris et al., 2017b). As in the post-editing scenario, being able to apply OL techniques into an IMT framework might be beneficial in order to develop more adaptive and productive MT systems.

Conclusions

In this work, three OL algorithms have been proposed and applied to an NMT system. They rely on the Passive-Aggressive OL approach in which the MT model is updated after every sample in order to fulfil a correctness criterion while remembering previously learned information. Because these algorithms have a high number of hyperparameters to tune, an extensive grid search has been carried out in order to find the best combination. In addition, a study has been made to see how the different parameters affect each algorithm's performance. Results show that, although they depend on several hyperparameters, those critical to the well behaviour of the algorithms are, in every case, the aggressiveness hyperparameter and the learning rate. Other parameters such as the margin or the number of iterations per sample offer minor improvements in the overall performance of the algorithm.

These algorithms have been tested in a post-editing scenario where an already trained NMT system has to be refined as soon as new samples are available. Each new sample that arrives to the system is translated and corrected (we use the reference sentence instead). A loss function is computed in order to measure the distance between both sentences. Once we have this, we use it to adapt the system, aiming to avoid the committed mistakes. In order to evaluate how well these new algorithms behave, a comparison between these algorithms and well-known OL gradient descent variants has been carried out. Results show that OL techniques help to refine and adapt the system on-the-fly, increasing the quality of translations and reducing the human effort in a post-editing scenario. We have found that the proposed PA-based algorithms offer a competitive performance: they improve the quality of translations in all tasks. Nevertheless, adaptive SGD-based algorithms performed generally better.

Finally, part of the work developed in this thesis has been submitted to the 2017 Conference on Empirical Methods in Natural Language Processing (Peris et al., 2017a). Moreover, we plan to submit an article to a journal (yet to be determined).

Bibliography

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv: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*, 35:3–28.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3:1137–1155.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Bentivogli, L., Bisazza, A., Cettolo, M., and Federico, M. (2016). Neural versus phrase-based machine translation quality: a case study. *arXiv preprint arXiv:1608.04631*.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Blum, A. (1998). On-line algorithms in machine learning. In *Online algorithms*, pages 306–325. Springer.

Bottou, L. (1991). Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91.

Boyd, S., Xiao, L., and Mutapcic, A. (2003). Subgradient methods. *lecture notes of EE392o, Stanford University, Autumn Quarter*, 2004.

Britz, D., Goldie, A., Luong, T., and Le, Q. (2017). Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

Brown, P. F., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. (1990). A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85.

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

Castaño, A. and Casacuberta, F. (1997). A connectionist approach to machine translation. In *Fifth European Conference on Speech Communication and Technology*.

Chen, S. F. and Joshua, G. (1998). An empirical study of smoothing techniques for language modeling. Technical report, Harvard Computer Science Group.

Chiang, D. (2012). Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 13:1159–1187.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chollet, F. et al. (2015). Keras. https://github.com/fchollet/keras.

Chow, Y.-L. and Schwartz, R. (1989). The n-best algorithm: An efficient procedure for finding top n sentence hypotheses. In *Proceedings of the workshop on Speech and Natural Language*, pages 199–202. Association for Computational Linguistics.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

Crammer, K. and Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

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

Federico, M., Bentivogli, L., Paul, M., and Stüker, S. (2011). Overview of the IWSLT evaluation campaign. In *Proceedings of the International Workshop on Spoken Language Translation*, pages 11–27.

Gentile, C. (2001). A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242.

Golik, P., Doetsch, P., and Ney, H. (2013). Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Proceedings of the Interspeech*, pages 1756–1760.

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356.

Guo, J. (2013). Backpropagation through time. *Unpubl. ms., Harbin Institute of Technology*.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hutchins, J. (2005). The history of machine translation in a nutshell. *Retrieved December*, 20:2009.

Jean, S., Firat, O., Cho, K., Memisevic, R., and Bengio, Y. (2015). Montreal neural machine translation systems for wmt'15. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 134–140.

Jordan, M. I. (1986). Attractor dynamics and parallellism in a connectionist sequential machine.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kivinen, J., Smola, A. J., and Williamson, R. C. (2004). Online learning with kernels. *IEEE transactions on signal processing*, 52:2165–2176.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184. IEEE.

Knowles, R. and Koehn, P. (2016). Neural interactive translation prediction. In *Proceedings of the Conference of the Association for Machine Translation in the Americas (AMTA)*, volume 1, pages 107–120.

Koehn, P. (2004). Statistical significance tests for machine translation evaluation. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 388–395.

Koehn, P. (2010). Statistical machine translation.

Lavie, M. D. C. D. A. (2014). Learning from post-editing: Online model adaptation for statistical machine translation. *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL 14)*, page 395.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521:436–444.

Li, Y. and Long, P. M. (2000). The relaxed online maximum margin algorithm. In *Advances in neural information processing systems*, pages 498–504.

Lu, J., Zhao, P., and Hoi, S. C. (2016). Online passive-aggressive active learning. *Machine Learning*, 103:141–183.

Luong, M.-T. and Manning, C. D. (2015). Stanford neural machine translation systems for spoken language domains. In *Proceedings of the International Workshop on Spoken Language Translation*.

Madsen, M. W. (2009). The limits of machine translation (thesis). *Center for Language Technology, Univ. of Copenhagen, Copenhagen*.

Martínez-Gómez, P., Sanchis-Trilles, G., and Casacuberta, F. (2011). Online learning via dynamic reranking for computer assisted translation. *Computational Linguistics and Intelligent Text Processing*, pages 93–105.

Martínez-Gómez, P., Sanchis-Trilles, G., and Casacuberta, F. (2012). Online adaptation strategies for statistical machine translation in post-editing scenarios. *Pattern Recognition*, 45(9):3193–3203.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proceedings of the Interspeech*, volume 2, page 3.

Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing*, pages 5528–5531. IEEE.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*, volume 13, pages 746–751.

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

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *International Conference on Machine Learning (3)*, 28:1310–1318.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of Empirical Methods in Natural Language Processing*, volume 14, pages 1532–1543.

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

Peris, Á., Cebrián, L., and Casacuberta, F. (2017a). Online learning for neural machine translation post-editing. *arXiv preprint arXiv:1706.03196*.

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

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65:386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Samson, R. (2005). Computer-assisted translation. *Training for the New Millennium: Pedagogies for translation and interpreting*, 60:101.

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

Shen, S., Cheng, Y., He, Z., He, W., Wu, H., Sun, M., and Liu, Y. (2016). Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1683–1692.

Shor, N., Zhurbenko, N., Likhovid, A., and Stetsyuk, P. (2003). Algorithms of nondifferentiable optimization: Development and application. *Cybernetics and Systems Analysis*, 39:537–548.

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, pages 223–231.

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.

Tatsumi, M. (2009). Correlation between automatic evaluation metric scores, post-editing speed, and some other factors. *Proceedings of the Twelfth Machine Translation Summit (MT-Summit XII)*, pages 332–339.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

Tiedemann, J. (2009). News from opus-a collection of multilingual parallel corpora with tools and interfaces. In *Recent advances in natural language processing*, volume 5, pages 237–248.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

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