UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Online Learning Techniques for Neural Translation Systems

DEGREE FINAL WORK

Degree in Computer Engineering

Author: Kevin Montalvá Minguet

Tutor: Francisco Casacuberta Nolla
Álvaro Peris Abril

Course 2015-2016

Resum

Els sistemes de traducció automàtica s'han fet servir desde la seva concepció per traductors professionals per a accelerar i facilitar la seva tasca. Aquestos sistemes reben traduccions editades professionalment a través del seu ús, lo que pot pontecialment millorar el seu rendiment. En les últimes décades, Xarxes Neuronals Artificials s'han utilitzat per a desenvolupar sistemes de traducció automàtica complets.

En aquest treball s'han utilitzat dos algoritmes d'aprenentatge online per a millorar traductors neuronals, que ja havien acabat l'etapa d'entrenament. Una amplia gamma d'experiments s'han dut a terme per trobar els hiperparàmetres óptims per els algoritmes en cada tasca de traducció, i després el rendiment d'aquestos sistemes, adaptats amb cadascun dels algoritmes aprenentatge (configurats amb aquestos hiperparàmetres óptims), s'ha mesurat abans i després en quatre tasques de traducció, i s'han extret conclusions sobre com han millorat o empitjorat.

S'ha modificat el codi del traductor neuronal, s'ha implementat un dels algoritmes presentants i s'ha desenvolupat un nou codi per oferir una interfície per a fer experiments de forma automatitzada. Una altra familia d'algoritmes ha sigut implementada i probada amb els models i tasques disponibles, sense resultats positius.

Finalment, línies futures d'investigació en adaptació de traductors neuronals han sigut considerades i discutides al final d'aquest treball, a més de la seva situació en l'estat actual de la traducció automàtica.

Paraules clau: aprenentatge automàtic, xarxes neuronals, xarxes neuronals recurrents, traducció automàtica, traducció neuronal

Resumen

Los sistemas de traducción automática han sido utilizados desde su concepción por traductores profesionales para acelerar y facilitar su tarea. Estos sistemas reciben traducciones editadas profesionalmente a través de su uso, que pueden potencialmente mejorar su rendimiento. En las últimas décadas, se han utilizado redes neuronales artificiales para desarrollar sistemas de traducción automática completos con éxito.

En este trabajo se han usado dos algoritmos de aprendizaje online para mejorar redes neuronales ya entrenadas. Un amplio abanico de experimentos se ha llevado a cabo para encontrar el conjunto óptimo de hiperpárametros para los algoritmos en cada tarea, y se ha calculado el rendimiento de estos sistemas, adaptados con cada algoritmo con sus hiperparámetros óptimos encontrados de forma empírica. Se han extraído conclusiones acerca de la mejora o empeoramiento de los traductores.

Se ha llevado a cabo una modificación de la base de código del traductor neuronal, junto con la implementación de uno de los algoritmos y el desarrollo de una nueva herramienta para proporcionar una interfaz para realizer experimentos de forma automatizada. Otra familia de algoritmos ha sido implementada y probada con los modelos y tareas disponibles, con resultados insatisfactorios.

Finalmente, líneas potenciales de investigación en adaptación de traductores neuronales han sido consideradas y discutidas al final de este trabajo, junto con su situación en el estado actual de la traducción automática.

Palabras clave: aprendizaje automático, redes neuronales, redes neuronales recurrentes, traducción automática, traducción neuronal

Abstract

Machine Translation systems have been used since their inception by professional translators to speed up and ease their work. Those systems receive professionally edited translations through their use, which could potentially improve their performance. In the last decades, Artificial Neural Networks have been used to develop complete Machine Translation systems to great success.

In this work two online learning algorithms for Artificial Neural Networks have been used to enhance already trained neural translators. A wide array of experiments have been carried out to find the optimal hyperparameters for the algorithms in each task, and then the performance of those systems, adapted with each algorithm with their empirically found optimal set of hyperparameters, has been measured before and after in four translation tasks, and conclusions have been extracted on how they improved or worsened.

A modification of a neural translator codebase has been carried out, along with the implementation of one of the algorithms and the development of new codebase to provide an interface to perform experiments in an automated way. One additional family of algorithms has been implemented and tested with the available model and tasks to no avail.

Finally, possible future lines of research on adaptation of neural translators have been considered and discussed at the end of this work, along with their situation in the current state of Machine Translation.

Key words: machine learning, neural networks, recurrent neural networks, machine translation, neural translation

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1
Introduction

1.1 Background

In 1949 Warren Weaver laid the foundations for Machine Translation (MT) by proposing the use of computers to tackle the challenge of translation. Consequently, in the 50s and 60s, several attempts were made to create practical translators, and by the mid sixties the newly created ALPAC (Automatic Language Processing Advisory Committee)[15] published a report in which they painted the future of MT as bleak.

About 10 years later nonetheless, after the conception of some new approaches and the increase in processing power of computers, there was a renewal of the expectancies for MT, and research and interest rose back up.

So far, MT is a challenge that has been engaged with multiple approaches and methodologies, but most of them can, nowadays, be classified into three categories[10]: Rule-Based approaches, Corpus-Based ones and a hybrid approach in between. Rule-Based Systems (RBS) require the use of human knowledge on languages, and the use of grammars and vocabularies. Corpus-Based Systems (CBSs) use big parallel corpora of text for the system to learn them and reproduce their features. A big advantage of CBSs over Rule-Based Systems (RBSs) is that language data is readily available for almost any language in the world, a big amount of information in several languages. From books to news, movie scripts, trial transcripts and laws, there is a large number of sources from which data can be extracted that Corpus-Based translators can use to improve the quality of their product. Since CBSs are *trained* on data, they also have (usually) the possibility of scaling up and out with hardware, while RBSs are usually limited by the quality of their rules.

1.2 Motivation

Professional translators use MT systems regularly to speed up their task. The translations obtained through those systems though are of irregular quality, and many if not most of them must be amended by human translators to obtain a satisfactory result. Those corrections can be used to improve the MT system being used to, in the long term, reduce the number of corrections that the professional needs to apply to future translations. The limit being of course "perfect translations", where no corrections are needed.

The correction of machine-generated translations human translators is known as post-edition. In a post-edition context, the human translator inputs a source sentence, the software generates a hypothesis in the target language and the human corrects the translation. In a simple system this correction is only useful to the user, since the system will

not benefit from it. The goal here is to exploit this new information by feeding it back to the system to enhance it. Every pair of source sentence and post-edited translation is a sample of data that the system can use to try to fix its defficiencies, the parts of its model that decided that the imperfect, generated hypothesis was more desirable to the user than the corrected translation.

There are different approaches on how to modify the model to fit this new data. In this thesis, two different online learning algorithms for this task will be compared: AdaGrad and Stochastic Gradient Descent (SGD). A third family of algorithms called Passive-Aggressive will also be studied. A set of already-trained models will be adapted with the first two algorithms and the results will be compared to their previous performance, to gauge how much they have improved.

1.3 Goals

Having more than one online learning algorithms available for harnessing the data produced by professional translators, one of them has to be chosen for each task so that the systems are as capable as possible. Our objective in this work is to find out whether any of the algorithms being discussed in section 3.2 dominates the other (in these tasks) or whether they perform better or worse than one another depending on the task in question.

To that end, we will use different translation tasks to measure the quality of the translations before and during an online training scenario. The *before* measure represents the quality of the translations issued by the model without intervention of the retraining algorithms. The *during* measure is obtained by evaluating the quality of each translation obtained by the model before being retrained on the relevant sentence pair. That is, first we obtain a translation for a source sentence, then we retrain the model with that sentence and its respective target sentence. Thus, the *during* measure represents the quality of the translations that were issued by a model being retrained with a given algorithm, emulating the post-edition case that was presented before. It is to be expected then, that if the model improves through this process, a translator based on this model and using this retraining algorithm when receiving data through normal use will improve as well. We assume also that, when independently retraining the same model with two different algorithms and obtaining a higher degree of enhancement in the hypotheses generated by one of them than in the ones generated by the other, this relationship will hold also for two software translators based on that model and using the respective algorithms for retraining.

Since both algorithms we compare have hyperparameters, it is essential to find a good approximation of the hyperparameters that work best for each combination of algorithm and task. Otherwise, we could be using a set of hyperparameters that makes an otherwise effective algorithm aggressively change the model, making the algorithm look faulty or useless when it only has been badly configured.

Therefore, once we have found the appropriate hyperparameters for each algorithm and task, we will use this combination to obtain *before* and *during* measures of the retraining of the models on the respective tasks using the relevant algorithms and hyperparameters. With these results we will be able to judge the affirmations made at the beginning of this section.

To accomplish this, the relevant parts of the software that powers the models which will be used for experimentation will have to be modified to accommodate the new algorithm implementations and the experiments environment. Moreover, new code will have

to be produced to provide an interface used to conduct experiments in an automated and flexible way.

1.4 Structure

In Chapter 2 of this document, the basis of Neural Machine Translation (NMT) is laid out and the state of the art discussed. In Chapter 3, the aforementioned online learning algorithms are introduced, and their characteristics are explained. Chapter 4 describes the nature of the data that was used in the experiments, how those experiments were performed and their results. Also, information about the software used in this work and implementation details of the algorithms can be found here. Finally, Chapter 5 contains the conclusions drawn from the obtained results and what additional research is necessary to further the goals of this work.

CHAPTER 2
State of the art

Natural Language Processing (NLP) is a knowledge field within Artificial Intelligence and Computational Linguistics, devoted to the study of methods that involve computers understanding or manipulating human languages.

NLP comprises several tasks, the most popular of them usually being Speech Recognition, Text Recognition, Speech Synthesis and Machine Translation. This thesis is concerned with a very particular topic within the last of them.

2.1 Statistical Machine Translation

In MT, within the Corpus-Based Systems, there is Statistical Machine Translation (SMT). The goal of a SMT system is to find, given a source sentence x of length J in a source language $\mathcal{F}$, a target sentence y of length I in the target language $\mathcal{E}$. If $P(y|x)$ is the probability of y being the closest translation of x in the target language, we want to find $\hat{y}$ such that:

$$\hat{y} = \arg\max_{y \in \mathcal{E}} P(y|x) \tag{2.1}$$

Most of the popular MT systems use the so-called phrase-based models [17, 9] to model $P(y|x)$. Phrase-based translation works by breaking down the source sentence into phrases, translating those phrases and then reordering those translations to obtain the target sentence.

State-of-the-art SMT systems, though, use what is known as the log-linear approach [20]. In this approach, the candidate is selected as a weighted sum of statistical models:

$$\hat{y} = \arg\max_{y \in \mathcal{E}} \sum_{m=1}^{M} \lambda_m h_m(y, x) \tag{2.2}$$

where λ_m is the weight of model m, $h_m(y, x)$ is the feature function that represents this model and M is the number of models. Commonly used features in this approach are:

1. The target language model $p(y)$.
2. The phrase-based model and inverse phrase-based model, $p(y|x)$ and $p(x|y)$.

3. A reordering model.
4. Target word penalty (ωI) and phrase penalty (ρK), where I is the length of the target sentence, and K the number of phrases the source sentence is broken down into. This is to compensate for the system bias towards short sentences.

2.2 Neural Machine Translation

In the late 90s, there were attempts to use Neural Networks to tackle the task of MT [6]. They, however, have not been embraced by the MT academia until very recently. In 2014, Cho et al. and Sutskever et al. independently proposed two similar approaches to NMT. Both of them describe a system with sentences as input, that are projected into a constant-length continuous vector representation by a process known as *encoding*, and retrieved as translations by a process known as *decoding*.

The use of a continuous representation for words and sentences dates back to 1997, when it was used by Bellegarda[3] in an n-gram based statistical language model. This representation offers many advantages that are not found in classical, discrete representations of language. One of the most important ones is that continuous representations allow to account for much richer syntactic and semantic relationships. As seen in [18], those relationships have several degrees, and are preserved in the high-dimensional space where the words are projected. Traditional n-gram based approximations to language modeling fail to account for word similarity, and rely on the training set to contain most word combinations or segments of them. This property allows NLP systems that use this approach to find similar, or in some way related, words close to each other by some measure, that can be computed from just their feature vectors. As an example obtained from [18], if we denote the continuous vector representation of a word w as $\text{Vector}(w)$, it was found that the result of the operation $\text{Vector}(\text{“}King\text{”}) - \text{Vector}(\text{“}Man\text{”}) + \text{Vector}(\text{“}Woman\text{”})$ resulted in a vector that was close to $\text{Vector}(\text{“}Queen\text{”})$.

The approach followed in this work consists of the use of a matrix **E** for each language (one *source language matrix* and one *target language matrix*). **E** has as many rows as the vocabulary size of the language. The number of columns of **E** is a parameter, which represents the dimensionality of the continuous representation. The latter is set arbitrarily, trying to find a suitable balance between efficiency (high dimensionality will make it harder and longer to train, given the sparseness of the space and the increase in number of trainable parameters) and efficacy (low dimensionality will give little advantage over integer representations, given how little information can be stored/used in a few dimensions). Since this work deals with already trained models, the size and current values of the matrix **E** is provided along with the other parameters, and is retrained in the same way.

As an example, consider an already trained model is provided, with a vocabulary (of the source language) consisting of just two words: “King” and “Queen”, in that order. The dimensionality of the continuous representation had been chosen to be 3 (again, arbitrarily; and deliberately small for the purpose of the example). Any possible matrix **E** for this model will have two rows (the size of the vocabulary) and three columns (the dimensionality). A possible instance of it would be the following:

$$\begin{bmatrix} 1 & 2.1 & 3 \\ 4 & 5 & 6.4 \end{bmatrix}$$

$\text{Vector}(\text{“}Queen\text{”})$ is the row corresponding to the word “Queen” in the matrix. At this point $\text{Vector}(\text{“}Queen\text{”}) = \begin{pmatrix} 1 & 2.1 & 3 \end{pmatrix}$, so this matrix can also be thought as a lookup

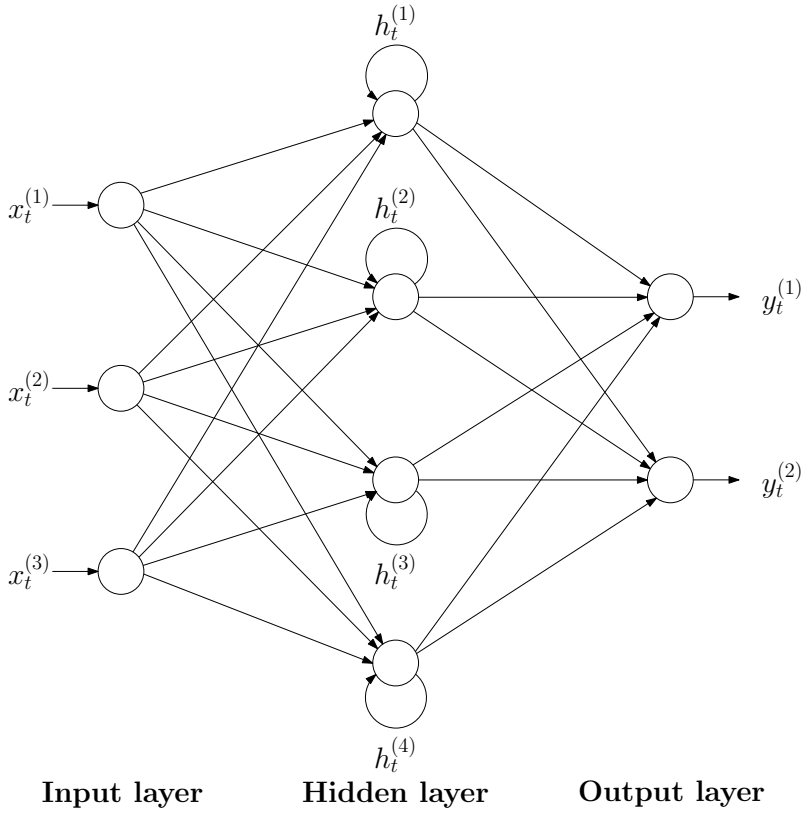

Figure 2.1: One example of 3-4-2 Recurrent Neural Network topology. $x_t^{(i)}$ stands for the i-th input at time-step t, $h_t^{(i)}$ is the i-th element of the hidden state at t and $y_t^{(i)}$ is the i-th output at t. The loops in the hidden layer are what set them aside from regular Neural Networks.

table. The initial values of the language matrices are arbitrary, and the whole matrix is modified like another weight matrix of the system in the training phase. That means that the vector representation of a given word may change repeatedly as part of the training.

There is another advantage to using continuous representations: since the words are real-valued vectors, one can expect the function to be learned by our model to have some local smoothness properties. Those properties are absent when words are discrete values, because a change in one of the words can result in drastic differences in the value of the function to be learnt[4].

2.2.1. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of Neural Networks where some connections of the network constitute a directed cycle. Those cycles allow the units involved to hold a sort of hidden state, that contains information of previous inputs of the network. This behavior makes RNNs the perfect candidate to process variable-length sequences, like sentences: we can process the words one by one taken into account other words in the sentence, independently of how long it is, because the representation and processing of those words are encoded in the hidden state.

Given this hidden state, the output vector $\mathbf{y}_t$ of a RNN is no longer a function of its input vector $\mathbf{x}_t$ only, but also of its hidden state at the current time-step $\mathbf{h}_t$. Since this hidden state is in turn a function of the current input and the previous hidden state (recursively, of the previous inputs), we can say that $\mathbf{y}$ is just a function of the hidden state, as indicated in Equation (2.3) and Equation (2.4).

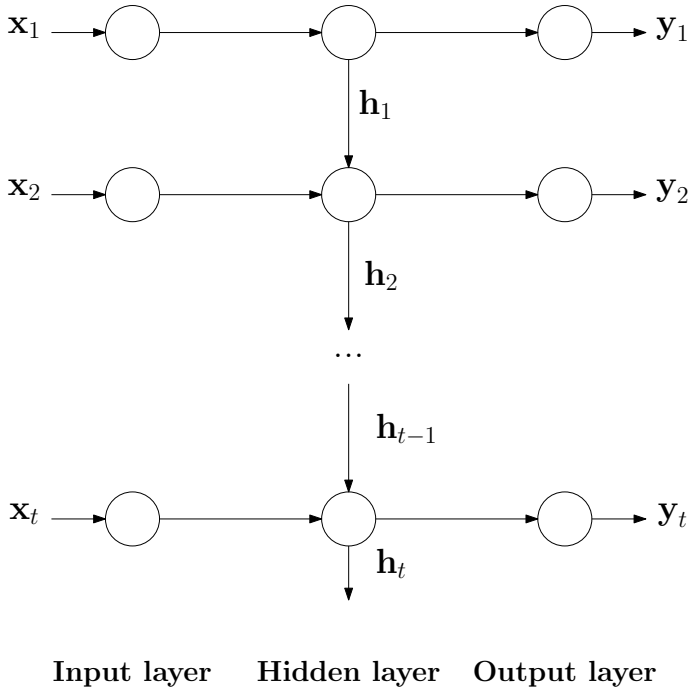

Figure 2.2: Visualization of a RNN unfolded in time. $\mathbf{x}_t$ stands for the t-th input processed by the network (alternatively, at the t-th iteration), $\mathbf{h}_t$ is the hidden state of the network at iteration t and $\mathbf{y}_t$ is the output of the network at iteration t.

$$\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{2.3}$$
$$\mathbf{y}_t = f_o(\mathbf{h}_t) \tag{2.4}$$

f_h and f_o depend on the architecture of the network and the choice of activation functions.

RNNs, however, have one well-known shortcoming when it comes to sequence processing: the vanishing gradient problem[5]. Gradient-based training algorithms update the network weights in proportion to the gradient of the error function with respect to each of them. Since layers "further away" from the output layer (in terms of connections) have smaller contributions to output values, this aforementioned gradient is in turn smaller. In the case of sequence-processing RNNs, this means that information from elements far "in the past" is almost lost. To solve this problem, many RNN-based systems have successfully used the RNN architectures that are about to be discussed.

In the late 90s, a family of RNNs appeared, called Long Short Term Memory (LSTM) networks [14], which have shown very good results in many sequence learning tasks. They make use of a set of gates to keep some kind of memory. LSTM units are trained to retain different degrees of memory, which allows RNNs to successfully remember long sequences. [25] used LSTMs to build a relatively simple neural translator that achieved results comparable to those of state-of-the-art SMT systems.

Another type of gated units are GRUs, developed in [7] for a phrase scoring system based on Neural Networks, as a part of a Phrase-Based translator. They are simpler than regular LSTM units, they have fewer gating units, thus reducing their training time. They, however, have been shown to be as powerful as LSTM.

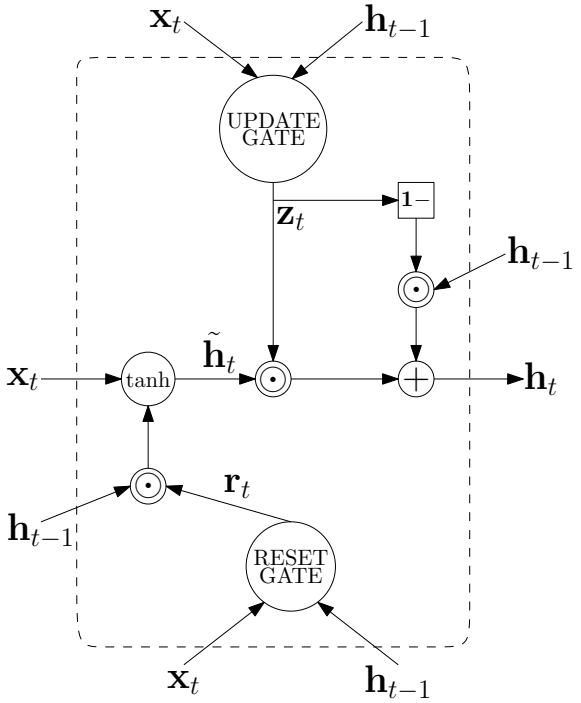

Figure 2.3: Illustration of the Gated Recurrent Unit (GRU). The reset gate adjusts how much of the previously stored information is retained and the update gate adjusts how much of the newly acquired (current iteration) information is kept. $\mathbf{x}_t$ and $\mathbf{h}_t$ follow the same notation as in Figure 2.2, while $\tilde{\mathbf{h}}_t$ represents the updated state at iteration t, $\mathbf{r}_t$ is the output of the reset gate at iteration t and $\mathbf{z}_t$ is the output of the update gate at iteration t.

At a given time t, a GRU cell holds a hidden state $\mathbf{h}_t$. Given its hidden state in the last iteration (at $t-1$), its current updated state $\tilde{\mathbf{h}}_t$ and the output of the update gate $\mathbf{z}_t$, the current hidden state will be computed as follows:

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{2.5}$$

where $\odot$ stands for the element-wise multiplication.

The updated state $\tilde{\mathbf{h}}_t$ can be computed from the current input $\mathbf{x}_t$, the previous hidden state $\mathbf{h}_{t-1}$ and the output of the reset gate $\mathbf{r}_t$:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}[\mathbf{r}_t \odot \mathbf{h}_{t-1}]) \tag{2.6}$$

where $\mathbf{W}$ and $\mathbf{U}$ are weight matrices, parameters of the model. Bias terms have been left out for readability.

Finally, the output of the reset and update gates are computed following these formulae:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1}) \tag{2.7}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1}) \tag{2.8}$$

where $\mathbf{W}_r$ are $\mathbf{U}_r$ reset gate weight matrices, $\mathbf{W}_z$ and $\mathbf{U}_z$ are update gate weight matrices and σ is the element-wise logistic function.

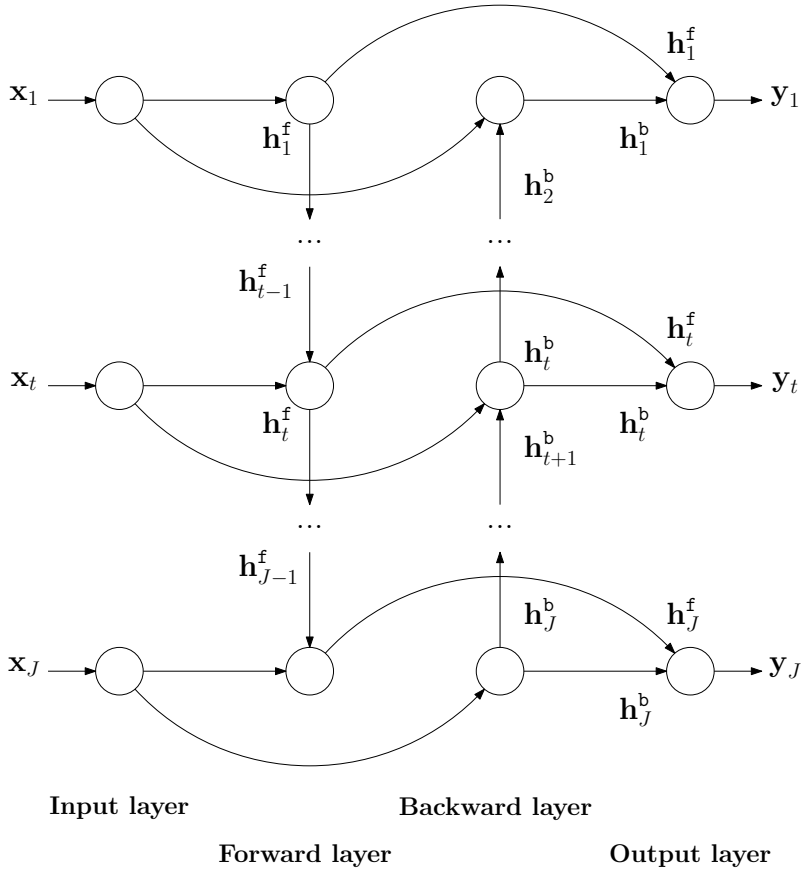

Figure 2.4: Structure of a BRNN.

2.2.2. Bidirectional Recurrent Neural Network

In order to improve the quality of the translations, when it comes to processing a given word in the middle of a sentence, we can choose to examine not only the words that precede but also the ones that follow it. In order to do that, we need to process the sentence both ways: forward and backwards. The current architecture of RNN that has been introduced only accounts for previous words, but we can include an additional hidden layer, independent of the previous one, that will process words from last to first. This architecture, introduced in [24] as Bidirectional Recurrent Neural Network (BRNN), will allow the network to take decisions based on the whole sentence context.

In this architecture, there are two hidden states: forward ($\mathbf{h}^{\mathsf{f}}$) and backward ($\mathbf{h}^{\mathsf{b}}$), that are computed as follows:

$$\mathbf{h}_t^{\mathsf{f}} = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{2.9}$$

$$\mathbf{h}_t^{\mathsf{b}} = f_h(\mathbf{x}_t, \mathbf{h}_{t+1}) \tag{2.10}$$

$$\mathbf{y}_t = f_o(\mathbf{h}_t^{\mathsf{f}}, \mathbf{h}_t^{\mathsf{b}}) \tag{2.11}$$

2.2.3. Encoder-Decoder model

In the model proposed by Cho et al. (2014) and Sutskever et al. (2014), we have a system composed of two RNNs: an Encoder and a Decoder. The model used in this work is the one proposed by Bahdanau et al. (2014), which is an extension of the aforementioned two.

The Encoder, which in our case is a BRNN, encloses the input sentence in a context vector. By means of reading each element of the input sentence, the hidden state of the network changes, and after completing the reading it is a compendium of the sentence. In our case, the context vector is obtained from the concatenation of the hidden states of the forward recurrent layer and the backward recurrent layer.

The Decoder in turn has the task to generate the output sentence from the aforementioned context vector. Iteratively, the Decoder will generate a word given the context vector and the previously generated words. Once the Decoder generates a special "end of line" word, the complete sentence is the system's hypothesis for the source sentence.

For each word x_j ($1 \leq j \leq J$) in the source sentence, belonging to the source vocabulary V_s, we produce a vector $\overline{\mathbf{x}}_j \in \{0,1\}^{V_s}$, where every entry is set to zero except the one corresponding to x_j, which is set to one. This is called one-hot codification. Then, the words are projected to a fixed-size continuous vector in the following way:

$$\mathbf{x}_j = \mathbf{E_s}\overline{\mathbf{x}_j} \tag{2.12}$$

where $\mathbf{x}_j$ is the embedding of word x_j and $\mathbf{E_s}$ is the source language projection matrix.

The sequence of word embeddings, represented as $\mathbf{x} = \mathbf{x}_1, ..., \mathbf{x}_J$, is the input of the Encoder. After processing each word $\mathbf{x}_j$, the hidden state of the Encoder $\mathbf{h}_j$ is recorded. Since we have opted for using a BRNN as the Encoder, our hidden state is actually

$$\mathbf{h}_j = [\mathbf{h}_j^{\mathsf{f}\intercal}; \mathbf{h}_j^{\mathsf{b}\intercal}]^\intercal \tag{2.13}$$

Once the sentence has been fully processed, an iterative process begins. At each step, a non-linear function q is applied to the sequence of hidden states and to the hidden state of the Decoder network at the previous step, in order to obtain a context vector $\mathbf{c}$. In this work an attention mechanism has been used, therefore the function q is a weighted sum of the hidden states. This works like an alignment model, implemented by a Multilayer Perceptron (MLP), between the source sentence and the target sentence, and thus, we have a different context vector $\mathbf{c}_i$ for each step i:

$$\mathbf{c}_i = q(\{\mathbf{h}_1, ..., \mathbf{h}_J\}, \mathbf{g}_{i-1}) \tag{2.14}$$

Thus, $\mathbf{c}_i$ is a dynamic representation of the input sentence, based in the state of the Decoder.

This context vector is then fed to the Decoder. The Decoder processes the context vector, and outputs V_s real numbers between zero and one, where the i-th output represents the probability of the i-th word in the target language to be next in the translation of the source sentence. This output depends on $\mathbf{c}_i$, $\mathbf{g}_i$ and the word embedding representation of the last emitted word. This is implemented through a softmax layer, which ensures that all the probabilities add up to one:

$$y'_k = \frac{y_k}{\sum_{l=1}^{V_s} y_l} \tag{2.15}$$

where y_i represents the i-th input of the softmax layer, and y'_i represents its i-th output, always between zero and one.

Therefore, the probability of a word at time-step i would be:

$$p(y_i|y_1, ..., y_{i-1}, x; \theta) = \overline{\mathbf{y}}_i^\intercal \boldsymbol{\varphi}(\mathbf{V}\boldsymbol{\eta}(y_{i-1}, \mathbf{g}_i, \mathbf{c}_i)) \tag{2.16}$$

where $\boldsymbol{\varphi}(\cdot)$ is a softmax function, $\overline{\mathbf{y}}_i$ is the one-hot vector representation of word y_i, $\mathbf{V}$ is the weight matrix and $\boldsymbol{\eta}$ is the output of a RNN with GRU units and a maxout output layer[12].

2.2.4. Training

Following Equation (2.1), our network aims to approximate $P(y|x)$ in the following way:

$$P(y|x) = \prod_{i=1}^{I} P(y_i|y_1, ..., y_{i-1}, x) \tag{2.17}$$

In order to maximize $P(y|x)$ for our training set, consisting of a bilingual corpus of $\mathcal{S}$ sentence pairs, and according to Equation (2.17), we need to find a set of parameters for our model $\hat{\theta}$ such as:

$$\begin{aligned}\hat{\theta} &= \arg\max_{\theta} \prod_{s=1}^{S} \prod_{i=1}^{I} p(y_i^{(s)}|y_1^{(s)}, ..., y_{i-1}^{(s)}, x^{(s)}; \theta) \\ &= \arg\max_{\theta} \sum_{s=1}^{S} \sum_{i=1}^{I} \log(p(y_i^{(s)}|y_1^{(s)}, ..., y_{i-1}^{(s)}, x^{(s)}; \theta))\end{aligned} \tag{2.18}$$

where $x^{(s)}$ and $y^{(s)}$ represent the s-th sentence of the training set in the source and target languages respectively, and I is the length of the s-th target sentence.

Since each word of the system's hypothesis depends on previously generated words and the context vector, and the context vector depends only on the source sentence, both Encoder and Decoder can be trained as a whole to maximize the conditional probability of the target sentences given the source sentences.

So far, we have introduced the knowledge field of Machine Translation. We have outlined the approaches that have been adopted in the last decades to advance the quality of translators, and we have described state-of-the-art techniques that power the top translation software in the field. Finally, we have reviewed in depth the Encoder-Decoder approach and the Neural Networks employed in it. What follows is a description of the work that was carried out in order to perform the experiments.

CHAPTER 3
Online learning

3.1 Training Neural Networks

Our objective in training is to maximize the log-likelihood of the data we use for training, in an attempt to produce a system that can generalize that set of translations into the overall translation task. By following Equation (2.18), we can tune the system parameters θ to maximize this sum of log-probabilities. The trainable parameters of RNNs are the weight matrices. Most Neural Network training algorithms are iterative, they update the network weights according to a rule step by step, until a given condition is accomplished. Update rules, step size and stopping condition define the different learning techniques. The following is a classification by step size[19]:

- **Batch learning** techniques are those that update the weights of the network after the whole training set has been processed. Once the system has evaluated every sample, its weights are updated to fit those, and a new training iteration begins. If the termination condition is reached, the training stops.
- **Mini-batch learning** techniques indicate that the updates must be applied after an arbitrary number of samples have been processed.
- **Online learning** techniques, finally, are those that update the network weights after every single sample. They are a particular interest of us, on account of them being a perfect fit for the situation described at the beginning of this work: improving a system after a new sample is obtained.

  With the goal of enhancing a neural translator with its use, a setup like the following can be adopted: the system generates translations for a human translator, who inputs corrected versions of those translations, which the system uses one by one to update the weights of its internal Neural Network.

3.2 Online learning algorithms

In the next sections we describe the algorithms we have chosen to compare.

3.2.1. Stochastic Gradient Descent

SGD is a learning algorithm[22] that approximates Gradient Descent by updating the weights of the system using the following rule:

$$\theta_s = \theta_{s-1} - \eta \nabla \ell_s(\theta_{s-1}) \tag{3.1}$$

This update is performed with the gradient for each sample of the training set, thus approximating the gradient of the whole set.

In (3.1) η is the *learning rate*, which sets the pace of the updates to the model. This is the only parameter we can tune in this algorithm. It is usually set to values lower than one, in order to modify the model in small steps towards minima in the error function. Every time the weights of the network are updated, the system tries to improve its performance towards the new sample, and in the process, its performance with previously seen samples may get worse. In order to try to achieve a good performance in the target data of the system as a whole, the learning rate is used. If the learning rate were too high, the system would aggressively try to fit new samples at the expense of past data, resulting likely in an overall bad performance. If it were too small, the network would conservatively learn the data, requiring a very high number of iterations, and thus a very long time, to be trained.

3.2.2. AdaGrad

AdaGrad is a family of adaptive, subgradient, online learning algorithms developed in [11], based on SGD, that is expected to outperform it for high-dimensional, sparse features. The implementation used in this work is an approximation obtained from [13]. Its update rule is as follows:

$$v_s = v_{s-1} - \eta G_s^{-1/2} \nabla_v \ell \tag{3.2}$$

$$G_s = G_{s-1} + (\nabla_v \ell)^2 \tag{3.3}$$

where v is any given weight of θ, $\nabla_v \ell$ represents the gradient of the loss function with the previous weight set with respect to weight v before processing sample s and G_s represents the sum of squared gradients before processing sample s. At any given time, $G_s = \sum_{i=1}^{s} \nabla_v \ell_i^2$.

In this case, we also have a *learning rate* parameter that we can tune in order to seek the optimal performance of the algorithm.

3.2.3. Passive-Aggressive

Passive-Aggressive are a family of margin-based online learning algorithms, proposed in [8]. The goal of those algorithms is to find, at each step, the model which, being as close as possible to the current one, achieves some given margin on the current sample. This is a constraint optimization problem that is solved by the Lagrange multipliers technique to find an update rule that meets the conditions. Since the margin requirement might be a hard one, the PA-II and PA-III algorithms include an "aggressiveness" hyperparameter that allows for a trade-off between the desired margin and the proximity to the current model.

These algorithms have the following update rule:

$$\boldsymbol{\theta}_{s+1} = \boldsymbol{\theta}_s + \text{sign}(\mathbf{y}_s - \hat{\mathbf{y}}_s)\boldsymbol{\tau}_s\mathbf{x}_s \tag{3.4}$$

where $\boldsymbol{\tau}_s$ depends on the particular algorithm:

$$\boldsymbol{\tau}_s = \begin{cases} \frac{\ell_s}{||\mathbf{x}_s||^2}, & \text{PA} \\ \min(C, \frac{\ell_s}{||\mathbf{x}_s||^2}), & \text{PA-I} \\ \frac{\ell_s}{||\mathbf{x}_s||^2 + \frac{1}{2C}}, & \text{PA-II} \end{cases} \tag{3.5}$$

where C is a parameter called *aggressiveness* in [8] and ℓ_s is the value of the loss function at time t.

As we can see in Equation (3.4) and Equation (3.5), PA-I has no hyperparameters and PA-I and PA-II have one: C.

In order to solve $\hat{\theta} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}_s||^2$ s.t. $\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) \leq 0$ we use the Lagrange multipliers technique:

$$\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) = \log p_{\hat{\boldsymbol{\theta}}}(\mathbf{h}_s|\mathbf{x}_s) - \log p_{\hat{\boldsymbol{\theta}}}(\mathbf{y}_s|\mathbf{x}_s) \tag{3.6}$$

We start by obtaining Lagrange function:

$$L(\boldsymbol{\theta}, \lambda) = \frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}_s||^2 + \lambda\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) \tag{3.7}$$

where λ is a Lagrange multiplier.

Next we obtain the gradient, which would be zero at the minimum:

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \lambda) = \boldsymbol{\theta} - \boldsymbol{\theta}_s + \lambda\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) = 0 \tag{3.8}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta}_s - \lambda\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) \tag{3.9}$$

Afterwards, we get the pseudo-dual function:

$$L_D(\boldsymbol{\theta}, \lambda) = \frac{1}{2}\lambda^2||\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2 + \lambda\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) \tag{3.10}$$

As we did before, we look for the minimum:

$$\frac{\partial L_D(\boldsymbol{\theta}, \lambda)}{\partial\lambda} = \lambda||\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2 + \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) = 0 \tag{3.11}$$

$$\hat{\lambda} = -\frac{\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2} \tag{3.12}$$

Which is the optimal solution for the Lagrange multiplier λ. Along with Equation (3.9), we can obtain the pseudo optimal solution:

$$\boldsymbol{\theta} = \boldsymbol{\theta}_s + \frac{\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2} \tag{3.13}$$

PA-I requires solving $\hat{\theta} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}_s||^2 + C\xi$ s.t. $\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) \leq \xi$, which is done similarly to equations 3.7 to 3.13:

$$
\begin{aligned}
L(\boldsymbol{\theta}, \lambda_1, \lambda_2) &= \frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}_s||^2 + C\xi + \lambda_1(\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) - \xi) - \lambda_2\xi \\
\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \lambda_1, \lambda_2) &= \boldsymbol{\theta} - \boldsymbol{\theta}_s + \lambda_1 \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) = 0 \\
\boldsymbol{\theta} &= \boldsymbol{\theta}_s - \lambda_1 \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) \\
\frac{\partial L(\boldsymbol{\theta}, \lambda_1, \lambda_2)}{\partial \xi} &= C - \lambda_1 - \lambda_2 = 0 \rightarrow C = \lambda_1 + \lambda_2 \\
L_D(\boldsymbol{\theta}, \lambda_1, \lambda_2) &= \frac{1}{2}\lambda_1^2 ||\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2 + C\xi \\
&\quad + \lambda_1 \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) - (\lambda_1 + \lambda_2)\xi \\
\frac{\partial L_D(\boldsymbol{\theta}, \lambda_1, \lambda_2)}{\partial \lambda_1} &= \lambda_1 ||\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2 + \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) = 0 \\
\hat{\lambda}_1 &= \min(C, -\frac{\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2}) \\
\boldsymbol{\theta} &= \boldsymbol{\theta}_s - \min(C, -\frac{\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2}) \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)
\end{aligned} \tag{3.14}
$$

whereas PA-II requires solving

$$
\hat{\theta} = \arg\min_{\boldsymbol{\theta}} \frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}_s||^2 + C\xi^2 \text{ s.t. } \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) \leq \xi
$$

$$
\begin{aligned}
L(\boldsymbol{\theta}, \lambda) &= \frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\theta}_s||^2 + C\xi^2 + \lambda(\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) - \xi) \\
\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \lambda) &= \boldsymbol{\theta} - \boldsymbol{\theta}_s + \lambda \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) = 0 \\
\boldsymbol{\theta} &= \boldsymbol{\theta}_s - \lambda \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) \\
\frac{\partial L(\boldsymbol{\theta}, \lambda)}{\partial \xi} &= 2C\xi - \lambda = 0 \rightarrow \xi = \frac{\lambda}{2C} \\
L_D(\boldsymbol{\theta}, \lambda) &= \frac{1}{2}\lambda^2 ||\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2 + C\Big(\frac{\lambda}{2C}\Big)^2 \\
&\quad + \lambda \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) - \lambda\Big(\frac{\lambda}{2C}\Big) \\
\frac{\partial L_D(\boldsymbol{\theta}, \lambda)}{\partial \lambda} &= \lambda ||\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2 + \frac{\lambda}{2C} + \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) - \frac{\lambda}{C} = 0 \\
\hat{\lambda}_1 &= -\frac{\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2 - \frac{1}{2C}} \\
\boldsymbol{\theta} &= \boldsymbol{\theta}_s + \frac{\ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s) \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \mathbf{x}_s, \mathbf{y}_s, \mathbf{h}_s)||^2 - \frac{1}{2C}}
\end{aligned} \tag{3.15}
$$

As can be seen in Equation (3.13), $\boldsymbol{\theta}$ is found in both sides of the equation, thus the need for the approximation using fixed-point iterators, which can be seen in Algorithm 3.1 for the PA algorithm, while PA-I and PA-II are identical, requiring only a change in the update line for the corresponding formula, to be like Equation (3.14) and Equation (3.15).

Although the implementation is further explained in Section 4.1.2, it is worth noting that in the preliminary experiments no promising results were achieved with either of the three versions of the algorithm, and therefore it was dropped from the experimentation plan towards the end of the project.

Input:
Parameters at the beginning of the iteration $\boldsymbol{\theta}_s$
Source sentence $\mathbf{x}_s$
Target sentence $\mathbf{y}_s$
Hypothesis $\mathbf{h}_s$
Output: $\boldsymbol{\theta}_{new}$
Initialization: $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_s$
repeat
1. $\boldsymbol{\theta}_{old} = \boldsymbol{\theta}_{new}$
2. $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_s + \frac{\ell(\boldsymbol{\theta}_{old},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)\nabla_{\boldsymbol{\theta}=\boldsymbol{\theta}_{old}}\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}=\boldsymbol{\theta}_{old}}\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)||^2}$

until $\boldsymbol{\theta}_{old} == \boldsymbol{\theta}_{new}$

Algorithm 3.1: Passive-Aggressive approximation pseudocode.

CHAPTER 4
Experiments

4.1 Software

The NMT system used in the experiments of this work are built in a Python library called GroundHog, and the pertinent modifications were made with Theano, another Python library on which GroundHog is based.

4.1.1. Theano

Theano is a Python library that allows for the compilation and optimization of symbolically-specified linear algebra functions[2, 26]. It also allows for the execution of those operations in GPUs easily, which greatly speeds up computations when a GPU is available.

It was extensively used on top of GroundHog to implement the AdaGrad and Passive-Aggressive algorithm in this work. The SGD algorithm was already implemented on GroundHog.

The main part of the training algorithms is the update rule, and its application is implemented through Theano functions. A Theano function, in a simplified way, is a routine that has input values, output values and update rules. Those update rules were used to implement the weight and parameter updates in the learning algorithms. They are executed "in parallel", meaning that if a variable was updated in the function but a different update depended on that variable, the latter update would be performed as if the variable had not been updated yet, using its previous value.

4.1.2. GroundHog

GroundHog[1] is a library developed in Python that uses the library Theano. It provides tools to build RNNs with ease, keeping track of trainable parameters and showcasing how to use its parts.

GroundHog was developed by the Lisa lab in the Université de Montréal by Razvan Pascanu, KyungHyun Cho and Caglar Gulcehre; and is licensed under the 3-clause BSD license.

This project began with the analysis and experimentation with GroundHog in order to understand its structure and information flow. The neural translation system implemented in GroundHog is completely configured through a big dictionary of options that rule the networks' architecture, the input processing, the training details... The use of a

[1] https://github.com/pascanur/GroundHog

huge data container that is being accessed by all parts of the library makes it difficult to understand the dependencies between modules, and consequently the first part of this work included small refactorings that limited the visibility of the state and enhanced in general the structure of the library.

Moreover, since most of the functionalities provided by the library were tightly coupled in scripts, a separated code repository was created in order to fulfill the role of some of those features and allow for nimble experimentation without having to deal with binary state files. Before this change, a GroundHog NMT system would consist of a file containing the weight values of the networks and a state file, representing a dictionary of the model structure and training details; both stored as binary files. With this latter repository of code, the state would be stripped of its role in the training and a command-line interface would assume the responsibility of dealing with the choice of training algorithm, its hyperparameters, input processing and input and output files, among others.

The creation of this tool was essential to the project, because it allowed for the automatization of batches of experiments and their logging.

Secondly, once the algorithms detailed in Chapter 3 had been researched, they were implemented in the GroundHog repository, for coherence with the other learning algorithms already provided. An implementation detail is provided below:

SGD

An implementation of SGD was already available in GroundHog main repository, and very few modifications were necessary:

- The original implementation introduced **noise** in both the input data and the weights of the networks, and it had been programmed with mini-batch learning in mind. This had the effect of quickly distorting the model even when a very small learning rate was in use, because the noise was not regulated by the learning rate, and in online learning the noise injections in the weights were several hundreds of times more frequent than in the mini-batch tasks that the model was tested on.

  The needed modification was to include a flag in the training process which the algorithm implementation would read to know whether to inject noise in the model and data. This flag was disabled for all the experiments in this work, so noise was effectively absent during the retraining.

- Some of the code in the implementation was redundant and was thus trimmed down to its essence in order to avoid bugs and bad performance.

AdaGrad

Given how close the update rule of the approximation of AdaGrad is to the update rule of plain SGD, the implementation of AdaGrad was heavily based on the provided SGD implementation. A small modification was needed at first to store the accumulated squared gradients of the weights, for each weight, which in Equation (3.2) is represented by G_s.

Theano update rules are applied in parallel, so it is not possible to update a value and readily use that value for a different value update. Thus, our coded version of the update rule is equivalent to Equation (3.2), but in order to improve the performance by using only one Theano function instead of two, the following transformation is applied:

$$v_s = v_{s-1} - \eta(G_{s-1} + (\nabla_v \ell)^2)^{-1/2} \nabla_v \ell \tag{4.1}$$

This way, all the weights and G_s for all s are updated in parallel with their correct values.

Passive-Aggressive

The Passive-Aggressive approximation algorithm is different to the previous two and the other algorithms available in GroundHog in that it is iterative. For each sample, it was expected to be executed repeatedly until some degree of convergence was achieved. This difference called for evaluation of the loss and the norm of the updates in each training step and branching on the result.

Another difference present in this algorithm is that it explicitly required the hypotheses emitted by the model to update the weights. This difference warranted a change in the interface of the training algorithms to include the last generated hypothesis in each training step.

Moreover, as can be observed in Algorithm 3.1, we need to store a copy of the weights at the beginning of each iteration ($\boldsymbol{\theta}_s$) to use them in the internal iterative process of the algorithm. To that end, a new set of matrices with the same shape as those of the weights of the networks were created and their value is assigned at the beginning of each iteration.

Input:
Training data: $(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)$
Parameters of the model: $\boldsymbol{\theta}$
Aggressiveness: C (only for PA-I and PA-II)
Output:
Retrained parameters of the model $\boldsymbol{\theta}$
Set of hypotheses generated by the model during the process: $(\mathbf{h}_1, ..., \mathbf{h}_N)$
for each $(\mathbf{x}_s, \mathbf{y}_s)$ in the training data **do**
  Obtain hypothesis from $\mathbf{x}_s$: $\mathbf{h}_s$
  $\boldsymbol{\theta}_s = \boldsymbol{\theta}$
  $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}$
  repeat
    $\boldsymbol{\theta} = \boldsymbol{\theta}_{new}$

$$\boldsymbol{\theta}_{new} = \begin{cases} \boldsymbol{\theta}_s + \frac{\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)||^2} & \text{for PA} \\ \boldsymbol{\theta}_s - \min(C, -\frac{\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)||^2})\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s) & \text{for PA-I} \\ \boldsymbol{\theta}_s + \frac{\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)}{||\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta},\mathbf{x}_s,\mathbf{y}_s,\mathbf{h}_s)||^2 - \frac{1}{2C}} & \text{for PA-II} \end{cases}$$

  until $\boldsymbol{\theta} == \boldsymbol{\theta}_{new}$
end for

Algorithm 4.1: Passive-Aggressive approximation execution flow.

4.2 Experimentation framework

A model already trained with GroundHog was used for each of the following four translation tasks: Xerox English to Spanish, Xerox Spanish to English, EU English to Spanish and EU Spanish to English. Those models (one for each task) underwent the following process of online training: for each sample (x, y) in the development set of the task, the model was fed the source sentence x, a hypothesis h was obtained from it, the hypothesis was stored and the model was retrained with the pair (x, y). The performance of the

Corpus	Task	Set size		Preprocessing	V_s	V_t
		Development	Test			
Xerox	En-Es	1012	1125	Lowercase	11425	14480
	Es-En	1012	1125	Lowercase	14480	11425
EU	En-Es	400	800	None	30001	30001
	Es-En	400	800	None	30001	30001

Table 4.1: Corpora characteristics. Set size for development and test is given in number of sentence pairs.

model was obtained by comparing the list of hypothesis h and the list of target sentences y in the development set.

The result of this comparison is the BiLingual Evaluation Understudy (BLEU) score[21]. BLEU is based on the *n-gram precision* and aims to measure the correspondence between machine-generated translations and human-generated ones. [21] shows that this measure is highly correlated with human evaluation, and it is relatively inexpensive to obtain. It is widely used in the MT academia [27, 7, 20, 1, 13].

Although BLEU is defined for a candidate sentence with one or more reference sentences, in this work only pairs of sentences are used, therefore each candidate sentence is only assigned a single reference sentence.

In order to compute the BLEU score, first we compute its modified n-gram precision. We will only consider n-grams up to n = 4. To that end, we count all the n-grams of a given order in the candidate sentence. We will call that number $\text{Count}(\text{n-gram})$, for each n-gram. For each of those n-grams, we count how many times they appear also in the reference sentence, and we call the minimum of those two numbers $\text{Count}_{clip}(\text{n-gram})$. Finally, we compute $\frac{\sum_{\text{n-gram}} \text{Count}_{clip}(\text{n-gram})}{\sum_{\text{n-gram}} \text{Count}(\text{n-gram})}$ to obtain the BLEU score of the pair.

When applied to a set of sentences, the formula used is the following[21]:

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{\text{n-gram} \in C} \text{Count}_{clip}(\text{n-gram})}{\sum_{C \in \{Candidates\}} \sum_{\text{n-gram} \in C} \text{Count}(\text{n-gram})} \tag{4.2}$$

where p_n is the modified n-gram precision of degree n.

We compute the brevity penalty BP as

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{4.3}$$

where r is the reference corpus length and c is the length of the candidate translations.

Finally, we compute the BLEU score as follows:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{4.4}$$

where w_n are weights used in the weighted geometric average of the modified unigram precisions. In our case, $N = 4$ and $w_n = 0.25$ for $1 \leq n \leq 4$.

Two bilingual corpora were used for two tasks and one language-pair. The tasks were *Xerox* (Xerox printer manuals) and *EU* (Bulletin of the European Union), and the chosen language pair was English-Spanish.

Corpus	Task	Initial BLEU	
		Development set	Test set
Xerox	English-Spanish	66.0	55.2
	Spanish-English	65.3	46.1
EU	English-Spanish	37.2	36.8
	Spanish-English	35.1	35.7

Table 4.2: Initial BLEU obtained in each set of each task.

For each task, all source sentences were processed to obtain a list of hypotheses, and the BLEU obtained from comparing this list with the target sentences list. The result is used to gauge the improvement or worsening of the performance of the models, and is included in the tables of the following section.

Additionally, since our models have a computational cost proportional to the size of vocabulary used, the vocabulary used for translation tasks is reduced to a reasonable size. Some of the models have different vocabulary size and this affects both the number of features and their density this data have been provided in Table 4.1 as well. For example, in the case of the EU models, we are using only the 30000 most frequent words of the corpus. Words that are not found in the vocabulary that appear in the source sentence are all grouped into a special token or symbol, usually called UNK. The model handles UNK as any other word.

The time it took for each of them to finish the training has also been recorded (CPU time) for the sake of comparison, in the case of experiments that involve the *test set*.

4.3 Hyperparameter search

As we have seen in Chapter 3, both studied algorithms have a single hyperparameter: the learning rate. In order to make a useful comparison between the algorithms, we are going to try to find the learning rate that works best with each model and task by performing several experiments with the *development set* of each corpus. Once we have found the optimal hyperparameter for each case, we will evaluate the performance of both algorithms with their best hyperparameters for each task, using this time its *test set*.

Additionally, we consider also the possibility of performing several updates after each sample, since this has shown to improve the results in preliminary experiments and is possible within the post-edition context of this work. Therefore, there are two hyperparameters we can tune for our experiments. In the following tables there is the exploration that has been carried out to find the best combination of those for each translation task. In bold are represented the hyperparameters that were chosen for experimentation with the *test set*, and their results in the *development set*.

4.3.1. Xerox task

The *Xerox* corpus[23] consists of Xerox printer manuals in three different language pairs: Spanish-English, French-English and German-English. In this work only the first one is used, in both translation directions (Spanish to English and English to Spanish). The corpus was tokenized and transformed into lower case, as can be seen in Table 4.1. This last transformation is expected to reduce the complexity of the task and allow the translator to achieve a better performance. Hence the BLEU obtained before retraining was as high as 66.

Iterations	Learning rate				
	0.05	0.1	0.2	0.4	0.8
1	66.2	66.0	65.9	64.2	61.0
3	66.0	66.2	-	-	-
5	**66.4**	66.2	65.3	60.9	-
10	66.5	66.4	66.1	60.8	-
20	66.6	66.8	-	-	-

Table 4.3: Hyperparameter search for the Xerox task, from English to Spanish, with SGD algorithm.

Iterations	Learning rate				
	0.05	0.1	**0.2**	0.4	0.8
1	69.9	69.9	69.8	68.7	63.7
3	70.1	69.8	**70.6**	68.4	61.3
5	70.5	70.0	69.5	69.3	60.0

Table 4.4: Hyperparameter search for the Xerox task, from Spanish to English, with SGD algorithm.

SGD

Table 4.3 shows general but small improvement using a learning rate lower than 0.2. It is possible that slightly better results could be achieved with even lower learning rates, but those experiments fell outside the scope of this work. The set of hyper parameters $(0.05, 5)$ was selected as a trade-off between performance and quality, since the best result (66.8) was achieved performing 20 iterations per sample, setup which would be liable of slowing down too much translation software.

Table 4.4 nonetheless showed much better results than Table 4.3, arguably because of differences in the models (different number of hidden nodes in their networks, different training time...), or maybe because this task was easier than the former (is translating from English to Spanish harder than from Spanish to English?). The best result in this batch of experiments achieved an improvement of 5.3 points in the BLEU score.

AdaGrad

According to Table 4.5, the model in the English-Spanish task improved the most by performing 20 training iterations per sample with a learning rate of 0.0001, and just as well by performing 3 iterations per sample with a learning rate of 0.0005. Since speed

Iterations	Learning rate			
	5e-5	1e-4	**5e-4**	1e-3
1	67.5	67.7	69.0	67.7
3	67.7	68.2	**69.4**	68.0
5	67.8	68.8	69.3	68.3
10	68.3	69.1	68.6	67.6
20	68.9	69.4	68.9	66.6

Table 4.5: Hyperparameter search for the Xerox task, from English to Spanish, with AdaGrad algorithm.

Iterations	Learning rate				
	5e-5	1e-4	**5e-4**	1e-3	5e-3
1	69.9	69.8	71.0	70.8	59.8
3	69.8	70.2	71.1	-	-
5	69.8	70.1	**71.7**	70.5	57.6

Table 4.6: Hyperparameter search for the Xerox task, from Spanish to English, with AdaGrad algorithm.

Iterations	Learning rate				
	0.05	0.1	**0.2**	0.4	0.8
1	35.5	35.3	35.4	34.8	33.3
3	35.3	35.5	**36.1**	35.6	35.4
5	35.4	35.6	35.8	36.1	-

Table 4.7: Hyperparameter search for the EU task, from English to Spanish, with SGD algorithm.

is important in the context of this work, the latter has been chosen as the optimal set of hyperparameters for AdaGrad in this task.

It is worth noting that none of the results of this table result in a decline in the performance. Moreover, the average improvement was of 2.3 percentage points, which shows that AdaGrad achieved a much better overall improvement in this set than SGD. Nevertheless, this task is being used for hyperparameter search, so conclusions must not be drawn from this comparison, given that we purposely select the best result of each of them.

In Table 4.6 we see improvements of the same order of Table 4.4, which leads to think that this task is quite adept at being retrained.

4.3.2. EU task

The *EU* corpus[16] was obtained from the Bulletin of the European Union, which is publicly available in all the official languages of the European Union. As in the Xerox tasks, only Spanish to English and English to Spanish were used. The corpus was tokenized as well, but not transformed to lower case as we did in Section 4.3.1.

SGD

Table 4.7 shows a case not encountered so far: none of the results show an improvement in the quality of translations. There are several factors that can contribute to this phenomenon. First of all, this set is much smaller than that of the previously shown tasks: 400 sentence pairs versus 1012 in Xerox. Since the BLEU score is a measure that attempts to match human judgement when averaged over a corpus[21], its value when the set is small can be expected to be less reliable than the case where it is applied to a big corpus.

Furthermore, the domain of the EU corpus is in all likelihood more complex than the domain of Xerox. Xerox corpus is full of short sentences, with very repetitive words, like printer features and options. Numbers in Xerox are most of the time model identifiers, which do not change. EU bulletin is very diverse, has long sentences, a high quantity of numbers of records, dates, percentages, file sizes, and so on. Also, the initial BLEU in both could not be any more different, and the vocabulary size in both languages in Xerox is less than half than in the EU models.

Iterations	Learning rate				
	0.05	0.1	0.2	**0.4**	0.8
1	35.1	35.1	35.1	35.3	34.4
3	35.0	34.8	35.2	**36.0**	35.5
5	35.1	35.7	35.8	35.8	34.8

Table 4.8: Hyperparameter search for the EU task, from Spanish to English, with SGD algorithm.

Iterations	Learning rate			
	5e-5	**1e-4**	5e-4	1e-3
1	35.9	35.4	35.7	35.0
3	36.1	35.7	36.1	35.5
5	35.8	**36.2**	35.7	33.7

Table 4.9: Hyperparameter search for the EU task, from English to Spanish, with AdaGrad algorithm.

We can say without a shade of doubt that the Xerox translation models in this work are more effective than their EU counterparts. It remains a question whether this makes the task of retraining easier or harder, or whether it is inmaterial to it. Further research would be necessary to answer it.

Finally, given the assumptions of correlation between model and algorithm performances in different tasks that were issued in the introduction of this work, we must assume too that the experiment with the *test set* that involves this particular configuration will deteriorate the model performance, but not as much as with the other parameters that were tried.

A small improvement can be observed in some of the entries of Table 4.8. Even those setups that resulted in a decline in quality of translations did so only by a very small amount. This could be attributed to the small size of the development set, in comparison to the other corpus.

AdaGrad

The results in Table 4.9 appear to be very similar to those in Table 4.7: no improvement in any case. This can likely be attributed to the same hypothetical reasons that were given for the results in Table 4.7. The best result of the table was chosen for furhter experimentation.

Table 4.10 shows slightly more promising results than Table 4.8, even though slightly fewer cases were attempted.

Iterations	Learning rate			
	5e-5	1e-4	**5e-4**	1e-3
1	35.3	35.7	35.8	35.0
3	35.5	36.1	35.4	35.5
5	35.7	35.9	**36.3**	35.8

Table 4.10: Hyperparameter search for the EU task, from Spanish to English, with AdaGrad algorithm.

Task	BLEU		
	Initial	SGD	AdaGrad
Xerox En-Es	55.2	55.9	57.1
Xerox Es-En	46.1	51.7	50.4
EU En-Es	36.8	36.4	36.4
EU Es-En	35.7	34.8	36.0

Table 4.11: BLEU score obtained in the experiments on the *test set* of each task, both without retraining and by retraining with each algorithm using the hyperparameters selected in Section 4.3.

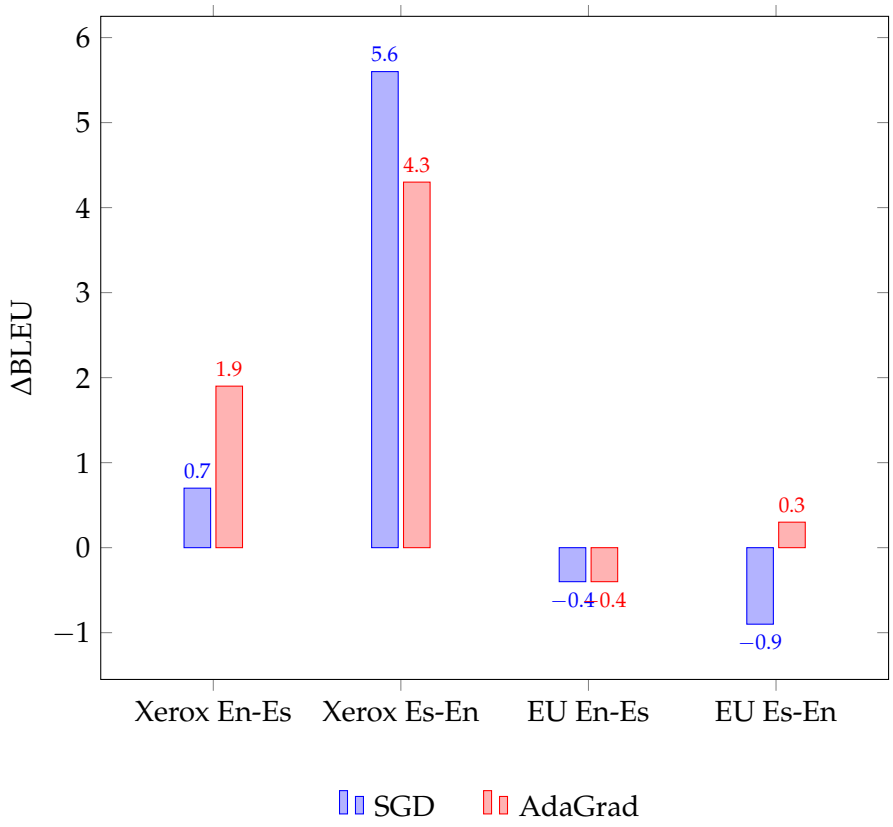

Figure 4.1: Differences in BLEU obtained in the test set of each task by retraining with each algorithm, using the hyperparameters selected in Section 4.3. The values used can be found in Table 4.11.

4.4 Test set experiments

From the data in Table 4.11 we see that AdaGrad outperforms SGD in two of our four cases, and matches it in another one. The data, however, is insufficient to draw conclusions about AdaGrad being in general better for this scenario. SGD outperformed AdaGrad in Xerox Spanish-English by a relatively large margin, in comparison with the other results. The improvement achieved by each algorithm on each model is plotted in Figure 4.1.

Both algorithms showed considerably better results in the Xerox tasks than in the EU ones, which could be attributed to, among other things, the task complexity and/or the already-existing model performance. The models that obtained a high BLEU initially improved their results, they probably took profit of the data and enhanced their parameters as to better fit the translation task. The models that obtained a relatively low BLEU in the initial measurement may have not been able to assimilate the new data and may have

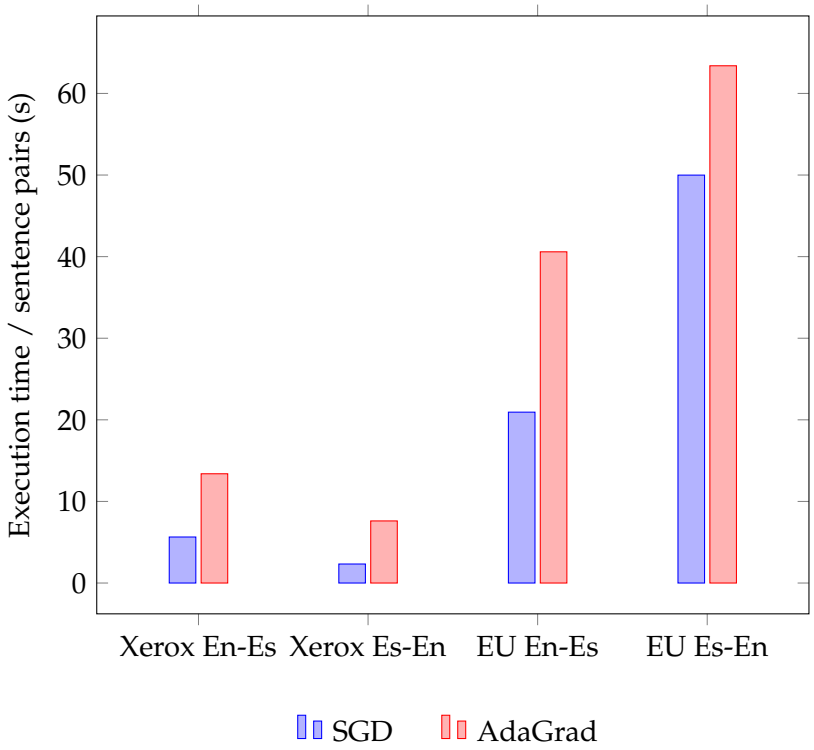

Figure 4.2: Execution time of experimens in the test sets.

been negatively affected by the parameter updates (as they may have in the experiments with the *development set*), but even then, the quality of the hypotheses was not much worse off, and AdaGrad algorithm could achieve an improvement in the EU Es-En task

The decline in performance observed by retraining in both EU tasks in previous experiments is consistent with these results. The Xerox English-Spanish translation task has produced much different results than prior experiments, obtaining much lower improvements, while Xerox Spanish-English has maintained a high degree of improvement with both algorithms.

In Figure 4.2 we can observe the different execution times of each algorithm, averaged over the number of sentence pairs of the relevant set. We can observe that the execution time for EU tasks was much longer than for Xerox tasks, likely due to their longer sentences and higher vocabulary size (which makes their models even larger).

CHAPTER 5
Conclusions and future work

We have developed a full-fledged experimentation environment for online adaptation of neural translators, able to perform automated batches of experiments with proper handling and storing of the results. We have selected and processed data for experiments, and performed several of them with different sets of hyperparameters, and have found the ones among them that work best for each combination of task and algorithm. With a different set but for the same task, we have performed experiments with both algorithms configured to their empirically-found best capacities and have obtained a comparison of how the algorithms can improve or worsen the quality of neural translators.

We have obtained an iterative approximation of the Passive-Aggressive family of online learning algorithms, produced an implementation for our experimentation framework and observed the lack of promising results. Due to this, no further experimentation was carried out with those algorithms.

In three out of four translation tasks, we observed a slight correlation between the results in the *development set* and the results in the *test set*. AdaGrad algorithm resulted more promising than SGD, but the latter outperformed the former in one task. All things considered, we cannot state that AdaGrad would be the best choice for every task, but *a priori* it is a better candidate.

Several questions remain unanswered, such as to why is there such disparity in the results between *development set* and *test set* in the Xerox English-Spanish task, or why the EU tasks put up such a challenge against both algorithms. Further research is required to answer those questions.

More experiments, involving more datasets and more language pairs are required, since neural translators are usually trained and used for single language pairs, and the results observed using one of them may not correspond at all with results obtained from a different one. In this work we have used two corpora and one language-pair (in both directions), and even then we have found significant differences between tasks (especially between both directions of the Xerox English to/from Spanish tasks).

The size of the EU *development set*, especially when compared to that of the Xerox corpus, raises the question of whether the hyperparameter search for the EU tasks may have been compromised, or at the least whether the size of this set has handicapped the hyperparameter search for those models. More similar pairs of sets could be used to help dispel this doubt.

Finally, the approximation of Passive-Aggressive algorithms must be revised in search of alternative methods to fixed point iterators, to eventually test them against the challenges presented in this work, and compare them to the other two algorithms we have used.

Acronyms

A. Acronyms

BLEU . . . . . BiLingual Evaluation Understudy

BRNN . . . . . Bidirectional Recurrent Neural Network

CBS . . . . . Corpus-Based System

GRU . . . . . Gated Recurrent Unit

LSTM . . . . . Long Short Term Memory

MLP . . . . . Multilayer Perceptron

MT . . . . . Machine Translation

NLP . . . . . Natural Language Processing

NMT . . . . . Neural Machine Translation

RBS . . . . . Rule-Based System

RNN . . . . . Recurrent Neural Network

SGD . . . . . Stochastic Gradient Descent

SMT . . . . . Statistical Machine Translation

Bibliography

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

[2] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[3] Jerome R Bellegarda. A latent semantic analysis framework for large-span language modeling. In *EUROSPEECH*, 1997.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

[6] M Asunción Castano, Francisco Casacuberta, and Enrique Vidal. Machine translation using neural networks and finite-state models. *Theoretical and Methodological Issues in Machine Translation (TMI)*, pages 160–167, 1997.

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

[8] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, December 2006.

[9] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *ACL (1)*, pages 1370–1380. Citeseer, 2014.

[10] Bonnie J Dorr, Pamela W Jordan, and John W Benoit. A survey of current paradigms in machine translation. *Advances in computers*, 49:1–68, 1999.

[11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[12] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[13] Spence Green, Sida I Wang, Daniel M Cer, and Christopher D Manning. Fast and adaptive online training of feature-rich translation models. In *ACL (1)*, pages 311–321, 2013.

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

[15] John Hutchins. Machine translation: A concise history. *Computer aided translation: Theory and practice*, 2007.

[16] S Khadivi and C Goutte. Tools for corpus alignment and evaluation of the alignments (deliverable d4. 9). Technical report, Technical report, TransType2 (IST-2001-32091), 2003.

[17] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

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

[19] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[20] Franz Josef Och and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 295–302, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[21] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[22] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

[23] SA SchlumbergerSema. Intituto tecnológico de informática, rheinisch westfälische technische hochschule aachen lehrstul für informatik vi, recherche appliquée en linguistique informatique laboratory university of montreal, celer soluciones, société gamma, and xerox research centre europe. 2001. *TT2. TransType2-computer assisted translation. Project technical annex*.

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

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[26] The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[27] Álvaro Peris Abril. Continuous spaces in statistical machine translation. Master's thesis, Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital, 2015.