Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Translation Rescoring through Recurrent Neural Network Language Models

Proyecto Final de Carrera

Ingeniería Informática

Autor: Álvaro Peris Abril

Director: Francisco Casacuberta Nolla

Codirector: Daniel Ortíz Martinez

September 3, 2014

Abstract

This work is framed into the Statistical Machine Translation field, more specifically into the language modeling challenge. In this area, have classically predominated the n-gram approach, but, in the latest years, different approaches have arisen in order to tackle this problem. One of this approaches is the use of artificial recurrent neural networks, which are supposed to outperform the n-gram language models.

The aim of this work is to test empirically these new language models. For doing that, the translation rescoring of three tasks of different complexity has been performed: in first place, the translation problem has been solved by means of the classic n-gram language models. Next, the different translation hypotheses have been rescored through the language models based on neural networks and the results have been compared.

This comparison shows that the translations produced by the neural network language models have a better quality in all the experiments: the perplexity of the language models has been lowered and the BLEU score of the translations outputted by the system has yielded higher values with the neural network language model than with the classical n-gram language model.

Keywords: Recurrent Neural Networks, Statistical Machine Translation, n-gram Language Model, BLEU.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Natural Language Processing

Natural Language Processing (NLP) is a knowledge field of Artificial Intelligence and Computational Linguistics, concerned with the formulation and investigation of efficient methods for generating and understanding interactions between computers and human natural languages, both oral or written. A natural language is a language written or speeched by humans for general communication purposes with each other.

NLP is a very wide knowledge field, where there are two main focuses, namely language processing and language generation. The first focus involves language analysis with the goal of generating a meaningful representation for computers. The latter, refers to the computerized language production from a representation. There is a wide range of applications referred to Natural Language Processing, such as automatic summarization, discourse analysis, machine translation, morphological segmentation, natural language generation and understanding, speech recognition, topic segmentation and recognition, information retrieval, dialog systems, question answering, etc.

1.2. Machine Translation

The field of Machine Translation (MT) researches the use of software with the aim of translating text or speech from a source natural language to another target natural language. This research area arose in 1949, from Warren Weaver's idea of apply some of the Information Theory ideas from Claude Shannon. During the 1950s and 1960s, there was a general optimism regarding the machine translation [Hut95]. Main researches were polarized between the empirical (which use statistical methods) and the theoretical approaches. There was spent much effort to improve the hardware of that time and to achieve effective programming methods, able to deal with the problem. However, the results were not satisfactory. The ALPAC (Automatic Language Processing Advisory Committee) report, released in 1966, concluded that the machine translation was not worth and there was no sign of a feasible future. From here, the research decreased, though some approaches for tackling the MT problem were developed. But 10 years later than the ALPAC report, the Machine Translations prospects improved, and the research and translation systems acquired importance once again. Until the late 1980s, the main approach was the Rule-Based approach (see next section). Nevertheless, since 1989, the hegemony of theses systems has been broken by the irruption of the so-called corpus-based systems.

During these more than 60 years of research, different main strategies have been applied to tackle this problem. These strategies can be categorized under different criteria [OM11]:

- According to the input type: speech or text.
- According the type of the application which uses the translations: applications that translate the input into a database query; applications that generate an approximated translation for its later correction by the user, in a post-edition translation stage; applications that perform an user-interactive translation; and fully automated translation systems.
- According to the translation technology. There are two main strategies: Rule-Based Systems and Corpus-Based Systems. Furthermore, there are approaches that combine both strategies and compose the so-called Hybrid Machine Translation approach.

1.2.1. Rule-Based Systems

Rule-Based Machine Translation (RBMT) was one of the firsts approaches used in machine translation (the first RBMT system was developed in the early 1970s), therefore, it is a relatively mature area. This paradigm is based on the extraction of linguistic information from dictionaries and grammars, and on the creation of translation rules by human translators, in order to generate the translations.

These systems are usually split in three stages: analysis step, transference step and generation step. First, the analysis step extracts information from the source text. The transference step is an optional stage, where the result of the analysis is transformed into an abstract representation. Finally, the generation step produces the target text.

RBMT systems have the following advantages:

- No bilingual texts are required: it is possible to develop translation systems for languages with no texts in common or with no digitalized data.

- Rules are domain independent: most of the rules are written in a domain independent way, thus they work in every domain.

- There is no quality top: each error, even it is very unusual, can be treated and corrected with an specific rule. In Statistical Machine Translation, this is not possible.

- Total control: rules are handwritten, thereby the system is easily debuggable, in order to watch exactly systems errors.

The main disadvantage of these systems is their cost: the development of the rules is expensive and it needs expert knowledge from human linguistics, both in source language and in target language. Moreover, some linguistic information must be manually set and it is hard to deal with rule interaction in big systems, with ambiguity and idiomatic expressions.

1.2.2. Corpus-Based Systems

Corpus-Based systems use sets of translation examples (also known as corpora or parallel texts) from one language to another. These translation examples are used to deduce the translation of the source text. We can establish the following classification of the corpus-based systems:

- **Example-Based Machine Translation (EBMT)**: this approach uses a set of translation examples as its knowledge base and it works by means of analogies. The translation process is divided into two steps, namely, comparison and recombination steps. First, the comparison step is carried out. Here, a set of hypotheses that are similar to the source text are extracted from the whole corpus. Second, at the recombination step, the hypotheses are combined in order to generate the final translation of the source text.

- **Statistical Machine Translation (SMT)**: the translations are based on statistical models and other models from Information Theory. This approach demands the availability of a large amount of parallel text with relevant information for the translation process. This parallel text is used to estimate the parameters of the referred models. Once this parameters have been estimated, they are used to infer the translation of new source text.

- **Other Corpus-Based Systems**: there are alternative approaches for implementing corpus-based system, such as the finite state approach, which applies tools from automata theory; or the context-free grammar approach, which tackles the Machine Translation problem by means of context-free grammars.

The main advantages of these systems regarding to Rule-Based Systems fall on a better resource usage, since there is a big amount of natural language in a machine-readable format and it is not necessary to manually develop rules for generate the translations. Additionally, these systems can be used for every pair of languages, provided that there exist parallel texts between those languages; human linguists in the languages involved in translations, who are costly, are not needed. Finally, Corpus-Based Systems produce more natural translations than Rule-Based System, whose translations are often literal or have nonsensical and obvious errors.

Nevertheless, this translations paradigm has shortcomings too:

- Corpus generation can be expensive.

- Results are not predictable: the SMT systems do not deal correctly with the fluency of the produced translations.

- In languages with substantially different word orders, such as English and Japanese, SMT systems do not work properly.

1.3. Statistical Machine Translation

Statistical Machine Translation is nowadays the most studied machine translation field. It tackles the translation problem using a statistic formalization thereof. For that purpose, it uses statistical models which parameters are estimated from parallel texts.

More formally, the goal of SMT is, given a source sentence $f_1^J \equiv f_1 ... f_j ... f_J$ in the source language $\mathcal{F}$, to find its equivalent target sentence $e_1^I \equiv e_1 ... e_i ... e_I$ in the target language $\mathcal{E}$. From all the possible sentences of $\mathcal{E}$, we want to choose that with the highest probability, according to the equation:

$$\hat{e}_1^{\hat{I}} = \arg\max_{I, e_1^I} \{Pr(e_1^I | f_1^J)\} \tag{1.1}$$

The problem of modeling this probability distribution has been approximated in many ways. First works were based on the so-called *generative models*. A generative model is a full probability model of all statistical variables involved in random generation of observable data. These models apply the Bayes decision rule in order to decompose $Pr(e_1^I | f_1^J)$. Considering that $Pr(f_1^J)$ is independent on e_1^I, we arrive to the following equation:

$$\hat{e}_1^{\hat{I}} = \arg\max_{I, e_1^I} \{Pr(e_1^I) \cdot Pr(f_1^J | e_1^I)\} \tag{1.2}$$

This expression is known as *fundamental equation of machine translation* [BPPM93], and states that, in order to obtain the equivalent sentence in target language from a source sentence, we must carry out an exploration of all sentences of the target sentence, computing, for each sentence e_1^I, first, the probability $Pr(e_1^I)$; and second, the conditional probability $Pr(f_1^J | e_1^I)$. Finally, we will choose the sentence of highest probability, $\hat{e}_1^{\hat{I}}$, which is the most likely translation. Hence, this most likely translation must maximize the product of these two factors: 1. the chance that someone would say e_1^I in first place and 2. once e_1^I has been said, the chance that someone would translate it into f_1^J. This way of proceeding fits into the *noisy channel model.*

The noisy channel model is a framework which aims to find a target word given a word which letters have been mixed somehow. Formally, the noisy channel model define the spelling problem as follows: given an alphabet $\mathcal{F}$, a dictionary $\mathcal{E}^*$ consisting of strings in $\mathcal{F}^*$, and a string $f_1^J \notin \mathcal{E}^* \wedge f_1^J \in \mathcal{F}^*$, we want to find the string $e_1^I \in \mathcal{E}^*$ that it is the most likely to have been generated f_1^J [BM00]. Writing this to probabilities, the goal is to compute $\arg\max_{e_1^I} Pr(e_1^I | f_1^J)$. Applying the Bayes' rule, this expression can be rewritten as: $\arg\max_{e_1^I} Pr(f_1^J | e_1^I) \cdot Pr(e_1^I)$. If we consider a translation as a mixture of letters of a source sentence, it is obvious that the translation problem can be tackled with the noisy channel means.

Thus, $Pr(e_1^I)$ represents the probability of generate the target sentence, and $Pr(f_1^J | e_1^I)$ represents the probability of generate the target sentence e_1^I

given the source sentence f_1^J. Since the real probability distributions $Pr(e_1^I)$ and $Pr(f_1^J|e_1^I)$ are unknown, they must be estimated through parametric statistical models. These models have a set of parameters Θ, linked to either a known probability density function or a probability mass function, expressed by $Pr(\cdot|\Theta)$. Given a set of training samples, $\mathcal{X} = \{e_1, e_2, ..., e_N\}$, we define the *log-likelihood function* as:

$$\mathcal{L}(\Theta, e) = \log p(e_1, e_2, ..., e_N|\Theta) = \sum_{i=1}^{N} \log p(e_i|\Theta) \tag{1.3}$$

Typically, the estimation of those parameters is carried out by means of the *maximum-likelihood* estimation method, which estimates the set of parameters Θ finding the value of Θ which maximizes $\mathcal{L}(\Theta, \mathcal{X})$. This value is called *maximum-likelihood* (**ML**) *estimator* of Θ:

$$\hat{\Theta} = \arg\max_{\Theta} \mathcal{L}(\Theta, \mathcal{X}) \tag{1.4}$$

Generally, both probability distributions which are involved in equation 1.2, are modeled separately. Thereby, $Pr(e_1^I)$ is modeled by the so-called *language model* and $Pr(f_1^J|e_1^I)$ by the *translation model*, both with their corresponding set of parameters, Θ_{LM} and Θ_{TM}. Following the previous reasoning and applying the maximum-likelihood estimation method, we define the training set as sentence pairs $\mathcal{X} = \{(f_1|e_1), (f_2|e_2), ..., (f_N|e_N)\}$, and we get to the following expressions for the different ML estimators:

$$\hat{\Theta}_{LM} = \arg\max_{\Theta_{LM}} \left\{ \sum_{n=1}^{N} \log p(e_n|\Theta_{LM}) \right\} \tag{1.5}$$

$$\hat{\Theta}_{TM} = \arg\max_{\Theta_{TM}} \left\{ \sum_{n=1}^{N} \log p(f_n|e_n; \Theta_{TM}) \right\} \tag{1.6}$$

Thus, there are three main computational challenges of SMT [BPPM93]:

1. Estimating the *Language Model probability*: $Pr(e_1^I)$.
2. Estimating the *Translation Model probability*: $Pr(f_1^J|e_1^I)$.
3. Finding and effective and efficient search search method for the string which maximizes the product, i.e, computing efficiently $\arg\max_{I,e_1^I}$

To build a translation system, based on the Bayes rule, it is necessary to assemble these models, adding a pre/postprocess stages for the sentences, which will increase the performance of the system. Figure 1.1 shows this scheme ([OM11]).

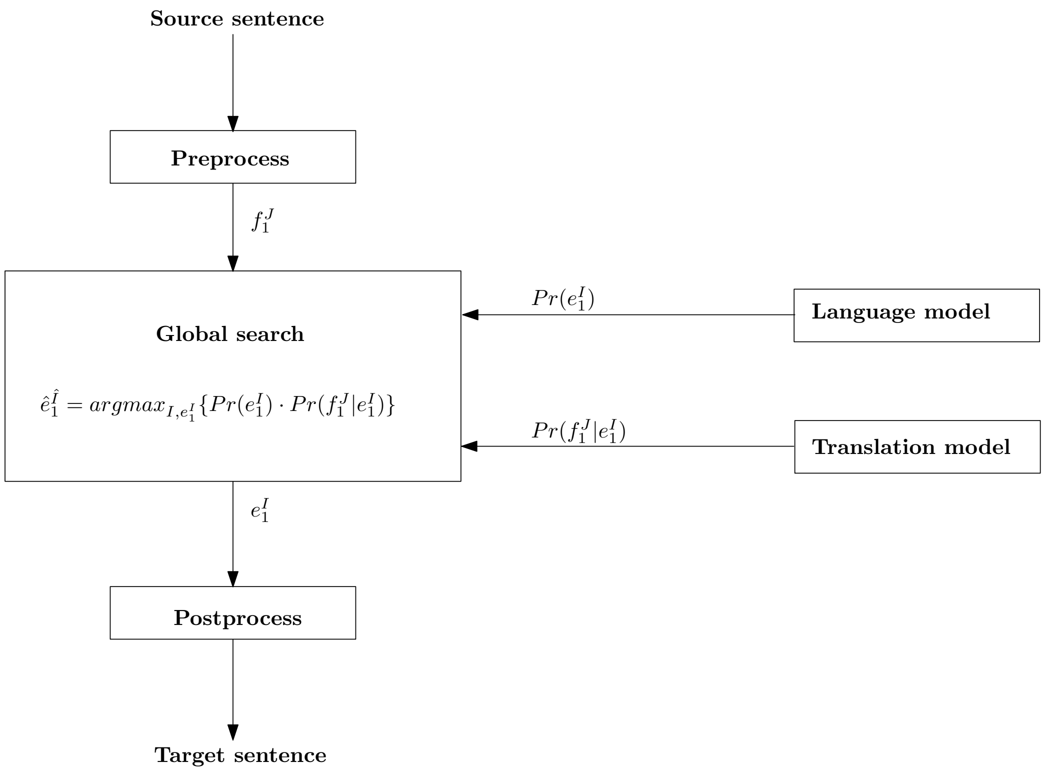


Figure 1.1: Architecture of the translation process based on the Bayes rule.

Up to here, we have introduced the Machine Translation knowledge field. We have reviewed the main paradigms historically developed to solve the problem presented by MT, classifying them according the technology that use. We have deepened into the Statistical Machine Translation approach: its goals and main reasoning ideas of SMT have been stated, and three main challenges have emerged, namely, the language modeling, the translation modeling and the search problem.

In the next chapter we will present statistical models and algorithms which will try to tackle these challenges. In addition, we will define some evaluation techniques for the machine translation output.

Chapter 2

Statistical Models for Machine Translation

As seen above, two of the challenges presented by equation 1.2 refer to build good statistical approaches, which should be able to model properly both probability distributions of the equation, language model and translation model. In the following sections, the models used in literature to deal with this problems will be defined and explained. In addition, some metrics to measure the goodness of the statistical models will be referred.

2.1. Language Models

First, we will focus on the language model problem: we want to estimate the probability distribution $Pr(e_1^I)$. Classically, in SMT have predominated the so-called *n-gram language models*, but in recent times, other language models have appeared, such as neural-network-based language models, which are supposed to outperform the classic n-gram models [Mik12]. Statistical language models are formulated as a probability distribution $p(e_1^I)$ for strings e_1^I. This probability distribution measures the well-formedness of a string e_1^I.

2.1.1. n-gram Models

Basically, an *n-gram* consists in an n-word substring. If we consider $n = 2$, the model is called *bigram*. If we consider $n = 3$, *trigram*. First, we will introduce the concept of n-grams with bigrams, and we will extend it later to higher order n-grams.

Let the sentence e_1^I be composed of the words $e_1 e_2 ... e_I$. We can denote, without loss of generality, $Pr(e_1^I)$ as:

$$Pr(e_1^I) = p(e_1) \cdot p(e_2|e_1) \cdot p(e_3|e_1 e_2) \cdot ... \cdot p(e_I|e_1 ... e_{I-1}) = \prod_{i=1}^{I} p(e_i|e_1 ... e_{i-1}) \tag{2.1}$$

In bigram models, since $n = 2$, the probability of a word depends solely on the preceding word:

$$Pr(e) = \prod_{i=1}^{I} p(e_i|e_1 ... e_{i-1}) \approx \prod_{i=1}^{I} p(e_i|e_{i-1}) \tag{2.2}$$

To estimate the probability $p(e_i|e_{i-1})$, that is, the probability of e_i given the previous word e_{i-1} we count the number of occurrences of the bigram $e_{i-1}e_i$ in the training text and normalize. This matches with a ML estimator:

$$p(e_i|e_{i-1}) = \frac{c(e_{i-1}e_i)}{\sum_{e_i} c(e_{i-1}e_i)} \tag{2.3}$$

Where $c(e_i)$ corresponds to the number of occurrences of the word e_i in the source text. In other words, we just divide the number of times we see the string $e_i e_{i-1}$ by the number of times we see the word e_{i-1}.

The extension to n-gram is natural: instead of computing the probability of the word based only on the preceding word, that probability is computed according the last $n-1$ words:

$$p(e_1^I) = \prod_{i=1}^{I-1} p(e_i|e_{i-n+1} ... e_{i-2} e_{i-1}) \tag{2.4}$$

The n-gram probability estimation is an extension of bigram estimation (equation 2.3). Let us use the following notation: e_{i-n+1}^{i-1} represents the segment of source sentence which starts at the $(i-n+1)$'th word and finishes at the $(i-1)$'th word. Thus, n-gram probabilities estimation is given by the expression:

$$p(e_i|e_{i-n+1}^{i-1}) = \frac{c(e_{i-n+1}^{i-1})}{\sum_{e_i} c(e_{i-n+1}^{i})} \tag{2.5}$$

Words e_{i-n+1}^{i-1} are called *history* of the n-gram and the value of n is called *order* of the n-gram. Usually, the order of n-grams fluctuate between 2 and 5.

2.1.2. Smoothing

As seen above, n-gram language models count occurrences of words, but what happens when, in test phase, they see new n-grams that they have never seen before?

N-gram models by themselves cannot assign non-zero probabilities to sentences that have not been seen during the training stage. Thus, if they find a sentence e_1^I such that $p(e_1^I) = 0$, the whole system would fail. In order to prevent this failure, we use the so-called *smoothing.* This term describes techniques for adjusting the maximum likelihood estimate of probabilities to produce more accurate and softer probabilities. These techniques tend to produce more uniform distribution, increasing low probabilities (zero or almost equal to zero) and decreasing high probabilities (one or almost equal to one). Besides preventing zero probabilities, smoothing methods also improve the performance of the model significantly. In [CG98] is performed a study of the main smoothing techniques developed. In the following sections, we will briefly explain some smoothing techniques related with this work and used to carry out our experiments.

To introduce the smoothing techniques, we will begin with an extremely simple technique: the so-called *additive smoothing*, which pretends that each n-gram happens δ times more than it actually does:

$$p(e_i|e_{i-n+1}^{i-1}) = \frac{\delta + c(e_{i-n+1}^{i-1})}{\sum_{e_i} \delta + c(e_{i-n+1}^{i})} = \frac{\delta + c(e_{i-n+1}^{i-1})}{\delta \cdot |V| + \sum_{e_i} c(e_{i-n+1}^{i})} \tag{2.6}$$

Where V is the vocabulary, the set of all words considered. This is a very simple method, but it has a poor performance in general. From here, many other smoothing techniques have been developed. In the next section, we will take a brief look to these methods.

Good-Turing Estimate

The *Good-Turing Estimate* fixes the basis of many smoothing techniques. It proposes that, for any n-gram that occurs r times, the estimate pretends that it occurs r^* times, where:

$$r^* = (r+1) \cdot \frac{n_{r+1}}{n_r}$$

being n_r the number of n-grams that occur exactly r times in the training data. We must convert this count into a probability, hence we have to normalize: let α be an n-gram that occurs r times, and $N = \sum_{r=0}^{\infty} n_r r^*$, the

Good-Turing estimate states that:

$$p_{GT}(\alpha) = \frac{r^*}{N}$$

In practice, this smoothing method is not used for n-gram smoothing, but, as told above, it is used as central tool in other techniques.

Jelinek-Mercer Smoothing

In general, it is useful to interpolate higher order n-grams with lower order n-grams, because when there is insufficient information for the higher order n-gram, the information provided by the lower order n-gram is mostly helpful. Jelinek-Mercer described a general class of interpolated models and in [BPP+92] is given a recursive way of executing this interpolation:

$$p_{interp}(e_i|e_{i-n+1}^{i-1}) = \lambda_{e_{i-n+1}^{i-1}} \cdot p_{ML}(e_i|e_{i-n+1}^{i-1}) + (1 - \lambda_{e_{i-n+1}^{i-1}}) \cdot p_{interp}(e_i|e_{i-n+2}^{i-1}) \tag{2.7}$$

That means that the nth-order smoothed model ($p_{interp}(e_i|e_{i-n+1}^{i-1})$), is defined recursively as the linear interpolation between the nth-order maximum likelihood order model ($p_{ML}(e_i|e_{i-n+1}^{i-1})$) and the ($n - \mathbf{1}$)th-order smoothed model ($p_{interp}(e_i|e_{i-n+\mathbf{2}}^{i-1})$). As base case of the recursion, it is used the smoothed 1st-order model to be the maximum likelihood distribution or the smoothed 0th-order model to be the uniform distribution $p_{unif}(e_i) = \frac{1}{|V|}$.

To estimate the value of $\lambda_{e_{i-n+1}^{i-1}}$ that maximizes the probability of the data, it is performed an efficient search, based on different information extracted from the data used to train the language model. In our experiments, we used the *Downhill Simplex* method (see Section 2.4.3) in order to lower the perplexity of the language model of the different tasks (see Section 3.1.2).

Witten-Bell Smoothing

Witten-Bell smoothing is an instance of Jelinek-Mercer smoothing. Starting with a definition similar to equation 2.7:

$$p_{WB}(e_i|e_{i-n+1}^{i-1}) = \lambda_{e_{i-n+1}^{i-1}} \cdot p_{ML}(e_i|e_{i-n+1}^{i-1}) + (1 - \lambda_{e_{i-n+1}^{i-1}}) \cdot p_{WB}(e_i|e_{i-n+2}^{i-1}) \tag{2.8}$$

The idea behind Witten-Bell smoothing is to use the higher order model if the corresponding n-gram occurs in the training data. If not, we will back off to the lower order model. For that purpose, we will set the term $(1 - \lambda_{e_{i-n+1}^{i-1}})$

to the probability that a word not observed after the history e_{i-n+1}^{i-1} in the training data occurs after that history. To model that, we must redefine the way these interpolation parameters $\lambda_{e_{i-n+1}^{i-1}}$ are estimated. Here, we define $N_{1+}(e_{i-n+1}^{i-1}\bullet)$ as the number of *unique words* that follow the history e_{i-n+1}^{i-1}. The parameters $\lambda_{e_{i-n+1}^{i-1}}$ are estimated according:

$$1 - \lambda_{e_{i-n+1}^{i-1}} = \frac{N_{1+}(e_{i-n+1}^{i-1}\bullet)}{N_{1+}(e_{i-n+1}^{i-1}\bullet) + \sum_{e_i} c(e_{i-n+1}^{i})} \tag{2.9}$$

Substituting on Equation 2.8 we arrive to the Witten-Bell smoothing equation:

$$p_{WB}(e_i|e_{i-n+1}^{i-1}) = \frac{c(e_{i-n+1}^{i}) + N_{1+}(e_{i-n+1}^{i-1}\bullet) \cdot p_{WB}(e_i|e_{i-n+2}^{i-1})}{\sum_{e_i} c(e_{i-n+1}^{i}) + N_{1+}(e_{i-n+1}^{i-1}\bullet)} \tag{2.10}$$

Proceeding in this way, we will accomplish our goal: we will use the higher order model with probability $\lambda_{e_{i-n+1}^{i-1}}$ and the lower order model with probability $(1 - \lambda_{e_{i-n+1}^{i-1}})$.

Kneser-Ney smoothing

Kneser-Ney Smoothing is an extension of the so-called *Absolute Discounting method*. The Absolute Discounting method is similar to the Jelinek-Mercer smoothing, involves the interpolation of higher-order and lower-order models; but instead of interpolating them with a factor $\lambda_{e_{i-n+1}^{i-1}}$, in Absolute Discounting, the higher-order distribution is created by reserving a probability mass from each nonzero count, that is, we subtract to each count a fixed discount $D < 1$.

The extension introduced in the Kneser-Ney smoothing is the combination of the lower-order model with the higher-order model. So far, lower-order distributions were smoothed versions of the lower-order maximum likelihood distributions. But this is an important factor in the combined model in the case when few or no counts are present in the higher-order distribution. Thus, if there are not many counts in the training data, we are going to waste an important part of the power of the higher order-model, using the lower-order models. The idea of Kneser-Ney smoothing is to optimize the lower-order distributions in order to have a good performance in this situation.

Kneser and Ney proposed the following smoothing equation:

$$p_{KN}(e_i|e_{i-n+1}^{i-1}) = \begin{cases} \frac{max\{c(e_{i-n+1}^{i})-D,0\}}{\sum_{e_i} c(e_{i-n+1}^{i})} & \text{if } c(e_{i-n+1}^{i}) > 0 \\ \\ \gamma(e_{i-n+1}^{i-1}) \cdot p_{KN}(e_i|e_{i-n+2}^{i-1}), & \text{if } c(e_{i-n+1}^{i}) = 0 \end{cases} \tag{2.11}$$

where $\gamma(e_{i-n+1}^{i-1})$ is chosen to make the sum of the probability for all possible event equal to 1. Therefore, they interpolate the lower-order distribution with all words, not only with the words with zero counts in the higher order distribution.

2.1.3. Neural Network Language Models

So far, we have studied the n-gram language model which computes conditional probabilities for the next word based on a big number of *contexts*, i.e. combinations of the last $n-1$ words. Those models were able to tackle the non-seen sample problem using an smaller context size and generalizing the acquaintance obtained from the samples seen in the training phase to new test samples. However, there are two main issues that decrease the performance of n-gram models [BDVC03]:

1. **Restricting context to few words**: typically, the order of an n-gram model is set to 3. Sometimes it can be increased until 5. That means it usually takes into account 2 words, 4 at most. That is a small number if we deal with complex sentences from large corpora, where we have more information avaliable which is not exploited by the n-grams.

2. **Not considering *similarity* between words**: since n-gram models are merely based on word counts, they cannot extract semantic and grammatical information from vocabulary. For example, we have seen on the training phase the sentence `"A car is crossing the road"` and then in the test phase we see the sentence `"The truck was driving on the highway"`. Then it is clear that we should be able to tag the test sentence as likely, because the words involved in both sentences have similar grammatical and semantic roles:
`"A"`$\leftrightarrow$`"The"`, `"car"`$\leftrightarrow$`"truck"`, `"road"`$\leftrightarrow$`"highway"`, etc. But if we use n-gram models, we are unable to extract that information.

For the purpose of tackling these problems, many approaches have been developed. A family of them belongs to the so-called *connectionist approach* to the MT and is based on the use of Artificial Neural Networks in order to generate a language model [BDVC03].

As seen above, the goal of a language model is to learn the joint probability of a sequence of words in a language. That implies to model the joint distribution of many discrete random variables and generalize this model. The use of continuous variables (like multi-layer neural networks) helps to perform an easier generalization, because the function to learn is expected to have some local smoothness properties.

More formally, let us define the training set as a sequence $e_1 ... e_I$ of words $e_i \in V$, where $V \subseteq \mathcal{E}^*$ is a finite vocabulary set from the target language. Our goal is to learn a model:

$$f(e_i, ..., e_{i-n+1}) = \hat{P}(e_i|e_1^{i-1}) \tag{2.12}$$

which produces a low perplexity. Because we are working with distribution probabilities, we must add the following constraint to our model: for any choice of e_1^{i-1} we must ensure that $\sum_{j=1}^{|V|} f(j, e_{i-1}, ..., e_{i-n+1}) = \mathbf{1}$ with $f > 0$.

We can decompose this model ($f(e_i, ..., e_{i-n+1}) = \hat{P}(e_i|e_1^{i-1})$) in two parts:

1. A mapping for each word index of the vocabulary to a real distributed feature vector, i.e. a function $\mathbf{C} : \mathbb{N} \rightarrow \mathbb{R}^{\mathbf{m}}$ from any element i from V to a real vector $C(i) \in \mathbb{R}^m$, where m is the size of the feature vector. This mapping C is represented by a $|V| \times m$ matrix of free parameters.
2. A probability function over words, expressed by means of the previous mapping C: we define a function $\mathbf{g} : \mathbb{N} \times \mathbb{R}^{\mathbf{m}}_{\mathbf{1}} \times ... \times \mathbb{R}^{\mathbf{m}}_{\mathbf{n-1}} \rightarrow \mathbb{R}^{\mathbf{I}}$ which maps, for the next word e_i, expressed by its index in the vocabulary vector, and an input sequence of feature vectors for words in context $(C(e_{i-n+1}), ..., C(e_{i-1}))$, to a conditional probability distribution over words in V. The output of g is a vector whose jth element estimates the probability $\hat{P}(e_i = j|e_1^{i-1})$.

The model function $f : \mathbb{N}_1 \times ... \times \mathbb{N}_n \rightarrow \mathbb{R}^I$ is a composition of these two functions: it receives an element from V, e_i, represented by its index in the vocabulary, $j = e_i$, and the context of this word, $e_{i-1}, ..., e_{i-n+1}$, and outputs the vector computed by g , where the ith element is the conditional probability for the input word and its context:

$$f(e_i, e_{i-1}, ..., e_{i-n+1}) = g(j, C(e_{i-1}), ..., C(e_{i-n+1})) \tag{2.13}$$

Each one of these two functions has associated a set of parameters:

1. **Parameters of mapping C**: feature vectors of the words. They are represented by a matrix C, of size $|C| = |V| \times m$, where each row i corresponds to the feature vector $C(i)$ for the word i. It should be noted that this matrix is shared by all the words in the context.

2. **Parameters of function g**: this function is implemented by means of a neural network with parameters ω.

We can include those sets into an overall parameter set $\Theta_{LM} = (C, \omega)$. These parameters are estimated according the maximum likelihood estimator (see equation 1.5) for the training corpus, adding a regularization term $R(\Theta_{LM})$:

$$\hat{\Theta}_{LM} = \arg\max_{\Theta_{LM}} \left\{ \frac{1}{I} \sum_{i} \log f(e_i, e_{i-1}, ..., e_{i-n+1}; \Theta_{LM}) + R(\Theta_{LM}) \right\} \quad (2.14)$$

Feed-Forward Networks

The neural network language models gained attention after the publication of [BDVC03], although previous work on this area was done earlier. In this publication, the neural network used was a feed-forward neural network. Although these neural networks are able to address the n-gram problems of similarity between words and smoothing, they are still not free from the limitation history: their predictions for the next word are only based on the previous state of the hidden layer. With these neural networks, the problem is not solved and it is necesary to use other type of neural network.

Recurrent Neural Network Language Model

Because of the problem of representing the history, Thomas Mikolov proposed to use a *recurrent neural network* instead of a feed-forward network for modeling the previous function g [Mik12].

In general, a recurrent neural network is a class of artificial neural network where the connections between units form a directed cycle. This creates an internal state of the network which allows it to model temporal behaviours. From here, subindex i will refer to the current state of the network, while subindex $(i-1)$ will refer to the previous network state and so on. The network is composed by a set $Y = \{y_{i,1}, ..., y_{i,d_Y}\}$ of d_Y units and a set $W = \{w_{i,1}, ..., w_{i,d_W}\}$ of d_W inputs [Cas14]. These connections are weighted by the sets of weights ω^W (for the weights between input and output units) and ω^Y (for the weights between the output units of the previous state and the current output units). ω^W contains, for each input unit, connections to each output unit, hence, $|\omega^W| = d_X \times d_Y$. On its part, ω^Y contains, for each output unit, the connections to all the output unit of the previous state, thereby, $|\omega^Y| = d_y \times d_Y$. The system gets as input samples $w_i \in \mathbb{R}^{d_W}$ and it outputs $y_i \in \mathbb{R}^{d_Y}, (i = i_0, i_0 + 1, i_0 + 2, ...)$. Figure 2.1 shows the general architecture of a recurrent neural network.

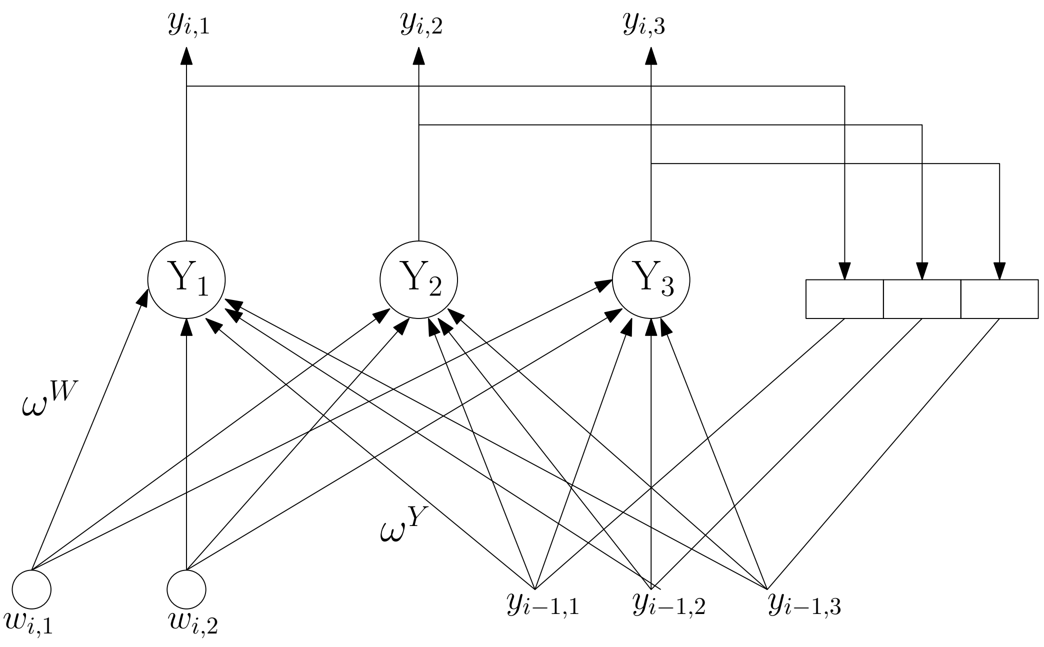


Figure 2.1: General recurrent neural network architecture. Here, $d_Y = 3$ and $d_W = 2$.

The approach presented by Mikolov makes use of the so-called Elman Networks. This is a neural network architecture which belongs to the group of *augmented neural networks.* This group of neural networks are distinguished by being the composition of a recurrent neural network with a feed-forward neural network. In Elman networks, the feed-forward neural network has only one layer, thus the network consists of a set $W = \{w_{i,1}, ..., w_{i,d_W}\}$ of input units, a set $S = \{s_{i,1}, ..., s_{i,d_H}\}$ hidden units and a set $Y = \{y_{i,1}, ..., y_{i,d_Y}\}$ of output units. As before, the connections are weighted by sets of weights. Since here is one more layer, we have one more set. Thus, the three sets are: the set of weights between the input layer and the hidden layer, ω^W, of size $d_W \times d_H$; the set of weights between the current hidden layer and the previous steps hidden layer, ω^S, of size $d_H \times d_H \times T$, where T is the number of times that the hidden layer state is stored; and finally, the set $\omega^Y, |\omega^Y| = d_H \times d_Y$, of weights between the hidden layer and the output layer. Figure 2.2 shows the general architecture of an Elman network.

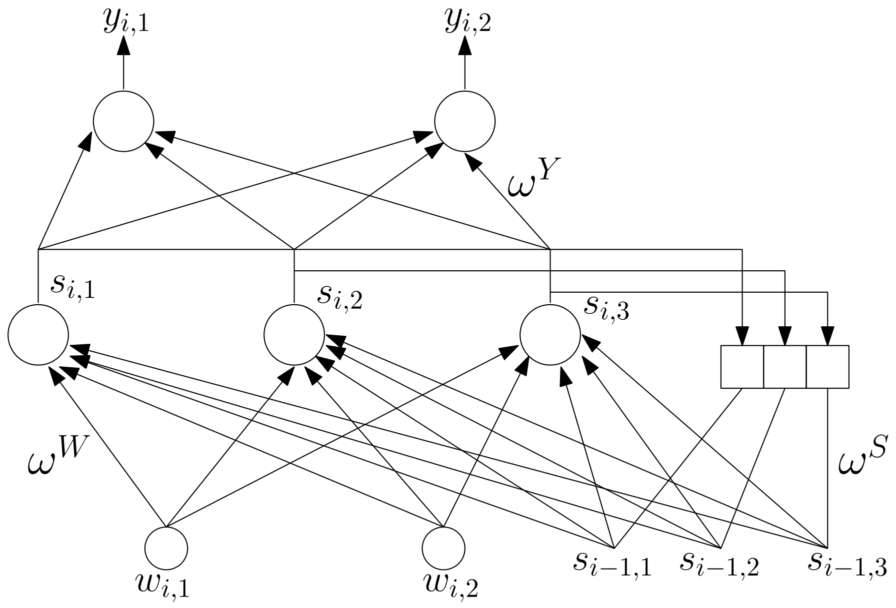


Figure 2.2: Elman recurrent neural network architecture, with $d_W = 2$, $d_S = 3$ and $d_Y = 2$.

The notation used in the next sections is as follows: matrices are represented by uppercase letters, while a vector of the matrix is represented by the lowercase letter of the matrix, together with its index.

When we tackle the language modeling problem, we have a vocabulary V and a sequence of words $e_1^I = e_1, ..., e_I$. These words enter to the network in a sequential way, from 1 to I and are all included in the vocabulary $(e_1^I(i) \in V, \forall i : 1 \leq i \leq I)$. We can encode all the words of the sequence e_1^I from 1 to $|V|$ with a binary vector of size $|V|$, where all the elements are equal to 0 except the index that represents the word e_i, which is equal to 1. Hence, the whole sequence of words is encoded through a 2-dimensional matrix of size $I \times |V|$ where the row i of the matrix contains a vector $\mathbf{w}(i)$ which represents the current word e_i. This vector is the input layer to the network, thereby, $|\mathbf{w}(i)| \equiv |V|, \forall i : 1 \leq i \leq I$, i.e. the input layer has the size of the vocabulary.

Each layer of the network has associated its corresponding weight matrices, namely **U** (between input and hidden layer), **W** (between previous-step hidden layer and current hidden layer) and **V** (between hidden and output layer). For a schematic overview of the network, see Figure 2.3.

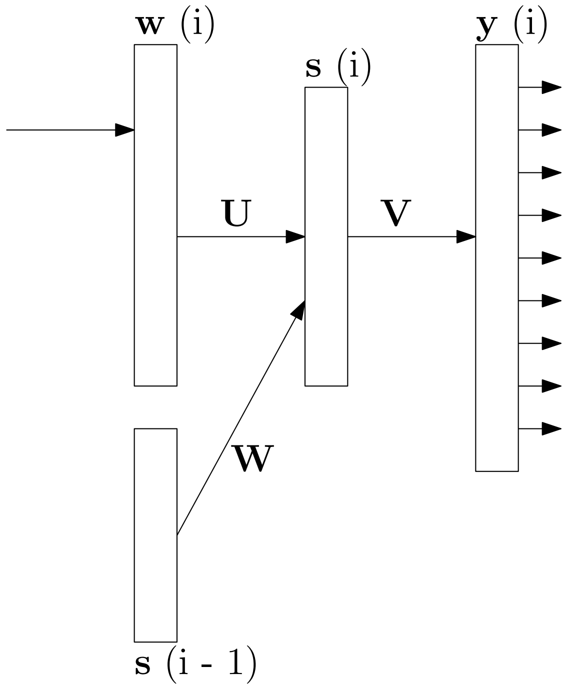


Figure 2.3: Recurrent neural network language model scheme.

Let us define the vector s as the output values in the hidden layer. Thereby $s(i)$ are the hidden layer values of the current step and $s(i-1)$ are the hidden layer values of the previous step. The output layer is represented by the vector $y(i)$ and it expresses the value of $\hat{P}(e_i + 1|e_i, \mathbf{s}(i-1))$, that, as shown in equation 2.12 is the probability that we are looking for.

According to Elman networks, we compute the output values of the hidden layer, for $1 \leq h \leq d_H$ as:

$$s_h(i) = f\left(\sum_j^{|V|} \mathbf{w}_j(i) \cdot u_{hj} + \sum_l^{d_H} s_l(i-1) \cdot w_{hl}\right) \tag{2.15}$$

where d_H is the number of units of the hidden layer and f is the sigmoid activation function: $f(z) = \frac{1}{1+e^{-z}}$.

The output values of the output layer, for $1 \leq o \leq d_Y$ are computed as:

$$y_o(i) = g\left(\sum_h^{d_H} s_h(i) \cdot v_{oh}\right) \tag{2.16}$$

where d_H is the number of hidden units, and g is the softmax activation function: $g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$. Note that that softmax function ensures a valid probability distribution (all its elements are between 0 and 1 and their sum is 1), necessary for the output of the system.

The last two equations can also be expressed by means of these matrix and vector products:

$$s(i) = f(\mathbf{U} \cdot \mathbf{w}(i) + \mathbf{W} \cdot \mathbf{s}(i-1)) \tag{2.17}$$

$$y(i) = g(\mathbf{V} \cdot \mathbf{s}(i)) \tag{2.18}$$

The training algorithm used for this network is *backpropagation through time*. For more details about training, see Section 2.4.2.

Factorization of the output layer

Training the network can be costly. Each step of the training process has a computational time complexity of $\mathcal{O}(d_H \times d_H + d_H \times |V|)$, where d_H is the size of the hidden layer and $|V|$ is the size of the vocabulary. It should be noted that $d_H << |V|$, therefore the main bottleneck is the factor $d_H \times |V|$. The reduction of this huge factor has been tackled through various approaches. One of them, which leads to a good performance without a big impact on the network's accuracy, is the factorization of the output layer. This technique is based on the Goodman's idea of using C classes for improve the training speed of its maximum entropy models [Goo01] . It assigns each word, e_i, of the vocabulary set to one of the C classes. Considering Equation 2.4, the probability for a word can be rewritten as:

$$P(e_i|e_{i-n+1}...e_{i-1}) = P(class(e_i)|e_{i-n+1}...e_{i-1}) \times P(e|e_{i-n+1}...e_{i-1}, class(e)) \tag{2.19}$$

The idea behind this equation is that, the prediction of a word can be computed as the prediction of its class given the previous words histoy and the probabililty of the own word, given its class and history. Thus, there are two models: one which computes the class of a word and other which, given this class, predicts the word.

In the neural network language model it is possible to proceed in the same way [MKB+11]: instead of compute the outputs of all the vocabulary, first is computed the class distribution and later, the distributions of the words which belong to a certain class. Complexity factor $|V|$ is reduced to $C + |V'|$ where $|C|$ is constant and $|V'| < |V|$. $|C|$ represents the number of classes and $|V'|$ represents the number of words that belong to the particular class.

The language model developed by Mikolov includes this improvement: before the training process starts, the learning algorithm assigns to each word e_i from the vocabulary a single class c_i. Then, it is computed the probability distribution over the classes, C and next, it is computed the probability distribution for the words which belong to each class. The probability of a word is computed as Goodman proposed:

$$P(e_{i+1}|s(i)) = P(c_i|s(i)) \cdot P(e_i|c_i, s(i)) \tag{2.20}$$

where, as before, $s(i)$ is the state of the output layer in the moment i, e_i is an index of the predicted word and c_i is its class. During the training phase, the hidden layer is the responsible of compute both words and class probabilities.

Interpolation with other language models

Another significant improvement can be done to the output result of the recurrent neural network language model [MKD+11a]: perform the interpolation of the neural network language model with other language models, such as n-gram language models. One of the simplest interpolation methods, linear interpolation, has yielded important improvements. The linear interpolation states that, given two language models M_1, M_2 and an interpolation weight $\lambda, 0 \leq \lambda \leq 1$, the interpolation between both models is computed as:

$$P_{M_{1,2}}(e) = \lambda \cdot P_{M_1}(e) + (1 - \lambda) \cdot P_{M_2}(e) \tag{2.21}$$

As it is shown in Chapter 3, this technique yields very good results.

2.2. Translation Models

Once we have formulated some language models, it is time to focus on the translation model. If we take a look back to the *fundamental equation of machine translation* (equation 1.2), we are now interested on the probability $Pr(f_1^J|e_1^I)$. This probability must be large for strings e_1^I that are good translations of f_1^J, even if they are not correctly formed according to the target language $\mathcal{E}$; correctness is checked by the language model, therefore we do not have to concern about it now.

2.2.1. Single-Word Alignment Models

The first approaches in modern SMT were the *single-word aligment models*. They are based on the concept of *alignment*. According to [BPPM93], an alignment between two strings, f_1^J in source language $\mathcal{F}$ and e_1^I in target language $\mathcal{E}$, is a correspondence that indicates, for each word of the string f_1^J, the word from the e_1^I string from which it arose:

$$a \subseteq 1...J \times 0...I$$

Where $a_j = i$ if the jth source position (f_j) is aligned with the ith target position (e_i). In addition, an artificial position is generated and placed in index 0 for convention, and means that the word from f_1^J which is aligned with this null position, is aligned with no word from sentence e_1^I. These alignments can be represented graphically by connections between words, as it is shown in Figure 2.4.

Let $\mathcal{A}(f_1^J, e_1^I)$ be the set of all possible alignments between two sentences f_1^J and e_1^I and let $Pr(f_1^J, a_1^J|e_1^I)$ be the probability of, given the alignment hidden variable a_1^J, translate the sentence f_1^J into e_1^I.We can rewrite the probability we were interested in by means of this probability:

$$Pr(f_1^J|e_1^I) = \sum_{a_1^J \in \mathcal{A}(f_1^J, e_1^I)} Pr(f_1^J, a_1^J|e_1^I) \tag{2.22}$$

Since $|e| = I$ and $|f| = J$, there are $I \cdot J$ possible connections between the strings, hence there are 2^{IJ} alignments in $\mathcal{A}(f_1^J, e_1^I)$.

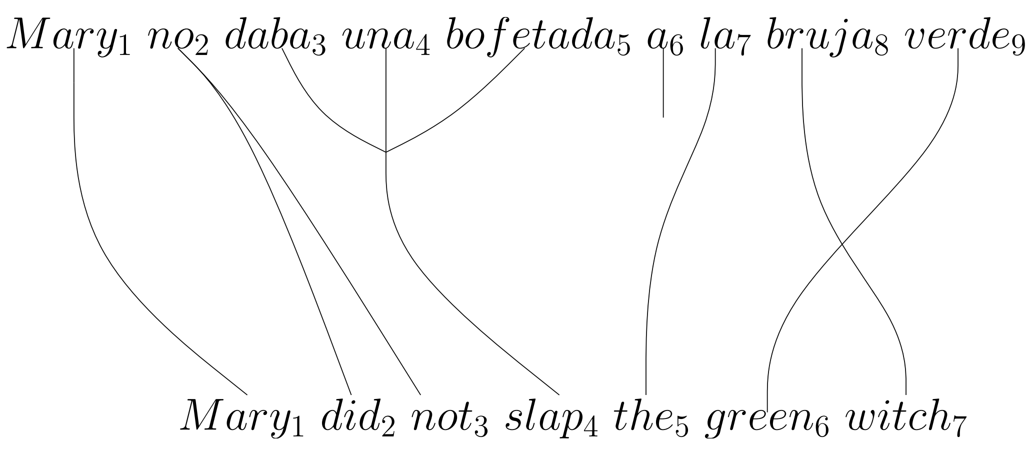

$Mary_1$ no_2 $daba_3$ una_4 $bofetada_5$ a_6 la_7 $bruja_8$ $verde_9$
$Mary_1$ did_2 not_3 $slap_4$ the_5 $green_6$ $witch_7$

Figure 2.4: Alignments between an English and a Spanish sentence. Note that f_6 (a), has no correspondence with any word in English. Hence, it is aligned with position 0.

IBM Models

In 1993, researchers from IBM's *Thomas J. Watson Research Center* developed the so-called *IBM models* [BPPM93]. There are five models, based on the above concept of alignment between words. They are all based on the following derivation of equation 2.23:

$$Pr(f_1^J, a_1^J|e_1^I) = Pr(J|e_1^J) \prod_{j=1}^{J} Pr(a_j|f_1^{j-1}, a_1^{j-1}, e_1^I) \cdot Pr(f_j|f_1^{j-1}, a_1^j, e_1^I) \quad (2.23)$$

This equation states that, regardless of the form of $Pr(f_1^J, a_1^J|e_1^I)$, it can always be decomposed in a product of terms in this way. It establishes a sequential way of processing it. When we generate a string f_1^J we:

1. Choose the length of f_1^J, given our knowledge of e_1^I ($Pr(J|e_1^J)$).

2. Choose where to connect the jth position of f, (a_j), given our knowledge of the built-so-far sentence and alignments from f (f_1^{j-1}, a_1^{j-1}) and the source string (e_1^I).

3. Finally, we choose the identity of the jth word in the target string (f_j) given our knowledge of the source string e_1^I, the built-so-far target sentence f_1^{j-1} and the current alignments, include the one generated in the last step (a_1^j).

It is clear that we must iterate over points 2 and 3. As we step over the target string, we make decisions which build the target string up, based on the previous knowledge from the string. In all five models, the points 1 and 3 are common. Probability $Pr(f_j|f_1^{j-1}, a_1^j, e_1^I)$ is approximated by a statistical dictionary of words $p(f_j|e_{a_j})$. Models variate their assumptions over step 2, specifically over probability $Pr(a_j|f_1^{j-1}, a_1^{j-1}, e_1^I)$. Shortly, these differences are:

- **Model 1**: the distribution $Pr(a_j|f_1^{j-1}, a_1^{j-1}, e_1^I)$ is meant to be uniform.
- **Model 2**: the distribution $Pr(a_j|f_1^{j-1}, a_1^{j-1}, e_1^I)$ is estimated by a zero-order model $p(a_j|i, J, I)$ which defines dependencies between absolute word positions of target and source sentences.
- **Model 3**: a *fertility model* $p(\phi|e)$ is included. It represents, for each target word e, the probability that e generates ϕ source words. The choice of ϕ depends only on e. The distribution $Pr(a_j|f_1^{j-1}, a_1^{j-1}, e_1^I)$ is approximated by a zero-order model, called *distortion model* $p(i|a_j, J, I)$, which models the dependencies between absolute word positions of source and target sentences.
- **Model 4 and 5**: they use a first-order a distortion model with dependencies between relative word positions of source and target sentences.

IBM models are estimated by means of the *Expectation-Maximization* algorithm (EM). For more information about this algorithm, see Section 2.4.1.

Already in 1993, IBM's researchers realized themselves that their models have some shortcomings: they defined their models as *deficient*. Deficiency is the property of a model of not concentrating all of its probability on the events of interest, but wasting its probability on the so-called generalized strings (strings with alignment positions with many words and positions with none). Models 3 and 4 are deficient theoretically while models 1, 2 and 5 are deficient on *spirit*. Deficiency can be seen as the price paid for having computationally tractable models.

The greater disadvantage of these models is that they, by ignoring alignments larger than one word, miss the contextual information. For tackling this problem, a new family of models were developed: the *multi-word alignment models*, which are able to capture contextual information. These models are studied in the following section (Section 2.2.2).

2.2.2. Multi-Word Alignment Models

As said in the previous section, the single-Word alignment models, by ignoring multi-word alignments, are not able of capture contextual information. In order to solve this problem, a new group of techniques emerged: the so-called *multi-word alignment models*. These models set alignments between groups of words from source and target sentences, instead of just single word alignments.

The basic idea behind multi-word alignment models is the concept of *phrase*. A phrase is defined as a set of one or more consecutive words of the source or the target sentences. Given the source and target sentences f_1^J, e_1^I, we use the following notation to denote an unspecified phrase: $\widetilde{f}$ and $\widetilde{e}$.

Phrase-Based Models

Phrase based models make use of the so-called *statistical dictionaries of phrase pairs* [ZON02]. These dictionaries are sets of bilingual phrases. A bilingual phrase is a pair of m source words and n target words. If they are extracted from a bilingual corpus, the following constraints must be satisfied:

1. The words in the phrase are consecutive.
2. They are consistent with the word alignment matrix (A): the m source words must be aligned only to the n target words and vice versa.

An additional hidden variable must be added to use bilingual phrases in the translation model, the *bisegmentation* of the source and target sentences. A bisegmentation of length K of a sentence pair (f_1^J, e_1^I), denoted by $B(f_1^J, e_1^I)$, represents a segmentation of the sentence pair (f_1^J, e_1^I) into K phrases: $(\widetilde{f}_1^K; \widetilde{e}_1^K)(1 \leq K \leq min(I, J))$. A bisegmentation is a phrase-level alignment of a sentence pair. With this, it is possible to rewrite the translation probability without loss of generality as:

$$Pr(f_1^J|e_1^I) = \sum_B Pr(f_1^J, B|e_1^I) = \sum_B Pr(B|e_1^I) \cdot Pr(f_1^J|B, e_1^I) \tag{2.24}$$

In order to manage the computation of this probability, various assumptions can be done. In [ZON02], it is assumed that all segmentations have the same probability $\alpha(e_1^I)$ and they allow only monotone translations, i.e. preserving the phrase order. This will result in an efficient search. Hence, the probability distribution is:

$$Pr(f_1^J|e_1^I) = \alpha(e_1^I)\sum_B \prod_{k=1}^{K} p(\widetilde{f}_k|\widetilde{e}_k) \tag{2.25}$$

where $p(\widetilde{f}_k|\widetilde{e}_k)$ is the phrase translation probability. It is estimated through relative frequencies:

$$p(\widetilde{f}|\widetilde{e}) = \frac{N(\widetilde{f},\widetilde{e})}{N(\widetilde{e})} \tag{2.26}$$

where $N(\widetilde{e})$ is the count of the phrase $\widetilde{e}$, and $N(\widetilde{f},\widetilde{e})$ is the count of the event that $\widetilde{f}$ has been seen as a translation of $\widetilde{e}$ in the training corpus. If during the test phase appears an unknown word, this is translated by itself.

It should be noted that the monotony constraint can lead to many translation errors in languages with different phrase order, for example, English-German. If monotonic alignments are not assumed, previous Equation 2.25 can be rewritten as [OM11]:

$$Pr(f_1^J|e_1^I) = \sum_{\widetilde{a}_1^K} \prod_{k=1}^{K} p(\widetilde{a}_k|\widetilde{a}_1^{k-1}) \cdot p(\widetilde{f}_k|\widetilde{e}_{\widetilde{a}_k}) \tag{2.27}$$

The search problem is tackled via the maximum approximation:

$$\begin{aligned} \hat{e}_1^I &= \arg\max_{e_1^I} Pr(e_1^I)\cdot\alpha(e_1^I)\cdot\sum_B Pr(\widetilde{f}_1^K, B|\widetilde{e}_1^K) \approx \\ &\approx \arg\max_{e_1^I,B} Pr(e_1^I)\cdot\alpha(e_1^I)\max_{K,B}\cdot Pr(\widetilde{f}_1^K, B|\widetilde{e}_1^K) \end{aligned} \tag{2.28}$$

If only monotonic translations are allowed, this search can be efficiently computed through dynamic programming techniques. Otherwise, other techniques must be used, such as using a reordering graph and pruning [ZON02].

2.3. Evaluation

Up to now, we have some models to produce automatic translations. We are now interested in measure the quality of these translations. We are evaluating the output of the machine translation system, instead of its performance or usability. The output metrics can be categorized into two main categories: human evaluation and automatic evaluation. Human evaluation follows adequacy and fluency criteria. The adequacy criterion represents how much of the meaning expressed in the source sentence is expressed in the target sentence and the fluency criterion measures how well-formed is a sentence, if it contains correct spellings, if it is an acceptable sentence in the target language, etc. These two criteria have an analogy with the aim of the translation models and language models, respectively; in the sense that a good language model will produce fluent outputs and a good translation model will produce adequate sentences. The human evaluation is very accurate and produces high quality metrics, but on the other hand, it is expensive and slow. It requires the establishment by human linguists of assessments of adequacy and fluency criteria. Then, human judges evaluate fluency and adequacy of the system translations [Fed08].

But MT researchers must receive a daily response from their systems. With human translation that is not possible, therefore, exist automatic MT metrics. They try to follow a similarity with human evaluation. They are inexpensive and fast, but the quality of these measures is lower than the human checking. Automatic metrics must be objective, informative, efficient and cheap. These methods compare the output of the system with high-quality human translations, called references. In the next sections, we are reviewing some of the main automatic metrics for SMT.

2.3.1. Perplexity

In the first place, we want to evaluate the quality of a language model, that means, how similar our estimation of the model distribution given by Θ_{LM} is with respect of the real probability distribution (but unknown) $Pr(e_1^I)$. For that purpose, we can obtain an empirical estimation of the *cross entropy* of those distributions: let $\mathcal{X} = \{e_1, e_2, ..., e_N\}$ be the set of test samples and Θ_{LM} is the set of language model parameters, the cross entropy is defined as:

$$H(Pr; p(\cdot|\Theta_{LM})) = -\sum_{i=1}^{N} \mathrm{Pr}(e_i) \cdot \log_2 p(e_i|\Theta_{LM}) \tag{2.29}$$

The most extended metric in evaluation of statistical language models is the *Perplexity*. It is computed as:

$$PPL(Pr; p(\cdot|\Theta_{LM})) = 2^{H(Pr;p(\cdot|\Theta_{LM}))} \tag{2.30}$$

The perplexity of a language model can be interpreted as the ability of a language model of predicting a sample. It also can be understood as the geometric average of the branching factor of the given language with respect the model. It measures either the model performance and the language complexity. Of course, the lower the perplexity a model have, the better it is.

2.3.2. BLEU

BLEU (BiLingual Evaluation Understudy) tries to model the correspondence between the output from a MT system and the one produced by a human. For BLEU score, the closer a machine translation is to a professional human translation, the better it is. This metric is applied to individual sentences, comparing them to their reference sentences.

BLEU score is based on the idea of *modified n-gram precision* [PRWZ02]. This concept relies on the *n-gram precision*, which counts up the number of candidate words (unigrams) from the system translation which appear in the reference sentence, and divides this count by the total number of words of the translation from the system. The problem is that the system can generate too many reasonable words. For example, if we have the system translation "`The the the the the the the.`" and the reference translation is "`The truck was driving on the highway.`", unigram precision will be 7/7. Obviously, this is absolutely not correct. Because of that, modified n-gram precision was defined as an extension of this procedure: first, the maximum number of times that a word happens in any reference translation is counted. Next, the count of each word in the sentence from the system is clipped by its maximum count. Clipped count operation is defined as: $Count_{clip} = \min(Count, Max\_Ref\_Count)$. Back to our example, the system translation would obtain an unigram modified precision of 2/7, because counts of word `the` in the reference translation (2) represent an upper bound for the count of this word in system translation (7).

This concept is extended to n-gram in a similar way: all candidate n-gram counts are computed and clipped by their corresponding value, added and normalized by the total number of candidate n-gram. If we move our example to bigrams, the n-gram modified precision of the system translation is 0, since counts of the bigram `the the` on reference sentence is 0.

The modified n-gram precision captures two important criteria of translation scoring, above mentioned: fluency and adequacy. A translation using the same words as in the reference sentence (unigram modified precision) will satisfy the adequacy criterion and a translation with longer n-grams will tend to achieve a high fluency [PRWZ02].

BLEU score uses as basic computation unit a sentence and extends the computation to the whole test corpus, so, finally, an unique modified precision score for the entire corpus is generated by 1. computing the n-gram matches sentence by sentence and 2. adding the clipped n-gram counts for all these candidate sentences. Finally, this result is divided by the number of candidate n-grams in the test corpus:

$$p_n = \frac{\sum_{\mathcal{C}\in\{Candidates\}} \sum_{n-gram\in\mathcal{C}} Count_{clip}(n-gram)}{\sum_{\mathcal{C}'\in\{Candidates\}} \sum_{n-gram'\in\mathcal{C}'} Count_{clip}(n-gram')} \tag{2.31}$$

In addition, it is obvious that a translated sentence should be neither excessively long or short, therefore this must be reflected on the metric. System translations longer than references are naturally penalized by modified n-gram precision, but for shorter translations, it is introduced the so-called *brevity penalty (BP)* factor: a penalty factor which penalizes short translations in the following way:

$$BP = \begin{cases} 1, & \text{if } c > \text{r} \\ \\ e^{1-\frac{r}{c}}, & \text{if } c \leq \text{r} \end{cases} \tag{2.32}$$

where c is the length of the system translation and r is the length of the corpus reference sentence.

BLEU score uses a weighted geometric mean of the n-grams for combining them. Each n-gram has a weight w_n such that $\sum_{n=1}^{N} w_n = 1$, but typically, this weight is set to $w_n = \frac{1}{N}$. Hence, the final BLEU score of a test corpus is computed as:

$$BLEU = BP \cdot exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{2.33}$$

2.3.3. Other evaluation measures

There are many other criteria for the evaluation of the output of a Machine Translation system. Here are four more automatic evaluation measures:

- **WER** (word error rate): this score is based on the Levenshtein distance, but it works at word level. WER is the minimum number of substitution, insertion and deletions operations that need to be performed to convert the system translation into the target sentence.

- **PER** (position-independent WER): is a metric derived from WER. While WER requires a perfect word order, PER compares the words in both sentences ignoring the order.
- **SER** (sentence error rate): SER is computed as the number of times that the sentence generated by the system corresponds exactly to one of the reference translations.
- **TER** (translation edit rate): TER computes the amount of edit operations needed to convert a translation hypothesis of the system into one of the reference sentences, normalized by the average length of the references [SDS$^+$06].

2.4. Parameter Estimation

Now that the statistical models involved in the MT translation have been defined, it is time to estimate their sets of parameters. This task belongs to the training problem of machine learning. Standard techniques are used to estimate these parameters. In this section, the main techniques used in this work will be explained.

2.4.1. Expectation-Maximization algorithm

The expectation-maximization algorithm (EM), also known as estimation-maximization, is used to find the maximum likelihood parameters of a statistical model where it depends on unobserved hidden variables. There are two main problems of statistical models for which EM algorithm is applied are:

1. These models involve latent variables, unknown parameters and a limited set of known data observations, i.e. usually there are missing values among the data.
2. Typically, finding the optimal maximum likelihood solutions implies the derivation of the likelihood function with respect all the unknown variables and solving the resulting equations. Usually, is not possible to carry this out analytically, due to the intractability of the likelihood function. Instead of this, likelihood function can be simplified by assuming the existence of additional hidden parameters.

More formally, given a statistical model, EM algorithm assume that data $\mathcal{X}$ is observed and generated by a distribution which has the set of parameters

Θ. There is a complementary set of unobserved data $\mathcal{Y}$, which, together with $\mathcal{X}$ compose the set of complete data $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$. Finally, a probability density function is assumed for the complete data set:

$$p(\mathbf{z}|\Theta) = p(\mathbf{x}, \mathbf{y}|\Theta) = p(\mathbf{y}|\mathbf{x}, \Theta) \cdot p(\mathbf{x}|\Theta) \tag{2.34}$$

From this equation, a new expression of the log-likelihood function is defined: $\mathcal{L}(\Theta, \mathbf{z}) = \mathcal{L}(\Theta, \mathbf{x}, \mathbf{y})$. This function is known as *complete data log-likelihood function* while the original log-likelihood function $\mathcal{L}(\Theta, \mathbf{x})$ is known as the *incomplete data log-likelihood function.* EM algorithm attempts to find the maximum likelihood estimator applying the following two steps:

1. Find the expected value (E) of the complete data log-likelihood function, $\log p(\mathbf{x}, \mathbf{y}|\Theta)$, with respect the conditional distribution of the hidden data and the previous estimation of the model parameters:

   $$Q(\Theta, \Theta^{t-1}) = E[\log p(\mathbf{x}, \mathbf{y}|\Theta)|\mathbf{x}\Theta^{t-1}] \tag{2.35}$$

   Where Θ^{t-1} are the current estimated parameters, Θ is are the new parameters that are being optimized to increase Q.

2. Find the set of parameters Θ that maximizes the quantity value, Q, computed at the previous step:

   $$Q(\Theta, \Theta^{t-1}) = \arg\max_{\Theta} Q(\Theta, \Theta^{t-1}) \tag{2.36}$$

This process is carried out iteratively, starting from initial parameter values, e.g. uniform parameter values. At each iteration of the algorithm, both steps are executed. Each iteration improves the log-likelihood of the incomplete data $\mathcal{L}(\Theta, \mathbf{x})$ or leaves it unchanged. For most models, the EM algorithm reaches a local maximum of $\mathcal{L}(\Theta, \mathbf{x})$.

2.4.2. Backpropagation Through Time

This is the training algorithm for the recurrent neural networks that are used in language modeling (see Section 2.1.3). Although typical Backpropagation algorithm can be used too in these networks, it does not take advantage of the capabilities of recurrent neural networks. With normal Backpropagation, the network tries to optimize its prediction of the next word based only on the previous word and the previous state of the hidden layer, but it does not store effectively valuable information for the future. In order to manage this capability of recurrent neural networks, it is necessary to extend the learning algorithm. This extension is known as *Backpropagation through time* algorithm.

The idea of the algorithm is as follows: an Elman recurrent network, with one hidden layer which is used for N time steps, can be seen as a feedforward network with N hidden layers (see Figure 2.5). This pseudo feedforward network can be trained by the normal gradient descent process: errors are propagated from the current hidden layer $s(i)$ to the previous step hidden layer $s(i-1)$ and the recurrent weight matrix $\mathbf{W}$ is updated. Weight matrices $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$ are initialized with small random numbers. The training of the network for one epoch is performed following Algorithm 20 [Mik12].

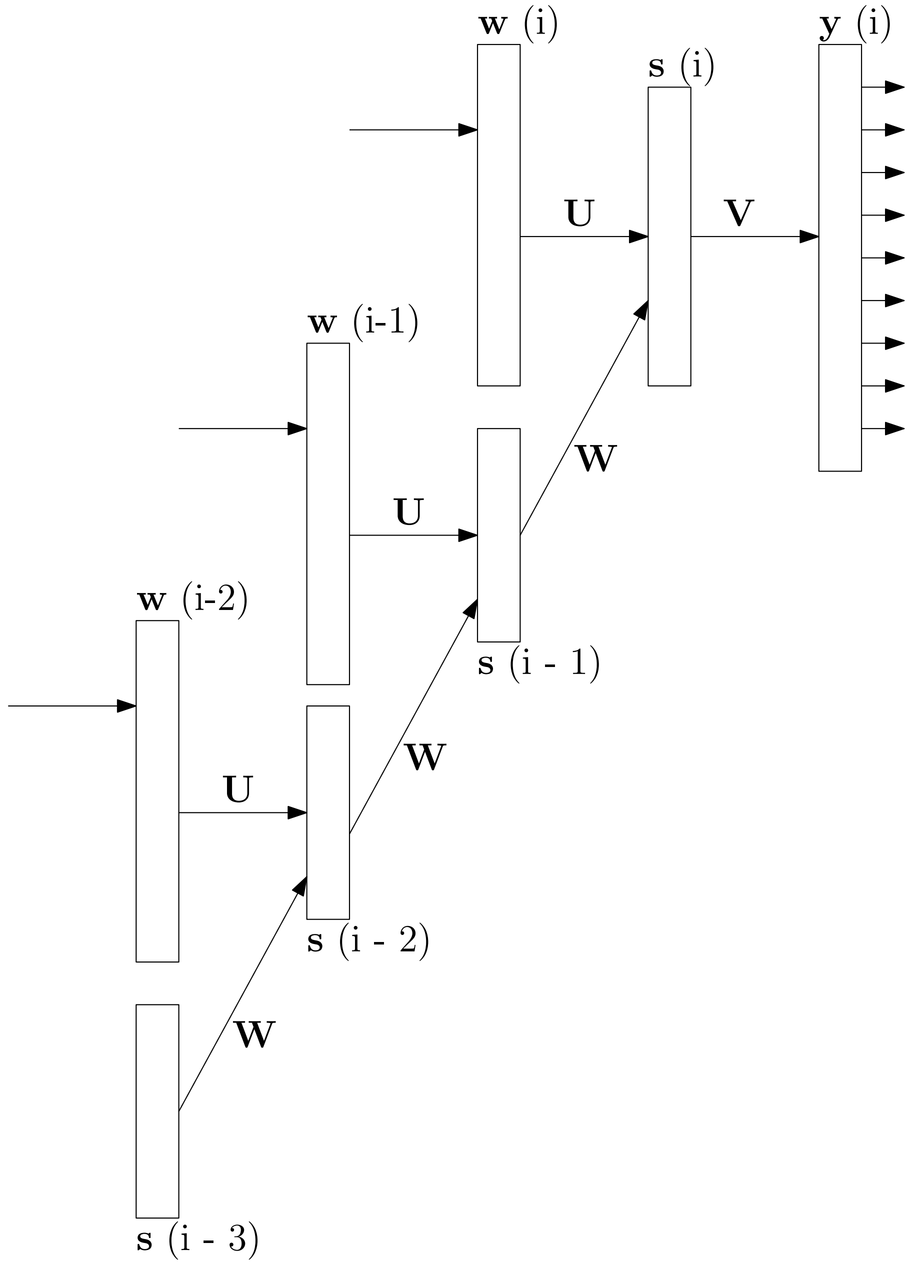


Figure 2.5: Recurrent neural network seen as a feedforward neural network unfolded 3 steps in time [Mik12].

Algorithm 1: Pseudocode for the backpropagation through time training algorithm.

```
Input: $\mathcal{X} = \{e_1, ..., e_n\}$ (training corpus),
$p_{RNNLM}(e) = \{$ $\mathbf{w}$ (input layer), $\mathbf{s}$ (hidden layer), $\mathbf{y}$ (output layer)$\}$
(neural network),
$\alpha$ (learning rate),
$T$ (steps back in time to propagate)
Output: $p_{RNNLM}(e) = \{$ $\mathbf{w}$ (input layer), $\mathbf{s}$ (hidden layer), $\mathbf{y}$ (output
        layer), $v_{ho}$ (weights between $s$ and $y$), $u_{jh}$ (weights between
        $w$ and $s$), $w_{lh}$ (recurrent weights)$\}$ (trained neural network)
1  Auxiliar: $i$ (Time counter),
2  $\delta_{hj}(e, i) = es_j(i)(1 - s_j(i))$,
3  $\mathbf{d}(i)$ (target vector to be predicted)
4  begin
5      $i = 0$ ;                              // Time counter initialization
6      $\mathbf{s}(i) = 1$ ;               // Initialization of the hidden units
7      foreach $e_i \in \mathcal{X}$ do
8          $i = i + 1$;
9          $\mathbf{w}(i) \leftarrow e_i$ ;  // Present current word to the input layer
10         $w(i) = \mathbf{s}(i-1)$ ;        // Copy the state of the previous
           hidden layer to the input layer
           // Compute output values of the hidden and output
              layer computation
11         $\mathbf{s}_h(i) = f\left(\sum_j^{|\mathcal{E}|} \mathbf{w}_j(i) \cdot u_{hj} + \sum_l^{d_H} \mathbf{s}_l(i-1) \cdot w_{hl}\right)$;
12         $\mathbf{y}_o(i) = g\left(\sum_h^{d_H} \mathbf{s}_h(i) \cdot v_{oh}\right)$;
           // Compute error gradient in the output layer
13         $\mathbf{E}_O(t) = \mathbf{d}(i) - \mathbf{y}(i)$;
14         $v_{ho}(i+1) = v_{ho}(i) + \mathbf{s}_h(i) \cdot E_o(i) \cdot \alpha$;      // Weights update
           // Error gradient in the hidden layer
15         $E_H(i - \tau - 1) = \delta_h(E_H(i - \tau)^T \mathbf{W}, i - \tau - 1)$;
16         if weight update necessary then
               // Weights update
17             $u_{jh}(i+1) = u_{jh}(i) + \sum_{z=0}^{T} \mathbf{w}_j(i-z) \cdot E_h(i-z) \cdot \alpha$;
18             $w_{lh}(i+1) = w_{lh}(i) + \sum_{z=0}^{T} \mathbf{s}_l(i-z-1) \cdot E_h(i-z) \cdot \alpha$;
19     end
20 end
```

If the update of the recurrent weight matrices is done at each training step, it would lead to a large computational complexity of the updating process: $\mathcal{O}(T \times W)$, where T is the number of steps back in time that are taken into account and W is the number of training words. This can be solved by performing an offline training, updating the matrices after processing all the training samples. With a batch training the complexity is lower, but the performance of the network also worsens [Mik12]. The best solution is to choose a compromise: the weight update will be performed in mini-batches, after processing 10-20 training samples). This can effectively remove the term T in the previous expression.

Each training step has a computational complexity $\mathcal{O}(d_H \times d_H + d_H \times |V|)$, where d_H is the size of the hidden layer and $|V|$ is the size of the training vocabulary. Thus, the complexity of the complete training of the model is proportional to $\mathcal{O}(I \times W \times (d_H \times d_H + d_H \times |V|))$, where I is the number of epochs performed before reaching to convergence and W is the number of sentences in the training corpus. It is possible to lower this complexity by reducing the different factors involved in the training algorithm: training epochs, number of training sentences, vocabulary size (see Section 2.1.3) or size of the hidden layer. However, not all reductions are good: the reduction of the training corpus or choosing a small hidden layer size can lead to the degradation of the neural network and therefore, bad results. Both other speedups do not degrade that much the network, hence it is worth to use them [MDP+11].

2.4.3. Downhill Simplex

Along the process of machine translation, it is necessary to optimize some functions. The Downhill Simplex method (also known as Nelder-Mead method) is a technique for performing this optimization. It was proposed in 1965 in [NM65] and it uses the concept of *simplex*.

A simplex is a polytope of $N+1$ vertices in N dimensions, e.g., a triangle in $\mathbb{R}^2$, a tetrahedron in $\mathbb{R}^3$, etc. Downhill simplex aims to minimize a function of n variables, based on the comparison of the function values at the $(n+1)$ vertices of a general simplex, followed by the replacement of the vertex with the highest value by another point. The simplex performs a sequence of transformations for adapting itself to the local landscape and trying to decrease the function values at its vertices. At each step, the transformation is determined by computing one or more test points, together with their function values and by the comparison of these function values with those at the vertices. The simplex becomes smaller gradually and the process finishes when finally the simplex contracts on to the final minimum or the function values f_j are close enough. It is an effective and computationally compact method.

This technique has been widely used in parameter estimation problems, where the function values are uncertain, subject to noise or the function is not tractable analytically. Hence, it is not necessary or possible to compute a highly accurate solution, but by performing an improvement (instead of a full optimization) in the function value is enough to solve the problem. Here is where downhill simplex method works well: it produces significant improvements at first iterations and it quickly obtains a good solution, with a relatively small number of function evaluations.

The main disadvantage of the method is that the lack of a solid convergence theory often produces a *numerical breakdown* of the algorithm: the method may take a huge amount of iterations with a despicable improvement of the function values, despite being way far from a minimum. This problem usually happens in early iterations and it is tackled through heuristics.

Chapter 3

Experiments

Theoretically, neural network language models are supposed to be stronger and more powerful than classic language models approaches, such as n-gram models [Mik12]. The aim of this work is to test this strength with real tasks and observe if there is a real improvement, not only reflected on the evaluation of language model (perplexity), but also in the whole translation. To perform this test, we are going to generate, for each sentence to be translated, a set of translation hypotheses. We will score these hypotheses with different language models and measure the quality of the translations produced by the best hypothesis.

In the following sections we will review the software that we have used for carrying these experiments out. In Section 3.1.1 we will give a brief outline of the main data structures, the experimentation process followed in this work will be explained in Section 3.1.2. In the following sections, we will offer an overview and we will present the results obtained in the experimentation of the three tasks studied in this work: Tourist (Section 3.2), Xerox (Section 3.3) and Europarl (Section 3.4).

3.1. Experimentation Framework

3.1.1. Software

We used three different toolkits: Thot, SRILM and RNNLM. In this section, we will shortly describe the features of each one of them.

Thot

Thot [OMC14] is an open source toolkit developed by Daniel Ortiz Martínez, member of the PRHLT research group from the Polytechnic University of Valencia. The project is currently supported by the European Union and also has received support from the Spanish Government. Thot is still under construction, but it already has implemented a state-of-art phrase-based translation system and tools to estimate all the statistical models involved in the machine translation process. Moreover, it is able to update incrementally and in real time its models, after receiving a new sample.

Thot implements the following remarkable features:

- Phrase-based statistical machine translation decoder (see Section 2.2.2).
- Computer-aided translation (post-editing and interactive machine translation).
- Incremental estimation of all of the models involved in the translation process.
- Robust generation of alignments at phrase-level (see Section 2.2.2).
- Single word alignment model estimation using the incremental EM algorithm (see Sections 2.2.1 and 2.4.1)
- Client-server implementation of the translation functionality.
- Scalable implementation of the different estimation algorithms using MapReduce.

Word Graphs and N-best lists

Since we are are on a rescoring task, we will need the list of the N-best translations for a given source sentence. These N-best translations will be evaluated, rescored and resorted by means of different language model approaches. At the end of this process, we may obtain as best translation a different sentence for each language model. This list of best translations is called *N-best list* and they are obtained from a data structure called *word graph*.

A word graph is a weighted directed acyclic graph, where each node represents a partial translation hypothesis and each edge is labelled by a word (or phrase) of the target sentence and it is weighted according the probabilities supplied by the SMT model involved in the translation process. The word graph has a set of initial nodes, which represent all the different beginnings of all the possible translation of the source sentence; and a set of final nodes, which represent all the different endings of the translation. A complete translation is a path from an initial node to a final node and it has associated a probability computed as the product of all the probabilities of its path edges.

For instance, Figure 3.1 shows a little example for an artificial word graph generated from the Spanish source sentence `"El hombre corre"`. As we can see, there are 2 initial nodes, which represent the words `"The"` and `"A"`. That means that all possible translations must begin either by `"The"` or `"A"`. The set of all possible translations is the set of all diferent paths in the graph from nodes 1 or 2, to final nodes 6, 7 or 8. Each edge has below its probability, therefore each path has its own probability:

1. `A kid jogs`: $0.2 \cdot 0.1 \cdot 0.2 = 0,004$
2. `A kid runs`: $0.2 \cdot 0.1 \cdot 0.7 = 0,014$
3. `A man runs`: $0.2 \cdot 0.7 \cdot 0.7 = 0,098$
4. `A man walks`: $0.2 \cdot 0.7 \cdot 0.1 = 0,014$
5. `The man runs`: $0.8 \cdot 0.7 \cdot 0.7 = 0,392$
6. `The man walks`: $0.8 \cdot 0.7 \cdot 0.1 = 0,056$
7. `The male walks`: $0.8 \cdot 0.2 \cdot 0.1 = 0,016$

Therefore, for this example, the most likely translation is `The man runs`.

The probabilities of the edges of the word graph are the weighted sum of the estimated scores (estimated log-probabilities) by the different statistical models that compound the MT system. Obviously, if these weights are changed, the probabilities of the word graph will change too, and other candidates will rise up to become the most likely translations. If we take the best translation of each sentence from the N-best list, we will obtain the translations of the input corpus provided by our system, and we will be able to score it with a metric, for example, with BLEU. We can execute a weight adjustment by means of the downhill simplex method, where the function to minimize is $(1 - BLEU)$ and its parameters are the weights of the word graph. For each iteration of the downhill simplex algorithm, a N-best list is generated and merged with the N-best list of the last iteration. As the downhill simplex finishes, we will obtain an *extended* N-best list for each word graph.

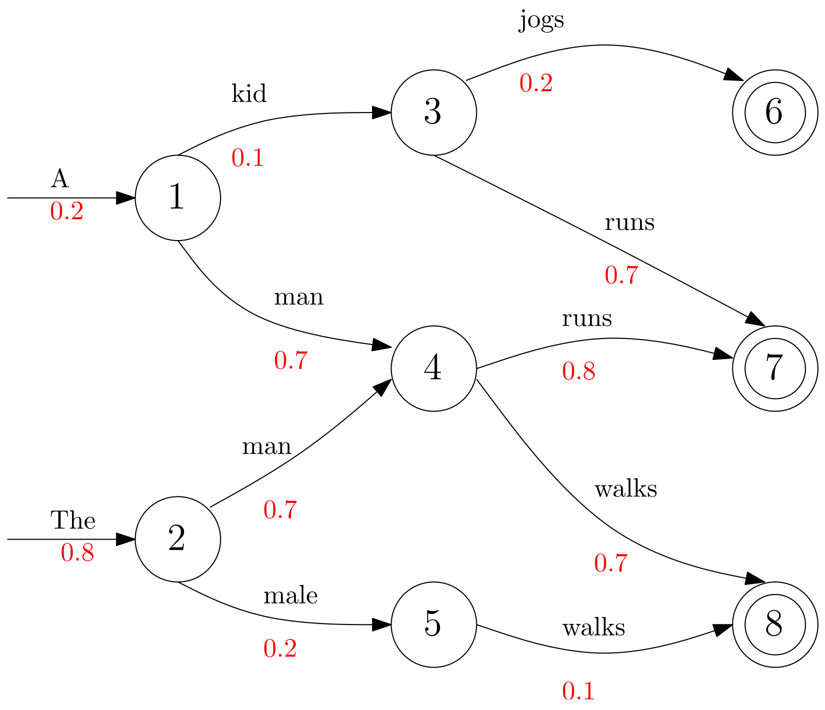


Figure 3.1: Word graph example for the source sentence `"El hombre corre"` using English as target language. From this word graph, we can determine that the best translation is: `The man runs`.

We have set the value of N to 200 so, in each iteration of the downhill simplex, the 200-best translations are computed and merged with the previous ones. We think that, even though this is not the optimal procedure, because potentially there are possible translations of the word graph not included in the N-best list, the majority of the probability mass of the word graph will be captured by the N-best list, hence the difference will be non-significant.

SRILM

SRILM [Sto02] is a toolkit which has been under development since 1995 by the *SRI Speech Technology and Research Laboratory.* This software has become a classical tool for language modeling. It has been applied in a wide variety of applications besides machine translation, such as speech recognition, tagging and segmentation, handwriting recognition or integrated development environments (IDEs) and language bindings. It implements many features, but for this work, we had basically used this tool for count and estimate *n*-gram language models and their interpolation with the recurrent neural network language model.

RNNLM Toolkit

This toolkit [MKD+11b], implements the above described language model based on recurrent neural networks (see Section 2.1.3). The software allows the creation, training, evaluation and usage of such models. It implements the training algorithm Backpropagation throught time (see Section 2.4.2). It also can be used together with SRILM to combine both language models.

3.1.2. Developed Experiment

We carried out 3 different tasks of increasing complexity: Tourist, Xerox and Europarl. Each one of this tasks has 3 separate corpora, namely training, development and test. For each one of the tasks, we followed a common procedure of machine learning: first, we trained our models with the training corpus. Next we adjusted their parameters for the development set and, finally, with this parameters, we tested their performance on the test corpus. Figure 3.2 shows a diagram of the executed experimentation.

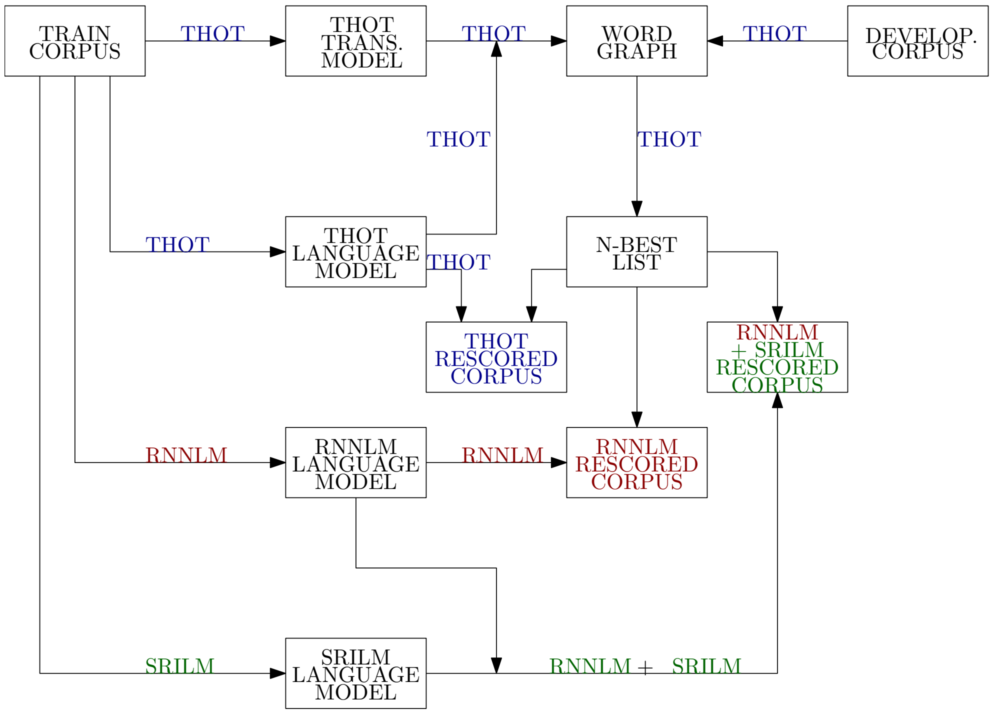


Figure 3.2: Schema for the performed experimentation. The labels on the edges indicate the toolkit used to generate the target element.

In this work, Thot has been used as baseline system. Since we are interested in the language model, we used the translation model provided by Thot for generating the word graphs and N-best lists for the development corpus. The rescoring of such sentences is performed by means of the recurrent neural network language model and its interpolation with the n-gram language model provided by SRILM. We use as main quality measure the BLEU score and all the weight adjustment task of the rescoring process is driven by this measure. Algorithm 2 shows the followed experimentation steps, for each task. This algorithm makes use of some auxiliary functions:

- $T(\mathcal{C})$: training of an n-gram language model for corpus C.
- $T_{\texttt{BPTT}}(\mathcal{C})$: training of a recurrent neural network language model with BPTT algorithm for corpus $\mathcal{C}$.
- $T_{\texttt{t}}(p(e), w, \mathcal{C})$: training of a translation model for corpus C using the language model $p(e)$ with weights w.
- $\texttt{word\_graph}((f_i, e_i), p(e), p(f|e))$: obtains the word graph for the pair of sentences (f_i, e_i), according the language model $p(e)$ and the translation model $p(f|e)$.
- $\texttt{N} - \texttt{best\_list}(\texttt{wg})$: obtains from the word graph `wg` its corresponding N-best list, as it is explained in Section 3.1.1.
- $\texttt{dhs}(f, p)$: applies for the function f and parameters p the downhill simplex optimization method, returning the optimal set of parameters.
- $\texttt{Reconstruct}(\texttt{NBL}, w)$: from the set `NBL` of N-best lists and the set of weights w, obtains the highest ranked sentence for each N-best list. Finally, all the best sentences are collected and form a new rescored corpus of size $|\texttt{NBL}|$.
- $\texttt{BLEU}(\mathcal{C}, \mathcal{C}_{ref})$: computes the BLEU score for the corpora $\mathcal{C}$ and $\mathcal{C}_{ref}$.
- $\texttt{PPL}(p(e), \mathcal{C})$: computes the perplexity for the language model $p(e)$ and the corpus $\mathcal{C}$.

Algorithm 2: Pseudocode for the experimentation process.

```
   Input: $\mathcal{X} = \{(f_1, e_1), ..., (f_n, e_n)\}$ (training corpus),
   $\mathcal{D} = \{(f_1^{'}, e_1^{'}), ..., (f_{n'}^{'}, e_{n'}^{'})\}$ (development corpus),
   $\mathcal{T} = \{(f_1^{''}, e_1^{''}), ..., (f_{n''}^{''}, e_{n''}^{''})\}$ (test corpus)
   Output: $\mathtt{BLEU}_n(\mathcal{D})$, $\mathtt{BLEU}_{\mathtt{RNNLM}}(\mathcal{D})$, $\mathtt{BLEU}_{\mathtt{inter}}(\mathcal{D})$,
   $\mathtt{BLEU}_n(\mathcal{T})$, $\mathtt{BLEU}_{\mathtt{RNNLM}}(\mathcal{T})$, $\mathtt{BLEU}_{\mathtt{inter}}(\mathcal{T})$
1  Auxiliar: $p_n(e)$ ($n$-gram language model),
2  $p_{\mathtt{RNNLM}}(e)$ (RNN language model),
3  $p_{\mathtt{inter}}(e)$(interpolation of language models),
4  $p(f|e)$ (translation model),
5  $w_m$ (parameters of model $m$),
6  $\lambda$ (interpolation weight)
7  begin
8      $p_n(e) := T(\mathcal{X})$;
9      $p_{\mathtt{RNNLM}}(e) := T_{\mathtt{BPTT}}(\mathcal{X})$;
10     $p_{\mathtt{inter}}(e) := \lambda \cdot p_{\mathtt{RNNLM}}(e) + (1 - \lambda) \cdot p_n(e)$ ;
11     $w_n := \mathtt{dhs}(PPL(p_n(e), \mathcal{D}), w_{p_n})$ ;
12     $p(f|e) := T_t(p_n(e), w_n, \mathcal{X})$;
13     foreach corpus $\in \{\mathcal{D}, \mathcal{T}\}$ do
14         foreach $(f_i, e_i) \in$ corpus do
15             $\mathtt{wg}_i$ := word_graph $((f_i, e_i), p_n(e), p(f|e))$;
16             $\mathtt{nbl}_i$ := $N$-best_list($\mathtt{wg}_i$);
17         foreach lm $\in \{n, \mathtt{RNNLM}, \mathtt{inter}\}$ do
18             NBL := $\emptyset$;
19             foreach $\mathtt{nbl}_i$ do
20                 forall the sentence $\in nbl_i$ do
21                     $p(e) := p_{\mathtt{lm}}$(sentence);      // Sentence rescoring
22                     NBL := NBL $\cup\, p(e)$ ;
23             if corpus $= \mathcal{D}$ then
24                 while $(\mathtt{dhs}((1 - \mathtt{BLEU}_{p_{\mathtt{lm}}}(\mathcal{D}', \mathcal{D})), w_{p_{\mathtt{lm}}}) \neq$ convergence)
                   do
25                     $\mathcal{D}'$ := Reconstruct(NBL, $w_{p_{\mathtt{lm}}}$);
26                     $w_{p_{\mathtt{lm}}} := \mathtt{dhs}((1 - \mathtt{BLEU}_{p_{\mathtt{lm}}}(\mathcal{D}', \mathcal{D})), w_{p_{\mathtt{lm}}})$;
27                 $\mathtt{BLEU}_{\mathtt{lm}}(\mathcal{D}) := \mathtt{BLEU}_{p_{\mathtt{lm}}}(\mathcal{D}', \mathcal{D})$
28             else
29                 $\mathcal{T}'$ := Reconstruct(NBL, $w_{p_{\mathtt{lm}}}$);
30                 $\mathtt{BLEU}_{\mathtt{lm}}(\mathcal{T}) := \mathtt{BLEU}_{\mathtt{lm}}(\mathcal{T}', \mathcal{T})$
31 end
```

For each task, we are going to present three main results: perplexity of the different estimated models, BLEU score for the development corpus and BLEU score for the test corpus. We performed a sampling between different size of the context which each language model handles. We collected results from 2-word contexts up to 5-word contexts.

The n-gram models provided by Thot implement the Jelinek-Mercer smoothing technique (see 2.7). According to [CG98], the smoothing technique which offers a best performance is Kneser-Ney smoothing technique (see 2.11), hence, by default we used this technique in our experiments. However, in [CG98] is shown that Witten-Bell smoothing technique has a good performance if the training corpus is larger, therefore we tested it too. Kneser-Ney smoothing performed better than the Witten-Bell smoothing in two of the three tasks, nevertheless, in the Xerox task, Witten-Bell smoothing obtained better results (see Section 3.3.2). The results shown in the following sections make use of the Kneser-Ney smoothing technique, unless otherwise stated.

To obtain the best network architecture, we tested many combinations of the network hyperparameters for each task, setting the backpropagation through time steps to 3. We computed perplexity for each one of the networks and we performed the interpolation between the neural network language model and a trigram language model. We selected as best network architecture that with a minimum perplexity. Then, we varied the backpropagation through time steps, from 2 up to 5.

The default interpolation performed in the experiments was between language models with the same order, i.e. we interpolated bigrams with recurrent neural networks trained with 2 steps back in time, trigrams with networks with 3 steps back in time, etc. Interpolations of higher order models were performed too: we enlarged the steps back in time with which the network was trained and we interpolated these new networks with the n-gram language model that achieved the best result for each task. The interpolation weight λ is task-dependent: for each task, we executed a scanning of this parameter, the perplexity of the resulting model for the λ value was evaluated and we selected the λ value which offered the lowest perplexity.

3.2. Tourist task

3.2.1. Overview

This was a toy task which contained sentences typically said by a tourist at the reception of a hotel. It is a very simple and small corpus used to build the rescoring system and test it easily. In this experiment, we used Spanish as source language, and we want to translate it to English. Training, development and test partitions were carried out by extracting randomly sentences from the corpus. Table 3.1 shows the different partitions size.

		Spanish	English
Training	Sentences	9 900	
	Running words	96 172	98 304
	Vocabulary	690	517
Development	Sentences	100	
	Running words	959	988
Test	Sentences	2 996	
	Running words	35 023	35 590

Table 3.1: Tourist corpora statistics: number of sentences, words and vocabulary size for each one of the three corpora: training, development and test, for both languages.

3.2.2. Results

We found that the best network architecture for this task had 50 hidden layer units and we set the number of classes to 200. We tested different number of classes, but the network accuracy did not improve, even using no classes. Since there were no improvements and the training and usage complexity was higher, we decided to set the classes parameter to 200.

Perplexity

Since this was a toy task, the perplexity was very low. Table 3.2 shows the perplexity values obtained by the difrerent language models, with different context sizes. As told above, the models involved in the interpolation had the same order. It is also reported the perplexity variation and variation percentage regarding the n-gram language model, which represents our baseline system. This variation is simply computed as:

$$Var(PPL_1, PPL2_2) = PPL_2 - PPL_1 \tag{3.1}$$

The variation percentage is computed as:

$$\%Var(PPL_1, PPL_2) = 100 \cdot \frac{PPL_2 - PPL_1}{PPL_1} \tag{3.2}$$

In all cases, we set as PPL_1 the value of the n-gram language model and as PPL_2 the value of the recurrent neural network language model.

Language model	**Order**	**Perplexity**	**Variation**	**% Variation**
N-gram	2	5.2	-	-
	3	3.5	-	-
	4	3.5	-	-
	5	3.5	-	-
Recurrent neural network	2	2.72	**-2.48**	**-47.7**
	3	2.73	-0.77	-22
	4	2.72	-0.78	-22.3
	5	2.36	-1.14	-33.6
Interpolation of models	2	3.00	-2.2	-42.3
	3	2.80	-0.7	-20
	4	2.76	-0.74	-21.1
	5	**2.07**	-1.43	-40.9

Table 3.2: Tourist task perplexities for the development and test corpora, variations and variation percentages regarding the n-gram baseline system with the same context size.

As we can see, even though the perplexity was very low, the neural networks language models were able to reduce it from a 22% up to a 47%. The interpolation between models had a similar performance than the solely use of neural networks. The interpolation factor was set to $\lambda = 0.7$. Such models reached a perplexity reduction greater than 40%.

BLEU

The BLEU scores of this task were extremely high: they were all over 86. Anyway, the recurrent neural network language model was able to overcome those results. It reached a top BLEU score for the development corpus of 91.68 and for the test corpus of 91.04. The neural network model improved the score of the n-gram language model in almost all scenarios - there is only one case where the BLEU remained constant. The results are shown in Table 3.3, where it is reported the BLEU score for each context size, the variations of the recurrent neural network language models regarding their corresponding n-gram language model and the variation percentage. These values are computed in a similar way than before, using now the BLEU score:

$$Var(BLEU_1, BLEU_2) = BLEU_2 - BLEU_1 \tag{3.3}$$

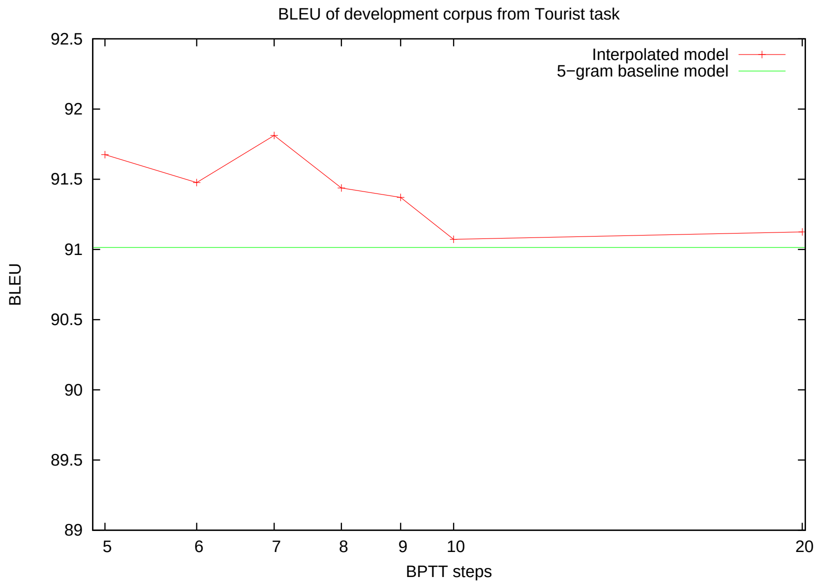
$$\%Var(BLEU_1, BLEU_2) = 100 \cdot \frac{BLEU_2 - BLEU_1}{BLEU_1} \tag{3.4}$$

As before, we set as $BLEU_1$ the one provided by the n-gram language model and as $BLEU_2$ that provided by the neural network language model or the interpolation between models.

		Development			Test		
Language model	Order	BLEU	Var.	%Var.	BLEU	Var.	%Var.
N-gram	2	89.64	-	-	86.98	-	-
	3	90.11	-	-	88.27	-	-
	4	91.01	-	-	89.39	-	-
	5	91.01	-	-	89.36	-	-
Recurrent neural network	2	91.19	**1.55**	**1.73**	**91.04**	**4.06**	**4.67**
	3	91.23	1.12	1.24	89.57	1.3	1.47
	4	91.01	0	0	89.85	0.46	0.38
	5	91.36	0.35	-	90.79	0.46	0.51
Interpolation of models	2	90.25	0.61	0.68	90.20	3.22	3.70
	3	91.11	1.00	1.11	89.40	1.13	1.28
	4	91.34	0.33	0.36	90.61	1.22	1.36
	5	**91.68**	0.67	0.73	90.28	0.92	1.03

Table 3.3: Tourist task BLEU scores for each language model and both corpora. It is also reported the variations of the neural-network-based language models with regard their corresponding n-gram language model, with the same context size.

Once we confirmed that the n-gram model interpolation with the RNNLM outperformed the baseline system, we decided to enlarge the size of the context computed by the neural network. For doing that, we rescored once again the best set of sentences (provided in this case by the 5-gram language model), variating the backpropagation through time steps. The results are shown in Figure 3.3. The best scores were provided by a context of 7 words. From here, if the context size is increased, the BLEU score is slightly reduced.



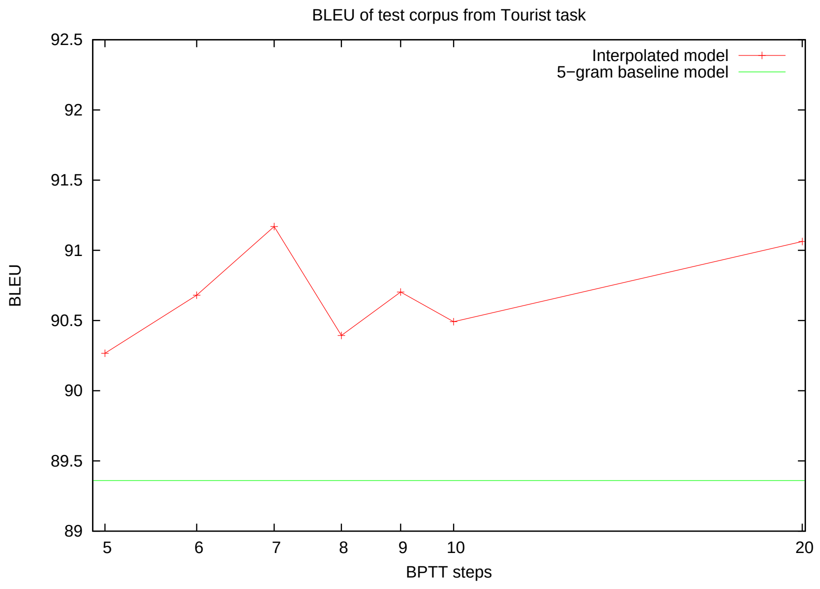


Figure 3.3: BLEU of the Tourist task, for development and test corpora, with larger contexts of the recurrent neural network language models. The interpolation was performed between the 5-gram language model and the neural network trained with the shown backpropagation through time steps.

3.3. Xerox task

3.3.1. Overview

This task consisted in the translation of sentences extracted from the user manuals of Xerox printers. The partitions were created in a similar way to the partitions of the tourist task. We translated from English to Spanish. Although it still consisted in a small task, it was much larger than the previous one. Table 3.4 shows the information related to the partitions.

		Spanish	English
Training	Sentences	55 675	
	Running words	746 955	661 639
	Vocabulary	17 148	14 510
Development	Sentences	1 012	
	Running words	15 952	14 279
Test	Sentences	1 125	
	Running words	10 084	8 348

Table 3.4: Xerox corpora statistics: number of sentences, words and vocabulary size for each one of the three corpora, training, development and test, and for both languages.

3.3.2. Results

We found that the best network architecture had 400 hidden layer units and 200 classes. Here the class clustering of the words was not indispensable: it was possible to train the network with no classes, although it was a really slow process and, as before, there was no significant improvement with respect to the use of classes.

Perplexity

Table 3.5 shows the perplexity of the different language models for this task, with different context sizes. As before, it is reported the variation of the perplexity with regard to the n-gram baseline system and the variation percentage, computed according Equations 3.1 and 3.2. As we can see, there was a big difference between the use of bigrams and higher order n-gram models. Anyway, the improvement of higher order n-grams reached a maximum with trigrams and neither 4-grams nor 5-grams were capable to reduce this perplexity. Meanwhile, recurrent neural network language models by themselves could only enhance the perplexity of contexts of size 2. If they gazed at more words, their perplexities were worse than the n-gram models ones. The greater perplexity reduction was reached by the interpolation between models: we found that the best result was obtained with an interpolation factor $\lambda = 0.3175$. With this technique, the perplexity of the task was reduced between a 21% and a 52%.

Language model	**Order**	**Perplexity**	**Variation**	**% Variation**
N-gram	2	46.9	-	-
	3	21.7	-	-
	4	21.7	-	-
	5	21.7	-	-
Recurrent neural network	2	38.59	-8.31	-17.71
	3	36.08	14.38	+66.26
	4	34.75	13.05	+60.01
	5	32.01	10.31	+47.51
Interpolation of models	2	33.32	**-13.58**	-28.96
	3	16.96	-4.74	-21.84
	4	12.15	-9.55	-42.40
	5	**10.41**	-11.29	**-52.03**

Table 3.5: Xerox task perplexities for the development and test corpora and the variation percentages with respect the n-gram baseline system with the same context size.

It is remarkable that the best perplexity scores were obtained using the Witten-Bell smoothing technique on the n-grams which are interpolated with the network. The differences in the perplexities of the models with both different smoothing techniques are shown in Figure 3.4 and they were around the 15% of the perplexity value.

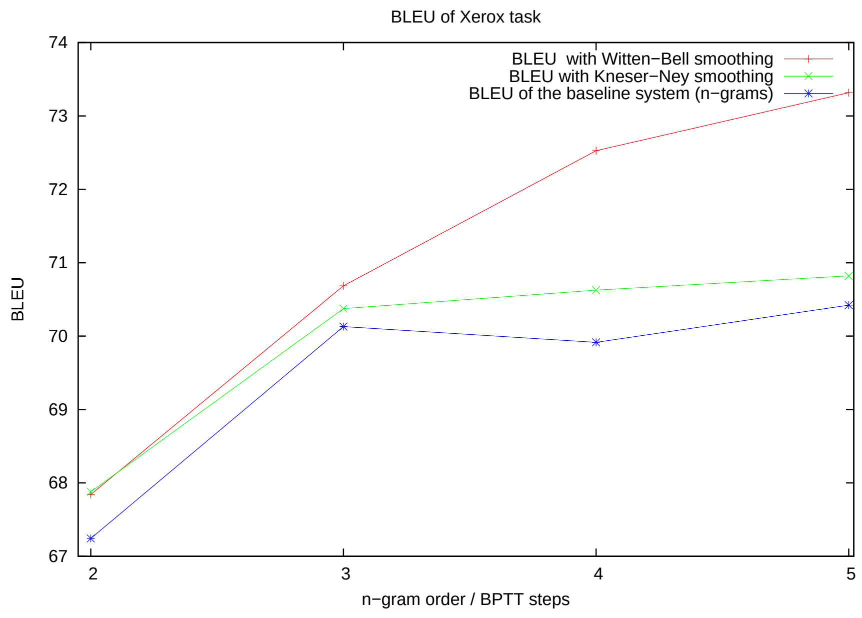


Figure 3.4: Perplexity of the development corpus from Xerox task using different smoothing techniques on n-gram language models which are interpolated with the recurrent neural network language model: Witten-Bell smoothing and Kneser-Ney smoothing.

BLEU

BLEU scores provided by the baseline system were not surpassed by the recurrent neural network model, excepting the 2-words context, where the improvement of the network model was 0.46 points in the development corpus and bigger than 1 point in the test corpus. The rest of experimentations produced poor results for the network system: the best values were obtained using a context of 5 words. Here, BLEU score of the test corpus was improved by 0.3 points. Considering that we came from a model with lower perplexity, it is positive that the network was able to overcome this gap and it performed an acceptable rescoring result.

Now, we will focus on the interpolation between models: this system had a lower perplexity than the n-gram model, thus it is reasonable to expect some improvement. With models of order 2 and 3, the network enhanced the n-gram scores, for both corpora, development and test. The gain of these models was around 0.6 points, except for the case of trigrams, where this value fell to 0.1. If we move to higher order models, the enhancement of the models was much higher: for the models of order 4 and 5 improvements of 2.61 and 2.89 points respectively were reached for the development set. By its side, there was also an important enhancement of the test set score: 0.92 in case of 4-order models and 1.54 for 5-order models. The results, BLEU score and the enhancement with respect the n-gram system for both development and test corpora, are shown in Table 3.6 and they are computed following again Equations 3.3 and 3.4.

		Development			Test		
Language model	Order	BLEU	Var.	%Var.	BLEU	Var.	%Var.
N-gram	2	67.24		-	53.11	-	-
	3	70.13	-	-	55.91	-	-
	4	69.92	-	-	56.72	-	-
	5	70.42	-	-	56.34	-	-
Recurrent neural network	2	67.70	0.46	0.68	54.13	1.02	1.92
	3	69.30	-0.83	-1.18	55.50	-0.41	-0.73
	4	69.62	-0.3	-0.43	56.38	-0.34	-0.60
	5	70.20	-0.22	-0.31	56.64	0.3	0.53
Interpolation of models	2	67.84	0.6	0.89	53.74	0.63	1.19
	3	70.68	0.55	0.78	56.01	0.1	0.17
	4	72.53	2.61	3.73	57.64	0.92	1.62
	5	**73.31**	**2.89**	**4.10**	**57.88**	**1.54**	**2.73**

Table 3.6: Xerox task BLEU scores for each language model and both corpora. It is also reported the variations of the neural-network-based language models with respect their corresponding n-gram language model, with the same context size.

As well as in the perplexity evaluation, the differences between the n-gram smoothing technique were tested. Using the BLEU metric, the Witten-Bell smoothing performed also better than the Kneser-Ney smoothing. The results for the development set of the different smoothing techniques are shown in Figure 3.5. It can be seen that, although there were no big differences in low order models, as we increase the order of the model, these differences were larger too, with a difference greater than 2 BLEU points in the case of a context of size 5.

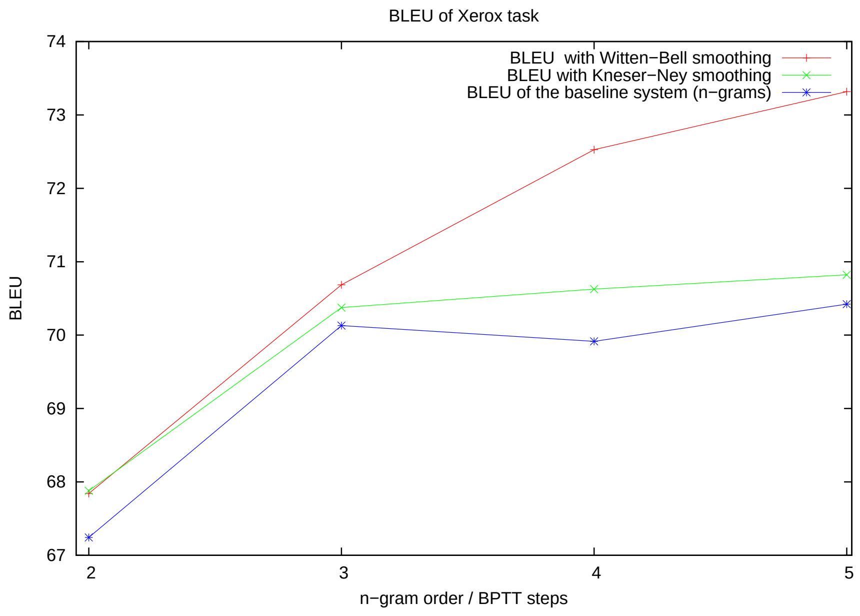
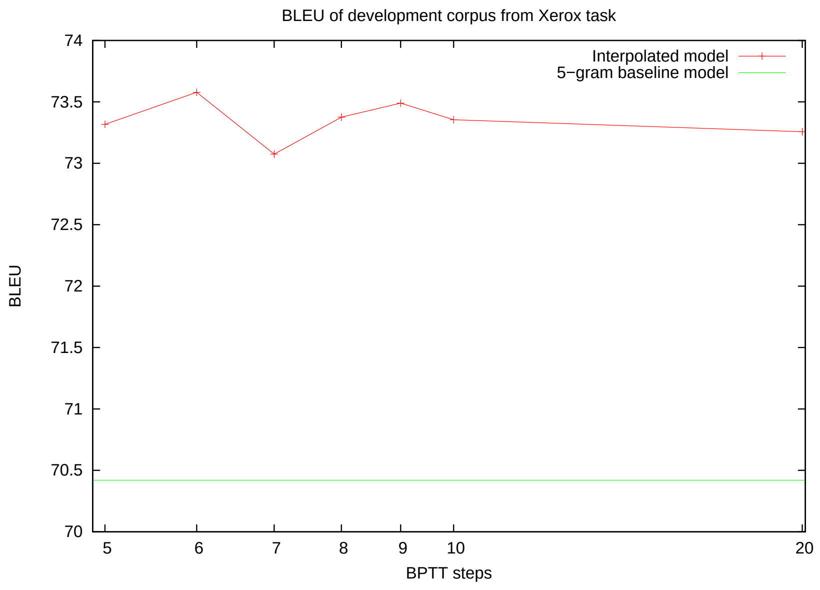


Figure 3.5: BLEU of the development set from the Xerox task using different smoothing techniques on the n-gram language models which are interpolated with the recurrent neural network language model: Witten-Bell smoothing and Kneser-Ney smoothing.

As before, we increased the context size of the RNN models. We used the experiment with the best result (models interpolation, n-gram order equal to 5), and, from here, we increased the context size. The results are shown in Figure 3.6. These results were similar to those obtained in the previous task: the score of the task rose slightly as we use greater context, but for a short time: from 6 steps back in time until 9, the results lightly outperformed the lower context results. From here, there was produced a little degradation of the score, as the context size was increased.



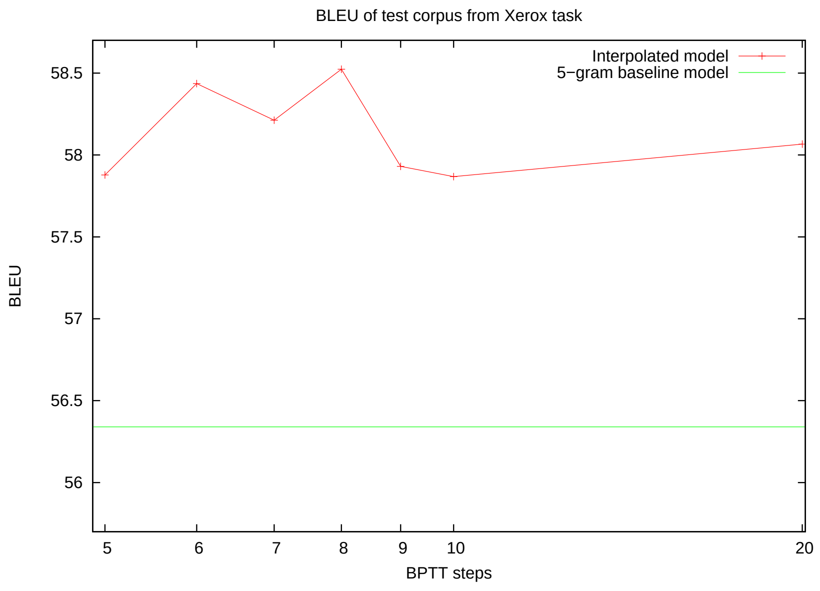


Figure 3.6: BLEU of the Xerox task, for the development and test corpora, with larger contexts of the recurrent neural network language models. The interpolation was performed between the 5-gram language model and the neural network trained with the corresponding backpropagation through time steps.

3.4. Europarl task

3.4.1. Overview

This parallel corpus was extracted from the proceedings of the European Parliament. This was a real task, significantly greater than both previous tasks. It is remarkable that the corpus used in the experiments was a bounded version of the complete one: the sentences were truncated to a size of 40 words. Despite this, the corpus was still large and complex. As in the previous task, we translated from English to Spanish. The development and test sets consisted in a collection of news articles related to the Parliament from the years 2012 and 2013, respectively. The statistics of this corpus can be seen in Table 3.7

		Spanish	English
Training	Sentences	1 547 596	
	Running words	34.0M	33.1M
	Vocabulary	146 292	96 745
Development	Sentences	3 003	
	Running words	78 814	72 954
Test	Sentences	3 000	
	Running words	70 383	64 808

Table 3.7: Europarl corpora statistics: number of sentences, words and vocabulary size for each one of the three corpora, training, development and test, and for both languages.

3.4.2. Results

The best network architecture found consisted in 400 hidden units and 400 classes. With a task of these dimensions, it was necessary to use the classes improvement in order to speed up the training process. Even so, this training process was slow: the average training time of the network drew near the 120 hours.

Perplexity

As it can be expected, the perplexity of the task was much greater than the previous ones. We found that the recurrent neural networks by themselves could not lower the n-gram perplexity. Except for the case of order

2, the rest of neural networks produced larger perplexity values than the n-gram language models of respective order. These differences were not large ($\pm 10\%$), but they did not produce any a priori sign of improvement of the translations.

Nevertheless, with the interpolation between n-gram models and the neural networks, larger enhancements were reached: the perplexity fell more than a 30% in all cases. We found that the λ value which resulted in a lower perplexity is $\lambda = 0.7$. The different perplexities of this task, for both development and test corpora, the perplexity variation and the variation percentage of the neural network language models with regard the n-gram baseline system, computed again according Equations 3.1 and 3.2, are shown in Table 3.8.

Language model	**Order**	**Perplexity**	**Variation**	**% Variation**
N-gram	2	296.4	-	-
	3	253.3	-	-
	4	243.8	-	-
	5	243.8	-	-
Recurrent neural network	2	265.10	-31.3	-10.56
	3	265.93	+12.63	+4.99
	4	265.54	+21.74	+8.92
	5	262.32	+18.72	+7.60
Interpolation of models	2	185.83	**-110.57**	**-37.30**
	3	174.86	-78.44	-30.97
	4	170.49	-73.71	-30.01
	5	**168.61**	-75.19	-33.49

Table 3.8: Europarl task perplexities for the development and test corpora and the variation percentages with respect the n-gram baseline system with the same context size.

BLEU

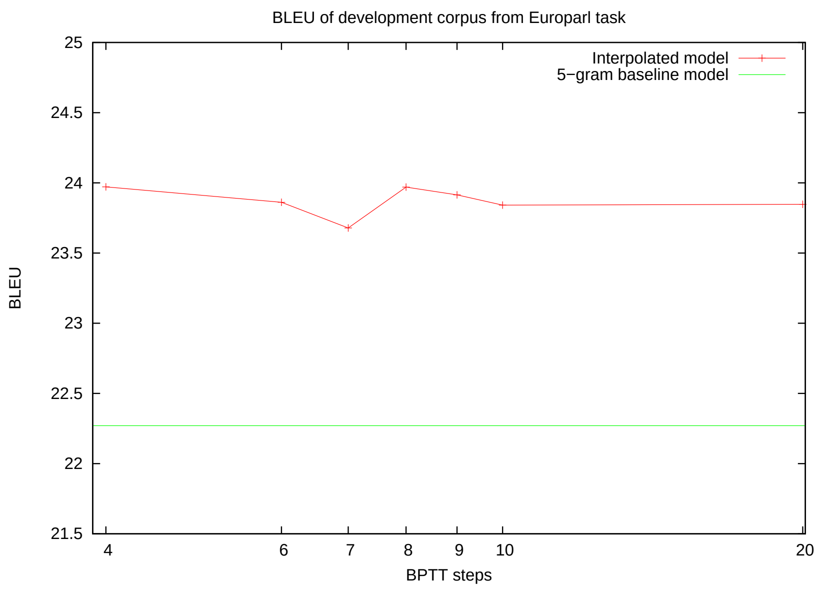
Despite that the perplexity of the recurrent neural network language models was higher than the n-gram perplexity, the first approach yielded better results than the latter, both in development and test experiments: there were improvements from 0.72 up to 1.74 BLEU points. The enhancements of the interpolation between models were even larger: they rise up to 1.82 points in the development experiment and 1.67 in the test experiment. Since the

BLEU scores were lower than in the previous experiments but the improvements provided by the recurrent neural network language model were similar (between 1 and 2 points), in this experiment the improvement percentages were greater: they rose up to 8.25% for the development corpus and up to 8.99% for the test set. The results of the experimentation are shown in Table 3.9. It is reported the BLEU score obtained by the different language models and the BLEU variation of the neural network language models with respect the n-gram laguage models and the percentage that this variation represents with respect the n-gram language model. As in previous sections, these results are computed following Equations 3.3 and 3.4.

		Development			Test		
Language model	Order	BLEU	Var.	%Var.	BLEU	Var.	%Var.
N-gram	2	18.62	-	-	15.59	-	-
	3	22.14	-	-	18.75	-	-
	4	22.27	-	-	18.83	-	-
	5	22.07	-	-	18.57	-	-
Recurrent neural network	2	19.53	0.91	4.88	16.35	0.76	4.87
	3	22.86	0.72	3.25	19.53	0.78	4.16
	4	23.69	1.42	6.37	20.08	1.25	6.64
	5	23.81	1.74	7.88	20.11	1.54	8.29
Interpolation of models	2	19.42	0.80	4.29	16.36	0.77	4.94
	3	23.26	1.12	5.06	19.93	1.18	6.29
	4	**23.97**	1.70	7.63	**20.35**	1.52	8.07
	5	23.89	**1.82**	**8.25**	20.24	**1.67**	**8.99**

Table 3.9: Europarl task BLEU scores for each language model and both corpora. It is also reported the variations of the neural-network-based language models with respect their corresponding n-gram language model, with the same context size.

As we increased the size of the context, we could see no further improvements. As Figure 3.7 shows, for the development set, there was a minimum rise of the BLEU for 8 steps back in time. The result for the test set, with 8 steps back in time, was lower than the result of a context of 4 steps. The best test result was obtained with 7 steps, but in this case, the performance for the development set fell from the best result. In conclusion, we could determine that the best context size for this task is 4 steps back in time. It should be mentioned that, as before, these results were computed based on the best previous result, i.e. using the recurrent netural network language model interpolated with a 4-gram language model.



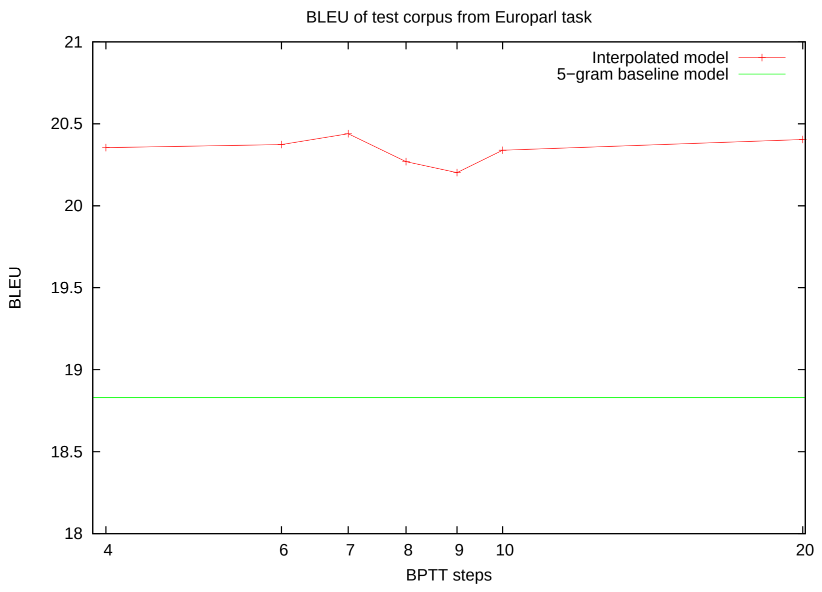


Figure 3.7: Europarl task BLEU scores for each language model and both corpora. It is also reported the variations of the neural-network-based language models with regard their corresponding n-gram language model, with the same context size.

Chapter 4

Conclusions and Future Work

4.1. Conclusions

We carried out a rescoring experimentation with satisfactory results: using the recurrent neural network language model solely, occurred a significant improvement in two of the three tasks, Tourist and Europarl. These models overcome the *n*-gram language models and offer, in these two tasks an average improvement of the BLEU score of 1.18 points. In the medium sized task, Xerox, this model lowered its performance, being unable to generally reduce the perplexity and increase the BLEU of the task.

The results of the interpolation between this model and *n*-gram language models were even better: BLEU and per plexity were enhanced in all cases, yielding the maximum improvements reached in this experimentation. Excepting the first task, where the BLEU score is extremely high (using both *n*-gram and recurrent neural network models) and the improvements were not excessively large, the enhancements of the interpolation between models were greater than 2.5 points in the case of the Xerox task and greater than 1.5 in the Europarl task.

We also tested the enlargement of the size of the context of the neural network language models. The results were similar to the shown in [Mik12], where the perplexity is compared. Although in this work the main metric used is BLEU, we can appreciate a relationship between both results. Our best results were achieved using contexts of 4-5 words. The system could enhance a little its performance using contexts of 7-8 words, but, from there, no further improvements were obtained. It can be concluded that context of 4-5 words should be enough to obtain good results.

It is remarkable that the improvements, since we used contexts of the same size for our experiments, are mostly due to the capability of the network to capture similarity between words. However, the neural network language model is able to manage larger contexts. We could have taken advantage of that if the N-best lists used in the rescoring process, were generated using higher order models. It is likely that this could lead to even better results. In spite of that, the theoretical advantages of the neural network language models were confirmed empirically with these experiments; giving to the future researches done in this field more guarantees.

4.2. Future work

4.2.1. Integration with Thot

The experiments carried out, showed that the neural network language models improved the quality of the translations provided by Thot. Hence, the next natural step would be to integrate this language model into the Thot decoder. In this work, we performed the integration via N-best lists rescoring. In the future, the neural network language model should be coupled into the Thot decoder, offering its features in translation runtime. In addition, it should preserve the Thot feature of estimating its models incrementally. It is planned to develop the integration in a near future.

4.2.2. Parallelization of the Recurrent Neural Network Toolkit

The training of the recurrent neural network is very costly, for instance, the training of the network used in the Europarl task with 8 steps back in time lasted for almost 160 hours. As it is pointed in [Mik12], it is necessary to parallelize the implementation of the neural network toolkit. Bengio et al. proposed a parallel implementation for their feedforward neural network model [BDVC03], hence the future work would consist in extending this parallel training to the recurrent neural network architecture and the backpropagation through time learning algorithm.

4.2.3. Bidirectional Recurrent Neural Networks

The bidirectional recurrent neural networks are an extension of recurrent neural networks, which try to take advantage of all available input data, both past and future information [SP97]. For achieving this goal, the state neurons of the recurrent neural network are split in two parts: one which is responsible of the positive time direction and another which is responsible of the negative time direction. The training algorithms of these networks are the same as the regular RNN, with minor modifications. Thus, the idea of this approach is to predict the next word based on previous words and the next words to come.

The utility of this bidirectional recurrent neural networks has been proved in the handwritten text recognition taks [GLF+09], where this approach overcomes the performance of classical approaches, such as Hidden Markov Models. It seems reasonable to think that this result could be extended to the machine translation task: if the predictions of the language model are based both in past and future history, it is likely that these predictions become more accurate than the ones which are based only on the past history.

Chapter 5

Bibliography

[BDVC03] Y. Bengio, R. Ducharme, P. Vincent, and C.Jauvin. A neural probabilistic language model. *Machine Learning Research*, 3:1137–1155, February 2003.

[BM00] E. Brill and R. C. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, Hong Kong, 2000.

[BPP$^+$92] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, J. C. Lai, and R. L. Mercer. An estimate of an upper bound for the entropy of english. *Comput. Linguist.*, 18(1):31–40, March 1992.

[BPPM93] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

[Cas14] F. Casacuberta. Advances in neural networks: Recurrent neural networks. University lecture: http://users.dsic.upv.es/~fcn/Students/ann/t3ann2p.pdf, 2014.

[CG98] F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Computer Science Group, Harvard U., Cambridge, MA, 1998.

[Fed08] M. Federico. Statistical machine translation. part iv: Mt evaluation. University lecture:

http://medialab.di.unipi.it/web/SMT/SMT-0508-part-4-pp.pdf, 2008.

[GLF+09] A. Graves, M. Liwicki, S. Fern'andez, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, May 2009.

[Goo01] J. Goodman. Classes for fast maximum entropy training. In *Proceedings ICASSP*, pages 561–564, 2001.

[Hut95] W.J. Hutchins. *Concise history of the language sciences: from the Sumerians to the cognitivists*, chapter Machine Translation: A brief history, pages 431–445. Oxford: Pergamon Press, 1995.

[Kni99] K. Knight. *A Statistical MT Tutorial Workbook*. Unpublished, 1999.

[MDP+11] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Èernocký. Strategies for training large scale neural network language models. In *Proceedings of the Automatic Speech Recognition and Understanding: Language Modeling and ASR Systems*, pages 196–201, Waikoloa, HI, 2011.

[Mik12] T. Mikolov. *Statistical Language Models based on Neural Networks*. PhD thesis, Brno University of Technology, 2012.

[MKB+11] T. Mikolov, S. Kombrink, L. Burget, J. Èernocký, and S. Khudanpur. Extensions of recurrent neural network language model. In *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 5528–5531, Prague, Czech Republic, 2011.

[MKD+11a] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and J. Èernocký. Empirical evaluation and combination of advanced language modeling techniques. In *Proceedings of the 12th Annual Conference of the International Speech Communication Association*, pages 612–615, Florence, Italy, 2011.

[MKD+11b] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and J. Èernocký. Rnnlm - recurrent neural network language modeling toolkit. In *Automatic Speech Recognition and Understanding: Demo Session*, Waikoloa, HI, 2011.

[NM65] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[OM11] D. Ortiz-Martínez. *Advances in Fully-Automatic and Interactive Phrase-Based Statistical Machine Translation.* PhD thesis, Universidad Politécnica de Valencia, 2011.

[OMC14] D. Ortiz-Martínez and F. Casacuberta. The new thot toolkit for fully automatic and interactive statistical machine translation. In *14th Annual Meeting of the European Association for Computational Linguistics: System Demonstrations*, Gothenburg, Sweden, April 2014.

[PRWZ02] K. Papieni, S. Roukos, T. Ward, and W. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, PE, July 2002.

[SDS+06] M. Snover, B. Derr, R. Schwartz, L. Micciulla, and J. Makhoul. A study of translation edit rate with targeted human annotation. In *Conferences of the Association for Machine Translation in the Americas*, pages 223–231, Cambridge, MA, 2006.

[SP97] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, November 1997.

[Sto02] Andreas Stolcke. Srilm - an extensible language modeling toolkit. pages 901–904, 2002.

[ZON02] R. Zens, F. J. Och, and H. Ney. Phrase-based statistical machine translation. In *Advances in artificial intelligence. 25. Annual German Conference on AI*, volume 2479 of *LNCS*, pages 18–32. Springer Verlag, September 2002.