Proximity-Based Error Correction in a Speech Recognition Transcript

Célia BEL

Ecole polytechnique Féderale de Lausanne
Electrical and Electronic Section

MASTER THESIS

Supervisor:
Dr. Andrzej DRYGAJLO
EPFL/LTS5

Advisor:
Dr. Wilhelm HAGG
SONY/SSG Sony

29/02/2016

Speech and Sound Group, Europeean Technology Center
Sony Deutschland GmbH, Stuttgart, Germany

Abstract

In speech recognition systems, speaker adaptation methods are often used to improve performances. Usually, they require gold transcriptions of the adaptation data, which are often manually made. Since this is time-consumming, we would like to generate them automatically, using for instance decodings from an independent Automatic Speech Recognition (ASR) system. However, results may contain errors. Therefore, the purpose of this work is to correct those errors. Among the different methods employed in the literature, we investigate an approach based on Recurrent Neural Networks (RNN) language model. Its basic idea consists in adapting a language model to each specific transcript.

More precisely, three different kinds of experiments are conducted. In the first one, we do not make use of any external data. A language model is elaborated using the transcript itself. In the second one, we train a language model on external data taken from the World Wide Web. To that end, we extract keywords from the transcript and select the best retrieved pages. Finally, we combine both methods in a last experiment: first with a simple interpolation, and then using the Kullback-Leibler divergence regularization. The selective web-data augmented language model and its interpolation with the book-specific model result in the best performances. In terms of Word Error Rate (WER), a reduction of more than 1% is obtained.

Keywords: language modeling, N-best rescoring, recurrent neural network, keyword extraction, BLEU score, Kullback-Leibler divergence.

Acknowledgements

I would like to express my sincere gratitude to my advisor Dr. Wilhelm Hagg for his constant support and wise advice all along these six months. His guidance and brilliant expertise really helped me for this work, and I have learnt a lot through discussions with him. I am grateful for all his patience, clear explanations and useful comments.

I would like to thank each member of the Speech and Sound Group (SSG) at Sony. Working in such a motivating and nice atmosphere was really rewarding and brought me a lot. I especially would like to thank Dr. Thomas Kemp for encouraging scientific curiosity and making me want to never stop learning.

I would like to express my deep gratitude to my supervisor at EPFL, Prof. Andrzej Drygajlo, for supervising my thesis, bringing me support, and initiating me to the Speech Recognition field. I am also grateful to EPFL's quality of education and teaching.

Finally, I would like to thank my parents and my sisters for being such a great source of love, and always giving me the willingness to continue and persevere.

Lausanne, February 2016

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context and problem description

In speech recognition standard procedure, systems are trained on a large amount of audio data, for a variety of different speakers. In order to improve performances, more specific speaker adaptation techniques are used. If supervised, those methods require the word level transcription of the adaptation data, which is often manually done. Ideally, this transcript could also be generated automatically, by directly using decodings from the independent Automatic Speech Recognition ASR system. However, the obtained transcription may contain errors. Thus, the purpose of this thesis is to detect and correct these errors.

A possible application scenario would be the following one: let us consider an ASR system in a phone device, which has to be adapted to his specific user. First of all, several hours of phone calls are recorded. Then, transcripts of these calls are generated and corrected. Finally, results are used as input for further processes, such as speaker adaptation of the acoustic model.

1.2 State of the Art

First of all, error correction should be distinguished from error detection, since it also implies candidate generation of possible corrections and ranking. Then, it is obvious that the error's type will influence the choice of a correction method. Kukich establishes in [12] a hierarchy of errors based on 5 levels: a lexical level, a syntactic level, a semantic level, a discourse level and a pragmatic level.

The lexical level concerns non-word errors, meaning words that do not exist. Their detection consists in using a word list or a dictionary: if the word do not appear in the list then, it is considered to be an error. Lexical level errors are corrected using various techniques, enumerated among others in [18]. For instance, it can be based on the minimum edit distance, which consists in the minimum operations (insertions, deletions, substitutions) required to transform one string to another. Similarity key techniques are also mentioned: the core idea is

to map every string to a key, such that similar strings will have similar keys. Thus, misspelled words give direct access to similar words which are possible candidates for correction.

The 4 other levels concern real-word errors: these are existing words improperly used in a given context. Real-word errors include typos (*from* becomes *form*), phonetic lapses (*there* becomes *they're*), syntactic mistakes (*eat* becomes *eats*), insertions and deletions. According to Kukich, real-word errors represent 25%-40% of spelling errors. A dictionary lookup can't detect those kind of errors. They require information from the surrounding words and context. That's why real-word errors are also called contextual spelling errors. More precisely: syntactic errors correspond to words grammatically incorrect, semantic errors refer to words incorrect in their meaning and discourse errors indicate text incoherence. For instance, the sentence *"There are three fruits on the table : one apple and one pear"*, contains a discourse error. Finally, pragmatic errors refer to sentences that are false, meaning that affirms something wrong. For instance, the sentence *"Paris is the capital city of Germany"* contains a pragmatic error. In this thesis, discourse and pragmatic errors are out of scope, since they require a high level of knowledge.

Syntactic errors can be dealt with taggers and parsers. Taggers associate each word with its type (noun, verb, adjective...) and parsers derive the syntactic structure of a sentence. The final goal is to associate a likelihood to a grammatical construct. A low likelihood implies a potential syntactic error. For instance, Atwell and Elliot adapted in [3] CLAWS part-of-speech tagger (developed in 1980 at Lancaster) to detect syntactic errors.

Semantic errors (or malapropism as defined in [9]) can't be handled with purely syntactic approaches. Alternatively, Pedler mentions in [23] methods based on "confusion sets". As defined in [11], confusion sets are "sets of words which are frequently misspelled or misused". For instance, *{they're, their, there}* forms a confusion set. Each time a word that belongs to a confusion set is encountered, rules are used to determine whether or not the word corresponds to the context. Those rules can be semantic or syntactic. For instance, methods based on Bayesian classifiers have been developed for that purpose (cf [7]). Also worth mentioning is the study conducted in 1991 by Mays and Al ([13]). In this work, they propose to group words together that differ by just one letter to form confusion sets ([13]). Then, they capture semantic and syntactic information using trigram (sequence of three words) probabilities. High frequent collocations are in that manner well corrected, but this method still can't catch all different word combinations.

Other approaches attempt to represent a semantic context. For this challenging task, knowledge of word associations is useful. In [9], Hirst introduces the notions of "linguistic cohesion" (repetitions, semantically related words...) and "lexical chains" (all words related to a same concept and which can be found through all the text). Hirst's intuition is that malapropism can't be linked to lexical chains and he explored this idea with St Onge in [10]. However, results were limited due to an inappropriate way of elaborating lexical chains. Some years after, in 2005, Hirst and Budanitsky developed a new algorithm, which this time considers text as a

"whole bag of words", and not just lexical chains [9]. This algorithm involves two mechanisms: first, given a word, it requires a system that returns a list of all plausible misspellings (insertion, deletion, transposition...). Then, it requires a measure quantifying the semantic relation between two words. This last point is very challenging: indeed, a semantic measure needs to catch all word's dependencies, but should also be able to reveal malapropisms. Hirst and Budanitsky experiment five different measures, which provide widely different performances (recall of 23-50% and precision of 18-25%). A need for a deep understanding of semantic relatedness is thus required.

In 1997, an attempt to combine confusion sets and Latent Semantic Analysis (LSA) is proposed in [11]. Usually, LSA requires a matrix representing occurrences of words in specific documents. Each column of the matrix is assigned to a document, and each line to a word. Then, LSA transcribes the relations between words, the "concepts" they carry and documents. In [11], Jones adapts this method and builds a LSA matrix for each confusion set. In this matrix, each column is associated with a sentence (and not a document). Then, for each word in the confusion set, LSA transcribes its consistency with the different sentences the word can be found in. In that way, a bad match reveals an error. As a result, Jones reported an average improvement over 18 confusion sets of 14% upon the baseline performance (achieved by a system ignoring context and just predicting the most frequent confusion word in the training corpus).

All those different methods use a restrictive amount of information, and are based on the selection of confusion sets. Recently, it appears that other approaches based on neural networks perform very well. Indeed, neural networks provide a wider and deeper representation of a semantic content. For instance, in a 2014 study [4], a RNNLM is built to catch long-term dependencies within and across utterances. By exploiting additionally a Deep Neural Network (DNN) ASR with a complementary Gaussian Mixture Model (GMM) ASR system, a 38% relative reduction in P(miss) (probability of missing an error) is observed at a 10% false alarm compared to the baseline (DNN ASR). Neural networks turn out to be powerful also for low resource language modeling, as shown in [6] (2014). With a training set of 33k words and a vocabulary size of 3312 words, a WER reduction of 0.7 is achieved by interpolating a modified Kneyser-Ney smoothed trigram with a feed-forward neural network. Some studies based on Recurrent Neural Networks also show some interesting results. In 2010, Mikolov reported a 18% WER reduction on the Wall Street Journal speech recognition task [15]. Thus, RNNLM seem very promising techniques to perform language modeling. Worth mentioning is however the long training time required for such kind of methods.

1.3 Objectives of the proposed work

Given a transcript generated by an ASR system, our objective is to reduce its Word Error Rate (WER) as much as possible. This will allow to perform further tasks, such as speaker adaptation.

In this work, we make the following assumptions: first of all, the transcript generated by the independent ASR system is quite short (between 15000 and 50000 words). Then, lexical errors are assumed to be solved. Finally, the transcript's content is clearly defined (no random sentences).

Intuitively, if the transcript is about the topic "navy", the word "ship" is more likely to appear than its homophone "sheep". Consequently, in this particular case, a good language model is a model that will attribute a higher probability to "ship" than to "sheep". In other words, it will make contextual words pop up. Thus, the main idea followed in this thesis is to use a language model specific to the transcript, i.e. to the user domain (also called target-specific domain).

By extension, given an audio utterance, scores can be attributed to its different hypotheses of transcription. By using the target specific language model, small scores are given to false hypotheses, whereas high scores are attributed to correct ones. Consequently, this method performs error detection as well as error correction.

However, one question arises : how to build this target-specific language model? One idea is to create it directly from the erroneous transcript. Of course, errors may be learned. But, for instance, if one word appears 10 times in its correct version, and only 2 times as an error, its correct version has still a higher weight. Thereafter, the 2 erroneous versions may be corrected.

An other idea is to perform topic modeling, that is to identify the transcript's main topic. Then external data can be added with an adequate topic oriented WEB-search, and a language model can be trained on them.

In the state of the art, neural networks have been applied successfully for language modeling tasks. Besides, as we focus here on context characterization, RNN turn out to be the best solution, since they don't limit the history information and catch long-term dependencies.

The solution proposed in this thesis is the following: by using a RNN, a language model adapted to the transcript is established. This model will then be used to identify errors and perform corrections (see 1.1).

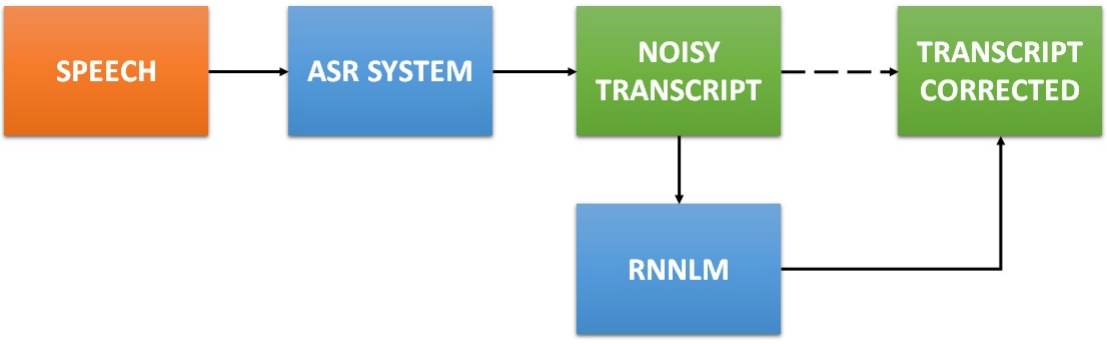


Figure 1.1 – System's Pipeline

1.4 Report outline

The rest of the report is organized as follows: Chapter 2 is dedicated to the theoretical background. Chapter 3 focuses on the toolkits used for implementation. In Chapter 4, the baseline system and its results are described. Chapter 5 presents a more advanced system based on web-data augmentation. Chapter 6 introduces improvement suggestions for this last system. Finally, Chapter 7 summarizes the results and makes propositions for further work.

2 Theoretical Background

This chapter focuses on different language modeling tasks. Indeed, with a good representation of the language, errors can be more easily found and corrected. Main references for this chapter can be found in [2] and [19].

2.1 Language Model - Description

The main goal of speech recognition is to transcribe a continuous speech signal into symbols. This requires the use of acoustic information (pronunciation, noise...) as well as prior information concerning the language. Consequently, speech recognizers mainly involve two models: the acoustic model and the language model.

From a mathematical point of view, the problem is to find the word sequence $\hat{\mathbf{W}}$, which is more likely to be produced from acoustic evidence $\mathbf{X}$

$$P(\hat{\mathbf{W}}|\mathbf{X}) = \underset{\mathbf{W}}{argmax} P(\mathbf{W}|\mathbf{X}) \tag{2.1}$$

Using the Bayesian rule, this equation can be transformed into:

$$P(\hat{\mathbf{W}}|\mathbf{X}) = \frac{P(\mathbf{X}|\mathbf{W}) * P(\mathbf{W})}{P(\mathbf{X})} \tag{2.2}$$

The acoustic model part corresponds here to $P(\mathbf{X}|\mathbf{W})$ and the language model part to $P(\mathbf{W})$.

Concretely, the acoustic model is often established using Hidden Markov Model (HMM) and the language model based on N-grams or neural networks. Since this master thesis focuses more on the language modeling task, this part will be described in more detail.

A language model assigns a probability to a sentence. For instance, in the context of spelling correction, this probabilistic model will put more emphasis on correct sentences than on false

ones:

P("Let us not exaggerate") > P("Lettuce not exaggerate")

From a mathematical point of view, since a sentence is basically a sequence of words, this equation can be rewritten in the following manner:

$$P(\mathbf{W}) = P(\mathbf{w_1}, \mathbf{w_2}, ...\mathbf{w_n}) = P(\mathbf{w_n}|\mathbf{w_{n-1}}, \mathbf{w_{n-2}}, ..., \mathbf{w_1}) \tag{2.3}$$

Then using the Chain Rule:

$$P(\mathbf{W}) = P(\mathbf{w_1}) * P(\mathbf{w_2}|\mathbf{w_1}) * ... * P(\mathbf{w_n}|\mathbf{w_{n-1}}, \mathbf{w_{n-2}}, ..., \mathbf{w_1}) = \prod_i P(\mathbf{w_i}|\mathbf{w_1}, \mathbf{w_2}...\mathbf{w_{i-1}}) \tag{2.4}$$

$P(\mathbf{w_i}|\mathbf{w_1}, \mathbf{w_2}...\mathbf{w_{i-1}})$ can be interpreted as the probability of the next upcoming word.

2.2 Types of Language Models

2.2.1 N-gram

Since the previous equation is hardly computable in practice, it is preferable to rewrite it, using the Markov assumption, as follows:

$$P(\mathbf{W}) \simeq \prod_i P(\mathbf{w_i}|\mathbf{w_{i-k}}...\mathbf{w_{i-1}}) \tag{2.5}$$

The idea is to limit the history to k previous words. This language model is called a N-gram. The most simple N-gram form is the unigram, which simply takes into account the occurence frequency of each word in the language, without any notion of context:

$$P(\mathbf{W}) \simeq \prod_i P(\mathbf{w_i}) \tag{2.6}$$

Bigram gives the probability of appearance of a word, knowing the previous word:

$$P(\mathbf{W}) \simeq \prod_i P(\mathbf{w_i}|\mathbf{w_{i-1}}) \tag{2.7}$$

Concretely, it is computed through a basic frequency count:

$$P(\mathbf{w_i}|\mathbf{w_{i-1}}) = \frac{count(\mathbf{w_{i-1}}, \mathbf{w_i})}{count(\mathbf{w_{i-1}})} \tag{2.8}$$

Trigram, 4-gram... are then defined in a similar fashion. The frequency count is rewritten in the following way:

$$P(\mathbf{w_i}|\mathbf{w_{i-k}}...\mathbf{w_{i-1}}) = \frac{count(\mathbf{w_{i-k}}, ..., \mathbf{w_{i-1}}, \mathbf{w_i})}{count(\mathbf{w_{i-k}}, ..., \mathbf{w_{i-1}})} \tag{2.9}$$

N-grams (and more specifically trigrams and 5-grams) are the most commonly used language models. However, they are subject to several limitations. First, they depend on the history length N: the longer, the better. Nevertheless, it should be noted that even high N-gram orders can't catch very long dependencies. Besides, they are strongly related to the genre of the training set. A language model that transcribes Shakespeare's style is different from a language model used for the newspapers. Finally, N-grams are also sensitive to data sparsity. Indeed, N-grams present in the test set may be absent from the training set, and consequently assigned to a zero probability, which will lead to poor results.

One particular case of data sparsity is the one of unigrams present in test set and absent from the training set. Those words are called OOV words, or simply unknown words. One way of dealing with the problem of unknown words is to define an open vocabulary: each word that does not belong to this vocabulary is mapped to the token <unk>. This "artificial" word <unk> is usually included in the training data, in order to estimate its probability.

For higher order of N-grams, a possible approach to deal with data sparsity is smoothing. The purpose of smoothing is to avoid to attribute zero probability to unseen events. One smoothing method consists in adding 1 to all N-gram counts (Laplace smoothing). However, this method is never used in practice, since it moves too much probability mass to the zero cases.

Another way to proceed is to use also the N-1, N-2 ... uni-grams. Either missing N-grams are replaced with the N-1 grams and so on (backoff system), or an interpolation of all lower order terms is performed. For instance, for a trigram system:

$$\hat{P}(\mathbf{w_i}|\mathbf{w_{i-2}}\mathbf{w_{i-1}}) = \lambda_1 * P(\mathbf{w_i}|\mathbf{w_{i-2}}\mathbf{w_{i-1}}) + \lambda_2 * P(\mathbf{w_i}|\mathbf{w_{i-1}}) + \lambda_3 * P(\mathbf{w_i})$$

$$\sum_i \lambda i = 1 \tag{2.10}$$

In the backoff system case, the point is to keep a correct probability distribution. To this purpose, probabilities of the higher N-gram orders are discounted to save some probability mass for the lower ones. One method used is called the Kneser-Ney algorithm (see [2]), which will not be explained in detail in this thesis. Compared to the simple backoff system, the basic idea in this algorithm is to introduce a probability of "continuation", which takes into account the different number of contexts (i.e. bigrams) in which a word occurs. In this method, unknown words can be added as a regular vocabulary entry, with a probability lower than $\frac{1}{|V|}$.

2.2.2 Neural Network

As mentioned in previous section, the higher the order of an N-gram, the better. However, it also means an increase of possible N-grams to consider. Thus, N-grams are limited since they can't catch long dependencies in history and become quickly computationally expensive.

To overcome this problem, an approach is to change the representation of the history. For instance, a context can be represented by a vector of N-1 words, but also by the projection of this vector in a lower dimensional space. This method is used for instance by Bengio in [5], where he builds a neural model as represented in 2.1 [1]

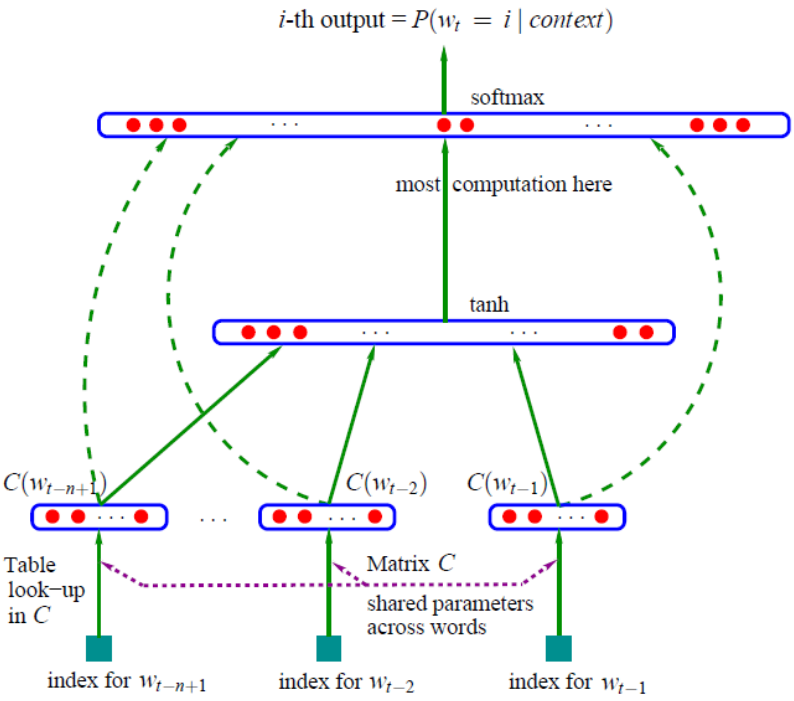


Figure 2.1 – Neural Model of Bengio

[1] Picture taken from [5]

Bengio's idea is to project the history of N-1 previous words (each word with a dimension corresponding to the vocabulary size) into a lower dimensional space, through a projection matrix C. Then, these new features feed a neural network (feed-forward or recurrent). Finally, the output gives a prediction for the next word.

Nevertheless, this method still limits the history model to the N-1 previous words. The RNNLM implemented by Mikolov [17] overcomes this problem. In this model, an effective representation of the history is learnt during training, using the state of the hidden layer. On the one hand, this layer is supposed to represent the whole "history" of the word (and not just the N-1 previous words). On the other hand, a reduced representation of the feature vector is still used in this model. Since we choose to use Mikolov's RNNLM in this master thesis, it will be explained in more detail in the next chapter.

2.3 Evaluation and Metrics

Language models are evaluated using two main metrics: the perplexity and the Word Error Rate WER.

2.3.1 Perplexity

The perplexity of a word sequence is the inverse probability of this sequence, normalized by the number of words. Given a sequence of words $W = w_1 w_2 ... w_N$, we have:

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 w_N)}} \tag{2.11}$$

Using the Chain Rule:

$$\begin{aligned} PP(W) &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 w_2 w_{i-1})}} \\ &= 2^{-\frac{1}{N}\Sigma_{i=1}^{N} log_2 P(w_i | w_1 w_2 ... w_{i-1})} \end{aligned} \tag{2.12}$$

This last expression can be interpreted in terms of cross-entropy. Indeed, cross-entropy has the following definition:

$$H(p,P) = -\sum_{w} p(w_i) log_2 P(w_i|w_1 w_2 w_{i-1}) \tag{2.13}$$

where $p(w)$ represents the true distribution of words in the language model. Since this probability is unknown, it has to be approximated. Then, the cross entropy of a sequence of words W can be expressed as follows:

$$H(W) = -\frac{1}{N}\sum_{i=1}^{N} log_2 P(w_i|w_1 w_2 ... w_{i-1}) \tag{2.14}$$

and thus:

$$PP(W) = 2^{H(W)} \tag{2.15}$$

The lower the cross-entropy, the better. Consequently, the perplexity should be as low as possible too.

A further interpretation of the perplexity introduces the concept of weighted average branching factor of a language. Concretely, given a sequence of words, it represents the number of possibilities for the next word. It ranges from 1 to the vocabulary size. Again, the smaller this number is, the better.

To conclude, perplexity is a straightforward indicator for a quick evaluation of a language model. However, considering a relative perplexity improvement, it is still hard to quantify the model's improvement. Therefore, a metric more close to our task is needed: the WER.

2.3.2 WER

The Word Error Ratte WER basically compares a word sequence with a reference. More precisely, it can be defined as follows:

$$WER = \frac{S+D+I}{N} \tag{2.16}$$

S stands for substitutions (words replaced by others), D deletions (words omitted), I insertions (words added) and N is the total number of words in the reference.

Although the WER gives a direct evaluation of the language model, it is computationally more expensive.

2.4 Decoding Graph and Lattices

2.4.1 Weighted Finite State Acceptor and Transducer

A Finite State Transducer (FST) is a finite automata, with input and output symbols on each transition. Thus, it "transduces", "maps" an input sequence into an output sequence.

A weighted FST includes weights on each transition, which encode the probabilities to go from one state to another. These probabilities depend on different factors, such as pronunciations and language modeling. Thus, the overall path from one initial state to a final state in the FST joins one input symbol with one output symbol with a specific weight. An example of weighted FST is shown in 2.2 [2].

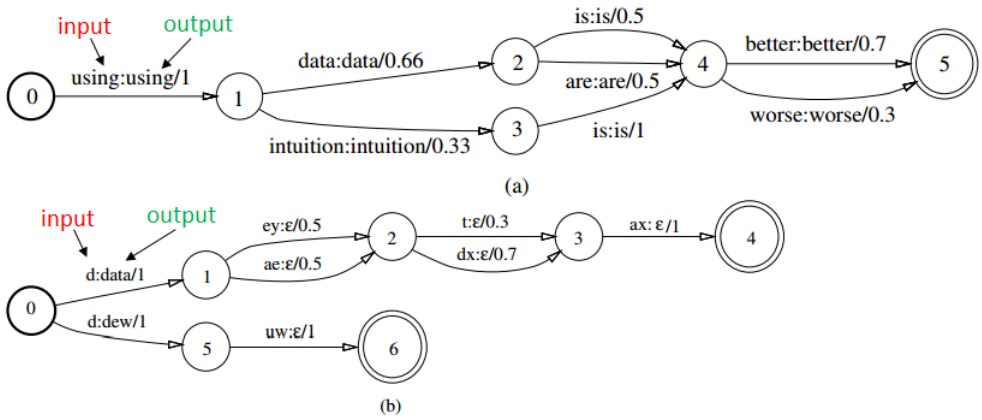


Figure 2.2 – Weighted Finite State Transducer

A weighted Finite State Acceptor (FSA) is a weighted FST which only has input symbols (thus only one kind of symbols). Instead of mapping one representation to an other, it "accepts", "recognizes" one string along a path. An example of weighted FSA can be seen in Figure 2.3.[2].

The transition between the weighted FSA and the weighted FST is quite easy: a weighted FSA can be interpreted as a weighted FST with identical input and output symbols. For instance, weighted FSA in Figure 2.3 (a) is transformed into a weighted FST in Figure 2.2 (a).

[2] Picture taken from [19]

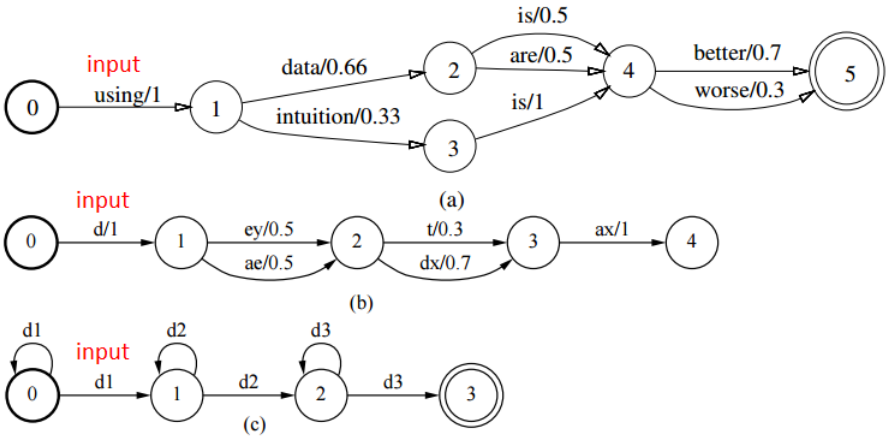


Figure 2.3 – Weighted Finite State Acceptor

2.4.2 Speech Recognition and Graph Decoder

A speech recognizer can be modelled by a transducer, composed of four smaller transducers as follows:

$$HCLG = H \circ C \circ L \circ G$$

H corresponds to the HMM, C to the context dependencies, L to the lexicon and G to the language model.

In particular: H transcribes transitions ids into context dependent phones (often triphones). It makes use of the acoustic model. Then, C transcribes context dependent phones into context independent phones. Thereafter, L maps phones into words. Finally, G introduces the language model. Since G maps words to words, it is in the form of a weighted FSA.

The global process can be seen in Figure 2.4.

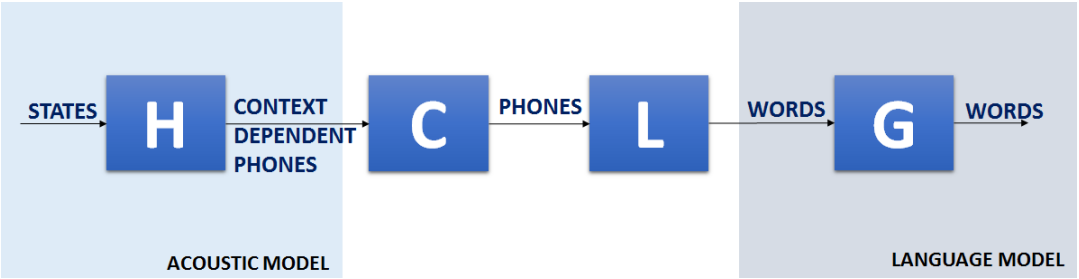


Figure 2.4 – Speech Recognition Pipeline

3 Implementation Considerations

In the scope of this thesis, two main toolkits will be used: the Kaldi toolkit and the RNNLM toolkit of Mikolov. In this chapter, the notion of lattices and their manipulation within Kaldi will be developed. Besides, a section will be devoted to present the RNNLM toolkit of Mikolov.

3.1 Lattice Rescoring

As introduced in the Kaldi toolkit [24] (a toolkit for speech recognition system which will be used in this work), a lattice is "a representation of the alternative word-sequences that are "sufficiently likely" for a particular utterance." In other words, a lattice is a graph which represents the different variants or possibilities of speech transcriptions given an ASR system. For instance, the sentence *"Let us not exaggerate"*, can also be transcribed as *"Lettuce not exaggerate"*, or as *"Let house not exaggerated"*. For simplicity, all those different possibilities are gathered in a specific representation: a lattice (see 3.1 [3]).

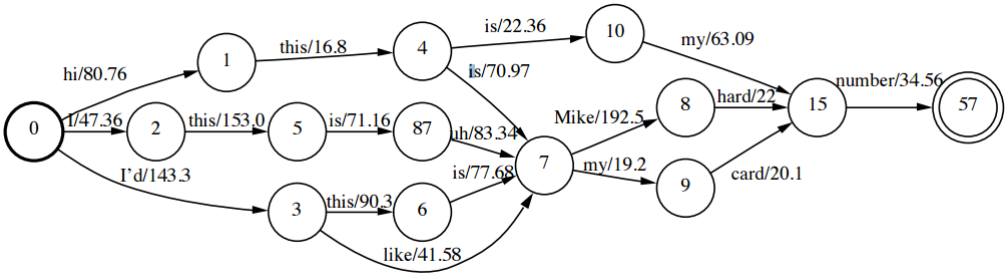


Figure 3.1 – Lattice Format

Lattices in Kaldi can be represented in two different forms: using a *Lattice* type or a *Compact-Lattice* type.

[3] Picture taken from Mohri, Lecture 12 in Speech Recognition, Lattices and Algorithms, www.cs.nyu.edu

A Lattice type is a weighted FST with the following characteristics: inputs are transition-ids, outputs are words, and weights correspond to two scores, the graph score and the acoustic score.

The acoustic score is given by the acoustic model.

The graph score is the sum of several factors: the Language Model (LM) score (corresponding to the G transducer introduced in the previous part), the pronunciation score, and some transition probabilities. It should be pointed out that the LM score is here mixed with other scores. Consequently, to change a language score for a new one, one has first to remove the old LM.

Graph score and acoustic score have different scales: any operation involving both should be preceded by a rescaling step. Indeed, scores stored in lattices are always unscaled.

To sum up, the format used for the Lattice type is the following :

[start id] [end id] [input symbol] [output symbol] [graph score/acoustic score]

For instance, see Figure 3.2 [4] :

```
11031_A_20120617_182613_000704
0       616     273     2402    10.9106,2539.22
0       667     345     3       4.09183,365.022
0       672     273     1268    13.7098,2659.86
0       726     273     758     12.6504,2686.83
0       780     273     3157    14.3474,2700.35
0       834     273     1992    16.2729,2677.13
1       888     281     3       10.8443,4811.55
```

Figure 3.2 – Lattice Type Format in Kaldi

The CompactLattice type contains exactly the same information as the Lattice type. However, it is represented in the form of an acceptor: the inputs and outputs are here both words, and the transition-ids are indicated in the weights, with the acoustic and graph scores.

To sum up, the format used for the CompactLattice type is the following:

[start id] [end id] [input symbol/output symbol] [graph score/acoustic score/transition ids]

It should be noted that the scores are expressed in both representations as negated log probabilities. Then, to retrieve the global score of a path, one just has to take the sum of all scores.

[4] Picture taken from the web site www.codingandlearning.blogspot.com

Since a lattice is often huge, only its N-best paths through it are usually kept. Knowing that each path is associated with a global score, the rescoring step consists in updating those scores. Basically, old LM scores are removed and new ones are added. Thus, the order of the N-best paths can change. Ideally, higher scores are attributed to correct transcriptions.

3.2 RNNLM of Mikolov

Although N-grams remain powerful tools for language modeling, Tomas Mikolov found out in 2010 that the use of RNN can get even better results. This section describes in more details the RNN developed by Mikolov, which will be used thereafter. Main references for this part can be found in [16] and [17].

3.2.1 Architecture

Mikolov's RNN is composed of one input layer, one hidden layer and one output layer. At the input layer, the word w_t at time step t is encoded with a vector of dimension V (with V the vocabulary size). This vector is concatenated with a vector $s(t-1)$ which represents the output values from the hidden layer at the previous time step. The hidden layer is composed of sigmoid activations. The output layer gives $P(w_{t+1}|w_t, s(t-1))$. Basically, the RNNLM predicts the most probable next word (see 3.3[1]).

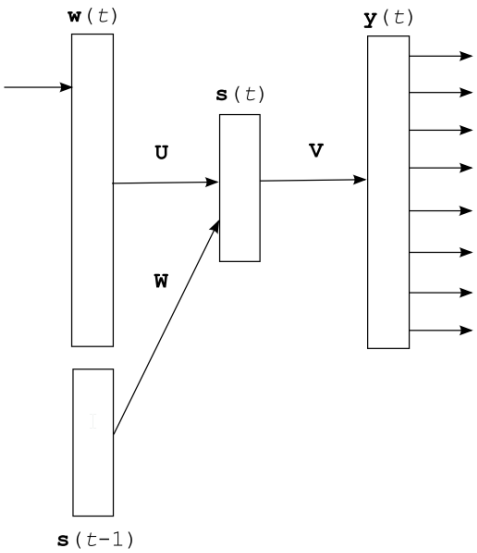


Figure 3.3 – Mikolov's RNNLM

The weight matrices are **W** (recurrent weights), **U** (between the input and the hidden layer)

[1] Picture taken from [16]

and **V** (between the hidden and the output layer). Output values at the hidden layers are computed as follows:

$$\mathbf{s}(\mathbf{t}) = f(\mathbf{U} * \mathbf{w}(t) + \mathbf{W} * \mathbf{s}(t-1)) \tag{3.1}$$

$$\mathbf{y} = g(\mathbf{V} * \mathbf{s}(t)) \tag{3.2}$$

where f and g correspond respectively to the sigmoid and softmax activation functions, define as follows:

$$f(z) = \frac{1}{1+e^{-z}} \tag{3.3}$$

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \tag{3.4}$$

3.2.2 Training

The RNN is trained using Stochastic Gradient Descent (SGD) with error backpropagation. In this case, weights are updated each time a training sample is presented (as opposed to batch update, where weights are updated only after all training samples of the batch have been presented). The cost function to minimize is the cross-entropy, which is well suited for classification problems with sigmoid and softmax non-linearities. On the contrary, Mean Square Error (MSE) is not employed since it's rather used for regression problems with continuous targets. The objective function can be written as follows:

$$CE = -\sum_i \mathbf{d}_{i+1} * log(\mathbf{y}_{i+1}) \tag{3.5}$$

where **d** the target probability and **y** is the estimated probability.

In the particular case where words are represented with sparse vectors, with a length corresponding to the vocabulary size, the target probability **d** equals 1 at the next word index (index $i+1$) and 0 otherwise. Thus the objective function can be simplified in:

$$CE = -log(\mathbf{y}_{i+1}) \tag{3.6}$$

This is explained in more detail in Figure 3.4, which is based on the sentence *"The cat is*

sleeping outside". The first upcoming word is *"The"*: it activates the input corresponding to its index. Then, the RNN tries to predict the next word: it results in a probability distribution at the output layer. In the figure, different colors represent different possible probabilities. The nodes left in white are probabilities of 0. The target distribution is a sparse vector: just one node, corresponding to the next word *"cat"*, is colored. This process continues from step to step, and tries to minimize the distance between the target distribution and the estimated one.

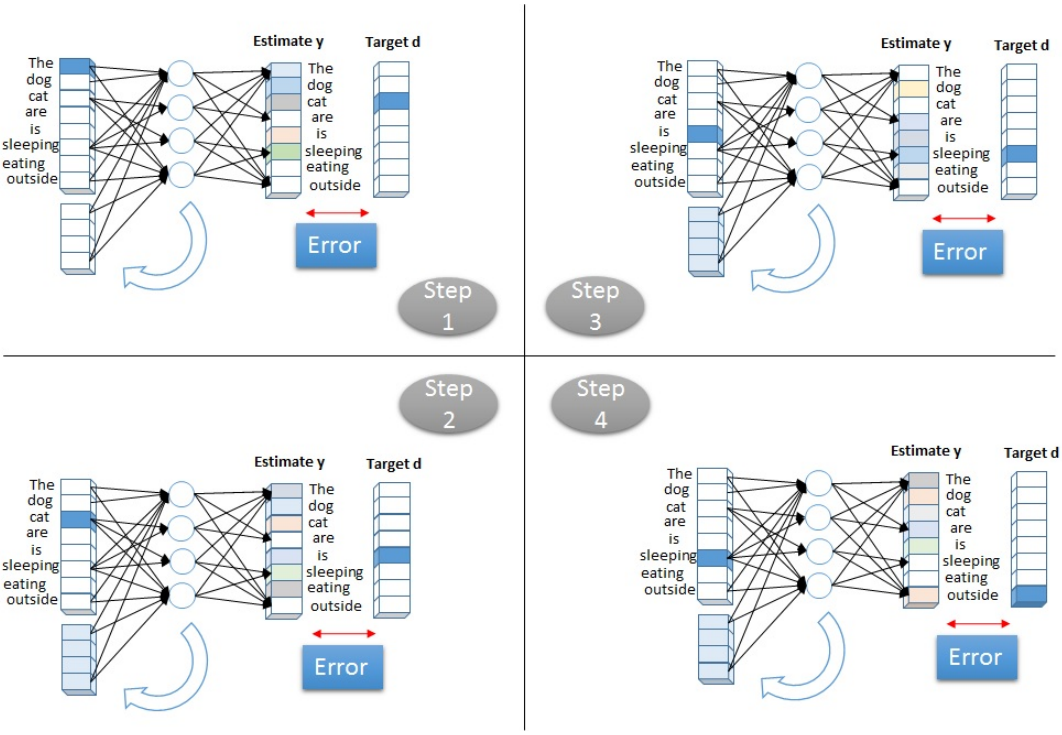


Figure 3.4 – RNN Principle

In our case, the error gradient at the output layer is equal to:

$$\mathbf{e_0}(t) = \mathbf{d}(t) - \mathbf{y}(t) \tag{3.7}$$

The error gradients are propagated from the output layer to the hidden layer as follows:

$$\mathbf{e_h}(t) = d_h(\mathbf{e_0}(t)^T \mathbf{V}, t) \tag{3.8}$$

where d_h corresponds to the derivative of the sigmoid function, and is applied element-wise

as follows:

$$d_{hj}(x,t) = x * s_j(t) * (1 - s_j(t)) \tag{3.9}$$

The weight update formulas are:

$$\mathbf{V}(t+1) = \mathbf{V}(t) + \mathbf{s}(t)\mathbf{e_0}(t)^T\alpha \tag{3.10}$$
$$\mathbf{U}(t+1) = \mathbf{U}(t) + \mathbf{s}(t)\mathbf{e_0}(t)^T\alpha \tag{3.11}$$
$$\mathbf{W}(t+1) = \mathbf{V}(t) + \mathbf{s}(t)\mathbf{e_0}(t)^T\alpha \tag{3.12}$$

One can modify those update formulas to keep the weights close to zero (for more efficient storing and better generalization), using L2 regularization. Mikolov uses regularization after each 10 steps (i.e. each 10 training words). The update formulas are then:

$$\mathbf{V}(t+1) = \mathbf{V}(t) + \mathbf{s}(t)\mathbf{e_0}(t)^T\alpha - \beta * \mathbf{V}(t) \tag{3.13}$$
$$\mathbf{U}(t+1) = \mathbf{U}(t) + \mathbf{s}(t)\mathbf{e_0}(t)^T\alpha - \beta * \mathbf{U}(t \tag{3.14}$$
$$\mathbf{W}(t+1) = \mathbf{V}(t) + \mathbf{s}(t)\mathbf{e_0}(t)^T\alpha - \beta * \mathbf{W}(t \tag{3.15}$$

Mikolov sets the parameter $\beta = 10^{-6}$

However, in our case, it should be noted that the Backpropagation (BP) algorithm is not optimal. Indeed, the prediction of the next word is only based on the previous word and the previous state of the hidden layer. Long context information, that can be actually useful, is not explicitly taken into account. That's why Mikolov uses an extension of the BP algorithm: Backpropagation Through Time (BPTT).

The basic idea is to capture longer history information by propagating the error further (for more than the layers of the current step). Concretely, the RNN is "unfolded", as shown in Figure 3.5[2].

Thus, for N time steps, the RNN can be unfolded in a feed forward network with N hidden layers. The unfolded recurrent weight matrices are the same, given by W. To train such a network, we can apply the usual gradient descent. From step to step, we propagate the error and update W.

Concretely, the recursive error propagation is done as follows:

[2] Picture taken from [16]

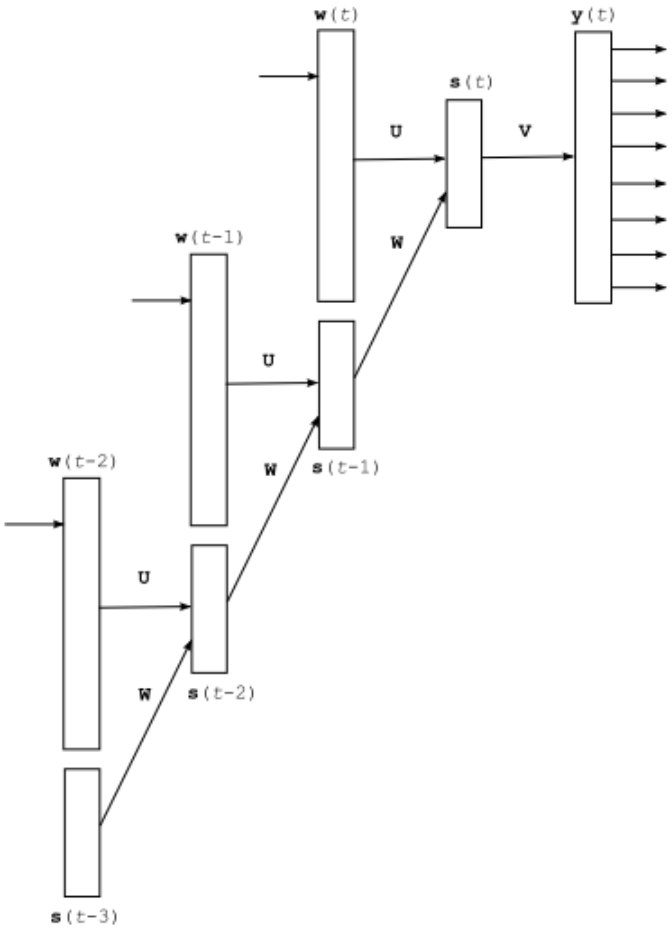


Figure 3.5 – RNN unfolded for 3 Time Steps

$$\mathbf{e}_(t-\tau-1) = d_h(\mathbf{e}_h(t-\tau)^T\mathbf{W}, t-\tau-1) \tag{3.16}$$

$$\tag{3.17}$$

and the matrices W (recurrent weights) and U (between input and hidden layers), can be rewritten:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \sum_{z=0}^{T} \mathbf{s}(t-z-1)\mathbf{e}_h(t-z)^T\alpha \tag{3.18}$$

$$\mathbf{U}(t+1) = \mathbf{U}(t) + \sum_{z=0}^{T} \mathbf{w}(t-z-1)\mathbf{e}_h(t-z)^T\alpha \tag{3.19}$$

If regularization is used, the equation becomes:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \sum_{z=0}^{T} \mathbf{s}(t-z-1)\mathbf{e}_h(t-z)^T \alpha - \mathbf{W}(t)\beta \tag{3.20}$$

$$\mathbf{U}(t+1) = \mathbf{U}(t) + \sum_{z=0}^{T} \mathbf{w}(t-z-1)\mathbf{e}_h(t-z)^T \alpha - \mathbf{W}(t)\beta \tag{3.21}$$

During the backpropagation through time, the error gradients quickly vanish. Thus, just a fix number of steps need to be unfolded. Here, this number is fixed at 6.

The number of epochs and the learning rate α are controlled by a validation set. For each epoch, as long as any improvement in perplexity on the validation set is observed, the learning rate is kept at $\alpha = 0.1$. As soon as the perplexity on the validation set stops improving (i.e. stops decreasing), α is divided by two, and that for each new epoch. Finally, the learning procedure stops as soon as the perplexity on the validation set stops improving again.

4 Baseline System

In this chapter, the baseline system implemented in this thesis is presented, as well as its performance on different experiments.

4.1 Description

4.1.1 Librispeech Corpus

The goal of this work is to correct a transcript generated by an ASR system. For instance, in a real word scenario, the input of this ASR system is a collection of several calls of a specific user. This one is characterized by its context, its main topic, but also its length (which is supposed to be shorter than several hours).

To avoid time-consuming preparation tasks (preprocessing, transcript generation...), the decision was made to directly work on a database already processed by Sony.

For that purpose, we choose the Librispeech Corpus (cf [21]). The Librispeech Corpus contains 1000 hours of read audiobooks sampled at 16kHz. Those audiobooks are themselves part of the LibriVox project. In our case, one book is equivalent to one speaker in the real world.

Audiobooks are well suited for this study for several reasons: first, the topic within a book is well defined, and a specific vocabulary will naturally pop up. Then, the recordings are also short (as for phone calls, not longer than several hours for one book), so the restriction concerning the small amount of data is respected. Finally, the author has a specific style, reflected by the book. Since the user will also have his own language "style" (he will prefer short sentences for instance, or always use the same grammar structures...), this is well adapted to our purpose.

It should be mentioned that, for each speaker of the Librispeech Corpus, the speech time is limited to 25 minutes. To conclude, in this thesis, a "context" will be modelled by passages from one single book and read by different speakers.

The Librispeech Corpus is divided into two parts: a "clean" part and an "other" part. Since the

"clean" part contains audiobooks with a higher recording quality, we will only use books of this category.

These audiobooks are processed in advance, and passed through a neural network based ASR system. The language model used for this process has been built on the Switchboard Corpus. It results in noisy transcriptions, which are used as input for this work.

4.1.2 Methodology

In the correction process, the core idea is to elaborate a new language model specific to the book (in a real word scenario, to the speaker). This is done by training a RNNLM on suitable data. Then, the lattices, or different possibilities of transcriptions for one utterance of the audiobook, are scored using this RNNLM. Finally, paths trough the lattices with the best scores are kept: this results in a corrected version of the book.

At this point, a question arises: which kind of data can be used to train the new language model? The strategy followed in this baseline system is to directly choose the noisy book itself. Of course, if a word is systematically wrong, it has no chance to be corrected without any external information. However, things are different if a word appears correct in 90% of the cases , and is poorly recognized only 10% of the time. The underlying assumption made here is that while catching information from the noisy book, correct cases will have more impact than wrong ones. In such a situation, it will be possible to correct the existing errors.

For instance, let's just consider a simple unigram: if the word *ship* appears 100 times correctly, and is recognized only 10 times as *sheep*, its probability is still higher in its right form. A similar reasoning can be applied with a RNNLM.

Consequently, our hypothesis is that some errors contain their correction directly in the book, and can thereafter be corrected. Even ideally, this baseline system does not aim at correcting completely the transcript. An improvement in terms of WER without any supplementary information is rather what is targeted.

Besides, given one noisy book, we want to catch information from this book and not from another one. Thus, a specific RNNLM is trained on this specific book and just for it. This RNNLM is not supposed to be used further for any other correction on other books. Generalization is not targeted.

The only thing that we aim at generalizing are the hyperparameters for each model. These are the parameters of the new language model fixed before any training phase (for instance, the number of hidden nodes of the neural network). To sum up, a use case application for this baseline system would be: a user's call is transcribed by an ASR system into a noisy transcript. Then, using the hyperparameters previously tuned, a RNNLM is trained on the transcript. In a final step, scoring is performed on the speech's lattices.

Since generalization is targeted for the hyperparameters, a test set is needed. In this study, 7 books are used as "training/validation set" (6+1) to determine the best hyperparameters, and 1 book is used as "test set" for those chosen hyperparameters (see Figure 4.1). More detailed information about those books can be found in Annexe A.

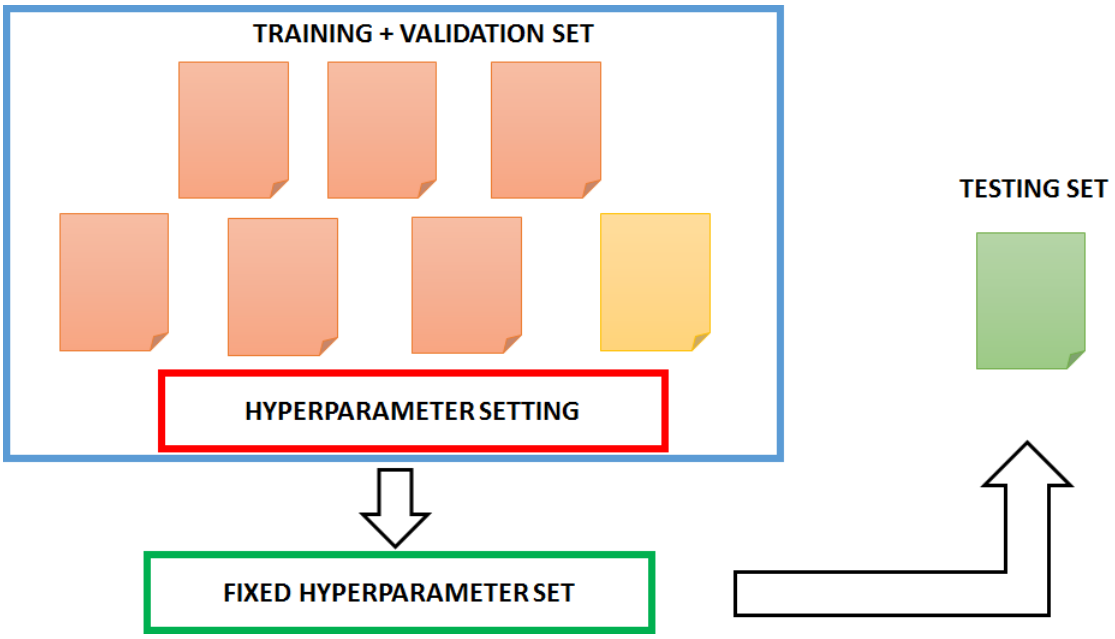


Figure 4.1 – Methodology

4.1.3 Baseline System

In this subsection, a general overview of the baseline system is given, based on the Figure 4.2.

The blue boxes are the existing data. Basically, a speech signal is transcribed by an ASR system into lattices, i.e. into all its different possible transcriptions. Each link in these lattices contain two scores: the acoustic score and the graph score (including the language model score and the pronunciation score). Scaled and combined together, these scores give a new global score. The best global score indicates the best transcription among the possible ones. As a result, we obtain a final transcript, which however still contains errors and that need to be corrected.

The red boxes on the figure correspond to our baseline system for error correction. The goal is to adjust the scores of the lattices produced by the ASR system. In this way, a new best path is found, which will ideally result in a better final transcript.

The green boxes represent the evaluation phase. Given the gold transcript (the book without errors), we can compute the WER for the noisy transcript WER1, and for the corrected transcript WER2. The system's performance is then estimated by the difference: WER 1 - WER 2, called *Gain in WER*. A positive gain indicates a good correction system, a negative gain a bad

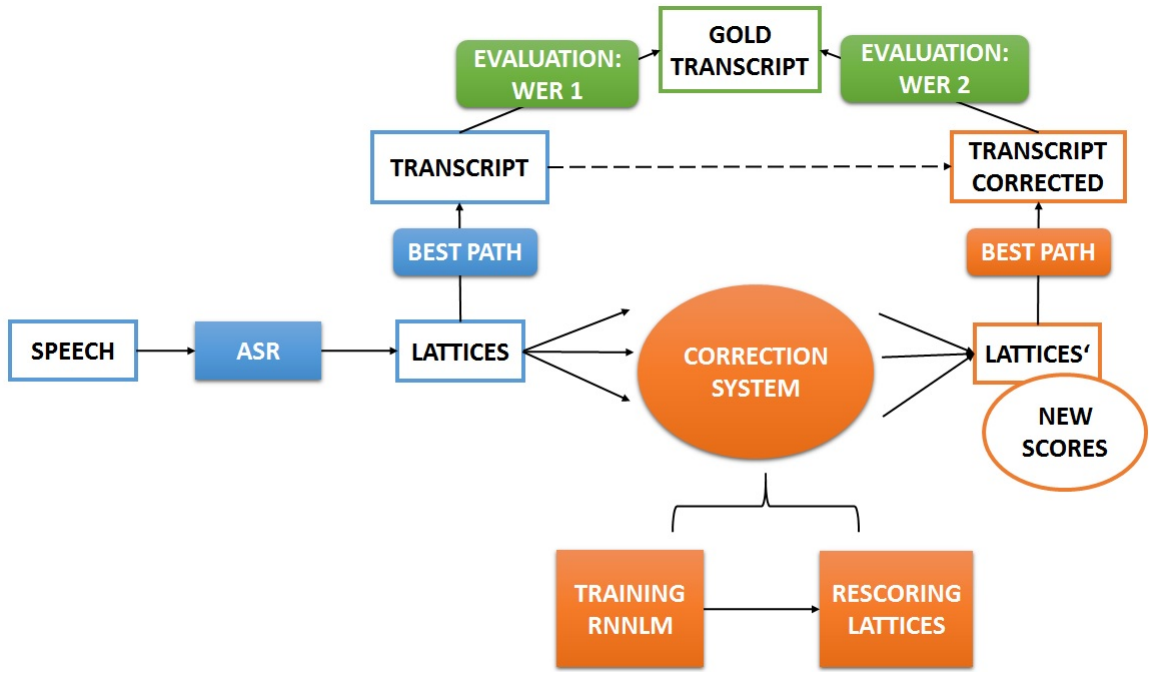


Figure 4.2 – Transcript Correction: General Overview

one.

The "correction system" involves in particular two main phases: a training phase and a rescoring phase. The training phase generates a new LM based on a RNN. Then, this LM is used on the rescoring phase: for all utterances in the transcript, a score is computed to replace the old one. These two phases, as well as the final correction/evaluation phase, will be detailed in the next subsections.

4.1.4 Training

First step in the correction process is to train a specific RNNLM.

The inputs are:

- the training set
- the validation set
- the speech lattices

The output is:

- the new language model G'

The books have sizes varying between 15k and 45k words. This corresponds to vocabulary sizes varying between 2k and 7k words.

The main hyperparameter to tune in this phase is the size of the hidden layer. During training, the "history" context (hidden layer fed back into the input layer) is reset to 0 after each utterance. The aim is to catch dependencies within utterances, and not across utterances. Indeed, RNN tends to forget information quickly, and it does not make so much sense to go beyond an utterance. Furthermore, the number of steps to unfold in the BPTT algorithm is set at 6, and the maximal number of epochs limited at 100.

The resulting RNNLM corresponds to the new language model G'.

For a comparison purpose, bigram and trigram LM have also been trained. For the discounted N-gram probabilities estimates, interpolation combined with a Kneser-Ney smoothing is performed.

4.1.5 Rescoring

Second step in the correction process is to rescore the lattices.

The inputs are:

- the speech lattices
- the ASR language model G
- the new language model G'
- the lexicon used L

G corresponds to the trigram model built on the Switchboard Corpus, and L to its lexicon.

The output is:

- the new scores for the speech lattices

The global rescoring process is shown on Figure 4.3.

For the sake of simplicity, the lattices are first reduced to the N-best paths (here we choose N = 100). This step is very sensitive to the acoustic scale, i.e. the scale applied before linking acoustic and graph scores. In this thesis, the scale = 1/15 is chosen, since it's the one which gives the best WER results for the first ASR system. The resulting output is specified in the Kaldi format "CompactLattice type".

Then, acoustic scores and graph scores are separated. The "ASR LM" scores given by G are subtracted from the graph scores. They are replaced by the new LM scores given by G'.

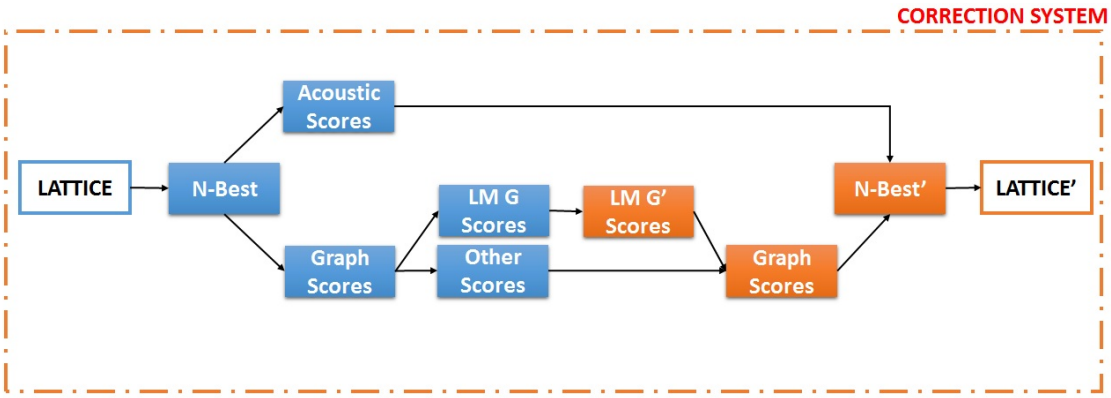


Figure 4.3 – Overview of the Correction System

Finally, acoustic scores and graph scores are combined together into new global scores. Again, this stage is very sensitive to the acoustic scale. We want to select the acoustic scale which will give the best WER results. To achieve scale selection, the method used by Kaldi assums this acoustic scale is a global constant of the model. Different acoustic scales from 1/9 until 1/20 are tried in parallel. The one which gives the best WER is then chosen (a point worth mentioning is that the WER computation involves the lexicon L). To ensure that the range [9-20] is correct, tests on several books are made. Since each scale within this range gives a WER evolution with a convex shape, this range is kept (see 4.4).

A last point to clarify concerns the generation of new LM scores based on G'.

For that purpose, Mikolov's toolkit enables in its testing mode to rescore lattices, and thus to output probabilities of whole sentences. The input required for such kind of mode is a list of utterances. Each one of them is identified with a unique ID. Furthermore, for all hypotheses of a given utterance, the IDs are the same. The example given by Mikolov in his thesis [16] is the following one:

1. WE KNOW
1. WE DO KNOW
1. WE DONT KNOW
2. I AM
2. I SAY

During N-best rescoring, all words that are unknown in the vocabulary (established during the training phase) are considered as "out of vocabulary" words (OOV). A "penalty probability" is assigned to them. After different tests, we set this penalty at -7 in Mikolov's code (see Figure 4.5). It should be pointed out that this value is in log10 basis.

It should be noted that the OOV words are mapped into a special token *OOVWORD*. This token has been chosen for OOV since it turns out that the default token <*unk*> appears in some noisy

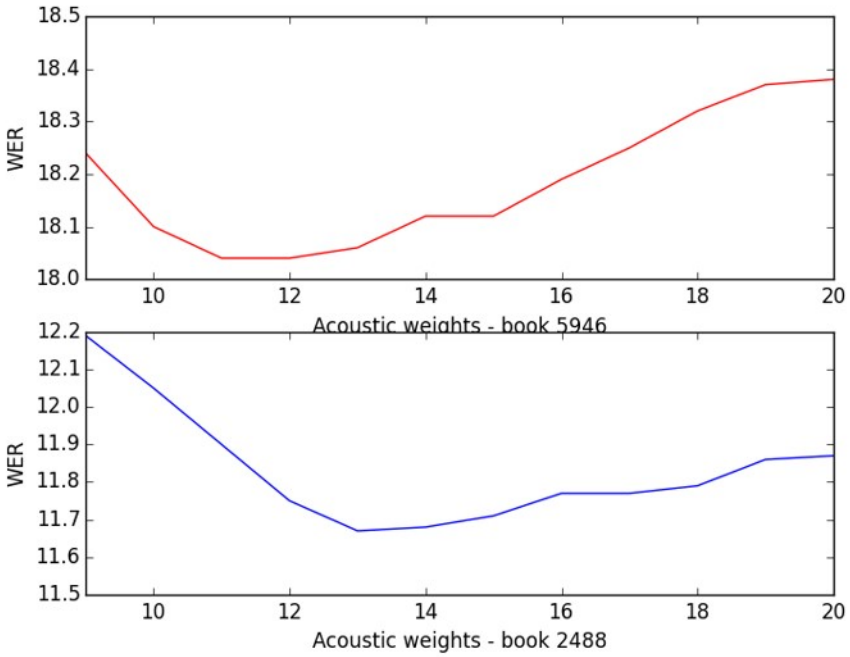


Figure 4.4 – Range for the Acoustic/Graph Scale

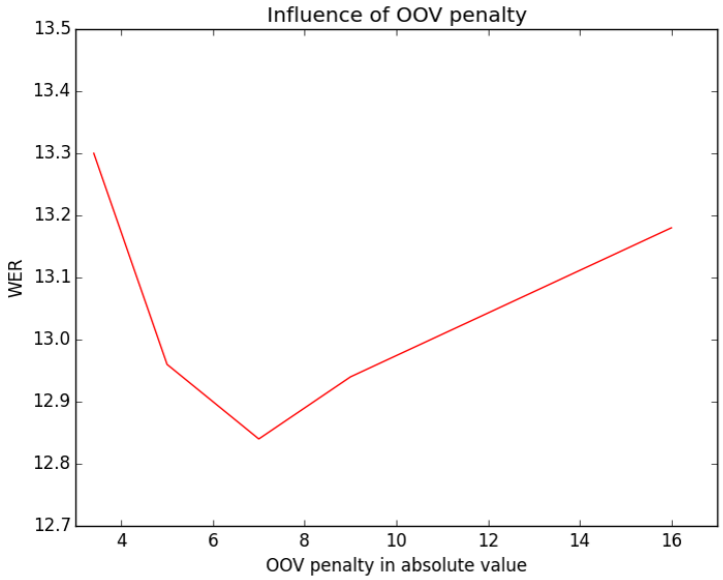


Figure 4.5 – WER according to OOV Penalty

books (perhaps due to the ASR decoding system). In these cases, <unk> is then considered as

a regular word and not a OOV token.

Worth mentioning is that the interpolations RNNLM/bigram have also been tested. Let's called λ the weight between RNNLM and the bigram. For $\lambda = 1$, the scores just come from the RNNLM. For $\lambda = 0$, the scores just come from the bigram. In the following experiments, λ is considered as a hyperparameter.

Scores output by the RNNLM toolkit of Mikolov are probabilities in log10 basis.

$$SCORES_{newLM} = log10(\lambda * P_{RNNLM} + (1 - \lambda) * P_{Ngram})$$

Eventually, the final scores are computed by interpolating the new LM scores from G' with the ASR LM ones from G. Indeed, we do not want to remove all previous information contained in G, but slightly change a part of it. The balance between the ASR LM and the new one is set with the hyperparameter *Model Weight*, which is tuned for the experiments.

$$SCORES_{final} = ModelWeight * SCORES_{newLM} + (1 - ModelWeight) * SCORES_{ASR-LM}$$

4.1.6 Correction and Evaluation

The inputs are:

- the speech lattices with new scores
- the lexicon used L

L is the lexicon created with the Switchboard Corpus.

The output are:

- the corrected version of the transcript
- its WER

The best path within all lattices (i.e. the one with the best total score) is selected. This path is transcribed into a new transcript using the lexicon L and the Kaldi tool $int2sym.pl$.

Then, this "corrected" transcript is compared with the gold one, and its WER is computed.

However, this WER does not really quantify the performance of our system. Indeed, it does not take into account all the words that couldn't be corrected due to their non-existence in the lexicon. It is the case, for instance, with some proper nouns. For example, with the book *Les Misérables* by Victor HUGO, it can be seen that the noun *Jean Valjean* never appears in the

noisy transcription in its right form. Since it does not exist in the lexicon, there is no chance for it to appear in the lattices, and thus to be corrected. This error is independent from the performance we want to measure, since it can't be handled at all in our correction pipeline.

That's why an other WER is defined. This WER does not take into account words that do not appear in the lexicon. Computing this score gives us a better idea of how the system performs according to what it is able to do.

Thus, a new gold version is prepared: in clean books' versions, all words that do not belong to the lexicon L are replaced with the token <OOV>. For the new WER computation, these <OOV> are skipped.

Finally, WER of noisy and corrected transcripts are compared. As already mentioned, we consider the *gain in WER* which is the difference between the noisy WER and the corrected WER. A positive gain indicates good system's performances. On the contrary, a negative gain indicates poor performances.

4.2 Experiments and Results

In this section, experiments conducted on the baseline system and their results are reported. Among others, hyperparameter tuning is performed. Hyperparameters are parameters fixed before any training phase. They have a large influence on the results. That's why they have to be optimized.

4.2.1 RNN Results

As a reminder, the RNN used is composed of one input layer, one hidden layer and one output layer. Thus the hyperparameter to tune here is the size of the hidden layer (its number of hidden nodes).

After the training stage, a first perplexity plot (see Figure 4.6) already shows the RNNLM global behaviour according to its number of hidden nodes. As expected, it appears that the perplexity declines with increasing number of hidden nodes. This decrease rapidly slows down around 10 hidden nodes, and becomes constant after 20 hidden nodes.

Consequently, a hidden layer size between 5 and 10 nodes may be a good choice.

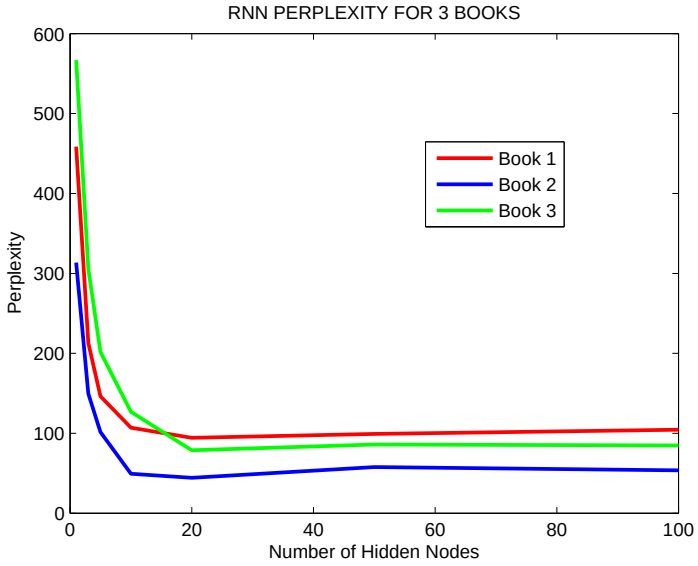


Figure 4.6 – RNN Perplexity for 3 different Books

4.2.2 Interpolation RNNLM and ASR LM

The most simple case is to interpolate the RNNLM with the previous, called "ASR", LM. The weight between both models constitutes an other hyperparameter. It is called: *Model Weight*. Intuitively, this process tunes, or refines the ASR language model using the RNNLM.

In practice, instead of using the Switchboard based model as ASR language model, we choose a trigram based on the larger COCA Corpus, which has a better language coverage. Of course, it results in adding external information to the process, and the WER is in that manner slightly improved. This "shift" in the standard system (system without any correction) is always denoted in the next curves as "Coca Gain". For simplicity however, the Coca LM is still designated as "ASR LM".

To reduce noise and fix the hyperparameters, results shown thereafter are averaged over 6 books. In particular, two hyperparameters are tuned in that approach: the number of hidden nodes and the model weight.

Hidden Nodes Influence

For the number of hidden nodes, values for the grid search range in [1, 3, 5, 10, 20, 30, 50]. The model weight is fixed at 0.3. Results are shown in Figure 4.7.

It seems that performance declines with increasing number of hidden nodes. Indeed, the training data size is too small, and the network can't learn efficiently. With a vocabulary size of 2k for instance, the total number of parameters is: $2 * 2000 * H$, with H the number of hidden nodes. However, the training set only contains around 20k words. It means that with already 6

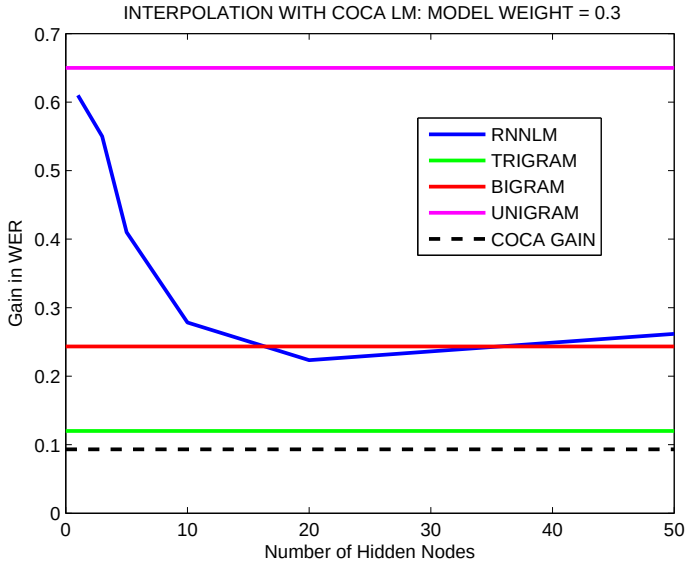


Figure 4.7 – Hidden Nodes Influence

hidden nodes, each parameter "sees" less than once each vocabulary word. This is a kind of bottleneck situation: too few parameters lead to poor results since they can't catch enough information. Too many parameters lead also to poor results since there are not enough data to enable good learning. This already indicates that it will be necessary to add external data (method explored in the next chapter).

Comparison with other N-grams trained on the noisy book and combined with the ASR model are also made. As can be seen in Figure 4.7, the RNNLM can't beat the unigram model. Their performances are equivalent on the case of one hidden node. This makes sense: in this situation, the ranking order for the output probabilities is independent from the input. Output values are all multiplied by the same factor (given by the activation of the hidden node). Of course, this factor changes from step to step depending on the input, but it has no influence on the output distribution order: the most probable words stay the most probable ones, and so on. For reminder, the RNNLM architecture with one hidden node is shown in Figure 4.8.

For higher number of hidden nodes, performances are equivalent to the bigram ones. This makes sense: in this case, with higher number of hidden nodes, history is well encapsulated. As a consequence, previous words influence the output distribution and the probability ranking of words.

Since the training dataset has a reduced size, trigram performs worse than unigram and bigram.

Finally, the COCA gain is indicated in black.

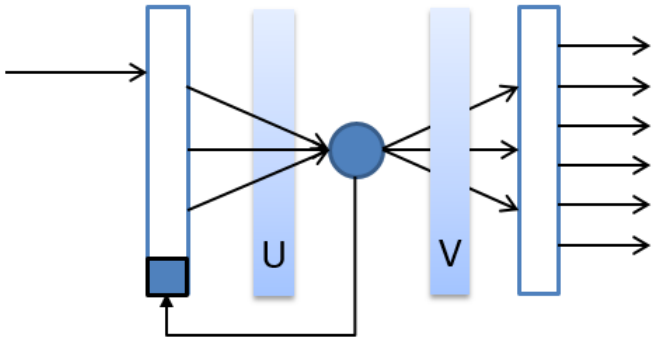


Figure 4.8 – RNN Architecture with 1 Hidden Node

Model Weight Influence

Values for the grid search range in [0, 0.1, 0.3, 0.5, 1.0]. The number of hidden nodes is fixed at 5. If the model weight equals 0, the ASR model (the one based on the COCA corpus), is the only one considered. If the model weight equals 1, the new RNNLM is the only one considered. Results can be seen in Figure 4.9

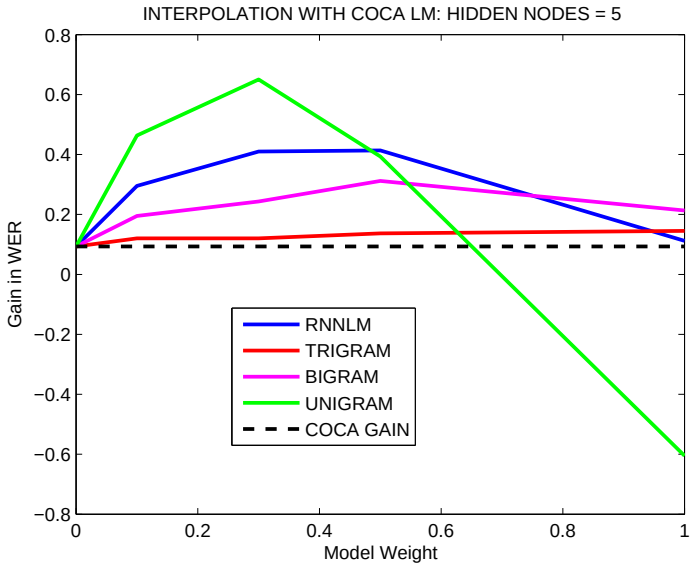


Figure 4.9 – Model Weight Influence

Good results are obtained for values around 0.3. Again, we can verify that the RNNLM wih one hidden node behaves almost the same as the unigram (see Figure 4.10). The slight differences might be due to the fact that learning is not perfect: the RNNLM with one hidden node can be interpreted as an appromixation of unigram.

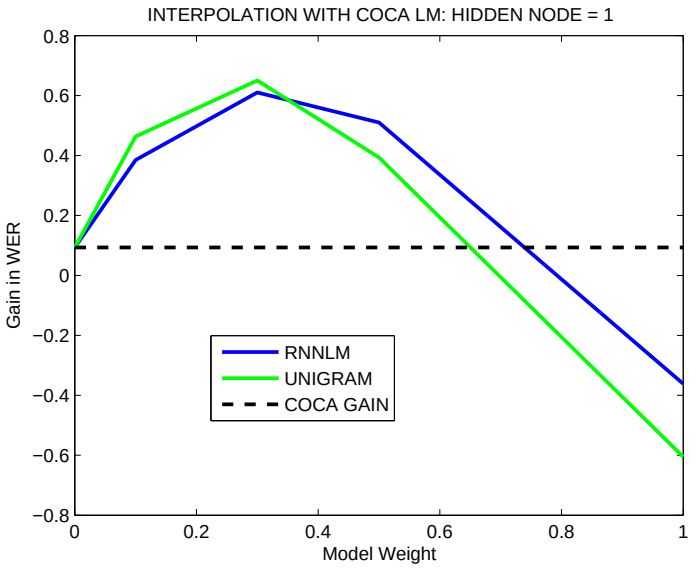


Figure 4.10 – Model Weight Influence - 1 Hidden Node

Besides, the weight matrices U and V are plotted in Figure 4.12, for one particular book, with a RNNLM of one hidden node. The weight distribution for U is quite Gaussian. This is however not the case for V, where the matrix seems rather sparse. Many weights have values close to 0, whereas a few ones have higher values. Considering Figure 4.11, besides the correlation, a kind of cut-off appears for very small frequencies. It seems that less frequent words are kind of discarded. This can also be seen in the table 4.1.

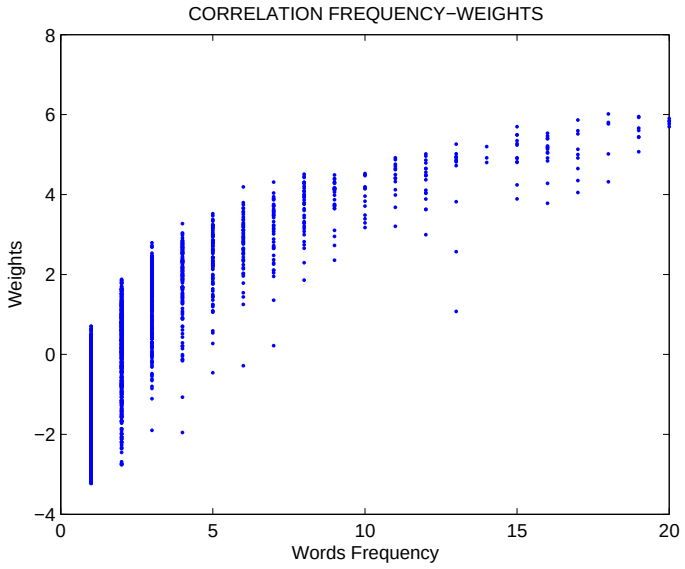


Figure 4.11 – Weights for Matrix V with Respect to Words Frequency

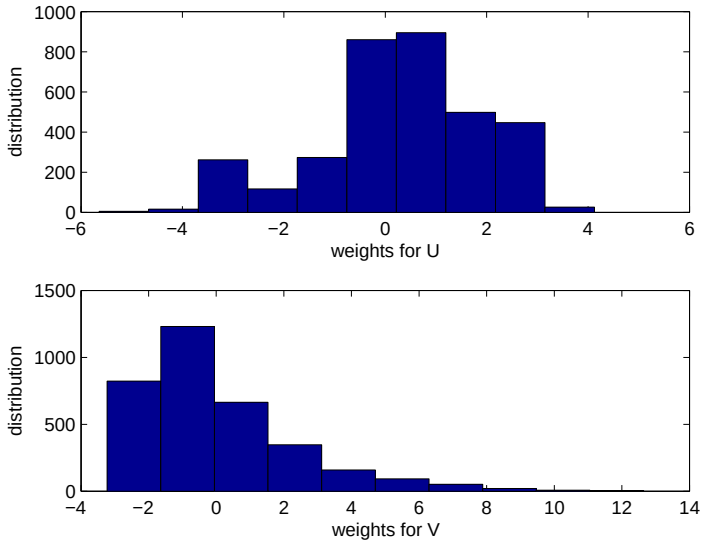


Figure 4.12 – Weights for the RNN Matrices

FREQ	WORDS	WEIGHTS
143	from	9.23
123	them	9.68
112	not	8.66
5	away	3.18
3	vast	0.17
2	struggle	-0.18
2	occasions	0.06
1	conquests	0.08

Table 4.1 – Part of Relation Words-Weights for Matrix V

Introduction of "cut-off" frequency

It leads to the following idea: what if all words appearing less than for instance 3 times where discarded from the training set, and put into a special token <MINCOUNT> ? This experiment is denoted further as "cut-off" case. Within this scope, a first perplexity plot can be seen in Figure 4.13.

Compared to the previous case without cut-off, perplexity is smaller (around 20 instead of 100), which is quite normal since the vocabulary size is also reduced (from around 2k to 500 words). Behavior in both cases are however similar.

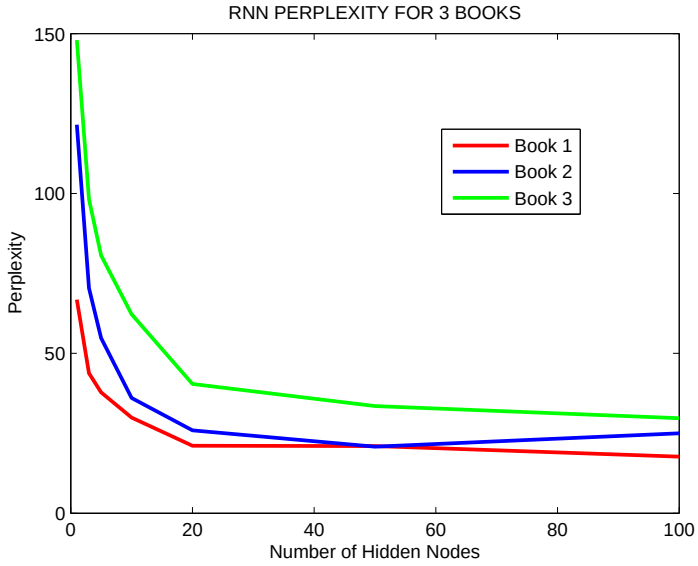
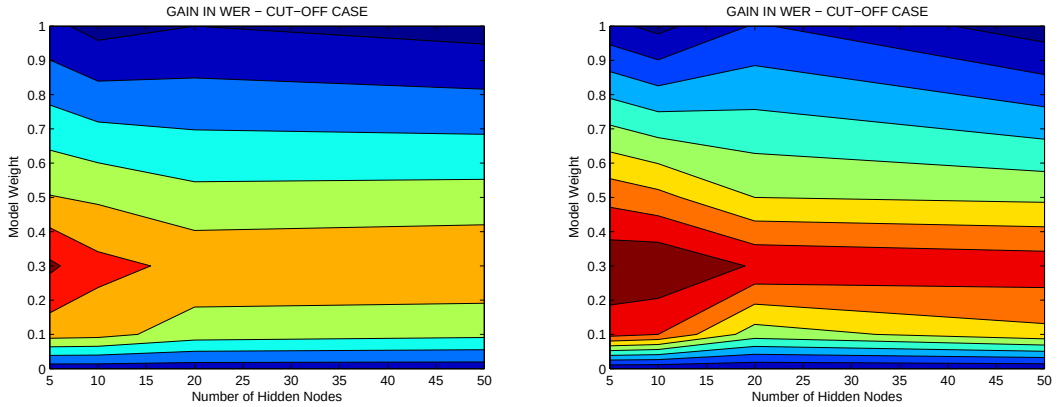


Figure 4.13 – RNN Perplexity for 3 different Books - Cut-off Case

Gird searches are made for several books. The hidden nodes values ranges in [1, 3, 5, 10, 20, 30, 50] and the model weight values in [0, 0.1, 0.3, 0.5, 1.0]. Results for two of them are shown in Figure 4.14.



(a) Example 1

(b) Example 2

Figure 4.14 – Grid Search in Cut-off Case

Again, good results can be seen for model weight around 0.3 and small number of hidden nodes.

Let's study these hyperparameters in more detail, for results averaged over 6 books.

Hidden Nodes Influence - Cut-off Case

In Figure 4.15, the influence of the number of hidden nodes in cut-off case can be seen. The model weight is fixed at 0.3. This time, the RNNLM performs better than the unigram. Good results are obtained for 5 hidden nodes. This can be justifed by the following consideration: let's consider a vocabulary size of 500 words. The number of parameters are: $2 * 500 * H = 1000H$ with H the number of hidden nodes. For a training set of 20k words, it means that each parameter "sees" 4 times each vocabulary word, which enables a quite good learning phase.

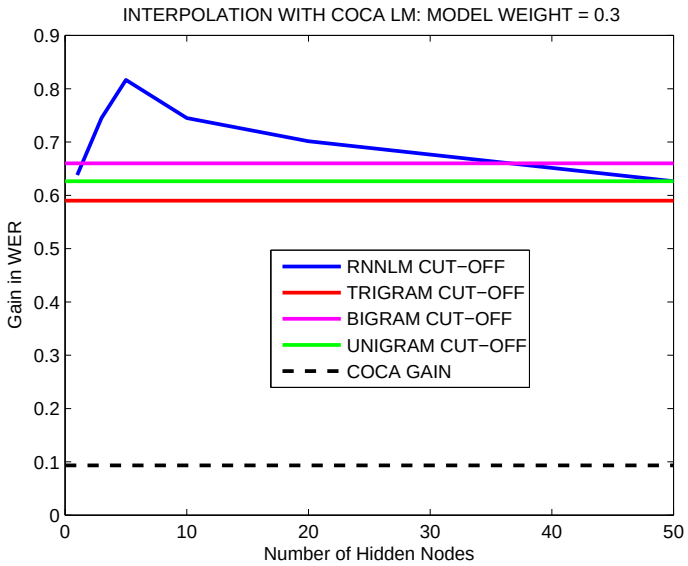


Figure 4.15 – Hidden Nodes Influence in Cut-off Case

Model Weight Influence - Cut-off Case

In Figure 4.16, influence of the model weight in cut-off case can be seen. The number of hidden nodes is fixed at 5. As expected, best results are obtained for weights around 0.3. Besides, RNNLM can beat the unigram.

Again, we verify that the RNNLM with one hidden node behaves the same as the unigram in Figure 4.17. This time, there is almost no difference between both curves. This might be due to the fact that learning is better with cut-off: irrelevant words appearing barely and which might disturb the learning process are removed.

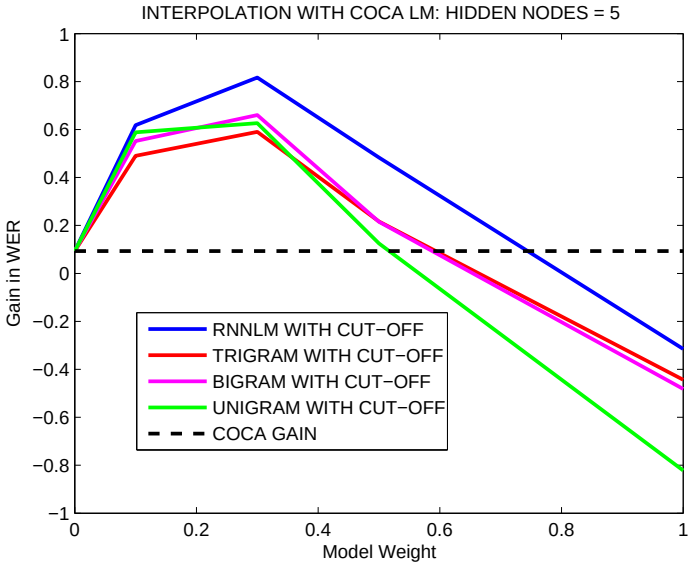


Figure 4.16 – Model Weight Influence in Cut-off Case

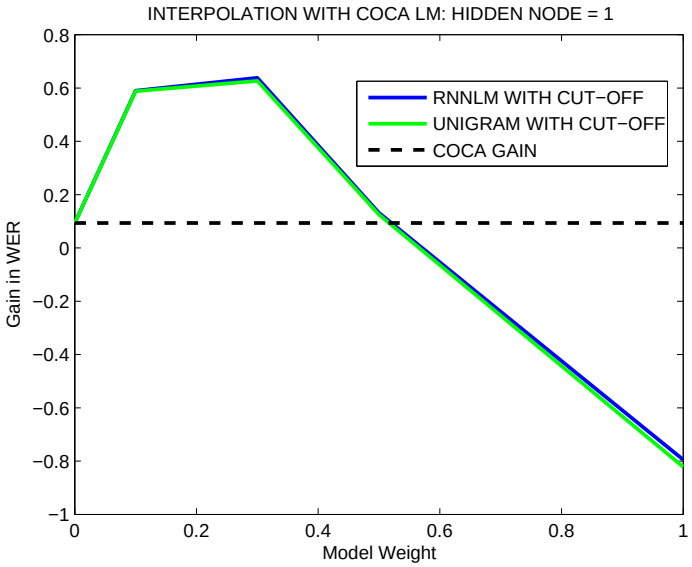


Figure 4.17 – Model Weight Influence in Cut-off Case - 1 Hidden Node

4.2.3 Influence of the insertion penalty

The influence of an insertion penalty is also studied. As mentioned in Ricardo Gutierrez-Osuna's lecture "Introduction to Speech Processing ", acoustic models have often a larger influence than language models. As a consequence, poorly articulated and short words are more likely to be inserted. Indeed, they are short, have a large variability, and may provide good acoustic scores. Therefore, insertion penalty are used to inhibit this effect, and to penalize the probability of transition between words. The larger this penalty, the fewer insertion errors appear, but also the more deletions occur. A trade-off between insertions and deletions should thus be found.

Several tests on different books are made to find the best insertion penalty value. The experiment is conducted on the normal case with values in [-1, 0, 0.8, 1, 1.5], and in the cut-off case with values in [-1, 0, 0.8, 1.5]. Results are shown in Figure 4.18. It seems that values around 0.8 lead to good performances.

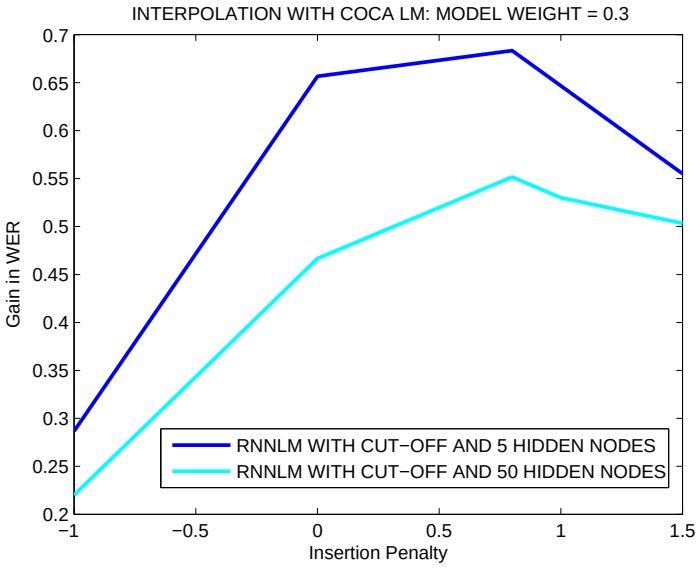


Figure 4.18 – Insertion Penalty Influence

Previous experiments conducted in cut-off case can be made with this insertion penalty. This leads to the Figures 4.19 and 4.20. In general, behavior are similar than without any penalty. Results are however slightly improved.

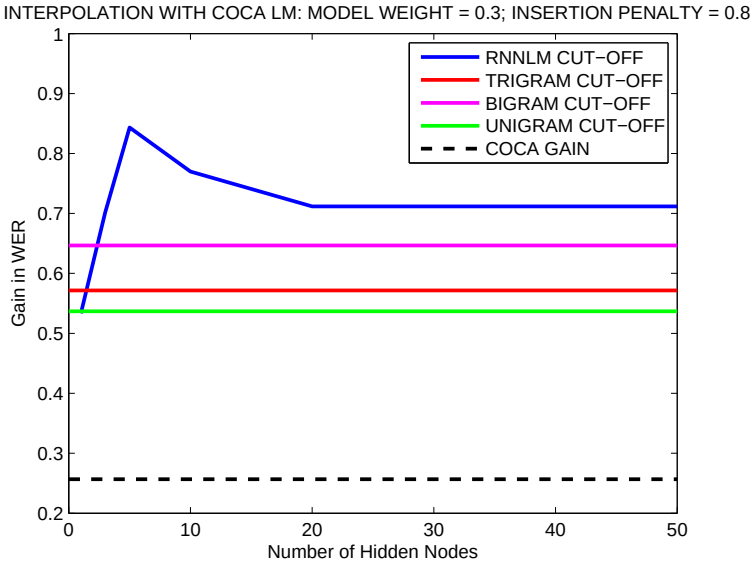


Figure 4.19 – Hidden Nodes Influence with Insertion Penalty and Cut-off

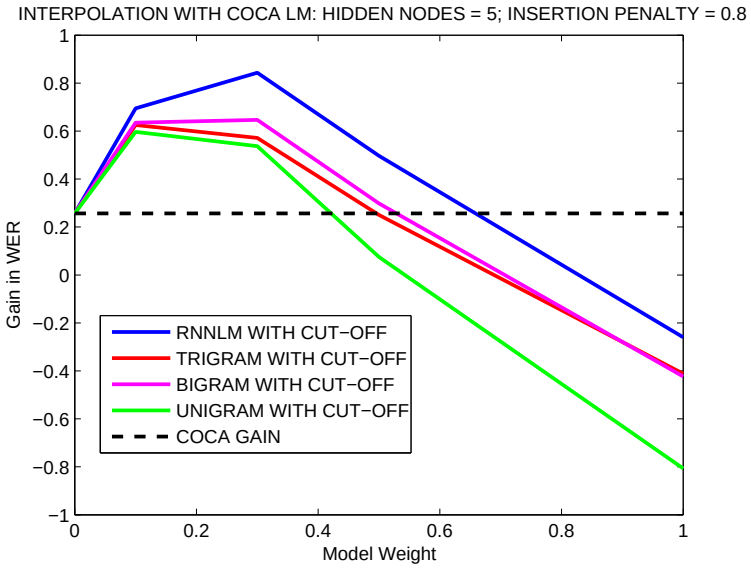


Figure 4.20 – Model Weight Influence with Insertion Penalty and Cut-off

4.2.4 Combination with other language models

In this subsection, an interpolation of three different language models is investigated: the RNNLM, the ASR LM (Coca Model) and also a bigram trained on the book.

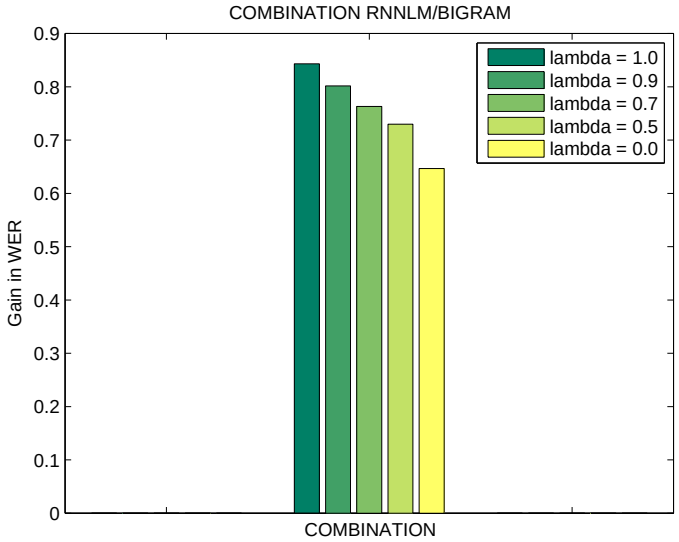


Figure 4.21 – Interpolation with Bigram Model

The resulting LM is computed in the following manner:

$$SCORES_{final} = ModelWeight * [\lambda * SCORES_{RNNLM} + (1-\lambda) * SCORES_{BIGRAM}] + (1 - ModelWeight) * SCORES_{ASR-LM}$$

λ corresponds to the weight given to the RNNLM with respect to the N-gram. If $\lambda = 1$, only the RNNLM is considered. If $\lambda = 0$, only the bigram is considered.

In this case, we set the insertion penalty at 0.8, the number of hidden nodes at 5 and the model weight at 0.3. We also work on the situation of "cut-off".

As can be seen in Figure 4.21, this interpolation does not really seem to help the global system. Best results are for the simple RNNLM without any interpolation.

4.2.5 Combination with inverted model

In this subsection, an interpolation of three different language models is investigated: the RNNLM, the ASR LM (Coca Model) and the RNNLM trained on the inverted book. The basic idea here is that information can be caught in the past as well as in the future. This

configuration will be denoted thereafter as forward/backward. More specifically, it is tested on the cut-off case.

The resulting LM is computed in the following manner:

$$SCORES_{final} = ModelWeight * [\mu * SCORES_{RNNLM} + (1-\mu) * SCORES_{RNNLM-BACKWARD}] + (1 - ModelWeight) * SCORES_{ASR-LM}$$

μ corresponds to the weight given to the RNNLM with respect to the RNNLM-BACKWARD. If $\mu = 1$, only the RNNLM-FORWARD is considered. If $\mu = 0$, only the RNNLM-BACKWARD is considered. Results are shown in Figure 4.22.

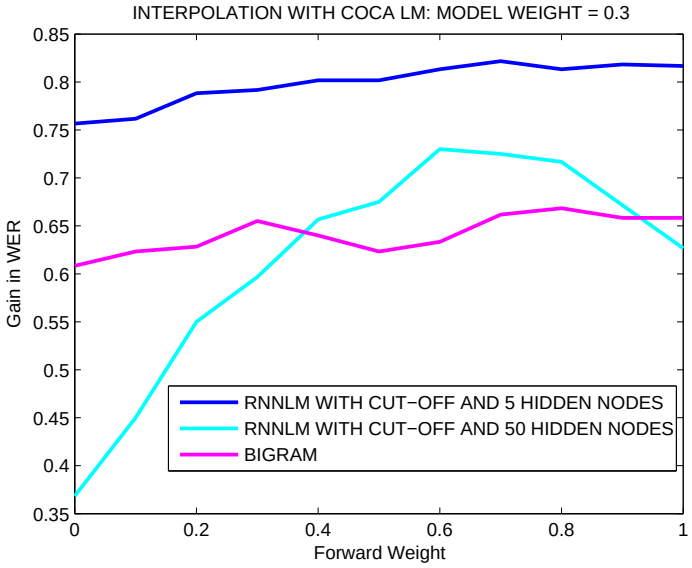


Figure 4.22 – Forward/Backward Setting

The insertion penalty is set to 0. The number of hidden nodes equals here 5, and the model weight 0.3.

It turns out that this interpolation does not help too. It should be noted that results in inverted and forward cases are quite similar for the RNNLM with 5 hidden nodes and the bigram. Worth mentioning is that this kind of behavior is strongly dependent on the language (here English). Here, information can be caught in the future as well as in the past.

However, the curve's shape is different for 50 hidden nodes. In this case, interpolation helps. It demonstrates how important hyperparameter optimization is (results decrease highly with a wrong number of hidden nodes).

RNNLM curve with 5 hidden nodes can be seen in more detail in Figure 4.23.

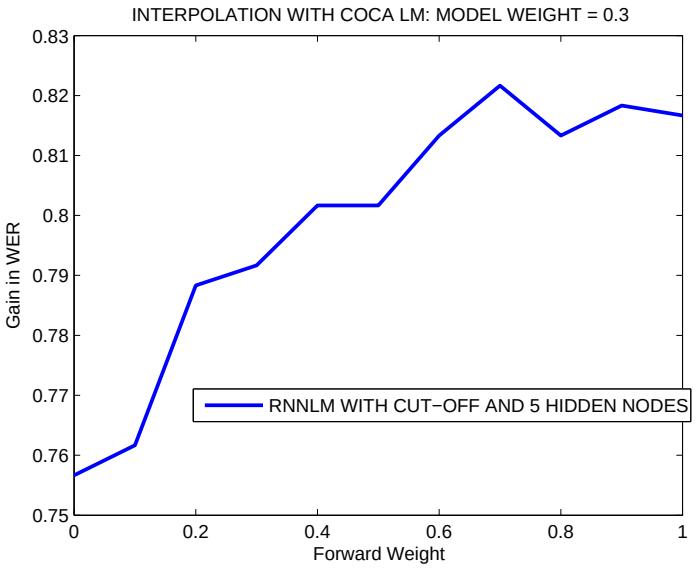


Figure 4.23 – Forward/Backward Setting - Details

Finally, the experiment with an insertion penalty of 0.8 is shown in Figure 4.24. Again, behaviors are similar. As expected, the case with insertion penalty performs better in the forward situation. Finally, the interpolation forward/backward does not seem to help either.

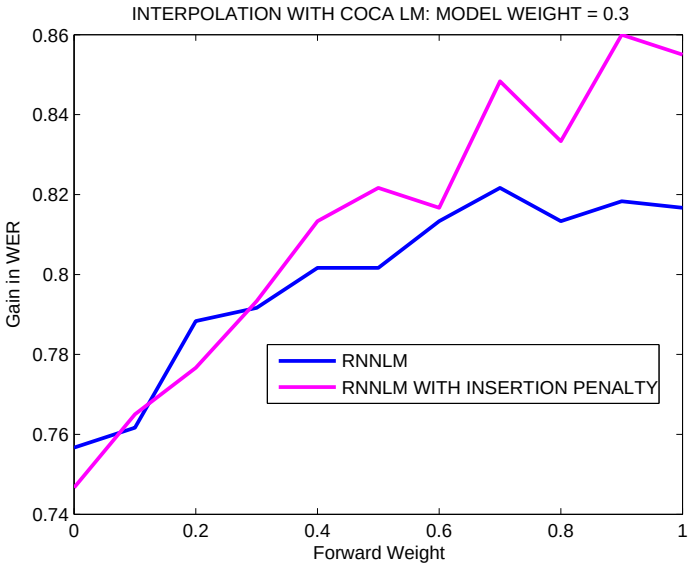


Figure 4.24 – Forward/Backward Setting - Insertion Penalty Case

4.2.6 Are 6 Books enough to fix the hyperparameters?

The only thing that has to be generalized in this work is the set of hyperparameters. In our case, they are the number of hidden nodes and the model weight. At that point, a question arises: how many books will be needed to fix those hyperparameters? In other words, given the grid search of 6 books: is this sufficient to set hyperparameters for the 7th one? How much will it deviate from the optimal case?

To answer these questions, the following procedure is done:

Let us give a set of 6 books. First of all, those books are considered separately. In this first step, the following question is investigated: is only one book sufficient to fix the hyperparameters?

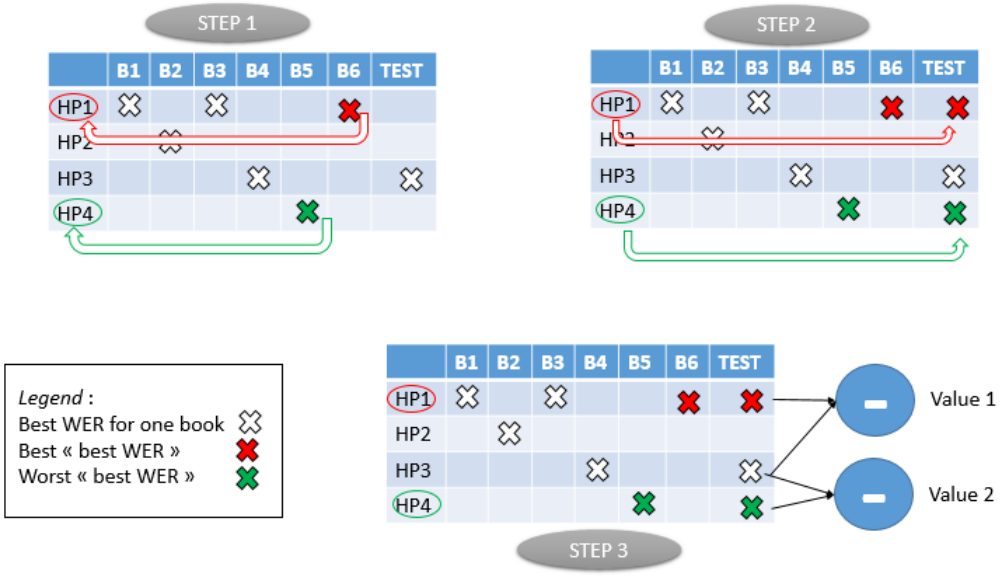


Figure 4.25 – Hyperparameter Setting Process

For each book, the best set of hyperparameters is determined. In Figure 4.25, its corresponding "Gain in WER" is indicated with a cross. The best "Gain in WER" is colored in red, and the worst one in green. Basically, instead of choosing a random book to fix the hyperparameters, we consider two situations: the one with the best book and the other with the worst one.

Here, in the first situation, the hyperparameters are fixed based on "Book 6". Thereafter, given an other "test" book: the "deviation", that is the distance between its best WER and the WER obtained with hyperparameters fixed on Book 6, is computed. This "deviation" gives an idea of "how much do I deter my results by having determined the hyperparameters with another book." In the second situation, the same procedure is followed, but this time with Book 5.

As a result, we obtain two values, which give an idea of the expected deviation for the test book.

The next steps concern subsets of 2, 3... 6 books. The question to answer is each time: are "N" books sufficient to fix the hyperparameters?

The process is exactly the same as for subsets of size one. Each time, the best and the worst subsets are considered, and hyperparameters are determined on them.

This process is realized in a similar way as for cross-validation in machine learning. In other words: among the 7 books, each one of them is considered in turns as "test" book, and the others are used for the hyperparameters' setting. Each time, we obtain two values, corresponding to the best situation and the worst one. At the end, the final value for each situation is obtained by averaging.

In pratice, we obtain the following result: if one determines the set of hyperparameters based on 6 books, further WER may deviate from the optimal one by an amount around 0.1%. This value represents a kind of "noise range". To sum up, results can be determined by using 6 books with a confidence of 0.1%.

4.2.7 Testing

Along this chapter, different configurations have been experimented on the "training set" of books. Furthermore, it has been determined than by using 6 books, one can evaluate the performance of its LM with a confidence of 0.1%.

For the testing, two configurations have been considered: the "basic forward" one, and the one with "cut-off" (removal of less frequent words in the book).

Test book is "The history Peloponnesian War" of Thucydides. Results and corresponding hyperparameters can be seen in Table 4.2. As a conclusion, the best system here turns out to be the configuration with cut-off. It results in a gain in WER of 1.08%.

Method	Hidden Nodes	Model Weight	Insertion Penalty	Gain in WER
forward	5	0.3	0.8	0.62
forward/cut-off	5	0.3	0.8	1.08

Table 4.2 – Testing Results

Even if it also take into account books from training and validation set, all results can also be seen in Figure 4.26 and in box plots 4.27. They confirm that the cut-off improves the gain in WER, but also increases the global distribution (worst cases become worse, best cases become better).

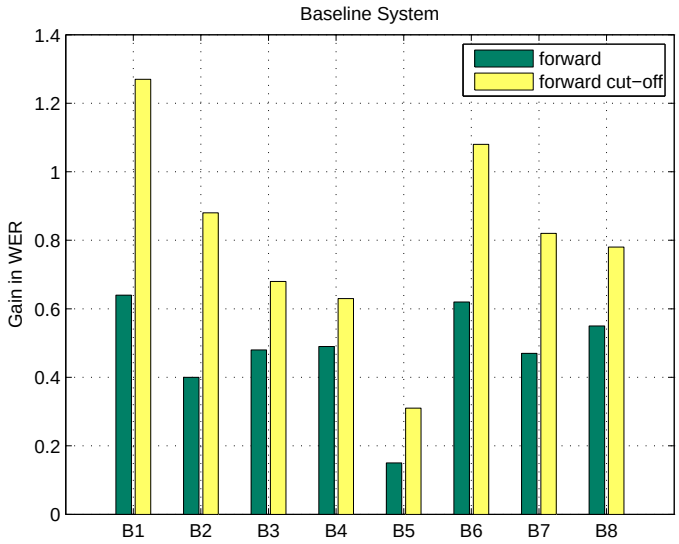


Figure 4.26 – Baseline System: Details

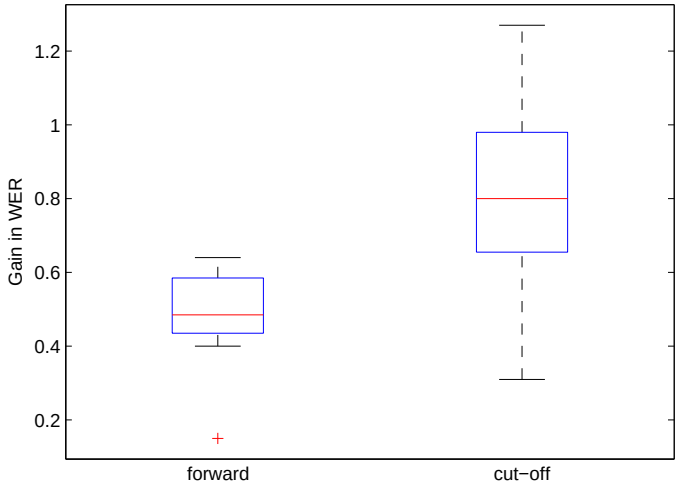


Figure 4.27 – Box Plots: Baseline System

On average over 8 books, we obtain a gain in WER of 0.48% in the forward case and of 0.81% forward case with cut-off.

5 System based on web-data augmentation

In the baseline system, the main limitation is due to the small amount of data. Indeed, with sets around 15k-45k words, RNN training is difficult. Therefore, this chapter is an attempt to cope with this issue. The main goal is to augment the size of the training set, with supplementary data taken from the World Wide Web (WWB).

5.1 Description

Due to its immense size, the WWB turns out to be a good resource for gathering data. Given a book transcript or a phone call transcription, it is however important to select data from the WWB which are close to the content of the transcript. This is why it has be chosen here to search for those data through adequate keywords. First of all, main keywords are extracted from the transcript. Those keywords should reflect the main topic of the book. Then, they are used as search queries. Finally, resulting WEB pages are selected and gathered in a new training set (see Figure 5.1).

5.1.1 State of the art in web-data augmentation

WEB-data augmentation involves two main challenges: extraction of relevant keywords, and selection of relevant text from the retrieved pages. Several different methods have been reported throughout the literature.

As mentioned in [8], keywords extraction can be divided into two steps: the extraction of candidate keywords itself, and the selection of the best ones. The extraction phase can be made by using heuristic rules, such as isolating N-grams for instance. The selection phase can be based for instance on Graph-Based Ranking methods, or on Language Modeling based methods.

One instantiation of the Language Modeling approach can be found in [1]. In this paper, three language models are used: a topic specific model, a background model and a rejection model.

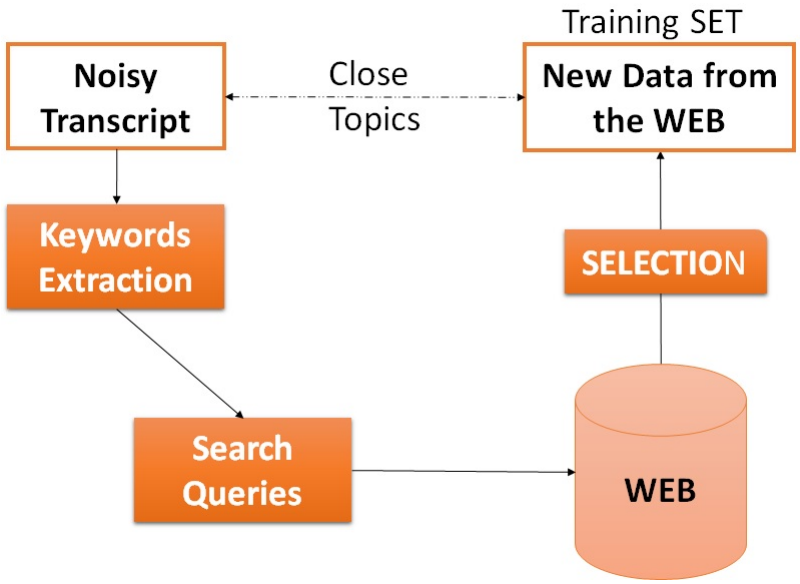


Figure 5.1 – Main Steps for the WEB-Data Augmentation System

A relative entropy measure enables to generate query strings, which are then used to download documents from the web. Based on the TFIDF/Naive Bayes methods, those documents are then classified into "topic specific" data, "background" data and "rejection" data. Language models are then updated, and new queries are generated in an iterative way.

A similar approach, used in [20], consists in creating topic clusters and training language models on each one of them. For the specific document, topic detection is then performed by comparison with the different clusters. Finally a new language model is built by interpolation with the relevant pre-trained language models.

The Graph-Based Ranking method measures the importance of a candidate by quantifying its relation with the other candidates. The more a keyword is related with other potential keywords, the better. This relatedness can be represented graphically. One of the most famous Graph-Based Ranking instantiation is the TextRank model (see [14]).

In this thesis, the Rapid Automatic Keyword Extraction (RAKE) toolkit is chosen for keyword extraction. Indeed, this method is computationally more efficient than TextRank, with a higher precision and similar recall. Furthermore, this toolkit is easy to use and the extraction process quite fast. Its exact principle is described in the next subsection.

For the document selection process, BLEU score metric is chosen, based on the paper "Rapid Language Model Development using External Resources for New Spoken Dialog Domains" (Sarikaya, 2005) [26]. The method developed there enables indeed to catch lexical as well as stylistic similarities, which is quite relevant for our task. BLEU score metric is also described in more details in this section.

5.1.2 RAKE: Rapid automatic keyword extraction

RAKE is a unsupervised, domain-independent, language independent method developed by Stuart Rose, Dave Engel, Nick Cramer and Wendy Cowley [25]. It requires as input a list of stopwords, phrases delimiters and words delimiters, and generates a set of candidate keywords as output.

RAKE algorithm can be divided into three steps: the parsing, the scoring and the extraction.

First, the parsing step splits the document into an array of words, by using the words' delimiters. Then, it identifies sequences of words, separated by stop words or phrase delimiters, forming the "candidate keywords".

For instance, the sentence "Compatibility of systems of linear constraints over the set of natural numbers" is tokenized into "Compatibility - systems - linear constraints - set - natural numbers".

Then, the scoring step assigns to each "candidate keyword" a score. This score transcribes the relevancy of the keyword. Its computation is based on two word level metrics: the frequency and the degree. The frequency corresponds to the number of time the word appears in the document. The degree takes into account the frequency of the word, and also the number of times the word co-occurs, i.e. appears with an other keyword. Thus, the degree favours words occurring often and in longer context keyword (indeed, they co-occur with more words than in shorter context keyword). Each word in the "candidate sentence" is given a score: $deg(w)/freq(w)$.

Finally, the score of a candidate keyword is defined as the sum of the scores of its member words. The degree and frequency of each word can be computed by using of graph of co-occurrences. The frequencies are read in the diagonal, and the degrees are the sum of each row.

For instance, let's consider the part of sentence "Compatibility of systems of linear constraints..." (taken from [25]). We have seen that its parsing leads to "Compatibility-systems-linear constraints". Table 5.1 shows its co-occurrence graph.

words	compatibility	systems	linear	constraints
compatibility	1			
systems		1		
linear			1	1
constraints			1	1

Table 5.1 – Part of Co-occurrence Graph

For those words, scores can be seen in Table 5.2

words	compatibility	systems	linear	constraints
deg	1	1	2	2
freq	1	1	1	1
deg/freq	1	1	2	2

Table 5.2 – Scores calculated from Co-occurrence Graph

Here, "linear constraints" scores 2, whereas "compatibility" scores 1. In this particular case, "linear constraints" is thus a better keyword than "compatibility".

In practice, the top T candidate keywords are selected. T often corresponds to the third of number of words in the co-occurrence graph.

5.1.3 BLEU Score Selection

The other main challenge in web-data augmentation is text selection from the retrieved pages. For this purpose, an approach in [26] makes use of the BiLingual Evaluation Understudy (BLEU) similarity measure. This metric is taken from the Machine Translation (MT) problem.

Basically, the BLEU score metric compares a "candidate sentence" with a set of "reference sentences". In our case, the candidate sentence is taken from the in-domain data, i.e. the noisy book. The reference sentences are taken from one particular retrieved web site. More precisely, the reference sentences are all the utterances from this web site containing at least one of the "content words" of the candidate sentence. Here, the definition of "content words" is restricted to common nouns and verbs.

Then, BLEU score metric compares N-grams of the candidate sentence with N-grams of the references. This comparison is done by using a " modified precision" metric (see [22]).

Let's give an example for modified precision computation in the unigram case:

Candidate: *the the the the cat sleeps*
Ref 1: *the cat eats*
Ref 2: *the cat is in the garden*

Step 1

First of all, each candidate word is associated with its maximum number of appearances in one single reference. Let's call this number T.

$$T_{the} = 2$$
$$T_{cat} = 1$$
$$T_{sleeps} = 0$$

Step 2

Then, each word in the candidate sentence is clipped by T. "Clipped" means its initial number of appearances N in the candidate sentence is reduced to the minimum between T and N.

Thus, the "clipped" candidate sentence is: *the the cat*

Step 3

Finally, the modified unigram precision is defined as the number of words in the "clipped" candidate divided by the number of words in the initial "candidate".

Here: Modified Unigram Precision = 3/6 = 1/2

This procedure can be extended to any order of modified N-gram precisions.

From a mathematical point of view, the BLEU score metric can be formulated as follows:

$$BLEU = BP * exp(\sum_{n=1}^{N} w_n log p_n) \tag{5.1}$$

N is the maximum N-gram order considered.
w_n is a weight defined as: $w_n = 1/N$
p_n is the modified N-gram precision.

BP is a brevity penalty. Thanks to it, a candidate sentence should match the references in length.

It is defined as follows:

$$BP = \begin{cases} 1 & \text{if } c > r \\ exp(1 - r/c) & \text{if } c \leq r \end{cases} \tag{5.2}$$

c is the candidate lengths
r is the closest distance between the candidate and the references.

To sum up, BLEU metric enables to quantify how close a web site and the noisy book are.

This similarity is based on lexical criteria (N-grams matches) but also on stylistic criteria (for instance length of the sentences). Its value is between 0 and 1, 1 meaning a perfect match.

From a practical point of view, in this thesis, BLEU score is implemented by using the toolkit NLTK. For one noisy book, N candidate sentences are selected. Here, N corresponds to half the total number of utterances in the book. Then, BLEU score is applied to each candidate sentence and a set of references from a web page. As a result, a list of BLEU scores is obtained. Then, the final score between the noisy book and the considered web site is computed as the average value of this list.

For the selection process, only web sites with a score higher than a specific threshold are selected. To set this threshold, BLEU score between some noisy books and latin texts have been computed. These "random" matches give values around 0.2. After some tuning trials, the threshold is set to 0.6, which is three times more than a random match.

Instead of selecting web pages as whole documents, only specific set of reference sentences within each web page might also be selected. The advantage is that the selection is more refined, since it is at utterance level.

5.2 Experiments and Results

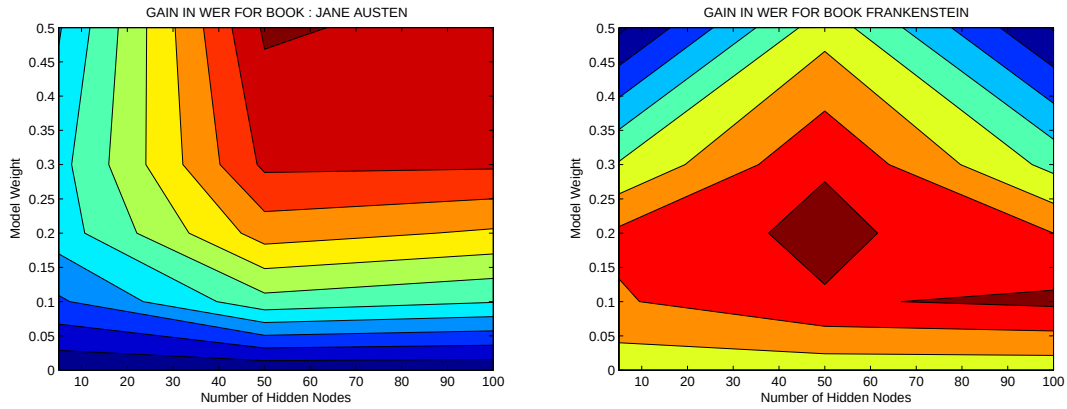
For each book to correct, a training set is built by downloading data from the web. Let us consider one book: first of all, keywords extraction is performed. Around 30 keywords are extracted. Then, pages are taken from the web. For each keyword, 7 pages are considered. All those external pages are concatenated into one training file. Thereafter, this size of the file is reduced to 0.5M words (each experiment on different books are in that manner comparable). Finally, the training file is post-processed (removal of frequent stop characters or strange punctuation).

5.2.1 Interpolation RNNLM and ASR LM

As with the baseline system, the most simple case here is to interpolate the RNNLM trained on the web data with the ASR LM. Hyperparameters here are, as in the previous chapter, the number of hidden nodes and the model weight.

Grid searches are made on different books. The ranges are [0.0; 0.1; 0.3; 0.5; 1.0] for the model weight and [5; 50; 100] for the hidden nodes. Results for two of them are shown in Figure 5.2.

These two plots show that 50 nodes seem to lead to good performaces. For instance, by considering a vocabulary size of 2k words and 50 hidden nodes, the total number of parameters is: $2 * 50 * 2000 = 200000$. Furthermore, the training set contains 0.5M words. Thus, it means that each parameter can "see" more than twice each vocabulary word.



(a) Gain in WER: "Jane Austen" Book

(b) Gain in WER: "Frankenstein" Book

Figure 5.2 – Hyperparameter Tuning: Web-Augmented System

The impact of the model weight is more difficult to analyze. Indeed, it seems to depend strongly on the book. For instance, 0.3 appears as a good value for the book "Frankenstein", whereas even 0.5 is insufficient for "Jane Austen". Actually, the impact is relatively constant after 0.5, as can be seen in Figure 5.3.

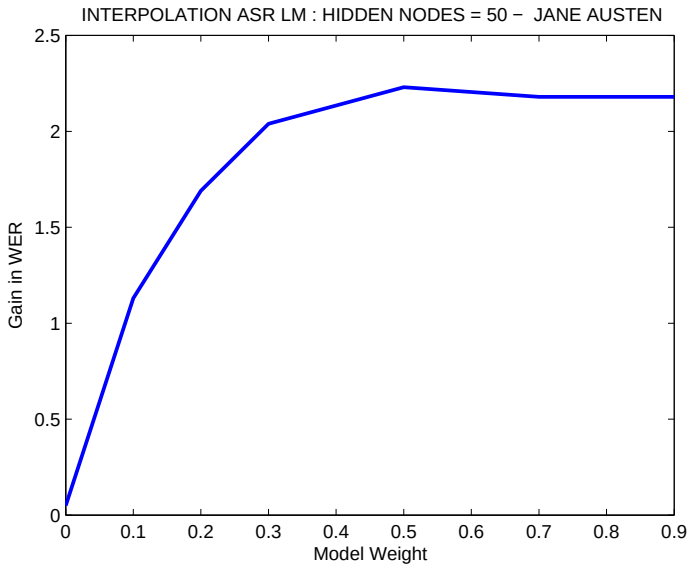


Figure 5.3 – Model Weight Influence - Details

5.2.2 Grid Search: Average over 6 Books

In this section, results concerning the hyperparameter optimization are presented. Each time, they correspond to an average over 6 books. The insertion penalty is set at 0.8.

In Figure 5.4, hidden nodes influence are shown (model weight fixed at 0.3). It seems like the more hidden nodes, the better. This makes sense: the training set is larger and needs thus a higher amount of parameter to learn efficiently. It could be expected to obtain even better results after 100 hidden nodes.

In Figure 5.5, the model weight influence is shown. Again 0.3 seems to be a good choice. It should be reminded however that the hyperparameters are really sensitive to the book which is considered. For instance, as shown in the previous subsection, "Jane Austen" book requires a model weight rather around 0.5.

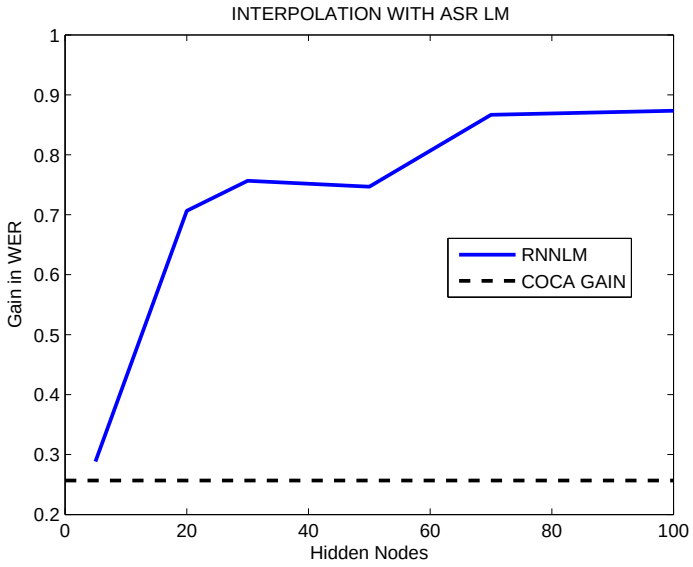


Figure 5.4 – Hidden Nodes Influence

5.2.3 Are 6 books enough to fix the hyperparameters?

As in the previous chapter, the following question arises: how many books are needed to fix the hyperparameters? Are 6 Books enough? Will the WER of the test book deviate too much from its optimal value? To answer these questions, the same procedure as the one followed for the baseline system is employed. For more details, see Chapter 4, Section 4.2, Subsection 4.2.6.

Unlike the baseline system, deviation from optimal WER is in this experiment quite high. Indeed,it reaches values around 0.3%. 6 books are thus not sufficient to fix the hyperparameters: performances depend too much on the books. Obviously, grid searches on other books are required. However, since RNNLM training is quite time-consuming with large sets, it has not been possible to perform enough grid searches in the scope of this thesis.

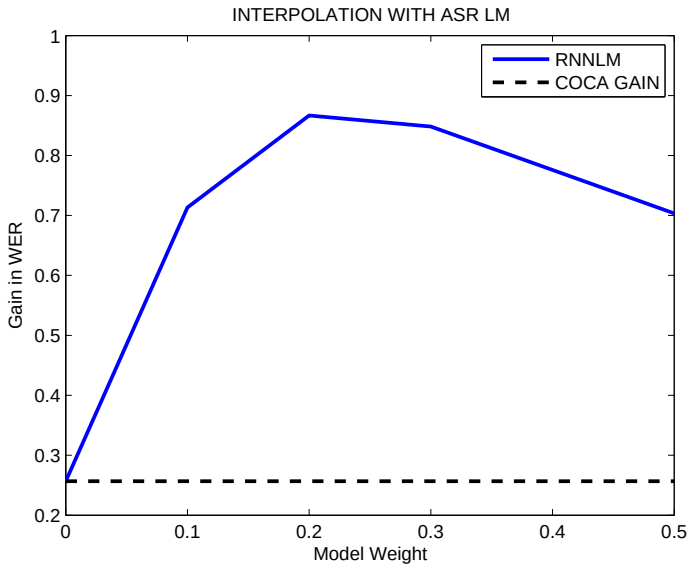


Figure 5.5 – Model Weight Influence

5.2.4 Comparison with Baseline system

Compared to the baseline system (see Chapter 4, Section 4.2), results are significantly improved by adding external data.

This can be seen in Figure 5.6, where hyperparameters used for the method based on the book are: 5 hidden nodes, a model weight of 0.3, an insertion penalty of 0.8 and hyperparameters used for the method based on the web-data are: 50 hidden nodes, a model weight of 0.3, and an insertion penalty of 0.8.

If the average over 8 books for the gain in WER results is increased (from 0.81% in the cut-off case to 0.94% in the web case), we can see however, in the box plots in 5.7, that the dispersion in the web case is also more spread than in the cut-off case.

To sum up: a gain in WER can be observed in most cases, but this gain still stays difficult to quantify precisely. Indeed, some corrections are really helped by web-data (for instance the Book 2 and the Book 8), whereas others are barely affected (Book 4) or even deterred (Book 7, Book 1).

A deeper look at Book 1 and Book 7 reveals that their corresponding web-data are polluted by "bad" web sites: for instance, some of them contain parts in other languages, or out of scope subjects ("bad" keywords). Obviously, a selection process is required.

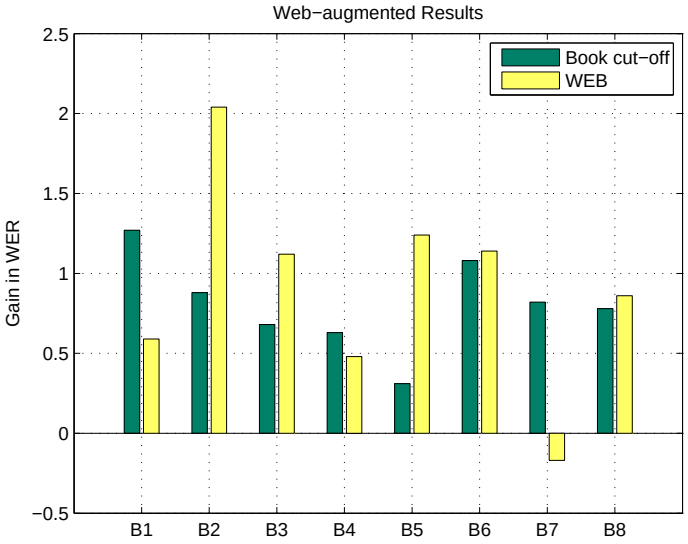


Figure 5.6 – Comparison Baseline System and Web-Augmented Data System

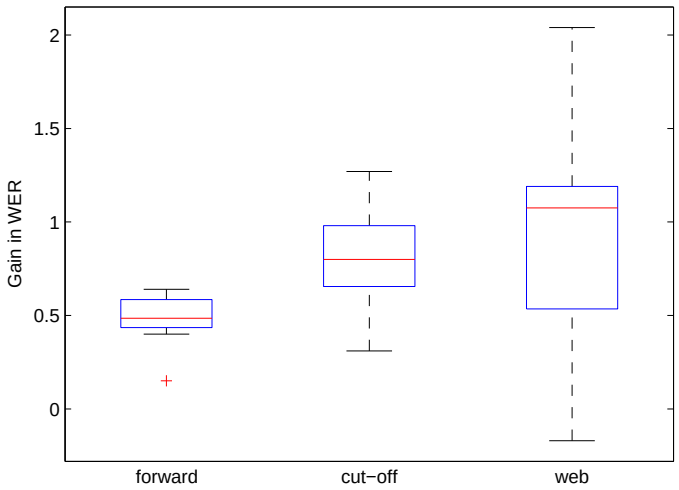


Figure 5.7 – Box Plots: with Web System

5.2.5 Results with BLEU score

As explained in Section 5.1, Subsection 5.1.3, the method used here for the selection process is the BLEU Score Metric. The global process employed in this subsection is explained in Figure 5.8.

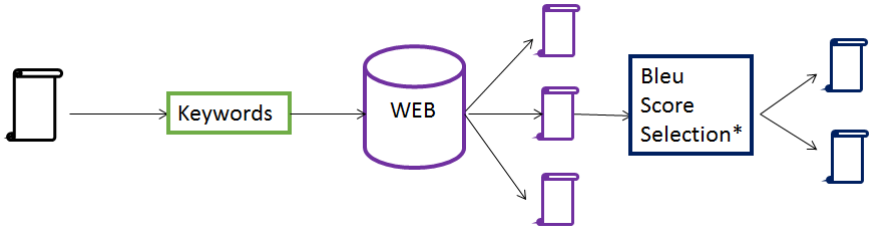


Figure 5.8 – Global Process for Web-Augmented System with BLEU Selection

Retrieved web pages scoring less than 0.6 are discarded. At the end, a training file of 0.5M is obtained. Results with the BLEU selected web pages are compared with those of the previous web system without any selection in Figure 5.9. Hyperparameters chosen here are: 50 hidden nodes, a model weight of 0.3, and insertion penalty of 0.8.

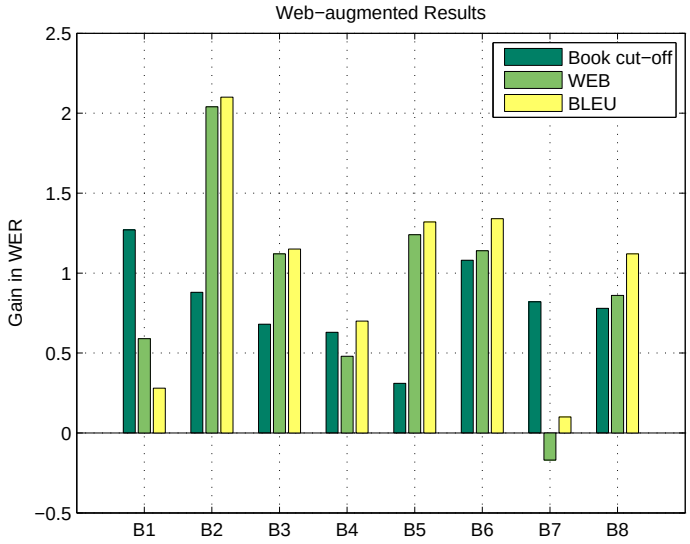


Figure 5.9 – Comparison: with and without BLEU Selection

Bleu score looks helpful and has a positive impact on almost all books. Even Book 7, which resulted in bad performances in the previous web-augmented system, is improved with the BLEU score. Indeed, strange web pages are filtered. Average results over 8 books is slighlty improved (gain in WER increased from 0.94% to 1.01%). We can also see in the box plots in 5.10 that the dispersion, compared to the web case, is more concentrated. Worst cases in the web case are improved with bleu selection.

However, this is not the case for Book 1, for which results are even worse than before. Indeed, some web documents contain parts close to the book, and other parts totally different. This suggests to perform BLEU selection not at document level, but at utterance level.

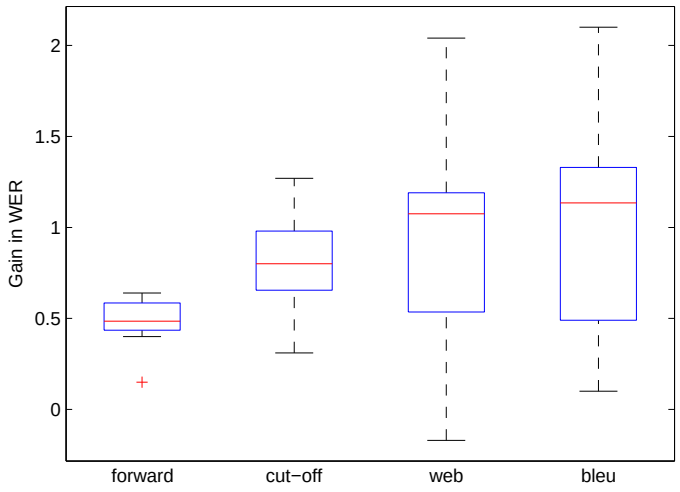


Figure 5.10 – Box Plots: with Bleu Selection

This experiment is conducted and results are shown in Figure 5.11. Hyperparameters are set at the same values as before (50 hidden nodes, model weight of 0.3 and insertion penalty of 0.8).

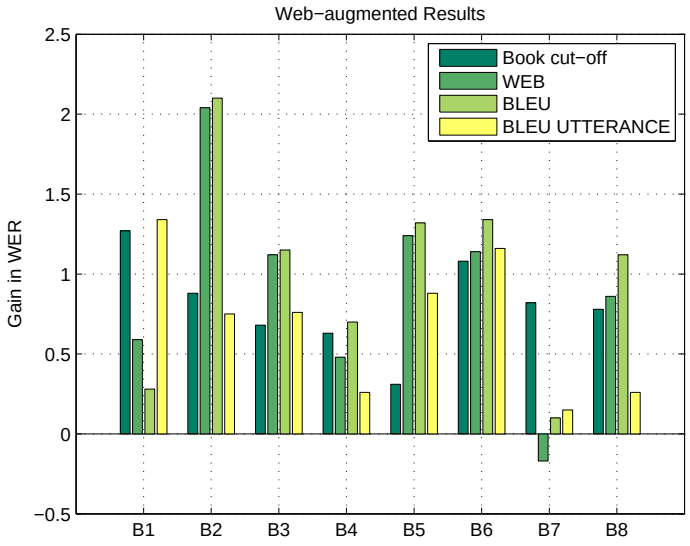


Figure 5.11 – Global Comparison

Surprisingly, results are not as good as expected. Average results over 8 books drops from 1.01% to 0.70% for the gain in WER. This method only helps Book 7 and Book 1. Indeed, in these cases, long articles from the web are taken: some of them contain parts really close to

the in-domain data as well as parts very distinct from it. Thus, selection of whole pages deters the gain in WER, whereas selection of single utterances improves it a lot.

On the contrary, for the other books, this process does not perform well: web articles are shorter, and utterances are thus extracted out of their context. It might explain an increase in WER compared to the other techniques. This can be seen in the box plots 5.12, where the median is significantly lower in the bleu selection at utterance level than at document level.

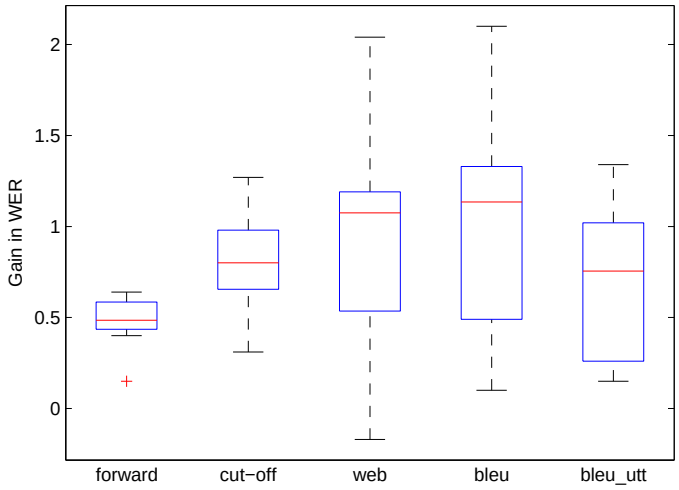


Figure 5.12 – Box Plots: with Bleu Selection at Utterance Level

5.2.6 Testing

Results for the testing book "The history of Peloponnesian War" are reported in Table 5.3. However, as already mentioned, it does not make much sense to use the hyperparameters fixed with only 6 books for the web-augmented system, since the deviation from the optimal WER is too high (around 0.3).

Method	Hidden Nodes	Model Weight	Insertion Penalty	Gain in WER
forward	5	0.3	0.8	0.62
forward/cut-off	5	0.3	0.8	1.08
WEB	50	0.3	0.8	1.14
WEB/BLEU	50	0.3	0.8	1.34

Table 5.3 – Testing Results

6 Adaptation System

In the previous chapters, two kinds of system are investigated.

The first one tries to catch information directly from the noisy book. It has the advantage to be really specific, and targets directly the in-domain data. However, it may learn errors, and is affected by a lack of data.

The second one has an opposite strategy. Instead of using data from the book, it searches for external data on the web. Of course, these data have to stay close the book, which is ensured by a keyword extraction and a selection process. Compared to the first system, results are better. Obviously, the selection process could be (and has to be) improved to further increase the WER.

This chapter makes a link between the independent system (based on the web data) and the in-domain system (based on the book). Two approaches are investigated: a naive interpolation between the independent and in-domain systems, and a method based on the Kullback-Leibler regularization.

6.1 Naive Interpolation

In this subsection, an interpolation between the RNNLM trained on the web data and the RNNLM trained on the book with frequency cut-off is made.

The resulting LM is computed as following:

$$SCORES_{final} = ModelWeight * [\phi * SCORES_{RNNLM-WEB} + (1-\phi) * SCORES_{RNNLM-BOOK}] + (1 - ModelWeight) * SCORES_{ASR-LM}$$

ϕ is the weight given to the RNNLM trained on the web data. The case $\phi = 1$ is equivalent to the web-augmented system. The case $\phi = 0$ is equivalent to the baseline system.

Results are shown in Figure 6.1. They are averaged over 6 books. For the web model, hyperparameters are set at: hidden nodes = 50, model weight = 0.3, insertion penalty = 0.8. For the book model, we consider the cut-off case, and the hyperparameters are set at: hidden nodes = 5, model weight = 0.3, insertion penalty = 0.8.

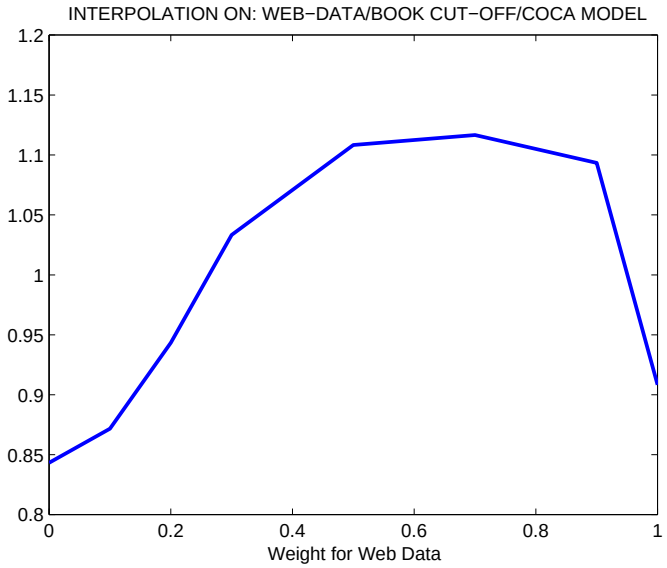


Figure 6.1 – Interpolation between Web-Augmented and Baseline Systems

Compared to the two previous approaches, this combination helps a lot. High weights for the the web data lead to the best gains. In details, results for each book can be seen in Figure 6.2.

It seems that this process behaves as if it was taking each time the best solution between training on web data and training on book data.

By looking at the box plots in Figure 6.3, the distribution is considerably reduced compared to the other methods, and the median slightly increased. Furthermore, the average results over 8 books is also improved, and reaches 1.14%.

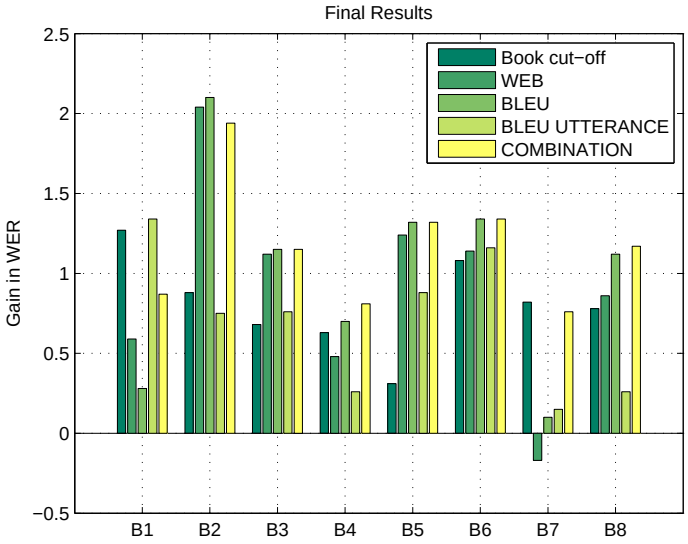


Figure 6.2 – Final Results

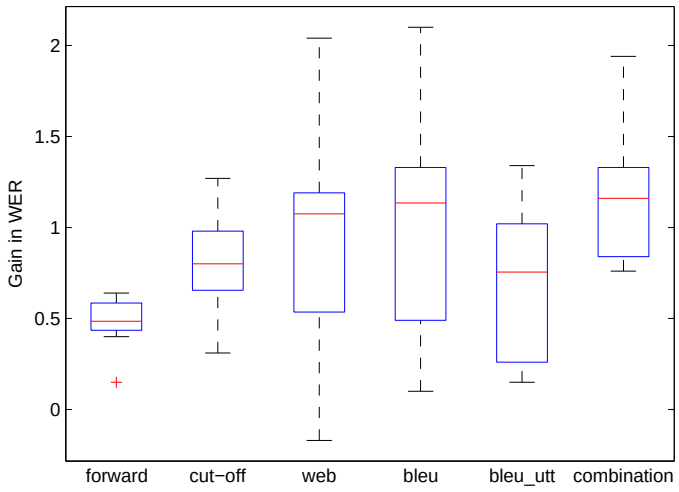


Figure 6.3 – Box Plots: with Final System

6.2 Kullback-Leibler Adaptation

6.2.1 Description

As explained in [27], an obvious approach to adapt a neural network trained on independent data to targeted data is to adjust its parameters by using the adaptation data. However,

during this process, the neural network may forget previously learned information on the independent system.

Thus, an other way of proceeding is developed: the Kullback-Leibler (KL)-divergence adaptation. The basic idea is to train the neural network on the adaptation data, but by still constraining it to stay close to the independent system.

From a mathematical point of view, this results in a change in the cost function:

$$\hat{D} = (1-\rho)\bar{D} + \rho\frac{1}{N}\sum_{t=1}^{N}\sum_{y=1}^{S} p^{SI}(y|x_t) * log\, p(y|x_t) \tag{6.1}$$

The probabilities coming from the independent system are expressed with $p^{SI}(y|x_t)$.

By rewriting this expression, this can be understood as a change in the target cost (see Figure 6.4). The new target probability is the following one:

$$p'_{target} = (1-\rho)\tilde{p}(y|x_t) + \rho * p_{SI}(y|x_t) \tag{6.2}$$

ρ is an hyperparameter to optimize. For $\rho = 0$, the neural newtork is only trained with the adaptation data. On the contrary, $\rho = 1$ means that no adaptation data is used, only the independent system is taken into account. For a small adaptation set, help will be needed from the independent model: a large ρ is required. On the contrary, for a large adaptation set, a small ρ should perform better.

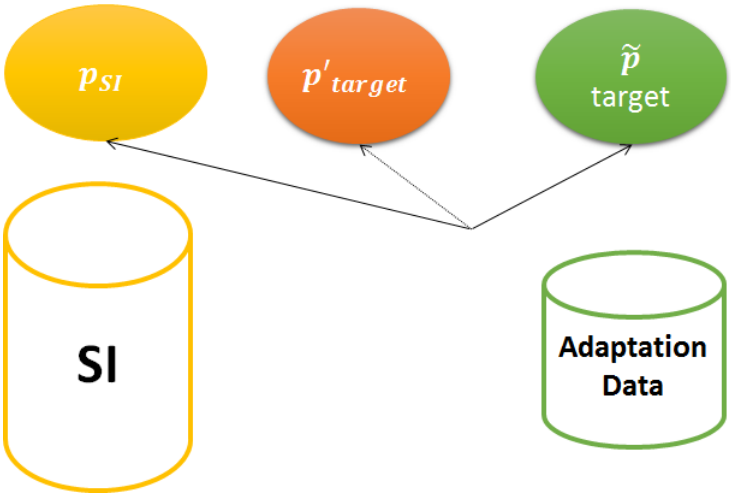


Figure 6.4 – Regularization with KL Divergence - Outline

In this thesis scope, the independent system corresponds to the RNNLM trained on the web data. The adaptation data correspond to the noisy book.

6.2.2 Results

Experiments are conducted on several books. Results for one of them are presented here. The others have a similar behaviour.

The hyperparameter tested is the weight ρ given to the independent system. Here, the independent system corresponds to the probabilities coming from the web model.

Furthermore, different sizes are investigated for the in-domain data. Training is thus performed on the whole training set, on 20 uterrances, 10 uterrances and only 5 utterances. The purpose is to investigate the impact of the KL divergence with respect to the amount of data.

ρ ranges in [0, 0.125, 0.2, 0.5]. For $\rho = 0$, baseline system is retrieved. Results can be seen in Figure 6.5.

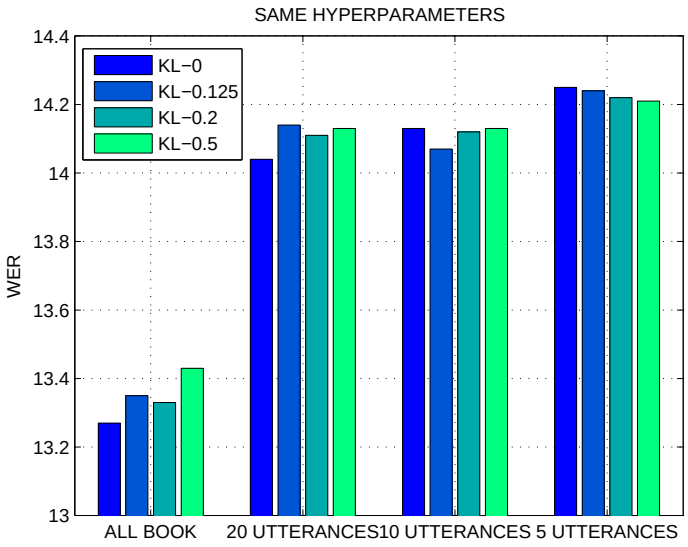


Figure 6.5 – Regularization with KL Divergence

It should be noted that the vertical axis does not represent the gain in WER, but directly the WER. Thus, the lower, the better.

Results are not as good as expected, since the KL regularization does not improve the gain in WER. However, with restricted amount of data, KL regularization has a positive impact. More precisely, high ρ helps even more with decreasing available data. It makes sense: if the in-domain model is too weak due to its restricted resources, more information is needed. This information is provided by the large independent system.

To conclude, KL-divergence adaptation helps in case of restricted amount of data.

6.3 Testing

In this subsection, results obtained on the test book ("The history Peloponnesian War" of Thucydides) are reported in the Table 6.1.

To conclude, best methods give in the test book a reduction of 1.34% of WER. This is obtained by training on the web data, selected with the BLEU score at a document level, and with the naive adaptation method.

Method	Hidden Nodes	Model Weight	Insertion Penalty	Gain in WER
forward	5	0.3	0.8	0.62
forward/cut-off	5	0.3	0.8	1.08
WEB	50	0.3	0.8	1.14
WEB/BLEU	50	0.3	0.8	1.34
Naive Adaptation	50	0.3	0.8	1.34

Table 6.1 – Testing Results

7 Conclusion

In this thesis, the problem of error correction in speech recognition transcripts has been investigated. More precisely, a context-based approach has been followed, attempting to catch information from proximity words to detect errors and correct them. In practice, extracts of transcripts of audiobooks have been used. They have the advantage to be language specific (due to the author's style) and context definite.

The method employed is based on language modeling and lattice rescoring. More precisely, it has been chosen to interpolate the COCA trigram with an RNNLM. This choice is justified by the fact that an RNNLM is very powerful, and can catch long dependencies within an utterance. Its main drawback concerns its training time for huge datasets. Since only small or middle sized databases were involved in this thesis, it turned out to be a good choice. In practice, the RNNLM of Mikolov has been chosen for the implementation.

The main challenge was here to find the best training set for the RNNLM. On the one hand, it has to be as close as possible to the book language. On the other hand, it must provide enough data for the training.

The first choice was to use the noisy book itself. Of course, errors may be learned. However, it is also the most specific kind of data one can uses. If the gain in WER is 0.62% for the test book, it is still limited, due to the restricted amount of data.

Moreover, throughout the experiments lead in this work, it seems that RNNLM performs better than book-specific N-gram models. Interpolating with a bigram model does not seem to be helpful either.

On the contrary, improvements have been obtained by applying a cut-off to the training set, removing all the words appearing less than three times in the document. Intuitively, it enables the RNNLM to learn only recurring words, which may be the most important ones. The less frequent ones do not transcribe much the language used, or might even be errors. Consequently, removing them enables better performances.

Finally, an interpolation between a forward trained and backward trained RNNLM has been explored. It results in slight reductions of the WER, and seems consequently to have a positive impact.

The second training set is composed of data taken from the web. First of all, keywords have been extracted from the noisy book, by using the RAKE toolkit . Then, data have been gathered from the web. Finally, they have been selectively concatenated into one training set, with a size of 0.5M words. In terms of WER, 1.14% of amelioration has been obtained on the test book.

This configuration revealed to be really efficient compared to the previous one. Of course, this is mainly due to the database size. However, further improvements need to be done, especially concerning the selection process of web pages.

Finally, an interpolation between both models (trained on the web data and on the book) has been investigated. It turned out to be the most efficient method, and reduction in WER of more than 1.34% has been obtained on the test book. An other kind of adaptation method, based on a Kullback-Leibler divergence regularization while training, has also been tested. This kind of process reveals to be efficient for very small amount of in-domain data.

To conclude, static information from the book as well as dynamic data from the web seem to be useful to perform an adequate language modeling. Besides, the total amount of data turned out to be a determining factor for an efficient RNNLM training. In this thesis scope, it was the most important limit. Another limit was related to the selection process, which revealed to be sometimes inefficient.

For future work, it may be necessary to investigate a better selection process for the retrieved web-pages. For instance, language models techniques could be used. Alternative methods more related to interpolation of topic-specific language models and topic detection could also be explored. Finally, if the Kullback-Leibleir divergence has been implemented for the adaptation process, it could also be used for the selection process as well.

Glossary

ASR Automatic Speech Recognition. i, vi, 1, 3, 4, 23–25, 27–30, 32–34, 41, 42, 52

BLEU BiLingual Evaluation Understudy. 50

BP Backpropagation. 20

BPTT Backpropagation Through Time. 20

DNN Deep Neural Network. 3

FSA Finite State Acceptor. 13, 14

FST Finite State Transducer. 13, 16

GMM Gaussian Mixture Model. 3

HMM Hidden Markov Model. 7, 14

KL Kullback-Leibler. 64

LM Language Model. 16, 17, 26–30, 32, 41, 42, 45, 52, 61

LSA Latent Semantic Analysis. 3

MSE Mean Square Error. 18

MT Machine Translation. 50

OOV Out Of Vocabulary. vii, 9, 29, 30

RAKE Rapid Automatic Keyword Extraction. 48, 49, 68

RNN Recurrent Neural Networks. i, vii, 4, 17–20, 26, 27, 31, 32, 37, 47

RNNLM Recurrent Neural Network Language Model. vii, 3, 11, 17, 24, 26, 27, 29, 31–34, 37–39, 41, 42, 45, 52, 55, 61, 63, 67, 68

SGD Stochastic Gradient Descent. 18

WER Word Error Rate. i, viii, 3, 11–13, 24–26, 28, 30–32, 45, 46, 55, 57, 58, 61, 64, 67, 68

WWB World Wide Web. 47

A Books General Details

ID-Thesis	ID-Librispeech	Title	Author
B1	84	Frankenstein	Mary Shelley
B2	121	Northanger Abbey	Jane Austen
B3	135	Les Misérables	Victor Hugo
B4	640	Story of the Emperor's new clothes	Hans Christian Andersen
B5	820	Edison : His Life and Inventions	Frank Lewis Dyer
B6	7142	The History of Peloponnesian War	Thucydides
B7	12897	The Man Who Laughs	Victor Hugo
B8	360	What is Property?	Proudhon

Table A.1 – Books Details

ID-Thesis	ID-Librispeech	Extract Size	Vocabulary Size	Vocabulary Size Cut-off
B1	84	19982	3600	728
B2	121	20758	3497	686
B3	135	45728	7132	1425
B4	640	16067	2309	582
B5	820	18637	3710	762
B6	7142	17582	3398	599
B7	12587	17185	4182	577
B8	360	17487	3525	612

Table A.2 – Books Sizes

ID-Thesis	ID-Librispeech	WER without Correction
B1	84	14.95
B2	121	14.94
B3	135	13.49
B4	640	9.95
B5	820	10.41
B6	7142	14.07
B7	12587	17.23
B8	360	12.53

Table A.3 – WER without any Correction Process

ID-Thesis	ID-Librispeech	Training Size - WEB	Vocabulary Size
B1	84	545983	68883
B2	121	547948	49142
B3	135	611701	30670
B4	640	668981	61883
B5	820	528461	33050
B6	7142	407167	31585
B7	12587	620715	43993
B8	360	625036	38550

Table A.4 – Training Set Sizes - Web Data

ID-Thesis	ID-Librispeech	Training Size - WEB	Vocabulary Size
B1	84	413779	36757
B2	121	473276	45456
B3	135	593533	29266
B4	640	556303	49308
B5	820	486662	30265
B6	7142	502592	40298
B7	12587	562652	34836
B8	360	592592	32010

Table A.5 – Training Set Sizes - Web Data BLEU Selection

ID-Thesis	ID-Librispeech	Training Size - WEB	Vocabulary Size
B1	84	687136	48687
B2	121	504694	45027
B3	135	600831	30492
B4	640	604119	58514
B5	820	502441	32534
B6	7142	512276	43453
B7	12587	566958	43829
B8	360	584442	37771

Table A.6 – Training Set Sizes - Web Data BLEU Selection - Utterance level

Bibliography

[1] BUILDING TOPIC SPECIFIC LANGUAGE MODELS FROM WEBDATA USING COMPETITIVE MODELS A. Sethy , Panayiotis G . Georgiou , Shrikanth Narayanan Speech Analysis and Interpretation Lab Department of Electrical Engineering-Systems Viterbi School of Engineering U.

[2] W. Allen and As Rus. Chapter 4. 2015.

[3] S. Atwell, E. S.; Elliot. Dealing with ill-formed English text. *The computational analysis of English: a corpus-based approach*, (January), 1987.

[4] Ravenswood Ave and Menlo Park. ASR ERROR DETECTION USING RECURRENT NEURAL NETWORK LANGUAGE MODEL AND COMPLEMENTARY ASR Yik-Cheung Tam , Yun Lei , Jing Zheng and Wen Wang Speech Technology and Research Laboratory , SRI International. pages 2331–2335, 2014.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A Neural Probabilistic Language Model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[6] Ankur Gandhe, Florian Metze, and Ian Lane. Neural Network Language Models for Low Resource Languages. *Interspeech-2014*, (3):2615–2619, 2014.

[7] Andrew R Golding. A Bayesian hybrid method for context-sensitive spelling correction, 1996.

[8] Ks Hasan and Vincent Ng. Automatic Keyphrase Extraction: A Survey of the State of the Art. *Utdallas.Edu*, 2011.

[9] Graeme Hirst and Alexander Budanitsky. Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, 11(December 2003):87–111, 2005.

[10] Graeme Hirst and David St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet - An Electronic Lexical Database*, (April):305–332, 1998.

[11] Mp Jones. Contextual spelling correction using latent semantic analysis. *Proceedings of the fifth conference on*, pages 166–173, 1997.

[12] Karen Kukich. Technique for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[13] Damereau Mays and Mercer. Context Based Spelling Correction. pages 187–194, 1991.

[14] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into texts. *Proceedings of EMNLP*, 4(4):404–411, 2004.

[15] T Mikolov, M Karafiat, L Burget, J Cernocky, and S Khudanpur. Recurrent Neural Network based Language Model. *Interspeech*, (September):1045–1048, 2010.

[16] Tomas Mikolov. Statistical Language Models Based on Neural Networks. *PhD thesis*, pages 1–129, 2012.

[17] Tomáš Mikolov, Stefan Kombrink, Anoop Deoras, Lukáš Burget, and Jan Černocký. RNNLM — Recurrent Neural Network Language Modeling Toolkit. *Proceedings of ASRU 2011*, pages 1–4, 2011.

[18] Ritika Mishra, Navjot Kaur, and A Dictionary Lookup Technique. A Survey of Spelling Error Detection and Correction Techniques. 4:372–374, 2013.

[19] Mehryar Mohri, New York, and Michael Riley. SPEECH RECOGNITION WITH WEIGHTED FINITE-STATE TRANSDUCERS 251 Mercer Street University of Pennsylvania 200 South 33rd Street Philadelphia , PA 19104 76 Ninth Avenue New York , NY 10011. *Speech Communication*, pages 1–31, 2000.

[20] Tim Ng, Mari Ostendorf, and Ivan Bulyko. Web-Data Augmented Language Models for Mandarin Conversational Speech Recognition. *ICASSP (International Conference on Acoustics, Speech, and Signal Processing)*, 1:589–592, 2005.

[21] Vassil Panayotov, Guoguo Chen, Daniel Povey, Sanjeev Khudanpur, and The Johns. Librispeech : An ASR Corpus Based On Public Domain Audio Books. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 2015.

[22] Kishore Papineni, Salim Roukos, Todd Ward, and Wj Zhu. BLEU: a method for automatic evaluation of machine translation. ... *of the 40Th Annual Meeting on* ..., (July):311–318, 2002.

[23] Jennifer Pedler. Computer Correction of Real-word Spelling Errors in Dyslexic Text. *ReCALL*, page 239, 2007.

[24] D. Povey, a. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely. The kaldi speech recognition toolkit. *IEEE 2011 workshop on Automatic Speech Recognition and Understanding*, pages 1–4, 2011.

[25] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. CO RI Automatic keyword extraction. *Text Mining: Applications and Theory*, pages 1—-277, 2010.

[26] R. Sarikaya and a. Gravano. Rapid Language Model Development Using External Resources for New Spoken Dialog Domains. *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, pages 573–576, 2005.

[27] Dong Yu, Kaisheng Yao, Hang Su, Gang Li, and Frank Seide. KL-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. *Icassp*, pages 7893–7897, 2013.