TR-IT-0270
Statistical language modeling based on variable length dependencies

Sabine Deligne

August, 1998

Internal Use only

Contents

Chapter 1

Introduction

In this report, we present a stochastic language modeling tool which aims at retrieving variable-length phrases (multigrams), assuming bigram dependencies between them. The phrase retrieval can be intermixed with a phrase clustering procedure, so that the language data are iteratively structured at both a paradigmatic and a syntagmatic level in a fully integrated way. Perplexity results on ATR travel arrangement data with a bi-multigram model (assuming bigram correlations between the phrases) come very close to the trigram scores with a reduced number of entries in the language model. Speech recognition scores are ranked accordingly. Also the ability of the class version of the model to merge semantically related phrases into a common class is illustrated.

The report is organized in the following way. First we review the field of statistical language modeling (chapter 2) ; especially we propose a classification to characterize and distinguish between some of the most recent works on variable-length modeling. We then introduce the bimultigram model and we propose a training procedure to estimate a probability distribution of the phrases and to define a partition of the phrases according to an ML criterion (chapter 3). We also show how to interpolate class based and non class based bimultigram models. Finally, we report on perplexity experiments and on speech recognition experiments with the bimultigram model and with conventional n-gram models, and we show some examples of the classes obtained in our experiments (chapter 4). Further details on the estimation of the probability distribution of the phrases are given in Appendix A. Also a notice of use of the toolkits "bimgramtk" and "interpoltk", which allow, respectively, to train bimultigram models, and to interpolate them is given in Appendix B.

Chapter 2

An overview of statistical language models

2.1 The conventional n-gram model

The most popular statistical language model is the n-gram model, where the assumption is made that each word depends on the $(n-1)$ previous words. For instance, the 3-gram likelihood of the string "a b c d" is computed as:

$$\mathcal{L}_{3-gram}(abcd) = p(a \mid \#)\ p(b \mid a)\ p(c \mid ab)\ p(d \mid bc)$$

The Maximum Likelihood (ML) estimates of the n-gram probabilities are computed as the relative counts of the co-occurrences of the words in a training database. Because the size of the training database is necessarily limited, many combinations of words cannot be observed, making it very difficult to collect reliable statistics.

One way to improve the robustness of the probability estimates, apart from smoothing them, is to reduce the dimensionality of the n-gram probability space by assigning the words to equivalence classes. In the class based n-gram model, the vocabulary is partitionned into a prespecified number of classes, and the likelihood of a sentence is computed by multiplying the transition probabilities between the classes to which the words belong, instead of the transition probabilities between the words. The discrimination between the words is usually made through the class conditionnal probability of the words. The class 3-gram likelihood of the string "a b c d" is computed as:

$$\begin{aligned}\mathcal{L}_{class\ 3-gram}(abcd) \;=\; & p(C_{q(a)} \mid \#)\, p(a \mid C_{q(a)})\ p(C_{q(b)} \mid C_{q(a)})\, p(b \mid C_{q(b)}) \\ & p(C_{q(c)} \mid C_{q(a)}C_{q(b)})\, p(c \mid C_{q(c)}) \\ & p(C_{q(d)} \mid C_{q(b)}C_{q(c)})\, p(d \mid C_{q(d)})\end{aligned}$$

Now, because the words belonging to the same classes are discriminated based on a unigram probability only, the class based models are usually not as accurate

as the word based models. To preserve the advantage of the accuracy of the word based model, while still taking benefit of the robustness of the class based model, the word and class estimates can be interpolated. A very common way is to use linear interpolation and to estimate the interpolation weights so as to maximize the likelihood of some held-out data [9]. More details about the interpolation of models are given in section 3.5.

2.2 Models with variable-length dependencies

2.2.1 Proposal of a Classification

There is currently an increasing interest in statistical language models, which, by contrast with the conventional n-gram models, aim at exploiting word-dependencies spanning over a variable number of words. Though all these models commonly relax the assumption of fixed-length dependency of the conventional n-gram model, they cover a wide variety of modeling assumptions and of parameter estimation frameworks. To help characterizing these models, we propose to classify them:

- depending on whether they are gram based or phrase based,
- depending on whether they are stochastic or deterministic,
- depending on the criterion they aim at optimizing, and on the optimization procedure (especially whether it is a heuristic or not).

2.2.2 Phrase based versus gram based models

In a gram based approach, models take into account variable-length dependencies by conditionning the probability of each word with a context of variable length. By contrast, in a phrase-based approach, sentences are structured into variable-length phrases and probabilities are assigned to phrases instead of words. The probability of each phrase may be conditionned by the preceding phrases, just the same way the probability of a word is conditionned by the preceding words in a gram based framework. In this report, we use brackets to discriminate between $p([abc])$, which refers to the probability of phrase $[abc]$ occurring, and $p(abc)$, the joint probability used in the gram based framework to refer to the probability of a, b, and c co-occuring. The main difference between the gram based and the phrase based approaches thus essentially lies in the fact that they assume different probability spaces. In the gram based approach, an event of the space of probability is the occurence of a word, whereas in the phrase based approach, an event of the space of probability is the occurence of a phrase. In the latter case, the notion of word no longer exists: instead, there are phrases which might be 1-word long. One consequence resulting from the choice of either a gram based framework, or of a phrase based framework is the

following:
In a gram based framework, the inequality

$$p(ab) \geq p(abc)$$

always holds, due to the fact that the constraint $\sum_X p(abX) = p(ab)$ must be satisfied in the probability space. In a phrase based framework, a similar inequality holds, but for phrases only, not for words. It is always the case that:

$$p([ab]) \geq p([ab][c])$$

since the constraint $\sum_{[X]} p([ab][X]) = p([ab])$ must be satisfied over the space of probability of the phrases. On the other hand, it may be the case that:

$$p([ab]) \leq p([abc])$$

2.2.3 Stochastic versus deterministic models

Regardless of whether they assume grams or phrases, models can be either deterministic or stochastic. In the stochastic approach, the ambiguity on the variable-length dependencies underlying a sentence is not solved: all possible hypothesis about how the dependencies underly the sentence are assigned a distinct likelihood value by the model. On the other hand, with a deterministic model, only one hypothesis is given a non zero likelihood value.
In a phrase based framework, non determinism is usually introduced via an ambiguity on the parse of the sentence into phrases. In practice, it means that even if $[abc]$ is registered as a phrase, the possibility of parsing the string abc as $[a]$ $[b]$ $[c]$, $[ab]$ $[c]$, or as $[a]$ $[bc]$ still remains, each of the parses being ranked according to a likelihood value. By contrast, in a deterministic approach, all co-occurences of a, b and c would be interpreted as an occurence of phrase $[abc]$.
In a gram based framework, non determinism may be introduced via an ambiguity on the length of the context conditionning the current word. These variable n-gram models are conveniently represented as stochastic automata, where the transition arcs between the states include, in addition to the usual n-gram transitions (backoff included), some so-called "escape" or "non-emitting" transitions which allow to reduce the size of the context without using backoff.

2.2.4 Optimization criterion and optimization procedure

Since statistical language models are used to help predicting each new word to come in a sentence, the parameters of a model are to be estimated so as to maximize its predictive capability. The predictive capability is evaluated by the perplexity measure, which is a function of the likelihood of the data on which it is computed. Therefore, the maximum likelihood (ML) criterion is a "natural" criterion to estimate the parameters of a language model. For instance, in the case of conventional n-grams, the ML estimates of the n-gram distribution are the relative frequencies of the n-gram counts. However, it is a well

knonw fact [25] that the ML criterion is prone to overlearning: optimizing the likelihood of some necessarily limited training data does not guarantee that the likelihood of any unseen but relevant data is also optimized.
To deal with this problem in the field of statistical language modeling, various criteria have been proposed to replace the ML criterion: the leaving-one-out likelihood, the mutual information, the Minimum Description Length, and the entropy.
The use of the ML criterion in a stochastic framework allows EM principled optimization procedures, for which convergence towards a (possibly local) optimum is theoretically guaranteed. The other criteria tend to reduce the risk of overlearning, but their optimization usually relies on heuristic procedures, like for instance greedy algorithms (e.g. word grouping via a greedy algorithm) for which convergence and optimality are not theoretically guaranteed.

2.3 Models with classes of phrases

Recently, class-phrase based models have gained some attention, but usually it assumes a previous classification of the words, be it based on grammatical part-of-speech or on data-driven classes. Typically, each word is first assigned a word-class label "$< \mathcal{C}_k >$", then variable-length phrases $[\mathcal{C}_{k_1}\mathcal{C}_{k_2}...\mathcal{C}_{k_l}]$ of word-class labels are retrieved, each of which leads to define a phrase-class label which can be denoted as "$< [\mathcal{C}_{k_1}\mathcal{C}_{k_2}...\mathcal{C}_{k_l}] >$". But in this approach only phrases of the same length can be assigned the same phrase-class label. For instance, the phrases "thank you for" and "thank you very much for" cannot be assigned the same class label. We will show, in chapter 3, that the class based bi-multigram model allows to address this limitation by directly clustering phrases instead of words. Other works allowing to overcome this limitation are the work just published in [21], and also the works combining a (possibly Stochastic) Context Free Grammar (SCFG) with the n-gram approach. In these works, the phrases usually result from a parse with a (S)CFG, so that they are "context free" phrases, and then n-gram probabilities between the phrase are estimated, thus introducing *a posteriori* a context dependency. Also, phrases can be assigned a class label based on the non-terminal corresponding to their derivation node. It results in a class phrase based model, where variable length phrases may be assigned the same class label. The introduction of a (S)CFG has the advantage of allowing to model the linguistic structure of the data, but on the other hand the derivation of the phrases is not optimized in view of optimizing a criterion related to the perplexity of the model, which is usually the goal in statistical language modeling. The main difficulty in optimizing the SCFG with respect to the n-gram framework is due to the computationnal complexity ; work in that direction has been reported in [13].

2.4 Classification of related bibliographical references

In this section, we give pointers on works on statistical language models exploiting variable-length dependencies which have been recently published. The references are characterized based on the classification presented in section 2.2.1.

Gram-based, deterministic, leaving-one-out likelihood criterion: [20] [27]

Gram-based, deterministic, Minimum Description Length criterion: [27]

Gram-based, stochastic, ML criterion: [22] [8]

Gram-based, stochastic, ML criterion, EM optimization: [24]

Phrase-based, deterministic, entropy criterion: [16] [17]

Phrase-based, deterministic, mutual information criterion: [28] [21]

Phrase-based, deterministic, leaving-one-out likelihood criterion: [23]

Phrase-based, stochastic, ML criterion, EM optimization: [4] [26] [5]

Class Phrase-based: [23] [20]

Class Phrase-based, with classes of variable-length phrases [21] [5]

Class Phrase-based, combining the use of a (S)CFG: [13] [19] [18]

Chapter 3

The bi-multigram model

3.1 Introduction

Using the classification proposed in chapter 2, the multigram model can be characterized as a stochastic phrase-based model, the parameters of which are estimated according to a likelihood criterion using an EM procedure. The multigram approach was introduced in [1], and in [4] it was used to derive variable-length phrases under the assumption of independence of the phrases. Various ways of theoretically releasing this assumption were given in [6]. More recently, experiments with 2-word multigrams embedded in a deterministic variable n-gram scheme were reported in [26].
In section 3.2, we further formulate a model with bigram (more generally $\overline{n}$-gram) dependencies between the phrases, by including a paradigmatic aspect which enables the clustering of variable-length phrases. It results in a stochastic class-phrase model, which, as shown in section 3.5, can be interpolated with the stochastic phrase model, in a similar way to deterministic approaches.

3.2 Formulation

In the multigram framework, the assumption is made that sentences result from the concatenation of variable-length phrases, called multigrams. The likelihood of a sentence is computed by summing the likelihood values of all possible segmentations of the sentence into phrases. The likelihood computation for any particular segmentation into phrases depends on the model assumed to describe the dependencies between the phrases. We call **bi-multigram model** the model where bigram dependencies are assumed between the phrases. In that case, the likelihood of a segmentation is computed by multiplying the transition probabilities between the phrases in this segmentation. For instance, by limiting to 3 words the maximal length of a phrase, the bi-multigram likelihood of the string "a b c d" is:

$$\sum \begin{cases} p([a] \mid \#)\ p([b] \mid [a])\ p([c] \mid [b])\ p([d] \mid [c]) \\ p([a] \mid \#)\ p([b] \mid [a])\ p([cd] \mid [b]) \\ p([a] \mid \#)\ p([bc] \mid [a])\ p([d] \mid [bc]) \\ p([a] \mid \#)\ p([bcd] \mid [a]) \\ p([ab] \mid \#)\ p([c] \mid [ab])\ p([d] \mid [c]) \\ p([ab] \mid \#)\ p([cd] \mid [ab]) \\ p([abc] \mid \#)\ p([d] \mid [abc]) \end{cases}$$

More generally, let W denote a string of words, and $\{S\}$ the set of possible segmentations on W. The likelihood of W is:

$$\mathcal{L}(W) = \sum_{S \in \{S\}} \mathcal{L}(W, S) \tag{3.1}$$

and, assuming $\overline{n}$-gram dependencies between the phrases (in the above example $\overline{n} = 2$), the likelihood of a segmentation S of W is:

$$\mathcal{L}\,(W, S) = \prod_{\tau} p(s_{(\tau)} \mid s_{(\tau-\overline{n}+1)} \cdots s_{(\tau-1)}) \tag{3.2}$$

with $s_{(\tau)}$ denoting the phrase of rank (τ) in the segmentation S. The model is thus fully defined by the set of $\overline{n}$-gram probabilities on the set $\{s_i\}_i$ of all the phrases which can be formed by combining 1, 2, ...up to n words of the vocabulary.

It is straightforward to define a class version of the model, where the set of phrases is partitionned into equivalence classes. Assuming that each phrase depends on the preceding phrases via its class only, the likelihood of a segmentation is computed by multiplying the transition probabilities between the classes and the class conditionnal probability of each phrase. Assuming again bigram dependencies between the classes, and denoting by q a class membership function, which specifies for each sequence s_i the class $\mathcal{C}_{q(s_i)}$ it belongs to, the likelihood of our exampled sentence "a b c d" is:

$$\sum \left\{ \begin{array}{l} p(\mathcal{C}_{q([a])} \mid \#)\, p([a] \mid \mathcal{C}_{q([a])})\, p(\mathcal{C}_{q([b])} \mid \mathcal{C}_{q([a])})\, p([b] \mid \mathcal{C}_{q([b])}) \\ p(\mathcal{C}_{q([c])} \mid \mathcal{C}_{q([b])})\, p([c] \mid \mathcal{C}_{q([c])})\, p(\mathcal{C}_{q([d])} \mid \mathcal{C}_{q([c])})\, p([d] \mid \mathcal{C}_{q([d])}) \\ \\ p(\mathcal{C}_{q([a])} \mid \#)\, p([a] \mid \mathcal{C}_{q([a])})\, p(\mathcal{C}_{q([b])} \mid \mathcal{C}_{q([a])})\, p([b] \mid \mathcal{C}_{q([b])}) \\ p(\mathcal{C}_{q([cd])} \mid \mathcal{C}_{q([b])})\, p([cd] \mid \mathcal{C}_{q([cd])}) \\ \\ p(\mathcal{C}_{q([a])} \mid \#)\, p([a] \mid \mathcal{C}_{q([a])})\, p(\mathcal{C}_{q([bc])} \mid \mathcal{C}_{q([bc])})\, p([a] \mid \mathcal{C}_{q([bc])}) \\ p(\mathcal{C}_{q([d])} \mid \mathcal{C}_{q([bc])})\, p([d] \mid \mathcal{C}_{q([d])}) \\ \\ p(\mathcal{C}_{q([a])} \mid \#)\, p([a] \mid \mathcal{C}_{q([a])})\, p(\mathcal{C}_{q([bcd])} \mid \mathcal{C}_{q([a])})\, p([bcd] \mid \mathcal{C}_{q([bcd])}) \\ \\ p(\mathcal{C}_{q([ab])} \mid \#)\, p([ab] \mid \mathcal{C}_{q([ab])})\, p(\mathcal{C}_{q([c])} \mid \mathcal{C}_{q([ab])})\, p([c] \mid \mathcal{C}_{q([c])}) \\ p(\mathcal{C}_{q([d])} \mid \mathcal{C}_{q([c])})\, p([d] \mid \mathcal{C}_{q([d])}) \\ \\ p(\mathcal{C}_{q([ab])} \mid \#)\, p([ab] \mid \mathcal{C}_{q([ab])})\, p(\mathcal{C}_{q([cd])} \mid \mathcal{C}_{q([ab])})\, p([cd] \mid \mathcal{C}_{q([cd])}) \\ \\ p(\mathcal{C}_{q([abc])} \mid \#)\, p([abc] \mid \mathcal{C}_{q([abc])})\, p(\mathcal{C}_{q([d])} \mid \mathcal{C}_{q([abc])})\, p([d] \mid \mathcal{C}_{q([d])}) \end{array} \right.$$

Using the same notation as in equation (3.2), the likelihood of a segmentation computed with the class version of the model is:

$$\mathcal{L}\,(W,S) = \prod_{\tau} p(\mathcal{C}_{q(s_{(\tau)})} \mid \mathcal{C}_{q(s_{(\tau-\overline{n}+1)})} \cdots \mathcal{C}_{q(s_{(\tau-1)})})\, p(s_{(\tau)} \mid \mathcal{C}_{q(s_{(\tau)})}) \qquad (3.3)$$

Be it in equation (3.2) or (3.3), the likelihood of a corpus is a function of the bigram distribution of the phrases and of the class membership function q:

$$\mathcal{L}\,(W) = \sum_{S \in \{S\}} \mathcal{L}(W,S) = F(\{\, p(s_j \mid s_i)\,\}_{i,j}\ ,\ q) \qquad (3.4)$$

Indeed, equation (3.2) corresponds to the case where the class membership function assigns each phrase to its own singleton class. Besides, the bigram distribution of the classes and the class conditionnal distribution used in (3.3) can be deduced from $\{\, p(s_j \mid s_i)\,\}_{i,j}$ and q. Thus, on the whole, equation (3.4) covers both the non class and the class versions of the bi-multigram model. In the following section 3.3, we will see that it is convenient to look at the data likelihood using equation (3.4), in order to define a training procedure for the model.

3.3 Training procedure

3.3.1 Overview of the training procedure

In this section, we define a procedure to train a bi-multigram model (bigram dependencies between the phrases) in a way which maximizes its predictive capability. Since the predictive capability is a function of the data likelihood, we adopt the ML criterion as a training criterion. We postpone to section 3.4 the question of how to accomodate the training procedure, in order to limit the

effect of overtraining, which is likely to result from the choice of the ML criterion.

Based on the formulation given by equation 3.4, the problem of learning a bi-multigram model can be stated as the problem of estimating a bigram distribution of phrases and of defining a class membership function partitionning the set of phrases, in a way which maximizes the training data likelihood:

$$\{ p^*(s_j \mid s_i) \}_{i,j} \ , \ q^* \ = \ \arg\max \ F(\{ p(s_j \mid s_i) \}_{i,j} \ , \ q) \qquad (3.5)$$

For lack of a trivial solution, we propose to solve equation (3.5) using an iterative procedure where the likelihood function is alternately optimized, first with respect to the bigram distribution of the phrases, and second, with respect to the class membership function. The training procedure is thus a 2-step process, where iteration $(k+1)$ consist of:

Step (1): estimation of the bigram distribution of the phrases

$$F(\{ p^{(k+1)}(s_j \mid s_i) \}, \ q^{(k)}) \ \geq \ F(\{ p^{(k)}(s_j \mid s_i) \}, \ q^{(k)})$$

Step (2): optimization the class membership function

$$F(\{ p^{(k+1)}(s_j \mid s_i) \}, \ q^{(k+1)}) \ \geq \ F(\{ p^{(k+1)}(s_j \mid s_i) \}, \ q^{(k)})$$

Therefore, at the end of iteration $(k+1)$, the model is characterized by an updated bigram distribution of the phrases and by an updated class membership function, for which the likelihood is garanteed to be higher than the likelihood of iteration (k).
In the following sections, we give more details about each of the 2 steps above.

3.3.2 Estimation of the bigram distribution of the phrases

The estimation of the bigram distribution of the phrases (step (1)) of each training iteration) can be addressed as an ML estimation problem from missing data [7]. The basic idea behind this approach is that the statistics which are missing to compute the ML estimates of parameters can be approximated by their expected values. Hence, these problems are usually solved by the so-called **EM** procedure: the **E** stands for the expectation step, during which the expected values of the missing statistics are computed, and the **M** stands for the maximization step, during which ML estimates are computed using the expected values of the missing statistics.
In the problem at hand, the ML estimate of $p(s_j \mid s_i)$ is the ratio of the number of occurences of "$s_i \ s_j$" and of the number of occurences of "s_i". But because, the segmentation underlying the data is not known, the counts of the phrases are missing data. The $(k+1)^{th}$ iteration of the EM procedure in that case can be stated as follows:

During the E step, the expected counts of the phrases are computed as:

$$E[\, n(s_i s_j) \; ; \; \{\, p^{(k)}(s_j \mid s_i) \,\} \text{ and } q^{(k)}] \; = \; \sum_{\{S\}} n(s_i s_j \mid S)\, \mathcal{L}^{(k)}(S \mid W)$$

where $n(X \mid S)$ is the number of occurences of "X" in the segmentation S, and where the conditionnal likelihood of a segmentation $\mathcal{L}^{(k)}(S \mid W)$ is computed from either equation (3.2) or (3.3) with the model of the previous iteration (i.e. with $\{\, p^{(k)}(s_j \mid s_i) \,\}$ and $q^{(k)}$).

During the M step, the ML estimates of the parameters are computed assuming that the counts of the sequences equal their expected values:

$$p^{(k+1)}(s_j \mid s_i) \; = \; \frac{\sum_{\{S\}} n(s_i s_j \mid S)\, \mathcal{L}^{(k)}(S \mid W)}{\sum_{\{S\}} n(s_i \mid S)\, \mathcal{L}^{(k)}(S \mid W)} \tag{3.6}$$

A less intuitive but more rigorous way of deriving equation (3.6), based on the use of an auxiliary function Q, is presented in the appendix A. In practice, the algorithm to estimate the bigram distribution of the phrases is not implemented by following the EM steps, but with a forward-backward algorithm, (or with its Viterbi approximation), so that the complexity of the algorithm is $O(n^2 T)$, with n the maximum size of a sequence and T the number of words in the corpus. The principle of the forward-backward algorithm is to re-arrange the terms in the numerator and in the denominator of equation 3.6, so that the summations are made over the time index of the words in the corpus, instead of being made over the set of segmentations. The forward-backward equation equivalent to equation 3.6 is given in equation (A.6) of the appendix A. An alternative training procedure, consists in approximating the sum over the set of segmentation with the term corresponding to the most likely segmentation only. In that case, the reestimation equation reduces to:

$$p^{(k+1)}(s_j \mid s_i) \; = \; \frac{n(s_i s_j \mid S^{*(k)})}{n(s_i \mid S^{*(k)})} \tag{3.7}$$

with $S^{*(k)} \; = \; \arg\max_{\{S\}} \; \mathcal{L}^{(k)}(S \mid W)$ being retrieved with a Viterbi algorithm. We will refer to this alternative training procedure as the "Viterbi" training, as opposed to the "forward-backward" training.

3.3.3 Optimization of the classification of the phrases

The optimization of the class membership function (step (2) of each training iteration) aims at partitionning the set of phrases in a way maximizing the data likelihood, given a known bigram distribution of the phrases and given a prespecified number of classes. This issue does not differ from the issue of the data driven clustering of words, which has been thoroughly studied already in the field of statistical language modeling. Therefore, any of the clustering techniques based on a data likelihood criterion which have been proposed for the

automatic classification of words applies to our problem. The only difference is that candidates to clustering are phrases instead of words, but this does not require to modify at all the clustering algorithms.

In [5], we present a version of the bimultigram model where the classification of the phrases is performed with the greedy algorithm proposed by Brown & al. in [2]. The algorithm is initialized by assigning each phrase to its own singleton class, and by computing the loss in average mutual information when merging 2 classes for every pair of classes. Each iteration of the clustering algorithm then consists in merging the 2 classes for which the loss in mutual information is minimal, and in updating the loss values. Iterations are stopped when the required number of classes is obtained.
A major drawback of this clustering technique, in the context of the training of a bi-multigram model, is that the classification starts from scratch again at step (2) of each iteration, i.e. each time a bigram distribution of the phrases has been reestimated. First, it makes the training of the model unnecessarily long, since the bigram distribution of the phrases may not have changed so much that the classification need to be completely reset. Second, at step (2) of the clustering algorithm, it was implicitely acknowledged that each new class membership function $q^{(k+1)}$ was obtained by **modifying** the previous class membership function $q^{(k)}$ in a way increasing the likelihood function $F(\{ p^{(k+1)}(s_j \mid s_i) \}, q^{(k)})$. This is not the case with Brown's clustering algorithm where $q^{(k+1)}$ is not derived from $q^{(k)}$, but from a partition into singletons. And because this algorithm is a heuristic the global optimality of which is not guaranteed, the likelihood function computed with the resulting class membership function $q^{(k+1)}$ is not theoretically bound to be greater than the one computed[1] with $q^{(k)}$.

For the sake of rapidity and to make the likelihood function theoretically bound to increase, we have decided to favor a clustering algorithm where it is possible to modify an existing classification, instead of having to start from scratch every time. This is the case of the clustering algorithm proposed in [14] and further exposed in [15], and which is the one implemented in the latest bi-multigram package. The principle of the clustering in [14] is, starting from any given classification, to remove each word from its current class and to assign it to the class for which the data likelihood is maximal. The algorithm applied at step (2) of the bi-multigram learning works as follows:

[1] Though it is in practice unlikely to become lower.

- Start with an initial class membership function $q^{(k)}$,
- Compute the initial train set likelihood,
- Do until the stopping criterion is met:
  - do for each phrase s_i having a non zero probability:
    - remove s_i from its class,
    - do for all existing classes $\mathcal{C}$:
      - compute the likelihood as if s_i was moved to $\mathcal{C}$
    - assign s_i to the class with the best likelihood
    - update all the counts
- end up with a re-optimized class membership function $q^{(k+1)}$

It is thus guaranteed that:

$$F(\{ p^{(k+1)}(s_j \mid s_i) \}, q^{(k+1)}) \geq F(\{ p^{(k+1)}(s_j \mid s_i) \}, q^{(k)})$$

The first time the exchange clustering algorithm is applied ($k = -1$), the initial class membership function assigns the $N_{\mathcal{C}}$ most frequent phrases to their own singleton class, $N_{\mathcal{C}}$ denoting the number of required classes. The remaining phrases are all assigned to an additionnal temporary class from which phrases can be removed only (not assigned). Thus, after all phrases have been examined once, the temporary class has been emptied and there remains exactly $N_{\mathcal{C}}$ classes. The clustering iterations are then applied till either the number of class exchanges becomes zero, or after a prespecified number of iterations. For $k \geq 0$, the initial class membership function is the one from the previous iteration.

3.4 Algorithm

In Table 4.3, we show the overall algorithm implemented in the bimultigram package. In addition to the estimation and classification steps explained earlier in this chapter, it includes some precautions required to limit the effect of overtraining. These precautions consist in:

- pruning the inventory of phrases by discarding the least frequent ones: the combinations of words occuring less than $thr1$ times are not registered at the initialization, and the bigrams made of a phrase occuring less than $thr2$ times during the iterations are removed from the bi-multigram inventory (except if the 2 phrases are of length 1),
- maintaining all the bigrams made of 2 phrases which are both of length 1 word: if the reestimated number of co-occurrences of two phrases which are both 1-word-long comes to fall to zero, it is reset to "1".

3.5 Interpolation of non-class and class based bi-multigram models

With a class model, the probabilities of 2 phrases belonging to the same class are distinguished only according to their unigram probability. As it is unlikely that

- Initialize:

  - register all phrases of 1 up to n words occuring more than $thr1$ times in the training data (1-word phrases are registered whatever their count is), with their counts: $n^{(0)}(s_i s_j)$ and $n^{(0)}(s_i) = \sum_j n^{(0)}(s_i s_j)$,

  - compute $p(s_j \mid s_i)^{(0)}$ based on the relative counts,

  - find a partition $q^{(0)}$ with the phrase exchange algorithm, and compute the distributions $p^{(0)}(\mathcal{C}_{q(s_j)} \mid \mathcal{C}_{q(s_i)})$ and $p^{(0)}(s_j \mid \mathcal{C}_{q(s_j)})$,

  - compute backoff coefficients for the bigram distribution of the classes,

  - set k:=0

- While the likelihood has not converged, or while $k < k_{max}$, repeat:

  - reestimate $n^{(k+1)}(s_i s_j)$ and $n^{(k+1)}(s_i) = \sum_j n^{(k+1)}(s_i s_j)$, based on the forward-backward equation (3.6) or on the Viterbi equation (3.7),

  - Prune:
    while there is a phrase s_i such that $n^{(k+1)}(s_i) \leq thr2$:
    - remove all the bigrams with s_i, except if it is made of 2 phrases both of length 1 word,
    - if the count of a bigram made of 2 phrases of length 1 word has become zero, reset the count to "1",
    - for all phrases s, update $n^{(k+1)}(s)$ as $\sum_j n^{(k+1)}(s\ s_j)$,

  - compute $p^{(k+1)}(s_j \mid s_i)$ based on $n^{(k+1)}(s_i s_j)$ and $n^{(k+1)}(s_i)$,

  - find a new partition $q^{(k+1)}$ with the phrase exchange algorithm, and compute the distributions $p^{(k+1)}(\mathcal{C}_{q(s_j)} \mid \mathcal{C}_{q(s_i)})$ and $p^{(k+1)}(s_j \mid \mathcal{C}_{q(s_j)})$,

  - compute backoff coefficients for the bigram distribution of the classes,

  - set k:=k+1

Table 3.1: Algorithm for the training of a bi-multigram model.

this loss of precision be compensated by the improved robustness of the estimates of the class distribution, class based models can be expected to deteriorate the likelihood of not only train but also test data, with respect to non-class based models. However, the performance of non-class models can be enhanced by interpolating their estimates with the class estimates. We first recall the way linear interpolation is performed with conventional word n-gram models, and then we extend it to the case of our stochastic phrase-based approach. Usually, linear interpolation weights are computed so as to maximize the likelihood of cross evaluation data [9]. Denoting by λ and $(1-\lambda)$ the interpolation weights, and by p_+ the interpolated estimate, it comes for a word bigram model:

$$p_+(w_j \mid w_i) = \lambda\, p(w_j \mid w_i) + (1-\lambda)\, p(C_{q(w_j)} \mid C_{q(w_i)})\, p(w_j \mid C_{q(w_j)}) \quad (3.8)$$

with λ having been iteratively estimated on a cross evaluation corpus W_{cross} as:

$$\lambda^{(k+1)} = \frac{1}{T_{cross}} \sum_{i,j} c(w_i w_j)\, \frac{\lambda^{(k)}\, p(w_j \mid w_i)}{p_+^{(k)}(w_j \mid w_i)} \quad (3.9)$$

where T_{cross} is the number of words in W_{cross}, and $c(w_i w_j)$ the number of co-occurences of the words w_i and w_j in W_{cross}.
In the case of a stochastic phrase based model - where the segmentation into phrases is not known *a priori* - the above computation of the interpolation weights still applies, however, it has to be embedded in dynamic programming to solve the ambiguity on the segmentation:

$$\lambda^{(k+1)} = \frac{1}{c(S^{*(k)})} \sum_{i,j} c(s_i s_j \mid S^{*(k)})\, \frac{\lambda^{(k)}\, p(s_j \mid s_i)}{p_+^{(k)}(s_j \mid s_i)} \quad (3.10)$$

where $S^{*(k)}$ the most likely segmentation of W_{cross} given the current estimates $p_+^{(k)}(s_j \mid s_i)$ can be retrieved with a Viterbi algorithm, and where $c(S^{*(k)})$ is the number of sequences in the segmentation $S^{*(k)}$. A more accurate, but computationally more involved solution would be to compute $\lambda^{(k+1)}$ as the expectation of $\frac{1}{c(S)} \sum_{i,j} c(s_i s_j \mid S)\, \frac{\lambda^{(k)}\, p(s_j \mid s_i)}{p_+^{(k)}(s_j \mid s_i)}$ over the set of segmentations $\{S\}$ on W_{cross}, using for this purpose a forward-backward algorithm. However in the bi-multigram interpolation package, only the algorithm based on equation (3.10) has been implemented.

Chapter 4

Experiments and results

Most of the results presented below are re-printed from [5], and were obtained using a previous version of the bimultigram software.
Only some of the experiments with the class based bimultigram model, and the speech recognition experiments, were obtained with the current bimultigram software package.

4.1 Perplexity experiments

4.1.1 Evalution criteria

A motivation to learn bigram dependencies between variable length phrases is to improve the predictive capability of conventional word bigram models, while keeping the number of parameters in the model lower than in the word trigram case. The predictive capability is usually evaluated with the perplexity measure:

$$PP = e^{-\frac{1}{T}\log \mathcal{L}(W)}$$

where T is the number of words in W. The lower PP is, the more accurate the prediction of the model is. In the case of a stochastic model, there are actually 2 perplexity values PP and PP^* computed respectively from $\sum_S \mathcal{L}(W,S)$ and $\mathcal{L}(W,S^*)$, where S^* is the most likely segmentation of W. The difference $PP^* - PP$ is always positive or zero, and measures the average degree of ambiguity on a parse S of W, or equivalently the loss in terms of prediction accuracy, when the sentence likelihood is approximated with the likelihood of the best parse, as is done in a speech recognizer.
In section 4.1.3, we first evaluate the loss ($PP^* - PP$) using the forward-backward estimation procedure, and then we study the influence of the estimation procedure itself, i.e. equation (3.6) or (3.7), in terms of perplexity and model size (number of distinct 2-uplets of phrases in the model). Finally, we compare these results with the ones obtained with conventional n-gram models (the model size is thus the number of distinct n-uplets of words observed), using

for this purpose the CMU-Cambridge toolkit [3].

In section 4.1.4, we compare class versions and interpolated versions of the bigram, trigram and bi-multigram models, in terms of perplexity values and of model size. For bigrams (resp. trigrams) of classes, the size of the model is the number of distinct 2-uplets (resp. 3-uplets) of word-classes observed, plus the size of the vocabulary. For the class version of the bi-multigrams, the size of the model is the number of distinct 2-uplets of phrase-classes, plus the number of distinct phrases maintained. We compare the results obtained with the hierarchichal clustering algorithm [2] with those obtained with the class exchange clustering algorithm [14].

In section 4.1.5, we show examples of classes obtained with a model allowing phrases of up to 5-words, to illustrate the potential benefit of clustering relatively long and variable-length phrases for issues related to language understanding.

4.1.2 Database

Perplexity experiments are run on ATR travel arrangement data (see Tab. 4.1). This database consists of semi-spontaneous dialogues between a hotel clerk and a customer asking for travel/accomodation informations. All hesitation words and false starts were mapped to a single marker "*uh*".

	Train	Held-out	test
Nb sentences	13 650	7 350	2 430
Nb tokens	167 000	69 500	29 000 (1 % OOV)
Vocabulary	3 525		+ 280 OOV

Table 4.1: ATR Travel Arrangement Data

4.1.3 Models without classes

Training parameters

Experiments are reported for phrases having at most $n = 1, 2, 3$ or 4 words (for n =1, bi-multigrams correspond to conventional bigrams). The bi-multigram probabilities are initialized using the relative frequencies of all the 2-uplets of phrases observed in the training corpus, and they are reestimated with 6 iterations. The dictionaries of phrases are pruned by discarding all phrases occuring less than 20 times at initialization, and less than 10 times after each iteration[1], except for the 1-word phrases which are kept with a number of occurrences set

[1]Using different pruning thresholds values did not dramatically affect the results on our data, provided that the threshold at initialization is in the range 20-40, and that the threshold of the iterations is less than 10.

to 1. Besides, bi-multigram and n-gram probabilities are smoothed with the backoff smoothing technique [10] using Witten-Bell discounting [29][2].

Results

Ambiguity on a parse (Table 4.2) The difference ($PP^* - PP$) usually remains within about 1 point of perplexity, meaning that the average ambiguity on a parse is low, so that relying on the single best parse should not decrease the accuracy of the prediction very much.

n	1	2	3	4
PP	56.0	43.9	44.2	45.0
PP^*	56.0	45.1	45.4	46.3

Table 4.2: Ambiguity on a parse.

Influence of the estimation procedure (Table 4.3) As far as perplexity values are concerned, the estimation scheme seems to have very little influence, with only a slight advantage in using the forward-backward training. On the other hand, the size of the model at the end of the training is about 30% less with the forward-backward training: approximately 40 000 versus 60 000, for a same test perplexity value. The bi-multigram results tend to indicate that the pruning heuristic used to discard phrases does not allow us to fully avoid overtraining, since perplexities with n =3, 4 (i.e. dependencies possibly spanning over 6 or 8 words) are higher than with n =2 (dependencies limited to 4 words).

Test perplexity values PP^*				
n	1	2	3	4
F.-B.	56.0	45.1	45.4	46.3
Viterbi	56.0	45.7	45.9	46.2
Model size				
n	1	2	3	4
F.-B.	32505	42347	43672	43186
Viterbi	32505	65141	67258	67295

Table 4.3: Influence of the estimation procedure: forward-backward (F.-B.) or Viterbi.

Comparison with n-grams (Table 4.4) The lowest bi-multigram perplexity (43.9) is still higher than the trigram score, but it is much closer to the

[2]The Witten-Bell discounting was chosen, because it yielded the best perplexity scores with conventional n-grams on our test data.

trigram value (40.4) than to the bigram one (56.0) [3]. The number of entries in the bi-multigram model is much less than in the trigram model (45000 versus 75000), which illustrates the ability of the model to select most relevant phrases.

Test perplexity values PP				
n (and n)	1	2	3	4
n-gram	314.2	56.0	40.4	39.8
bimultigrams	56.0	43.9	44.2	45.0
Model size				
n (and n)	1	2	3	4
n-gram	3526	32505	75511	112148
bimultigrams	32505	42347	43672	43186

Table 4.4: Comparison with n-grams: Test perplexity values and model size.

4.1.4 Models with classes

Training parameters

The non-class models are the same as in section 4.1.3. The class-phrase models are trained with 5 iterations of the algorithm described in section 3.4: each iteration consists in iteratively estimating a phrase distribution with equation (3.6), and in clustering the phrases into 300 phrase-classes. The bigrams and trigrams of classes are estimated based on 300 word-classes derived with the same clustering algorithm as the one used to cluster the phrases. Like for the phrase estimates of the previous section, the class estimates are smoothed with the backoff technique using a Witten Bell discounting. Linear interpolation weights between the class and non-class models are estimated based on equation (3.9) in the case of the bigram or trigram models, and on equation (3.10) in the case of the bi-multigram model.
The training and test data used to train and evaluate the models are the same as the ones described in Table 4.1: the models are trained on the train set, the interpolation weights on the held-out data and the perplexity values are computed on the test set.

Results obtained with the hierarchichal clustering algorithm [2] (Table 4.5)

We first show results based on the hierarchichal clustering algorithm proposed by Brown & al. [2], which was the clustering algorithm implemented first in the original bimultigram software. The perplexity scores obtained with the non-class, class and interpolated versions of a bi-multigram model (limiting

[3] Besides, the trigram score depends on the discounted scheme: with a linear discounting, the trigram perplexity on our test data was 48.1.

to 2 words the size of a phrase), and of the bigram and trigram models are in Table 4.5. Linear interpolation with the class based models allows us to improve each model's performance by about 2 points of perplexity: the Viterbi perplexity score of the interpolated bi-multigrams (43.5) remains intermediate between the bigram (54.7) and trigram (38.6) scores. However in the trigram case, the enhancement of the performance is obtained at the expense of a great increase of the number of entries in the interpolated model (139256 entries). In the bi-multigram case, the augmentation of the model size is much less (63972 entries). As a result, the interpolated bi-multigram model still has fewer entries than the word based trigram model (75511 entries), while its Viterbi perplexity score comes even closer to the word trigram score (43.5 versus 40.4). Further experiments studying the influence of the threshold values and of the number of classes still need to be performed to optimize the performances for all models.

Test perplexity values PP^*			
	non-class	class	interpolated
bigrams	56.04	66.3	54.7
bimultigrams	45.1	57.4	43.5
trigrams	40.4	49.3	38.6
Model size			
	non-class	class	interpolated
bigrams	32505	20471	52976
bimultigrams	42347	21625	63972
trigrams	75511	63745	139256

Table 4.5: Comparison of class-phrase bi-multigrams and of class-word bigrams and trigrams: Test perplexity values and model size.

Results obtained with the phrase exchange algorithm [14] (Table 4.6)

In Table 4.6, we show the perplexity values obtained with phrases having up to either 1, 2, 3, 4 or 5 words, when using the phrase exchange clustering algorithm [14] implemented in the latest version of the bimultigram package. During the classification, only the phrases occuring more than 3 times are clustered ; the other phrases are not clustered, but instead they are assigned to a common class.

The use of the phrase exchange clustering algorithm, in addition to being faster, allows to improve the perplexity values with respect to the hierarchichal algorithm: 53.4 instead of 57.4 for phrases of up to 2 words. It is interesting to note that the use of classes allows to overcome the overtraining effect observed with the non class models: indeed, the best perplexity values are obtained with phrases having up to 3 or 4 words, whereas in the case of models without classes, the best perplexity values were with phrases of maximum length 2 words.

	Test perplexity values PP^*				
n	1	2	3	4	5
class bimultigram	63.6	53.4	52.2	52.4	53.3
interpolated bimultigram	45.8	36.3	36.2	36.9	37.5

Table 4.6: Test perplexity values using class based bimultigram models (300 classes) trained with the phrase exchange algorithm.

4.1.5 Examples of classes of phrases

In Table 4.7, we show a few classes resulting from the training (with the latest version of the package) of a model allowing phrases of up to 5 words. It is often the case that phrases differing mainly because of a speaker hesitation (all hesitations were maaped to the marker "*uh*") are merged together. Clustering variable-length phrases may provide a natural way of dealing with some of the language disfluencies which characterize spontaneous utterances, like the insertion of hesitation words for instance.
Table 4.7 also illustrates another motivation for phrase retrieval and clustering, apart from word prediction, which is to address issues related to topic identification, dialogue modeling and language understanding [11]. Indeed, though the clustered phrases in our experiments were derived fully blindly, i.e. with no semantic/pragmatic information, intra-class phrases often display a strong semantic correlation. To make this approach effectively usable for speech understanding, constraints derived from semantic or pragmatic knowledge (like speech act tag of the utterance for instance) could be placed on the phrase clustering process.

4.2 Speech recognition experiments

4.2.1 Protocol

All recognition experiments were performed with the JANUS toolkit, using the acoustic models trained by Detlev Koll during his stay at ATR in 1997. In the case of the bimultigram model, phrases are limited to 2 words. All the phrases in the language model are added as multiword entries to the recognition lexicon, with all the possible combinations of the word pronounciation variants. The 2-gram and 3-gram models are trained with the CMU-Cambridge toolkit using the Witten Bell backoff scheme, which is the one also used in the bimultigram toolkit[4]. The train data used to estimate the language models, and the test data used to run the speech recognition experiments are shown in Table 4.8.

[4] we checked that the performances obtained with the bimgramtk toolkit, by limiting the size of a phrase to 1 word, were exactly the same as the ones obtained with the 2-gram model of the CMU-Cambridge toolkit: so that we can reliably assert that there is no performance bias due to the use of a different toolkit.

{ thank+you+very+much , thank+you+for+calling , thanks+a+lot }
{ hello , hi , good+afternoon , good+morning , hello+this+is+the , good+evening }
{ and+*uh* , but+*uh* , *uh*+but , *uh*+and , around+noon }
{ yes+*uh* , yeah , *uh*+yes+*uh* , *uh*+actually , *uh*+yeah }
{ yes+that's+right , you're+welcome , yes+that's+correct , hello+front+desk , yes+please , all+right+sir , you're+very+welcome , yes+that's+fine , yes+i+do , *uh*+yes+please , i+see+sir , congratulations }
{ okay+*uh* , i+see+*uh* , *uh*+then , *uh*+i+see+*uh* , i+see+well }
{ i+understand , let's+see , let+me+see , i+understand+*uh* , *uh*+so , that+sounds+good , *uh*+let's+see , understood , dear }
{ all+right+then , i+see+then , okay+so , all+right+so , okay+then , i+see+so , *uh*+in+that+case , well+then }
{ how+much , what+time , how+long , and+what+time , *uh*+how+much , lastly }
{ waiting , calling , expecting , looking+forward+to , in+a+hurry , awaiting , causing }
{ like , prefer , like+to+have , love , hate }
{ could+you+tell+me , do+you+know , can+you+tell+me , i'd+like+to+know , i+don't+know , finally , hostels }
{ tell+me , give+me , give , tell , explain , mention , write , spell , calculate , allow , consider }
{ i'd+like+to , can+i , would+you+like+to , i+would+like+to , can+we , should+i , we'd+like+to , shall+i , did+you , and+you+can , *uh*+can+i , you+have+to }
{ o'clock , p.m. , a.m. , o+two , o+six , years+old , o'clock+tomorrow+morning , o'clock+in+the+morning , kilometer }
{ a+reservation , your+reservation , a+room , a+twin+room , a+single+room , a+double+room , the+reservation , an+interpreter , that+one , a+baby+sitter , two+tickets , another+room , photographers , accommodations , ohps , preparations }
{ make+a+reservation , stay , pay , check+in , get+there , make+it , work , make+the+reservation , sleep , play , return+the+car , go+there , help+me }
{ thirty+minutes , an+hour , fifteen+minutes , five+minutes , ten+minutes , ten+thousand+yen , one+hour , two+hours , fifty+dollars , about+an+hour , a+week , one+day , chinatown , membership , lodging }
{ costs , up+to , it+costs , it+takes , it+will+be , under , we+charge }
{ we+have , there+is , there's , there+are , do+you+have , is+there , there+is+a , we+do+have , do+you+have+any , there+will+be , we+have+a , it's+a , is+there+a , there's+a , is+there+any , are+there+any , there'll+be , *uh*+we+have , there+is+no , there're , you+can+enjoy }
{ immediately , right+away , later , as+soon+as+possible , for+that , free+of+charge , anytime , to+my+room , with+that , easily , shortly }
{ ms.+suzuki , mr.+phillips , ms.+tanaka , mr.+suzuki , ms.+phillips }
{ i'm+sorry , oh , excuse+me }

Table 4.7: Example of classes of phrases, with a model allowing up to 5-word phrases (clusters are delimited with curly brackets)

	Train	test
Nb sentences	21,586	173
Nb tokens	277,560	2,101

Table 4.8: Data used to train the LMs, and test data used for the speech recognition experiments

4.2.2 Recognition results

The word accuracy obtained with the bimultigrams (cf. Table 4.2.2) is intermediate between the performance of the 2-grams and of the 3-grams, as could be expected from the ranking of the perplexity values (note that the perplexity values in Table 4.2.2 are computed on the test set on which recognition is performed; this test set is smaller than the one used in section 4.1, so that the values are quite different).

	PP^*	% WA
2-gram	35.2	78.9
bimultigram	21.8	81.8
3-gram	20.0	83.7

Table 4.9: Bimultigrams versus conventional 2-grams and 3-grams

We believe that one major weakness of our speech recognition experiments with phrase based language models lies in the way the phrases are added to the recognition lexicon. Currently, each phrase is associated with all the pronounciations which can be obtained by combining the pronounciation variants of the words in the phrase. This has for effect to considerably and unnecessarily increase the size of the recognition lexicon. On the other hand, the phrases could be used to help reducing the confusability between the lexicon entries. Indeed, the benefit of using phrases could be enhanced by taking advantage of the fact that the integration of a word in a phrase constrains its pronounciation. When adding phrases to the lexicon, pronounciation rules could be applied to limit the number of possible variant pronounciations for each word in the phrase, depending on the phrase where it occurs. In addition to reducing the confusability between the words, this would fasten the recognition process by reducing the number of hypothesis it has to keep track of.

In Table 4.10, we show recognition results obtained with either the CMU-Cambridge toolkit [3] or the CLAUSI toolkit used within the JANUS system. While the CMU-Cambridge toolkit uses a conventional backoff [10] based on the Witten Bell discounting [29], the CLAUSI toolkit computes the backoff distribution following the method proposed by Kneser & al. in [12], and it is based on an absolute discounting. From Table 4.10, it can be seen that better

	PP	% WA
CMU 2-gram	35.2	78.9
CLAUSI 2-gram	34.0	80.3
CMU 3-gram	20.0	83.7
CLAUSI 3-gram Kneser&Ney backoff	18.9	84.6
CLAUSI 3-gram without	19.3	83.3

Table 4.10: Perplexity values and word accuracies on the test set

scores can be obtained with the CLAUSI toolkit than with the CMU-Cambridge toolkit. However, if the Kneser's backoff of the CLAUSI toolkit is disabled (whith still an absolute discounting), then the score of the CLAUSI toolkit falls below the score of the CMU-Cambridge toolkit. This stresses the influence of the backoff strategy on the speech recognition performances, the Kneser's backoff outperforming conventional backoff, and, in the case of conventional backoff, the Witten Bell discounting [5] outperforming athe other discounting schemes, on our data.

[5] Actually, the Kneser's backoff and the Witten Bell discounting have this in common that they tend to discount more the probabilities of the words observed in many different contexts

Chapter 5

Conclusion and perspectives

In this report, we have presented a stochastic class phrase based model for statistical language modeling. We have proposed a training algorithm based on an ML criterion, which allows to integrate the estimation of the probability distribution of the phrases with the classification of the phrases, in a way which guarantees the convergence of the likelihood. One originality of this work with respect to other current works on class phrase based models is the possibility to assign common labels to phrases having a different length. Experiments on a task oriented corpus have shown that structuring sentences into phrases results in large reductions in the bigram perplexity value, while still keeping the number of entries in the language model much lower than in a trigram model, especially when these models are interpolated with class based models.

Possible directions to improve this work are the following:

- work on better strategies than the add hoc pruning presented in this report, in order to reduce the over-learning problem observed for long sequences; this might require to accomodate the training process to other optimization criteria (leaving-one-out criterion for example).
- integrate the model within a 3-gram framework, instead of a 2-gram framework, in order to improve the performances of conventional 3-grams: in [21], work on a phrase based model is reported, where it is shown that the phrase based model with bigram dependencies allows, like in our work, to greatly improve the conventional 2-grams, while still being less performant than the conventional 3-grams. However they show that the integration of the phrase based model in a 3-gram framework allows to slightly outperform the conventional 3-grams.
- solve the problem of the high combinatorial complexity of the model. This would also allow us to use it for very large vocabularies, and also is prob-

ably a pre-requirement before it can be efficiently used in a 3-gram framework. Solutions to reduce the combinatorial complexity might be to *a priori* prune the set of phrases, possibly by using some linguistic knowledge like the part-of-speech for some words, or to perform a partial word clustering, ...etc. Implementation aspects are also involved in this issue.

- integrate a partial word clustering, especially for the words which are proper names. For instance, the class "{ kyoto+station ; grand+central+station ; osaka+airport ; ...}" could be re-written as "{ **city_name**+station ; grand+central+station ; **city_name**+airport ; ...}", which would improve the generalization capability of the model.
- optimize the number of distinct classes
- explore the potentialities of the classes of phrases in the area of language understanding ; this might require to place semantic/pragmatic constraints on the clustering process.

Bibliography

[1] F. Bimbot, R. Pieraccini, E. Levin, and B. Atal. Variable-length sequence modeling: Multigrams. *IEEE Signal Processing Letters*, 2(6), June 1995.

[2] P.F. Brown, V.J. Della Pietra, P.V. de Souza, J.C. Lai, and R.L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[3] P. Clarkson and R. Rosenfeld. Statistical language modeling using the cmu-cambridge toolkit. *Proceedings of EUROSPEECH 97*, 1997.

[4] S. Deligne and F. Bimbot. Language modeling by variable length sequences: theoretical formulation and evaluation of multigrams. *Proceedings of ICASSP 95*, 1995.

[5] S. Deligne and Y. Sagisaka. Learning a syntagmatic and paradigmatic structure from language data with a bi-multigram model. *Proceedings of COLING-ACL 98*, 1998.

[6] S. Deligne, F. Yvon, and F. Bimbot. Introducing statistical dependencies and structural constraints in variable-length sequence models. In *Grammatical Inference : Learning Syntax from Sentences*, Lecture Notes in Artificial Intelligence 1147, pages 156–167. Springer, 1996.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistics Society*, 39(1):1–38, 1977.

[8] J. Hu, W. Turin, and M. K. Brown. Language modeling using stochastic automata with variable-length contexts. *Computer Speech and Language*, 11:1–16, 1997.

[9] F. Jelinek and R.L. Mercer. Interpolated estimation of markov source parameters from sparse data. *Proceedings of the workshop on Pattern Recognition in Practice*, pages 381–397, 1980.

[10] S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. on Acoustic, Speech, and Signal Processing*, 35(3):400–401, March 1987.

[11] T. Kawahara, S. Doshita, and C. H. Lee. Phrase language models for detection and verification-based speech understanding. *Proceedings of the 1997 IEEE workshop on Automatic Speech Recognition and Understanding*, pages 49–56, December 1997.

[12] R. Kneser and H. Ney. Improved clustering techniques for class-based statistical language modeling. *Proceedings of EUROSPEECH 93*, 1993.

[13] K. Mark, M. Miller, U. Grenander, and S. Abney. Parameter estimation for constrained context-free language models. *Proceedings of the speech and natural language workshop*, II:37–40, February 1992.

[14] S. Martin, J. Liermann, and H. Ney. Algorithms for bigram and trigram word clustering. *Proceedings of EUROSPEECH 95*, 1995.

[15] S. Martin, J. Liermann, and H. Ney. Algorithms for bigram and trigram word clustering. *Speech Communication*, 24:19–37, 1998.

[16] H. Masataki and Y. Sagisaka. Variable-order n-gram generation by word-class splitting and consecutive word grouping. *Proceedings of ICASSP 96*, 1996.

[17] S. Matsunaga and S. Sagayama. Variable-length language modeling integrating global constraints. *Proceedings of EUROSPEECH 97*, 1997.

[18] M. K. McCandless and J. R. Glass. Empirical acquisition of language models for speech recognition. *Proceedings of ICSLP 94*, 1994.

[19] M. Meteer and J. R. Rohlicek. Statistical language modeling combining n-gram and context-free grammars. *Proceedings of ICASSP 93*, 1993.

[20] T. R. Niesler and P. C. Woodland. A variable-length category-based n-gram language model. *Proceedings of ICASSP 96*, 1996.

[21] G. Riccardi and S. Bangalore. Automatic acquisition of phrase grammars for stochastic language modeling. *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 188–196, 1998.

[22] G. Riccardi, R. Pieraccini, and E. Bocchieri. Stochastic automata for language modeling. *Computer Speech and Language*, 10:265–293, 1996.

[23] K. Ries, F. D. Buo, and A. Waibel. Class phrase models for language modeling. *Proceedings of ICSLP 96*, 1996.

[24] E. S. Ristad and R. G. Thomas. Hierarchical non-emitting markov models. *Proceedings of ACL 97*, pages 381–385, 1997.

[25] J. O Ruanaidh and W. J. Fitzgerald. *Numerical Bayesian Methods Applied to Signal Processing*, chapter 2, pages 8–9. Springer-Verlag, New York, 1996.

[26] M. Siu. *Learning local lexical structure in spontaneous speech language modeling.* PhD thesis, Boston University, 1998.

[27] M. Siu and M. Ostendorf. Variable n-gram language modeling and extensions for conversational speech. *Proceedings of EUROSPEECH 97*, 1997.

[28] B. Suhm and A. Waibel. Towards better language models for spontaneous speech. *Proceedings of ICSLP 94*, 1994.

[29] I.H. Witten and T.C. Bell. The zero-frequency problem: estimating the probabilities of novel events in adaptative text compression. *IEEE Trans. on Information Theory*, 37(4):1085–1094, July 1991.

Appendix A

Estimation of the probability distribution of the phrases

A.1 Derivation of the reestimation equation

To derive the reestimation equation of the probability distribution of the phrases in this section, we assume $\overline{n}$-gram correlations between the phrases, and we note n the maximal length of a phrase. Let W denote a string of words, and $\{S\}$ the set of possible segmentations on W. The likelihood of W is:

$$\mathcal{L}(W) = \sum_{S \in \{S\}} \mathcal{L}(W, S) \tag{A.1}$$

and the likelihood of a segmentation S of W is:

$$\mathcal{L}\,(W, S) = \prod_{\tau} p(s_{(\tau)} \mid s_{(\tau-\overline{n}+1)} \cdots s_{(\tau-1)}) \tag{A.2}$$

with $s_{(\tau)}$ denoting the phrase of rank (τ) in the segmentation S. The model is thus fully defined by the set of $\overline{n}$-gram probabilities on the set $\{s_i\}_i$ of all the phrases which can be formed by combining 1, 2, ...up to n words of the vocabulary.

Maximum likelihood (ML) estimates of these probabilities can be obtained by formulating the estimation problem as a ML estimation from incomplete data [7], where the unknown data is the underlying segmentation S. Let $Q(k, k+1)$ be the following auxiliary function computed with the likelihoods of iterations k and $k+1$:

$$Q(k, k+1) = \sum_{S \in \{S\}} \mathcal{L}^{(k)}(S \mid W) \log \mathcal{L}^{(k+1)}(W,\ S) \tag{A.3}$$

It has been shown in [7] that if $Q(k,k+1) \geq Q(k,k)$, then $\mathcal{L}^{(k+1)}(W) \geq \mathcal{L}^{(k)}(W)$. Therefore the reestimation equation of $p(s_{i_{\overline{n}}} \mid s_{i_1} \ldots s_{i_{\overline{n}-1}})$, at iteration $(k+1)$, can be derived by maximizing $Q(k,k+1)$ over the set of parameters of iteration $(k+1)$, under the set of constraints $\sum_{s_{i_{\overline{n}}}} p(s_{i_{\overline{n}}} \mid s_{i_1} \ldots s_{i_{\overline{n}-1}}) = 1$, hence:

$$p^{(k+1)}(s_{i_{\overline{n}}} \mid s_{i_1} \ldots s_{i_{\overline{n}-1}}) = \frac{\sum_{S \in \{S\}} c(s_{i_1} \ldots s_{i_{\overline{n}-1}} s_{i_{\overline{n}}}, S) \times \mathcal{L}^{(k)}(S \mid W)}{\sum_{S \in \{S\}} c(s_{i_1} \ldots s_{i_{\overline{n}-1}}, S) \times \mathcal{L}^{(k)}(S \mid W)} \tag{A.4}$$

where $c(s_{i_1} \ldots s_{i_{\overline{n}}}, S)$ is the number of occurences of the combination of phrases $s_{i_1} \ldots s_{i_{\overline{n}}}$ in the segmentation S. Since each iteration improves the model in the sense of increasing the likelihood $\mathcal{L}^{(k)}(W)$, it eventually converges to a critical point (possibly a local maximum).

Reestimation equation (A.4) can be implemented by means of a forward-backward algorithm ; the forward-backward algorithm of the bimultigram model $\overline{n} = 2$ is explained in section A.2. In a decision-oriented scheme, the reestimation equation reduces to:

$$p^{(k+1)}(s_{i_{\overline{n}}} \mid s_{i_1} \ldots s_{i_{\overline{n}-1}}) = \frac{c(s_{i_1} \ldots s_{i_{\overline{n}-1}} s_{i_{\overline{n}}}, S^{*(k)})}{c(s_{i_1} \ldots s_{i_{\overline{n}-1}}, S^{*(k)})} \tag{A.5}$$

where $S^{*(k)}$, the segmentation maximizing $\mathcal{L}^{(k)}(S \mid W)$, is retrieved with a Viterbi algorithm.

A.2 Forward-backward algorithm

In the case of a bimultigram model ($\overline{n} = 2$) equation (A.4) can be implemented at a complexity of $O(n^2 T)$, with n the maximal length of a sequence and T the number of words in the corpus, using a forward-backward algorithm. Basically, it consists in re-arranging the order of the summations of the numerator and denominator of equation (A.4): the likelihood values of all the segmentations where sequence s_j occurs after sequence s_i, with sequence s_i ending at the word at rank (t), are summed up first; and then the summation is completed by summing over t. The cumulated likelihood of all the segmentations where s_j follows s_i, and s_i ends at (t), can be directly computed as a product of a forward and of a backward variable. The forward variable represents the likelihood of the first t words, where the last l_i words are constrained to form a sequence:

$$\alpha(t, l_i) = \mathcal{L}(W_{(1)}^{(t-l_i)} [W_{(t-l_i+1)}^{(t)}])$$

The backward variable represents the conditional likelihood of the last $(T-t)$ words, knowing that they are preceded by the sequence $[w_{(t-l_j+1)} \ldots w_{(t)}]$:

$$\beta(t, l_j) = \mathcal{L}(W_{(t+1)}^{(T)} \mid [W_{(t-l_j+1)}^{(t)}])$$

Model without classes: Assuming that the likelihood of a parse is computed according to equation (3.2), then the reestimation equation (A.4) can be rewritten as:

$$p^{(k+1)}(s_j \mid s_i) = \frac{\sum_{t=1}^{T} \alpha(t,\ l_i)\ p^{(k)}(s_j \mid s_i)\ \beta(t+l_j,\ l_j)\ \delta_i(t-l_i+1)\ \delta_j(t+1)}{\sum_t \alpha(t,\ l_i)\ \beta(t,\ l_i)\ \delta_i(t-l_i+1)} \tag{A.6}$$

where l_i and l_j refer respectively to the lengths of the sequences s_i and s_j, and where the Kronecker function $\delta_k(t)$ equals 1 if the word sequence starting at rank t is s_k, and equals 0 if not.

The variables α and β can be calculated according to the following recursion equations (assuming a start and an end symbol at rank $t = 0$ and $t = T+1$):

for $1 \leq t \leq T+1$, and $1 \leq l_i \leq n$:

$$\alpha(t, l_i) = \sum_{l=1}^{n} \alpha(t-l_i, l)\ p([W_{(t-l_i+1)}^{(t)}] \mid [W_{t-l_i-l+1}^{(t-l_i)}])$$

$\alpha(0,1) = 1$, $\alpha(0,2) = ... = \alpha(0,n) = 0$.

for $0 \leq t \leq T$, and $1 \leq l_j \leq n$:

$$\beta(t, l_j) = \sum_{l=1}^{n} p([W_{(t+1)}^{(t+l)}] \mid [W_{(t-l_j+1)}^{(t)}])\ \beta(t+l, l)$$

$\beta(T+1,1) = 1$, $\beta(T+1,2) = ... = \beta(T+1,n) = 0$.

Model with classes: In the case where the likelihood of a parse is computed with the class assumption, i.e. according to equation (3.3), the reestimation equation becomes:

$$p^{(k+1)}(s_j \mid s_i) =$$

$$\frac{\sum_{t=1}^{T} \alpha(t,\ l_i)\ p^{(k)}(C_{q(s_j)} \mid C_{q(s_i)})\, p^{(k)}(s_j \mid C_{q(s_j)})\ \beta(t+l_j,\ l_j)\ \delta_i(t-l_i+1)\ \delta_j(t+1)}{\sum_t \alpha(t,\ l_i)\ \beta(t,\ l_i)\ \delta_i(t-l_i+1)} \tag{A.7}$$

Besides, in the recursion equation of α, the term $p([W_{(t-l_i+1)}^{(t)}] \mid [W_{t-l_i-l+1}^{(t-l_i)}])$ is replaced by the corresponding class bigram probability multiplied by the class conditional probability of the sequence $[W_{(t-l_i+1)}^{(t)}]$. A similar change affects the recursion equation of β, with $p([W_{(t+1)}^{(t+l)}] \mid [W_{(t-l_j+1)}^{(t)}])$ being replaced by the corresponding class bigram probability multiplied by the class conditional probability of the sequence $[W_{(t+1)}^{(t+l)}]$.

Appendix B

The bimultigram package

B.1 How to get it

cp /home/atrh39/sdeligne/bimgramtk.tar $HOME/.
cp /home/atrh39/sdeligne/interpoltk.tar $HOME/.

tar xvf interpoltk.tar
cd interpoltk
make

tar xvf bimgramtk.tar
cd bimgramtk
make

It creates executables with a suffix corresponding to the operating system of the machine on which it is run.

B.2 Demonstration in bimgramtk

The demonstration files in the bimgramtk package are:

- the data file "digit.tr"
  it was created by drawing digits among the ten digits ("zero", "one",... "nine"), and by concatenating their spellings. The drawing was made in such a way that the spelling of an odd digit can follow the spelling of an even digit only, and that the spelling of an even digit can follow the spelling of an odd digit only. The boundaries between the digits were removed, and a blank was inserted inbetween each letter. The beginning and the end of each sentence are marked with the symbols $< s >$ and $< /s >$.
  A sample of the resulting file is:

< s > u n s i x u n < /s >
< s > t r o i s q u a t r e c i n q s i x s e p t z e r o c i n q z e r o t r o i s
q u a t r e s e p t d e u x < /s >
< s > t r o i s s i x s e p t z e r o u n q u a t r e c i n q s i x n e u f d e u
x s e p t h u i t < /s >
< s > n e u f d e u x t r o i s h u i t t r o i s s i x s e p t d e u x c i n q z
e r o u n s i x n e u f < /s >
< s > z e r o c i n q s i x n e u f z e r o t r o i s q u a t r e c i n q h u i t t
r o i s < /s >

- the data file "digit.te"
  it was made following the same procedure as for digit.tr. The distribution of the digits (even and odd) is thus the same, so that in average the distribution of the letters is the same.

- the list of file(s) "digit.list"
  it just contains the filename "digit.tr", and it is meant to be used as an argument with the perl program "get_vocabulary.perl" to create the vocabulary file of "digit.tr", i.e. the list of the strings of characters separated by blanks in "digit.tr".

- "digit.tr_config", "digit.tr_class_config", and "digit.te_config" are configuration files to train a bimultigram and a class based bimultigram model on the data "digit.tr", and to test an existing bimultigram model on the data "digit.te".

The demo allows you to:

- create a vocabulary file from the list "digit.list" with the command:
  "get_vocabulary.perl -list digit.list -voc digit.voc"

- train a class bimultigram model, based on the vocabulary previously defined:
  "Langmodel -config digit.tr_class_config"
  The specifications of the model in the configuration file "digit.tr_class_config" are: 2 classes, phrases of maximum length 6.
  Comments about the training process and its outcome are displayed: in this demo, the resulting model has identified the 10 spellings of the digits as sequences. Besides, because the set of the odd digits and the set of the even digits obey 2 distinct distributions, the model has assigned them to 2 distinct classes. Note that all sequences of length 1 (i.e. in that example, all sequences made of 1 letter) are kept in the inventory of sequences.
  It creates a file with extension ".lm" which is the language model file in arpa format, and a file with extension ".te.parse" which contains the best parse found on "data.te" with the model.

- train a bimultigram model without classes, based on the vocabulary previously defined:
  "Langmodel -config digit.tr_config"
  The specifications of the model in the configuration file "digit.tr_class_config" are: no classes, phrases of maximum length 6.

- compute the perplexity of "data.te" with a model previously trained:
  "Langmodel -config digit.te_config"

B.3 Demonstration in interpoltk

The demonstration files in the interpoltk package are:

- the data files "digit.cross" and "digit.te"
- a class phrase based model "digit.tr.bw.bimgram6_6000_500.class2_500.lm" and a non-class phrase based model "digit.tr.bw.bimgram6_6000_500.lm"
- the configuration files "digit.tr_config", and "digit.te_config" to compute ML interpolation weights between the demo models on the data "digit.cross", and to test the resulting interpolated model on the data "digit.te".

The demo allows you to:

- compute ML interpolation weights between 2 existing models (one of which is a class based model and the other one is without classes) on "digit.cross":
  "cd ../interpoltk ; Interpolate -config digit.tr_config"
  It creates a file with extension ".weight" in which the values of the estimated interpolation weights is specified.
- compute the perplexity of "digit.te" by interpolating the models with the weigths previously estimated:
  "cd ../interpoltk ; Interpolate -config digit.te_config"

B.4 How to create a vocabulary

Before training a model, you need to define the vocabulary. There is a perl tool "get_vocabulary.perl" to create a vocabulary file from a list of training data files (one filename per line).

usage:

get_vocabulary.perl -list < *list_of_data_files* > -voc < *output_vocfile* > -thr < *thr_value* >

only the entries with a nb of occurences greater than < *thr_value* > will be in the vocabulary (default value is 0)

B.5 How to train a model

To train a model, you need to write a training configuration file, and to use the perl command "Langmodel" with the name of the training configuration file as an argument to the option "-config".

The perl command "Langmodel" just reads the options in the configuration file, and calls the executable of the C program "Langmodel.c" with these options.

Mandatory options in a training configuration file are: -trfile, -vocfile, -mu. All others are optional (start the line in the configuation file with "#" to disable it). An example of a training configuration file is: digit.tr_class_config ; we give explanations about its entries below:

help option:

-help	# to get a help message

options specifying the data files:

-trfile=digit.tr	# training data
-tefile=digit.te	# test data
-vocfile=digit.voc	# vocabulary file

options specifying the format of the data files:

-sep=+	# symbol to join the units forming a phrase ("+" should be used with ATR files, default is "_")
-start=5	# symbol marking the beginning of a sentence in the data ("5" should be used with ATR files, default is "< s >")
-end=6	# symbol marking the end of a sentence in the data ("6" should be used with ATR files, default is "< /s >")

options specifying the training/evaluation algorithm:

-itereval	# compute perplexity on -tefile at each iteration of the training
-trtype=bw	# reestimate the phrase bigram distribution using the forward-backward ("bw") algorithm, or the viterbi ("vit") algorithm
-tetype=bw	# compute the perplexity on -tefile using the forward-backward ("bw") algorithm, or the viterbi ("vit") algorithm

options specifying the training parameters:

-mu=6	# maximum length of a sequence
-thr1=6000	# all the sequences occuring less than -thr1 at the initialization are discarded (default value is "0")
-thr2=500	# all the sequences occuring less than -thr2 during the iterations are discarded (default value is "0")
-niter=5	# number of iterations for the estimation of the bigram distribution of the phrases

options specifying the clustering parameters:

-cluster=2	# number of clusters into which the phrases are partitionned
-clu_niter=5	# number of iterations for the exchange clustering algorithm
-clu_thr=500	# clustering threshold: sequences occurring less are put in an "UNK" class, together with the out-of-vocabulary words
-final	# cluster only after the last iteration of the estimation of the bigram distribution of the phrases (faster but far less performant in terms of perplexity)
-display	# display the members of each class at the end of the training

B.6 How to compute a perplexity with an existing model

To test an existing model, you need to write a test configuration file, and to use the perl command "Langmodel" with the name of the test configuration file as an argument to the option "-config".

The perl command "Langmodel" just reads the options in the configuration file, and calls the executable of the C program "Langmodel.c" with these options.
Mandatory options in a test configuration file are: -tefile, -lm, -mu. All others are optional (start the line in the configuation file with "#" to disable it).
An example of a test configuration file is: digit.te_config ; we give explanations about its entries below:

help option:
-help # to get a help message

options specifying the test data file:
-tefile=digit.te # test data

options specifying the format of the data files:

-sep=+ # symbol to join the units forming a phrase ("+" should be used with ATR files, default is "_")
-start=5 # symbol marking the beginning of a sentence in the data ("5" should be used with ATR files, default is "< s >")
-end=6 # symbol marking the end of a sentence in the data ("6" should be used with ATR files, default is "< /s >")

options specifying the evaluation algorithm:
-tetype=bw # compute the perplexity on -tefile using the forward-backward ("bw") algorithm, or the viterbi ("vit") algorithm

options specifying the existing model:
-lmfile=digit.tr.bw.bimgram6_6000_500.class2_500.lm
name of the existing language model file
-mu=6 # maximum length of a sequence in the model -lmfile

B.7 How to estimate interpolation weights

The tools to estimate interpolation weights are in the directory "interpoltk". To estimate interpolation weigths between bimultigram models with and without classes, you need to write a training configuration file, and to use the perl command "Interpolate" with the name of the training configuration file as an argument to the option "-config".

The perl command "Interpolate" just reads the options in the configuration file, and calls the executable of the C program "Interpolate.c" with these options.

Mandatory options in a training configuration file are: -crossfile, -lm1, -lm2, -mu
All others are optional (start the line in the configuation file with "#" to disable it). An example of a training configuration file is: digit.tr_config ; we give explanations about its entries below:

help option:
-help # to get a help message

options specifying the data files:
-crossfile=digit.cross #training data
-tefile=digit.te #test data

options specifying the models:
-lm1=digit.tr.bw.bimgram6_6000_500.class2_500.lm
input lm model
-lm2=digit.tr.bw.bimgram6_6000_500.lm
input lm model

options specifying the training parameters:
-mu=6 # maximum length of a sequence
-niter=5 # number of iterations for the estimation of the bigram distribution of the phrases

-sep=+ # symbol to join the units forming a phrase ("+" should be used with ATR files, default is "_")
-start=5 # symbol marking the beginning of a sentence in the data ("5" should be used with ATR files, default is "< s >")
-end=6 # symbol marking the end of a sentence in the data ("6" should be used with ATR files, default is "< /s >")

B.8 How to compute a perplexity with existing models and existing interpolation weights

You need to write a test configuration file, and to use the perl command "Interpolate" with the name of the test configuration file as an argument to the option "-config".

The perl command "Interpolate" just reads the options in the configuration file, and calls the executable of the C program "Interpolate.c" with these options.

Mandatory options in a training configuration file are: -tefile, -lm1, -lm2, -mu, -weightfile
All others are optional (start the line in the configuation file with "#" to disable it). An example of a test configuration file is: digit.te_config ; we give explanations about its entries below:

help option:

```
-help                # to get a help message
```

options specifying the data files:

```
-tefile=digit.te     #test data
```

options specifying the models:

```
-lm1=digit.tr.bw.bimgram6_6000_500.class2_500.lm
                     # input lm model
-lm2=digit.tr.bw.bimgram6_6000_500.lm
                     # input lm model
-weightfile=digit.cross.weight
                     # input file with the interpolation weights
```

options specifying the training parameters:

```
-mu=6                # maximum length of a sequence
```

options specifying the format of the data files:

```
-sep=+               # symbol to join the units forming a phrase
                     ("+" should be used with ATR files, default is "_")
-start=5             # symbol marking the beginning of a sentence in the data
                     ("5" should be used with ATR files, default is "< s >")
-end=6               # symbol marking the end of a sentence in the data
                     ("6" should be used with ATR files, default is "< /s >")
```