Language Model Techniques in Machine Translation

Evaluating the effect of integrating long term context dependency language modeling techniques into Statistical Machine Translation

Diplomarbeit

Universität Karlsruhe / Carnegie Mellon University

Martin Raab
Advisors: Matthias Eck, Stephan Vogel

31st October 2006

Affirmation

Hiermit erkläre ich, die vorliegende Arbeit selbständig erstellt und keine anderen als die angegebenen Quellen verwendet zu haben.

Hereby I declare by that I created this thesis on my own and used no other references as the quoted ones.

Pittsburgh, the 31st October 2006 ....

Abstract

Automatic translation from one language to another is a highly ambituous task, and there is already a long history of people trying to solve this problem (Weaver, 1955). Yet there is no answer to this problem, but since Brown et al. (1993) Statistical Machine Translation (SMT) emerged as a promising candidate and is until now of primary research interest. Many work has been spent on components like the translation model and subtleties of this. Less work has been spent on language models, which are needed to derive well formed sentences in the target language.

Hence the leading idea of this thesis is to integrate and evaluate language modeling techniques to capture long term context dependencies. An excellent source of inspiration for this is the field of speech recognition. The reason is that language models have been studied thoroughly for speech recognition, where language models play a similar role. However, few of the numerous approaches for speech recognition language models have been evaluated with respect to machine translation. After analyzing the literature the following three approaches seemed promising and have been chosen to become investigated: class base language models, cache language models and sentence mixture language models. Each of these techniques is at least evaluated regarding perplexity and translation scores.

The structure of the paper is the following. At first an introduction to the area of machine translation is given. Then a literature review highlights different approaches for language models. In the next three sections, the actual experiments are described, starting with a short description of the implementation of the different models, continuing with the results of experiments performed and ending with an analysis of the results. Finally some conclusions are drawn and suggestions for future work are made.

Contents

Acknowledgments

First of all I have to thank Prof. Waibel and the InterAct program for giving me a stipend and the chance to write my diploma thesis at the Carnegie Mellon University in Pittsburgh.

For the academic supervision of my thesis I have to thank Dr. Stephan Vogel, Matthias Eck, Ying Zhang (Joy), Bing Zhao, Sanjika Hewavitharana and Almut Silja Hildebrand. For the original inspiration to this thesis I have to thank Dr. Rogina. I also would like to thank Friedrich Faubel and Thomas Schaaf for motivating me after making a series of unsuccessful experiments.

For a pleasant time in Pittsburgh I have to thank the Pittsburgh Council of International Visitors (PCIV), which is a great institution for all foreigners in Pittsburgh. Through them I made a lot of friends, and I'm certain that other towns would benefit if they would create something similar.

Chapter 1

Introduction

In order to understand the preconditions for this thesis, this introduction is first explaining some basics of machine translation. After this, the motivation for this thesis and its approaches are described. At the end an outline briefly describes the structure of the rest of the thesis.

1.1 Approaches to Machine Translation

Machine translation is the task of automatically translating either written or spoken (source) language to another target language. There are basically three different promising approaches to cope with the highly complex task of machine translation. These three approaches are depicted in figure 1.1.

The easiest to grasp is the *direct translation* approach. In many cases a precondition for this approach is to have a large amount of bilingual data. As a new sentence in the source language has to be translated, similarities of this sentence to sentences in the bilingual data are searched. How this actual search is performed can vary. The basic idea is to find matching words, and use the translations of these words. However, to know which word of the source sentence becomes translated to which target word is already a difficult research question. A similar approach, which has actually been of primary research interest in the last years is SMT. The advantage of SMT is that it makes soft decisions. This means that SMT does not state which translation is correct, but assigns probabilities to possible translations. Before details of SMT process are provided in 1.2.1, the discussion of figure 1.1 is continued.

The *transfer-based translation* approach is a little bit more abstract as the direct translation approach, as it first tries to make a syntactical analysis of the source sentence. Example steps of such an analysis can be a morphological analysis, anaphora resolution and multi word splitting. For interested readers, more information about these techniques can be found in

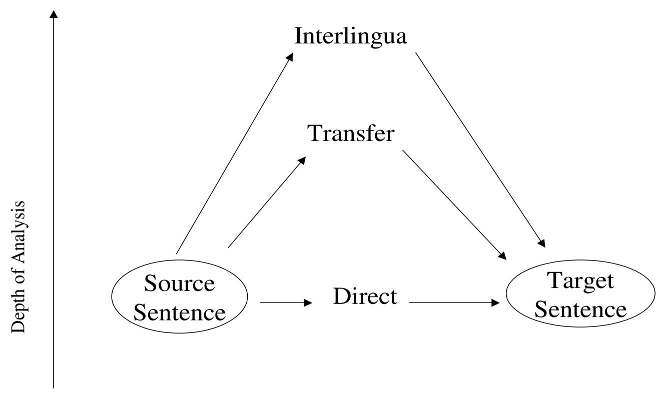


Figure 1.1: Three Approaches for Machine Translation

Arnold et al. (1994, chap.3). The advantage of transfer-based translation is that it can capture long-distance correlations between words and usually produces very good translation when the source sentence can be parsed. However, the analysis is complex, and automatic parsing of a sentence is far away from working well for all sentences. Thus transfer-based translation produces sometimes garbage when it is unable to parse the source sentence.

The desired machine translation would of course not only be able to resolve syntactical issues, but also be able to resolve ambiguities through semantic knowledge. In order to use semantic knowledge usually an interlingua is used. An interlingua is an idealized language, which abstracts from words in any language and tries to capture meanings and concepts. Apparently, there are many problems when trying to create an interlingua, hence there is no “one” interlingua yet. Rather different interlinguas have to be defined for every domain, to make it possible to reduce the nearly infinite number of concepts and meanings in human speech to an feasible amount. For convenience most of the interlinguas defined up to now used concepts defined in English words, but there is no reason why English should be better than any other language. More about the *interlingua translation approach* and the according problems can again be found in Arnold et al. (1994, pp. 80ff).

To conclude this broad introduction to machine translation, it has to be said that most systems still operate on the direct translation approach due to practicability. It would be nice to realize one of the other two approaches, but due to flexibility reasons the majority of work is done in direct translation. The interesting SMT, which is one technique of the direct translation approaches and of central interest for this paper is described in the next section.

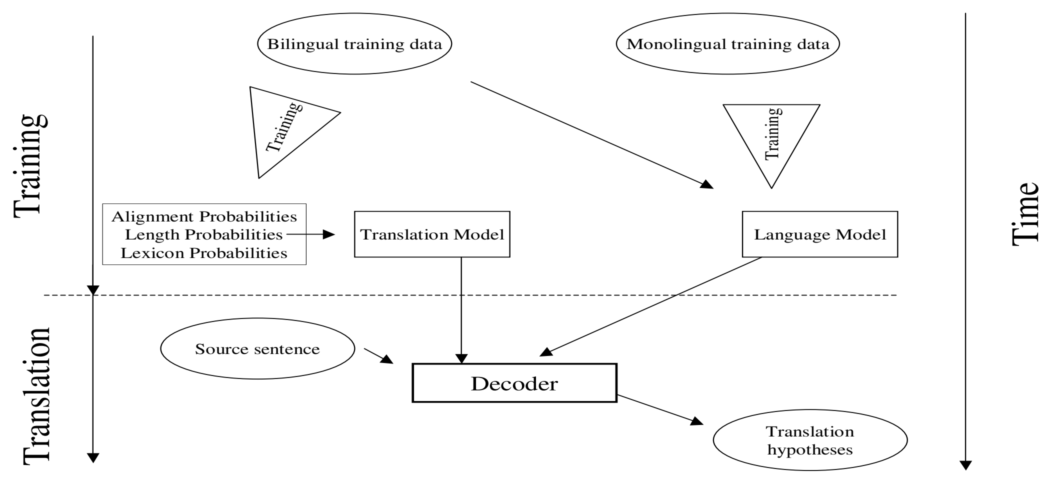


Figure 1.2: Components involved in SMT

1.2 Statistical Machine Translation

1.2.1 Statistical Machine Translation

Statistical Machine Translation is the most widely technique used to produce machine translation today. The key feature of Statistical Machine Translation is that is uses probabilities and hence makes only soft decisions which can be revised at a later point. In other words, they only decide with a certain probability at a time which translation might be the best. As there are multiple components, they can agree on which translation has the highest probability. Figure 1.2 is giving an overview about the components involved in SMT. There may be further components, depending on the actual translation system, however, the depicted components should be present in most SMT systems. Before describing the training process and introducing the mathematical background, the next sentences will briefly describe the translation itself to point out the connections between the components.

As shown at the bottom of figure 1.2, the translation is performed by a decoder. This decoder queries the translation model subsequently with the words from the source sentence. The translation model itself consists of several probability distributions. As response, the translation model returns translation hypotheses for every word. The decoder takes these hypotheses and merges them to a translation lattice as illustrated in 1.3. As can be seen, a translation lattice is essentially a compressed representation of many different hypotheses for possible translations. Each of these hypotheses has an associated probability. But there are still many hypotheses that are ill formed, which means they do not comply with standard grammatic rules. Task of the language model is to sort out these ill formed hypotheses. How a language model is achieving this is described in section 1.2.3. The hypothesis with the highest overall probability, both from the translation and language model is considered to be the best translation.

An evaluation how well the automatic translations is performed by comparing the output to human reference translations. Two very popular metrics to score similarity between reference and automatic translation are BLEU and NIST. NIST scores are strongly influenced by word matches in the translation, whereas BLEU is influenced stronger by matching n-grams. As this thesis is consistently evaluating BLEU scores more details about BLEU are given here. The basic idea is to calculate a geometric mean of n-gram occurrences in both reference and hypotheses translation. This score is then multiplied by a length penalty, which is calculated by comparing the full length of all candidate sentences to the full length of all reference translations. Due to this averaging of the length penalty over the whole corpus, a BLEU score is only meaningful when evaluated over a whole text. More information about BLEU and NIST can be found in Papineni et al. (2001) and NIST (2003).

Of course, before the above can be done, the involved models and probability distribution have to be trained first. Commonly used are the five IBM models introduced by Brown et al. (1993). Up to now, many machine translation programs still use the simplest IBM model 1. This model essentially uses lexical probabilities independent from sentence length and positions in the sentence to calculate probabilities for hypotheses. Today this basic model is usually combined with more sophisticated phrase alignment methods. A basic idea behind these techniques is to segment the source and target sentence into phrases and calculate phrase translation probabilities. Examples for these methods which have been used at CMU are integrated segmentation and alignment or phrase extraction via sentence alignment. These techniques are not discussed further here. The interested reader is referred to Brown et al. (1993), Zhang et al. (2003) and Vogel (2005).

1.2.2 Mathematical Foundation of SMT

As the reader should now have a grasp of the components participating in SMT, this section is reviewing the SMT process from a mathematical point of view. To recall, SMT tries to translate a sentence from a *source language* S into a *target language* T. As SMT decoders output a probability for the possible translations, usually the aim is to find a translation with

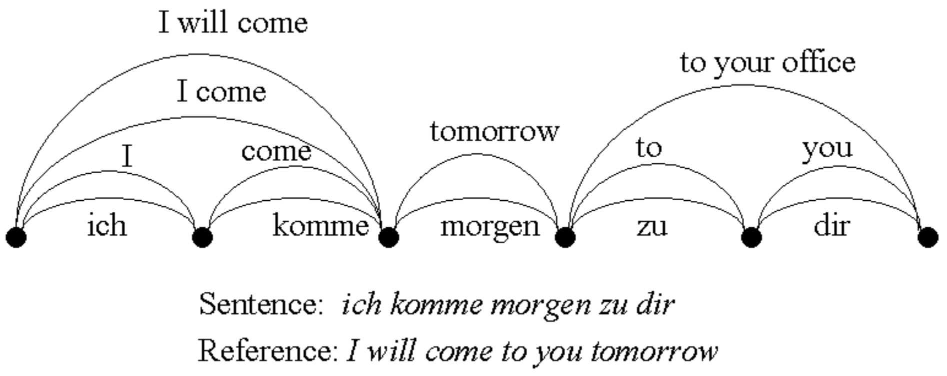

I will come
I come
I
come
tomorrow
to your office
to
you
ich
komme
morgen
zu
dir
Sentence: *ich komme morgen zu dir*
Reference: *I will come to you tomorrow*

Figure 1.3: Translation Lattice Example

highest conditional probability P(T|S). This probability has to be generated by a translation model. However, programming a computer algorithm which produces "correct" (meaning humans could agree with) P(T|S) probabilities is very hard. There are too many problems, and as stated before, a translation model will produce many translations, and many ill formed as well. In order to ameliorate the situation, the Bayes Theorem can be used:

$$P(T|S) = \frac{P(T)P(S|T)}{P(S)}$$

with P(T) and P(S) being the a priori probabilities that the target respectively the source sentence is occurring. The advantage the right side offers, is that there are now two separate models, or in other words two independent information sources. One is the translation model P(S|T), and the other one is the language model of the target language P(T). Thus the same situation as in figure 1.2 has been deduced, with the already noted advantage that the language model can sort out ill formed sentences. But the formula can be simplified further, because it usually only of interest to find the target sentence $\hat{T}$ with the highest probability P(T|S). As the maximum is independent from the a priori probability P(S), the problem to solve remains to find

$$\hat{T} = argmax_T(P(T)P(S|T))$$

This is the same result as in the previous section, as P(T) is given by a language model, and P(S|T) is given by a translation model.

1.2.3 Language Models in SMT

Language models are the key concept this thesis evaluates. But before the approaches of this paper to improve standard trigram language models become discussed in the next section, this section introduces the basic concepts of language modeling. The task of a language model is to judge sentences according to their grammatical correctness. Of course, there are many possible attempts to achieve this, n-gram language models are the established standard though. Basically n-gram language models count occurrence frequencies in large training data sets, and can use this knowledge to deduce how probable it is that a sentence is grammatically correct or not. More details about n-gram language models can be found in section 2.1.

To evaluate language models without influences from other models, usually the perplexity of a language model is measured. Interpreted in mathematical notation, perplexity is the geometric average of the inverse probability of words from a test set

$$\sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

Informally, the perplexity represents the average number of words which can follow any word. One effect of this is that the worst possible language model would not help at all to prefer any word and the average number of words which could follow any word would be the complete vocabulary size of available words.

1.3 Motivation and Approaches

As it is clear from the title of this thesis, the main area of interest is to analyze language modeling techniques for SMT. The following visualizations of the motivation and the approaches will all employ the same level of abstraction. The basic concept is that each line is representing a word, and each block is representing a word which is of interest. The problem is that a translation or other process is generating word hypotheses for a word position. This word position is marked with a rectangle which contains a question mark, and the word hypotheses are listed in a gray box next to it. Other rectangles indicate words that are useful to make decisions among the different word hypotheses. The deeper black a rectangle is, the stronger is its influence on the decision.

Having said this, the current approach to language modeling is depicted on figure 1.4. The rightmost block is the word which is currently investigated. A translation process has generated possible translations for this position and is now needing information to decide which of these possibilities are good. The answer has to be given from a language model. Most contemporary language models would only consider the two words to the left of the current word for their probability estimation. Recent enhancements in processing power has allowed to extend this limited dependency to include up to four words to the left of the current word. Figure 1.4 is clearly showing that there many words in the context of the word under investigation which have no effect on the returned probability. The whole thesis is evaluating approaches how more of these other words can be integrated in the probability estimation for the current word.

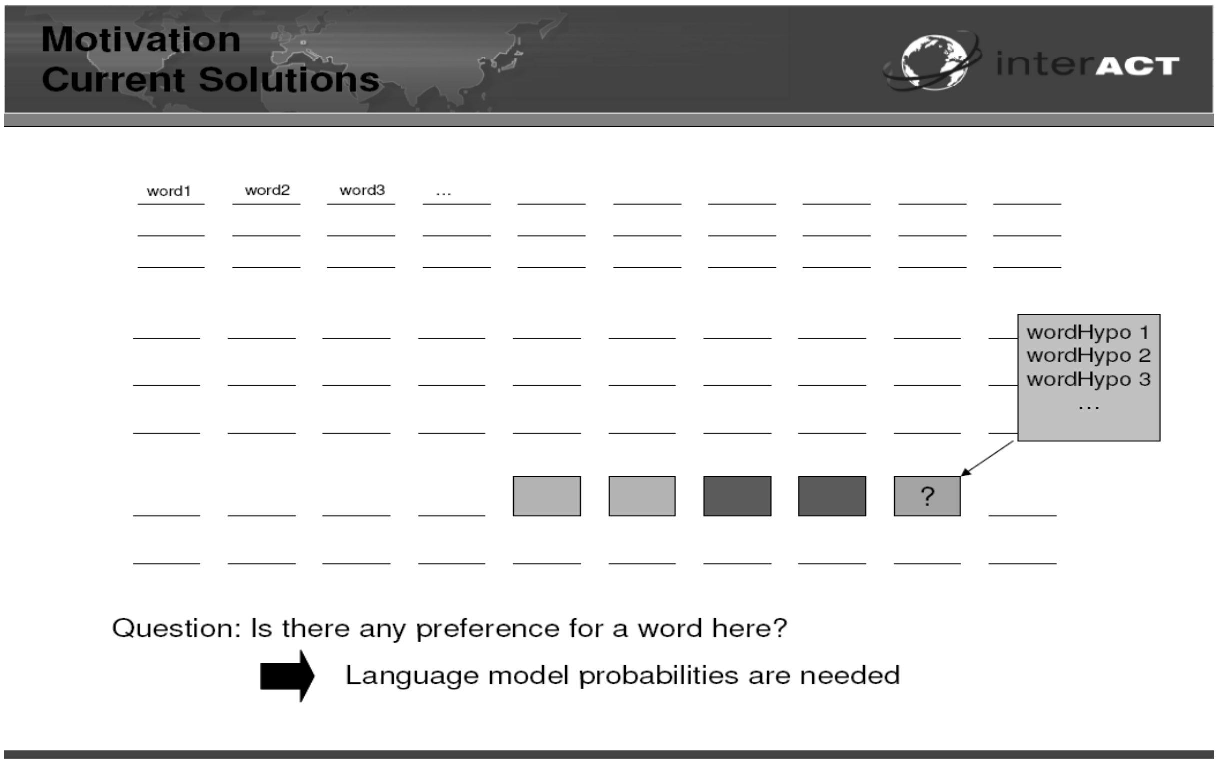
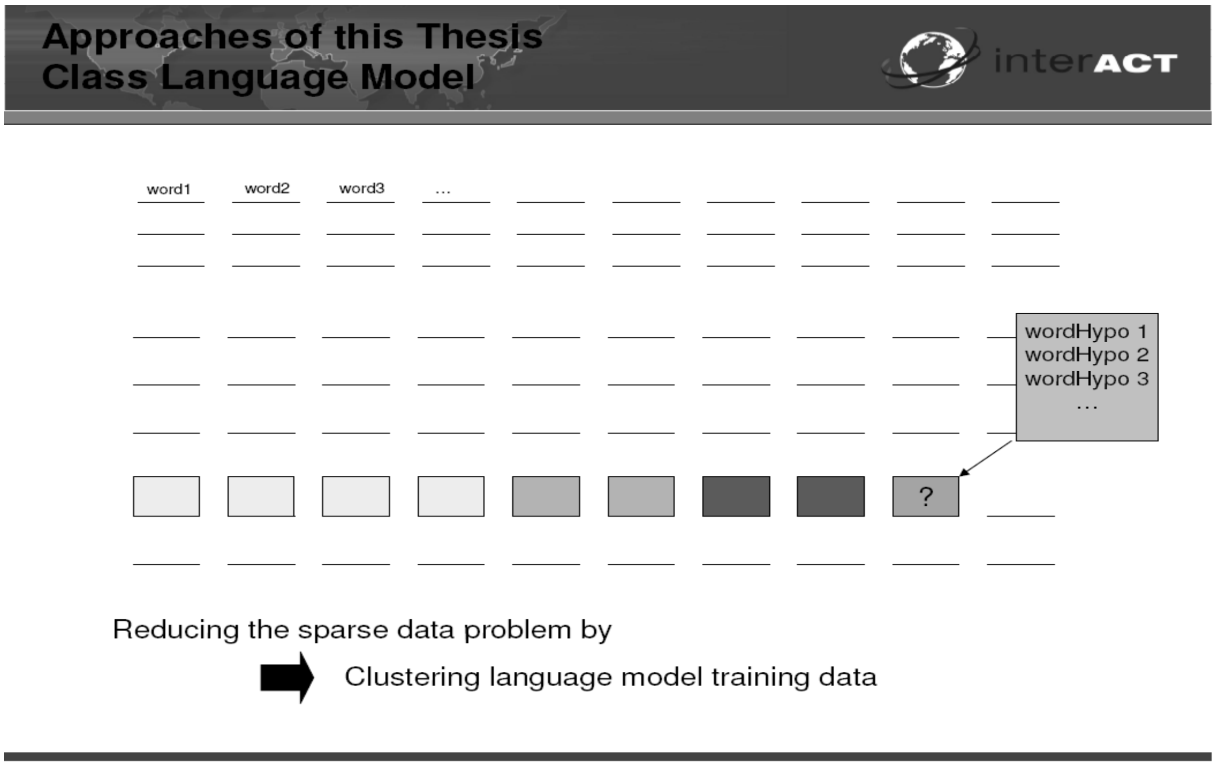

Motivation
Current Solutions
interACT
word1
word2
word3
...
wordHypo 1
wordHypo 2
wordHypo 3
...
?
Question: Is there any preference for a word here?
Language model probabilities are needed

Figure 1.4: Motivation

When looking again at 1.4, one question is for what reason, even with high processing power, only the last four words are considered. The answer lies in the underlying probability estimation, which would need more training data as is available up to now. The class language model (figure 1.5) is a technique which eliminates both the need for high processing power and large training data. Essentially this is achieved through clustering words to classes and applying only classes for its probability estimation. The key effect however is that this makes the use of longer histories possible.

Even after the application of the class language model, there are many words, that are not considered. Hence the cache language model (figure 1.6) is integrating influences from previous sentences in the probability estimation. This is achieved by boosting current words that have occurred in one of the last sentences.

The last approach is to capture similarities, both in topic and style between sentences. These so called sentence mixture language models make it possible to retrieve influences from the whole sentence, instead of being limited to a certain amount of words in the history.

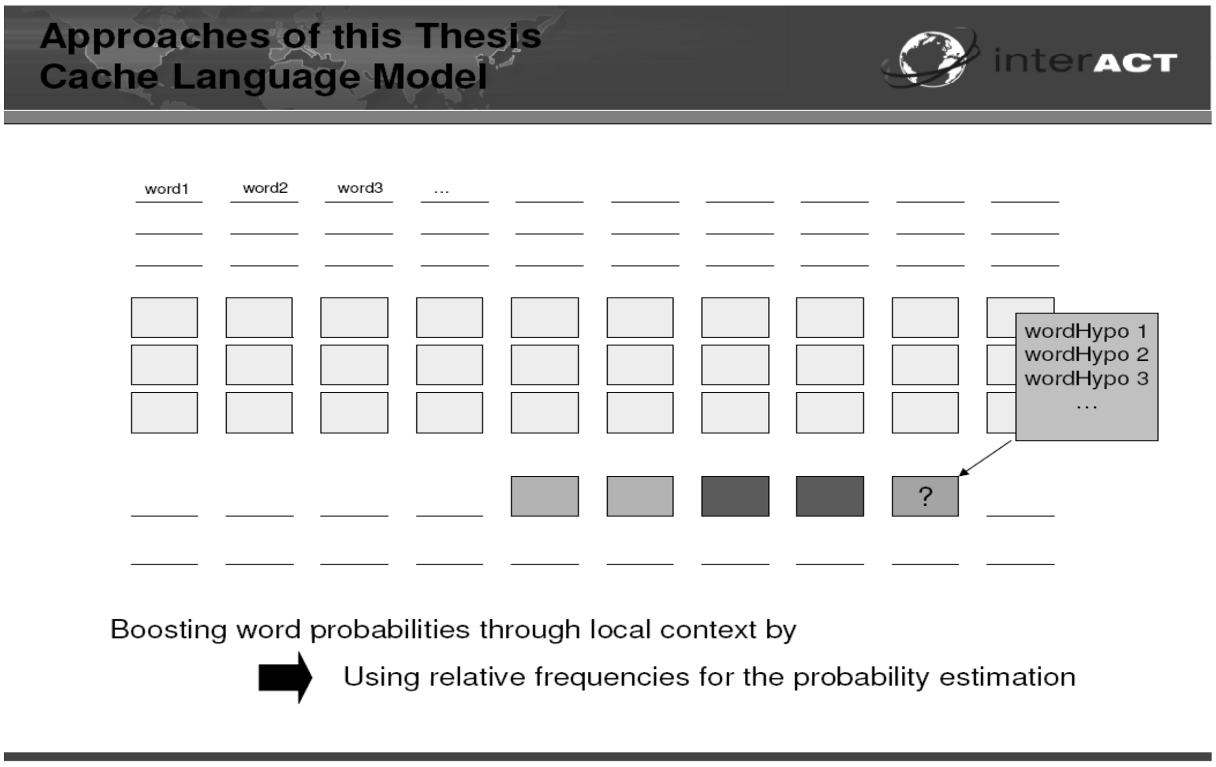


Approaches of this Thesis
Class Language Model

Reducing the sparse data problem by

➡ Clustering language model training data

Figure 1.5: Idea of Class Language Models



Approaches of this Thesis
Cache Language Model

Boosting word probabilities through local context by

➡ Using relative frequencies for the probability estimation

Figure 1.6: Idea of Cache Language Models

Figure 1.7 is showing this approach. After the sentence type has been decided, similar sentences from the training data will have more effect on language model scores. The training corpus is illustrated in the gray colored box.

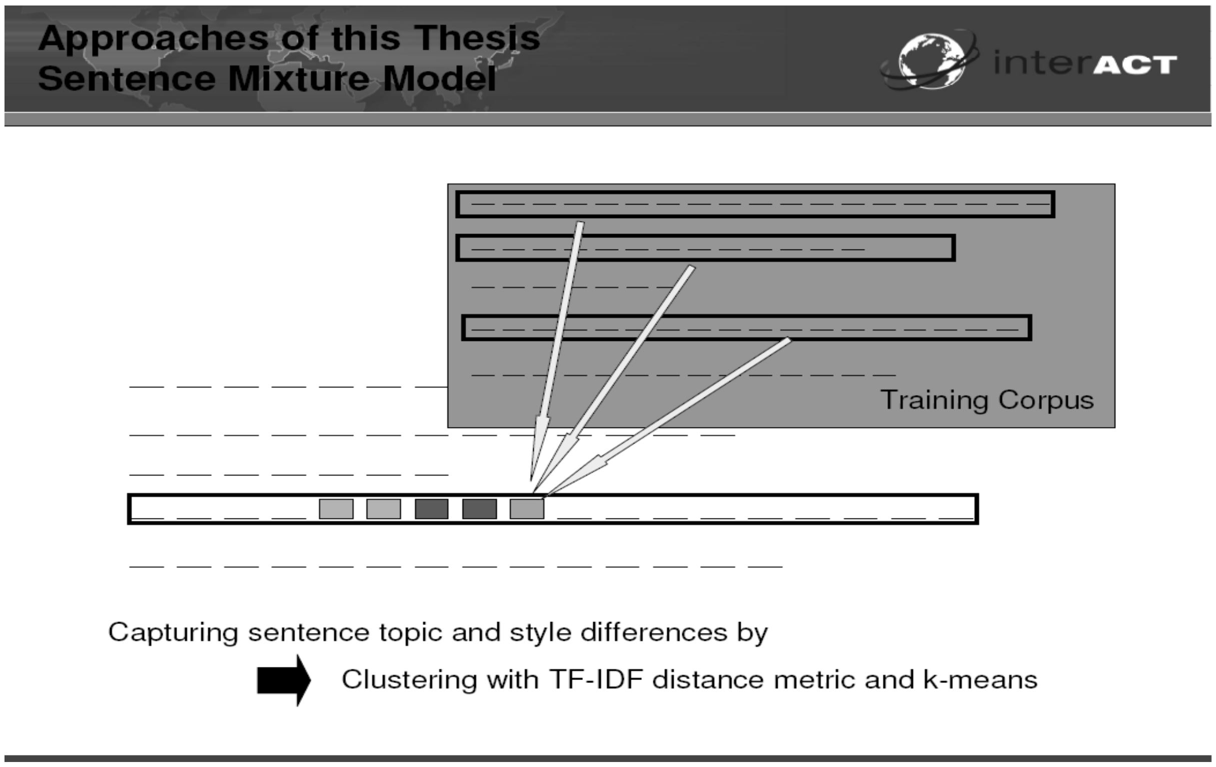

Approaches of this Thesis
Sentence Mixture Model
interACT
Training Corpus
Capturing sentence topic and style differences by
Clustering with TF-IDF distance metric and k-means

Figure 1.7: Idea of Sentence Mixture Language Models

It has to be said that all three approaches had already been know for language modeling before this thesis. This thesis, however, is going beyond previous work by first integrating them in SMT and second by evaluating their effect on SMT.

1.4 Outline

The aim of this introduction was to make the reader familiar with SMT and to motivate and introduce the approaches this thesis is analyzing. The rest of this document is structured as follows. The next chapter analyzes current literature about language models and improvements to language models. After the background is set, the actual implementation and data sets used become described. The next two chapters then describe performed experiments and discuss the results. In the last chapter conclusions are drawn and suggestions for future work are made.

Chapter 2

Literature Review

This literature review will highlight and discuss techniques for language modeling. First the standard n-gram language model is introduced. After that, several refinements for language models are described and their use for this thesis is evaluated by comparing previous results in the literature. The whole literature review is guided by Goodman (2001), who provided an excellent survey for language models in the domain of speech recognition. In addition, further literature is analyzed for the most promising candidates.

2.1 N-gram Language Models

2.1.1 Basics

A language model estimates probabilities for word sequences P($w_1 ... w_i$). N-gram language models achieve this with the use of partial word histories

$$P(w_1 ... w_i) = P(w_1) \times P(w_2|w_1) \times ... \times P(w_i|w_1 ... w_{i-1})$$

For practical reasons, quite often a trigram model assumption is made, and the history is only traced for the last two words.

$$P(w_1 ... w_i) \approx P(w_i) \times P(w_i|w_{i-1}) \times P(w_i|w_{i-2} w_{i-1})$$

The probability of a sentence S is the multiplication of the included trigrams.

$$P(S) = \prod_{i=0}^{\#T} P(T_i)$$

The needed probabilities are estimated from large training corpora.

However, the straightforward idea to count occurrences of trigrams in training data is flawed. Quite a test trigram which occurs in a test set is absent in the training data. As a consequence, this trigram would be assigned a zero probability, causing the sentence to be assigned a zero probability, too. Furthermore it is many times only coincidence if a trigram occurs once or not at all in the training text. Not allowing unseen trigrams,but allowing trigrams seen once is therefore rather inexact. Solutions to this issue are presented in the next subsection.

2.1.2 Smoothing techniques

Smoothing techniques try to make probability estimations for word sequences more robust. The main advantage is that smoothing can prevent n-grams to have zero probability and thus drop the whole language model score to zero. There are basically three techniques to achieve this: cutoffs, discounting and backoff.

Setting the cutoff to m means to ignore all word sequences that occur less or equal than m times in the training text. As all word sequences for the probability estimation occur now at least $m{+}1$ times, the probability can be estimated more robust. Discounting means to change counts of word sequences to change the probability estimation. Discounting is quite often used to remove probability mass, which is needed for the backoff algorithm. The aim of the backoff algorithm is to allow unseen word sequences to occur. Unseen means here that this n-gram has not occured in the training data. Yet, the backoff algorithm can generate a probability for this word sequence by calculating the probabilty of the (n-1)-gram. This process can be iterated, eventually falling back to the unigram probability of the current word if no longer n-gram matches. In order to do this falling back to lower length n-grams mathematically correct, probability mass has to to be redistributed from full length n-grams to shorter (n-x)-grams ($x \in 1, ..., n-1$). This can be achieved by removing some probability mass (discounting) from the training text.

Several sophisticated smoothing techniques are discussed in Chen and Goodman (1998). For the creation of the trigram language model in this thesis modified Kneser-Ney smoothing is used as it was shown to be the best smoothing technology by Goodman (2001). The supremacy of Kneser-Ney smoothing is founded in the fact that it is the only technique which considers the frequency how often a word can be seen in a new context. Regular Kneser-Ney uses only one discounting factor, while modified Kneser-Ney uses three different discounting factors depending on how often the investigated trigram was observed. However it needs many parameters as there are different discounting factors for different counts.

2.1.3 Higher order N-gram Language Models

Higher order n-gram language models try to capture longer distance correlations between words within a sentence better. They do this by loosening the common n-gram assumption to regard only two words of context and rather look at more context words. However, including more words for the n-gram probability estimation introduces two practical problems.

First of all, increasing the history length means exponentially increasing the needed amount of training data. Consider for example that a language model should be trained with a vocabulary of 100k ($= 10^5$) words. This means that there can be 100k to the power of three different trigrams, which are 10^{15} different possible combinations of words. This also means that a five-gram would need to estimate probabilities for 10^{25} word combinations. In other words, the five-gram model would need to estimate 10 billion times more parameters than the trigram model. The estimation of this large amount of probabilities is introducing the second practical problem, that even current computing system can hardly handle the amount of training data needed to estimate them. Both are only practical problems, and in times of the Internet the amount of training data available is anyway increasing in never imagined speed. The suffix array language model described in the next section is one possibility to cope with the large amounts of parameters needed.

Still, these practical problems for only including five words should make clear how hard it is to further extend n-grams, to for example seven-grams. More information about n-grams language models can be found in Knight (1999) and Stolcke (2002).

2.2 Suffix Array Language Model

Essentially a *Suffix Array Language Model* (SALM) is a regular n-gram language model. However, there is one big difference to the other n-gram language models: it is not pre-training probabilities. Instead the SALM builds a data structure for calculating the n-gram probabilities on the fly (Zhang et al., 2006)(Manber and Myers, 1993).

There are two advantages of SALMs. First of all their memory usage, which is a key issue in all static language models is only depending on the size of the training corpus, not on the length of n-grams. This makes it easy to use histories up to the length of 10-grams, whereas before a university could hardly go beyond 4-grams due to limited computing power. The second advantage is that, as they calculate the probabilities on the fly, other features can influence the language model probabilities. Previously, if there would be features like sentence mixture models, see 2.6, for each mixture model a separate language model had to be build. This was even further intensifying the language model memory problem. The SALM now allows to integrate dynamic features without the overhead which prevented them from being used in previous static language models.

Of course, SALMs have also some problems. The first issue is that they need more processing power than static language models. As of today, however, this is less a problem than memory usage and for currently common language model sizes trained on up to one billion words they run very fast. Secondly, as static language models have been used for decades, many techniques to improve them were implemented. Some of these techniques, like Kneyser-Ney smoothing (Chen and Goodman, 1998) can not easily be adopted by the SALM. However, the advantage of being able to use longer histories seems to be more important than the advanced smoothing techniques.

After all SALMs seem to be a promising improvement to static n-gram models, which dominated the language model applications for decades. This potential is the reason why this is a section on its own in the literature review, rather than being a subsection of the n-gram language models.

2.3 Class based Language Model

There are essentially two different types of class based models. The first type uses manually designed classes. In many cases this will be linguistically motivated classes, often part of speech classes. The second type is based on automatically derived classes, most of the time these classes will be uncorrelated to linguistic assumptions.

Their are different possibilities how class based language models estimate their probabilities. The most common one is

$$P(w_1...w_i) = P(w_i|W_i) \times P(W_i|W_{i-2}W_{i-1})$$

where the capital W's refer to the classes. In simple models each word is belonging to exactly one class, however more sophisticated models also do this mapping probabilistically. The approach shown here does not incorporate the real words history, consequently it is usually interpolated with a regular n-gram language model.

The advantage of using class based language models is that they reduce data sparsity. This is due to the fact that there are usually far less classes than words in the vocabulary. As an example consider the phrase “I was born in Berlin”. In regular language models, this five-gram would be relatively useless for statistical estimation of a five-gram probability, as it is likely to occur only once or twice in a corpus. If, however, there would be a (manually designed) class city, the phrase would be “I was born in [CITY]”. Consequently, this five-gram would be much more frequent in human speech, hence it can be estimated more reliably.

Previous work has shown that language models which use automatically derived classes (Niesler et al., 1998) outperform manually designed classes. As shown by Goodman (2001) the best class language models can lower perplexity up to 7% for large amounts of training data. For small amounts of training data the gains can be up to 13%, as it is essentially a technique to cope with data sparsity.

Another interesting approach to class based language models was introduced by Och (1999). The novel idea there was to build bilingual word classes. The reported results are very promising, however they rely on a certain overall translation approach which can benefit from these bilingual word classes.

Fortunately there are a number of toolkits available that support the automatic generation of word classes (Och, 1995) (Stolcke, 2002).

2.4 Cache Language Model

Cache language models promise to be the strongest enhancement compared to standard trigram language models. In Goodman (2001) perplexity improvements of up to 35% have been made for small training sets, and still gains of 15% for large training sets. However, it seems to be important to interpolate with bi- or trigram cache models, as they substantially outperform interpolation with word cache models.

The idea behind cache language models is to catch short time dependencies of language. Typical history lengths for cache models are between 2000 and 5000 words (Nepveu et al., 2004). Dependencies are extracted by bookkeeping of all recently seen words. The probability for these words gets boosted accordingly to their number of occurrences. How this boosting is actually performed can vary. For example one of the three following methods can be used

1. $P(w_1 ... w_i) = \lambda GLM(w_1 \dots w_i) + (1 - \lambda) CLM(w_i)$
2. $P(w_1 ... w_i) = (GLM(w_1 ... w_i)) \times (1 + \lambda CLM(w_i))$
3. $P(w_1 ... w_i) = GLM(w_1 ... w_i)^{1 + \lambda CLM(w_i)}$

The abbreviation GLM refers to the general language model score, CLM refers to the probability this word is been given by the cache model. λ is a weighting factor. To be correct, the CLM score should not only depend on w_i, and consider the whole context like the general language model. Experiments how bi- or trigram models affect the translation performance are performed in section 4.3.

A more sophisticated approach to incorporate short time word histories was performed by Kuhn and de Mori (1990). There approach used 19 different caches for 19 different part of speech classes. The corresponding cache was then selected to give a probability for a word after the part of speech of the word has been guessed. This cache probability for the word was then interpolated with a 3g-gram model, also described in Kuhn and de Mori (1990). This approach lead to an extremely high perplexity improvement of 67%. However, the disadvantage is that this approach needs both for the 3g-gram model and the adjustment of interpolation parameters part of speech tagged training data, which is only available in comparatively small amounts.

A problem which becomes evident when applying cache language models to real speech or machine translation problems is that it is never clear if the history used for the cache language model is correct. Yet cache language models, being neglected by research over the last decade are becoming increasingly popular again. Zhang et al. (2004) and Vaiciunas and Raskinis (2006) are two examples for current efforts, but both are only reporting perplexity improvements. Nepveu et al. (2004) is going a step further, and recently reported real translation improvements with interactive machine translation, however his experiments have the advantage of having a nearly correct history.

After all evaluating the performance of cache language models on machine translation tasks seems to be very worthwhile. Considerable improvements (in perplexity) have been reported in literature, yet as far as the author knows nobody evaluated cache language models on the relatively new field of machine translation. In addition, a recent survey (Rosenfeld, 2000) still mentions cache models as promising approach. The experiments in this paper try to shed light on the real benefits of cache language models for non-interactive machine translation, if there are any.

2.5 Skipping Models

Skipping models are another technique to cope with data sparsity which occurs when trying to use higher order n-grams. The idea of skipping models is to capture similar context instead of capturing exactly the same context as regular n-gram models do. This is achieved by ignoring words in the history (=skipping).

If for example a five-gram is considered usually the following probabilities would be needed to make a estimation for the probability of the following word w_i

$$P(w_i), P(w_i|w_{i-1}), P(w_i|w_{i-2}w_{i-1}), P(w_i|w_{i-3}w_{i-2}w_{i-1}), P(w_i|w_{i-4}w_{i-3}w_{i-2}w_{i-1})$$

Skipping Models can add to these probabilities the following ten probabilities where some words in the history are skipped

$$P(w_i|w_{i-2}), P(w_i|w_{i-3}w_{i-2}), P(w_i|w_{i-3}w_{i-1}), P(w_i|w_{i-4})$$

$$P(w_i|w_{i-3}w_{i-1}), P(w_i|w_{i-4}w_{i-1}), P(w_i|w_{i-4}w_{i-2})$$

$$P(w_i|w_{i-4}w_{i-3}w_{i-2}), P(w_i|w_{i-4}w_{i-3}w_{i-1}), P(w_i|w_{i-4}w_{i-2}w_{i-1})$$

Of course, it is not necessary for a skipping model to generate all these probabilities, it is also possible to add only some of these additional probabilities. Another nice feature of skipping models is the possibility to integrate longer contexts without the exponential increase of parameters needed. For example the following subset of probabilities considers the last four words in the history, but no probability is depending on more than two words.

$$P(w_i), P(w_i|w_{i-1}), P(w_i|w_{i-2}w_{i-1}), P(w_i|w_{i-4})$$

$$P(w_i|w_{i-2}), P(w_i|w_{i-3}w_{i-2}), P(w_i|w_{i-3}w_{i-1}),$$

According to Goodman (2001) skipping (trigram) models are good for small training data, but less effective for large amounts of training data where five-grams can be trained.

2.6 Sentence Mixture Models

A corpus can contain different kinds of sentences, these kinds can for example be distinct in topic or style. The idea behind sentence mixture models is to create different language model scores for each sentence type. This offers in addition to using longer n-grams another possibility to capture long-distance correlations in a sentence.

Goodman (2001) found perplexity improvements of 19% for large training sets with 284 million words by using a simple approach to estimate the sentence conditional probabilities. On a training set of 38 million word Iyer and Ostendorf (1999) use a more sophisticated technique for sentence clustering. In experiments where both papers use the same amount of sentence classes results by Iyer and Ostendorf (1999) reach about twice the perplexity reduction as Goodman (2001).

As far as the author knows sentence mixture models have not been used for machine translation. However, they might be an extremely valuable addition for machine translation. The reason for this is that in previous usage on speech recognition, there was the problem that sentence mixture models can only be used after the whole sentence has already been recognized. This is making them worthless for one pass recognition systems, and reduces their usage to n-best rescoring. Yet, in machine translation, there is the possibility to actually do the clustering on the *source side*. This provides the opportunity to use *completely correct* knowledge as starting point for assignment of a sentence to a cluster. Therefore, sentence mixture models might yield bigger improvements for machine translation, than they had for speech recognition.

This thesis refers to clustering on the source side as bilingual sentence mixture model, whereas the traditional clustering on a first pass translation is referred to as monolingual sentence mixture model. The following figures should help to understand the tasks involved in applying a bilingual sentence mixture model.

Figure 2.1 shows the initial situation. In the upper part of the figure the available training data is shown in dark grey boxes. In addition to the always needed language model training data on the right, there is some further training data needed. This training data must be bilingual and possess an aligned document structure between the languages. In the lower part of the picture it can be seen that the test set is also already known. The middle part of the figure is still empty, as the components there have first to be produced by clustering. The separation in original training data, models that are produced by clustering and actual test data is also indicated on the left border of the figure.

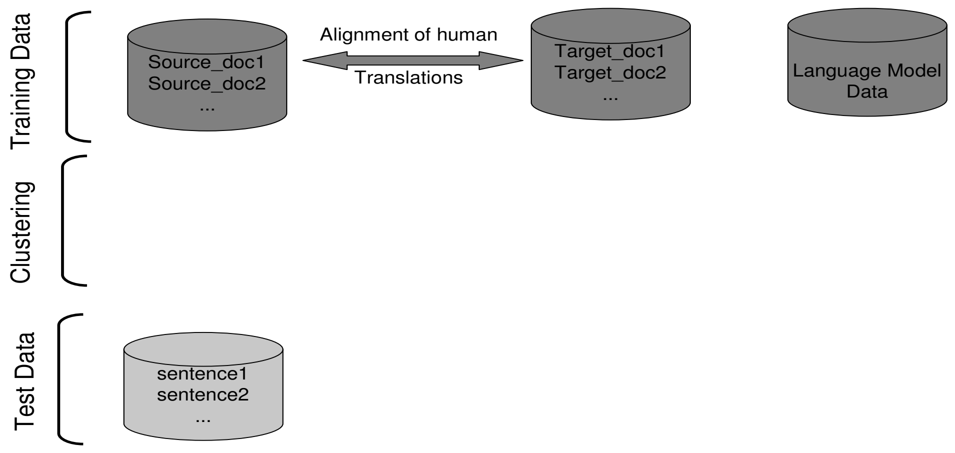


Figure 2.1: Idea of Bilingual Sentence Mixture Models I

The final situation is depicted in 2.2. At first a bilingual clustering on the training corpus is performed. This clustering can for example be performed by the Term Frequency Inverse Document Frequency (TFIDF) metric and k-means on top of it. The result is a mapping of documents IDs to clusters. As the documents are aligned between the languages, this mapping is enough to tell which documents from the training data in both languages belong to which cluster. Using this bilingual clustering information, both the existing language model data and the test set can be clustered. As a final step, each of the clusters of the test set is translated with a weighted combination of different language models. The different language models are build on the clustered parts of the language model training data.

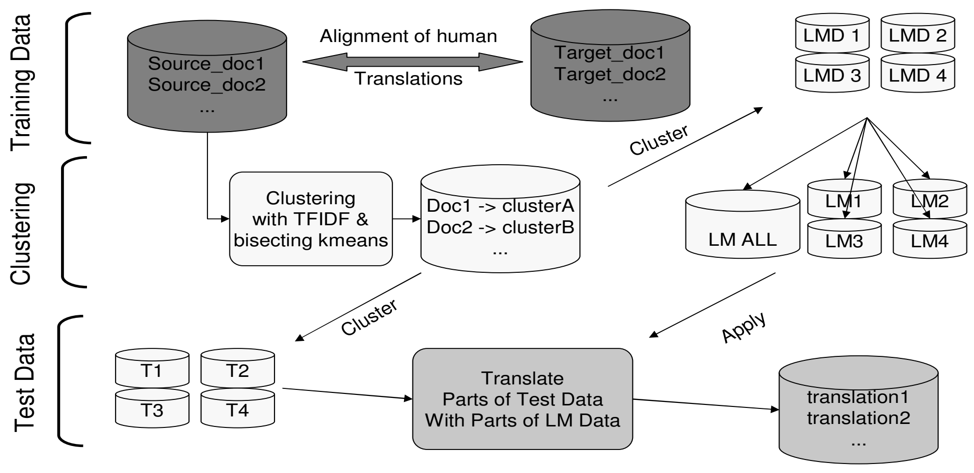


Figure 2.2: Idea of Bilingual Sentence Mixture Models II

There is also other work which states improvements by using topic information as in sentence mixture models. Wu and Khudanpur (1999) report with a very similar approach 12% relative perplexity rate reduction, but also state better gains were achieved when only considering content words.

2.7 Content based Adaptation of Language Models

Content based language model adaptation has been applied successfully to both speech recognition and machine translation (Eck et al., 2004) (Mahajan et al., 1999). Content based language model adaptation is usually building topic specific language models with a first pass translation output. Note that the sentence mixture model approach described in the previous section is also a topic and style adaptation of language models.

After this is done, the translation is done again with the adapted language model. Typically the adaptation is performed by using an information retrieval similarity metric. A common similarity measure for the adaptation step is the Term Frequency Inverse Document Frequency (TFIDF) metric. As the speaking name tells, this metric is able to cluster documents by collecting counts how often a word occurred in a document (term frequency). This term frequency is then modified by the inverse document frequency (IDF), which automatically reduces the impact of common words that occur in all topics on the classification. The use of an other similarity metric like Okapi does not have a strong effect on the result of the adaptation (Hildebrand, 2005).

The LEMUR Toolkit Callan and Croft (2006) is a ready to use toolkit that allows to perform the necessary information retrieval tasks. Eck et al. (2004) could report a nearly 4% improvement in NIST score with language model adaptation on a machine translation task.

2.8 Combination of Language Models

There are several ways to combine and evaluate the performance of a mixture of language models. A simple approach is to linearly interpolate the different language models and assign each model a specific weight according to the performance over a development set. More elaborated combination techniques try to combine concepts. An example of this would be that a combination of a skipping model with a cluster model would not only interpolate the two original models but further introduce a new skipping cluster model. This skipping cluster model is now skipping clusters instead of words in the history.

With an a approach like this Goodman (2001) was able to produce a perplexity reduction of 50% and the word error rate in speech recognition was reduced by 8,9%. These results have been achieved by conceptually combining 7 different language models. However, this improvement was only made by rescoring 100-best lists. There is hope that including this very large mixture of language models could result in even better word error rate reductions when included directly in the speech recognition decoder, or similarly in the machine translation decoder. Of course, to actually perform such a decoding would be very complex, and exceed the timeframe of this thesis. The next subsection will now present a further addition how the combination of separate language models can be improved.

2.9 Meta Cache Language Model

The term meta cache language is not yet used in the literature. However, it seems to be a fitting name for the following idea. The idea is to change weights of participating language models dynamically in a multi language model environment. At the beginning the sum of language model weights has to be one, during the process this can change. The rules for this dynamical adaptaion of the language model weights could be as follows

- If a model has suggested the winning hypotheses for the last three words, this model is regarded to fit the current text and its weight is increased by adding 0.02.
- If a model has suggested wrong hypotheses for the last three words, this model is regarded to be less useful for the current text and its weight is decreased by subtracting 0.01.
- To ensure that all models can continue to have influence on the resulting hypotheses, some borders are introduced to keep the weights in a certain zone. Of course, this zone is depending on the number of participating language models. The following formula could be used
$$\frac{1}{2x\#LM} < x < \frac{2}{\#LM}$$
, where #LM is the actual number of different language models in the current run and x stands for the possible values the weights of the language models can take.

The assumption is that if a language model is predicting *correct* (where correct still has to be determined) for some words, it seems that the language model fits the current situation. Thus the language model impact should be increased for the next words. In an easy attempt, *correct* could be defined as the winning word. Whenever a word is selected for the first best hypothesis, and the language model supported this word, the suggestion of the language model was correct. Of course, more sophisticated definitions of *correct* can be used, for example a confidence measure.

2.10 Summary

This literature review has analyzed several language modeling techniques for the integration of long term context dependencies. At first an introduction to the currently used n-gram language modeling was given, including its weaknesses in the integration of longer contexts.

The problem with large contexts is that the models usually need exponential more parameters to model them. Many parameters however need huge amounts of training data.

Thus some techniques to reduce data sparsity like the class based LM and skipping models where analyzed as well as techniques like suffix array LMs, which can cope better with larger training data. Other techniques discussed in this literature review where dealing with the consideration of topic (sentence mixture models, content based adaptation) and/or the combination of different language models (combination of LMs, meta cache LM).

From the several techniques, it was clear that this thesis can only analyze some regarding their use for machine translation. Suffix array LMs and content based language LM adaptation were already applied to machine translation. Hence there was no point in doing these experiments again. The same of is of course true for the regular n-gram approach.

With these techniques being out of the consideration, the following three approaches were chosen to become evaluated further in this thesis. The class based LM, the cache LM and the sentence mixture model. The reasons for the selection were the difficulty of the implementation and the expectations regarding their effect on SMT.

Chapter 3

Data and Implementation

3.1 Baseline Language Models

Of course, for some experiments new language models have been built, however, if there was no need to build a new one, most of the time one of the language models in table 3.1 was used. Different preprocessing refers for example to different treatment of number tokens

Name	#word	Corpora
Xinhua_M220_95-04	220 Million	Xinhua from Giga Word Corpus
xin_all.gpp.220M	220 Million	Xinhua from Giga Word Corpus, different preprocessing
IBM_All.370M	370 Million	Gale-IBM

Table 3.1: Different Language Models

and/or sentence marks. Yet the exact preprocessing is not very important for the rest of this thesis.

3.2 Evaluation corpora

3.2.1 Spanish BTEC Travel corpus

The Spanish to English medical BTEC translation task (500 sentences, 7.5k words). The original document boundaries of this data have not been available, and hence not been used. A bi-directional IBM model 1 lexicon has been trained on 123k sentences training data which contained roughly 900k words. Due to this relatively small IBM model 1, BTEC translation experiments could be set up to be pretty fast when translating with a small initial language model. Thus the BTEC corpus was the first choice for initial experiments and parameter tuning. The reference contained only one reference translation.

3.2.2 Japanese BTEC Travel Corpus

The Japanese to English medical BTEC translation task (500 sentences, 3.7k words). Similar to the Spanish BTEC Travel Corpus, there are no document boundaries available and the experiments run pretty fast. In addition to this there was also the IWSLT05 test set available, which was very similar in terms of its describing numbers (506 sentences, 3.8k words). The reference contains 16 reference translations for both test sets. The training data for the models consisted out of 160k words.

3.2.3 Darpa TIDES Data Sets

Many experiments were performed on test sets from the Translingual Information Detection, Extraction and Summarization (TIDES) DARPA research project. Only the Chinese-English and Arabic-English data has been used, most of the time the MT03 data. Though sometimes other data like the MT02 has been used due to comparability to previous results, which are occasionally not mentioned in this thesis. However, in each case four reference translations have been used for the evaluation.

The Chinese MT03 data consists out of 919 sentences. Together the references have 101,129 words and 3676 lines. The Arabic MT03 contains 663 sentences. The references have 2,652 lines and 70,012 words.The Chinese MT02 data has 3,512 lines and 106,925 words, whereas the Arabic MT02 data has 2,912 lines and 91,609 words. All of the mentioned test sets are partitioned in 100 documents, although document boundaries have only been used by experiments on the MT03 data.

Many of the n-best list rescoring experiments have been performed on n-best lists from the HIERO system. Some translation experiments used a phrase table provided from IBM for the Rosetta dry run in the GALE evaluation 2006. The phrase table contains about 160k phrases, each of them has five features. Only three of the features are originally provided from IBM ($P(S|T)$, alignment score, $P(T|S)$, alignment score, $P(T|S)$, relative frequencies), in addition to that the 4th feature is describing phrase target length and the last feature is for phrase target length balancing.

The MT05 data was used as real unseen translation test set. The Arabic MT05 data consists out of 1056 sentences, and is evaluated against four reference translations. The Chinese MT05 data consists out of 1082 sentences and also has four reference translations.

3.2.4 Gigaword Xinhua Corpus

This is a monolingual English corpus with document structure. More details about certain data sets which have been used in experiments are given in table 3.2.

name	#documents	#words
xin2003	117k documents	27 million words
100 docs	100	18k words
1000 docs	1000	193k words
10,000 docs	10,000	2 million words
100,000 docs	100,000	20 million words
xin2000 - xin2004	-	123 million words

Table 3.2: GigaWord Xinhua Data

3.2.5 Xinhua LDC Corpus

A Chinese English corpus with document structure, most of it is monolingual, but there are 20k bilingual documents available.

name	#documents	#words
English 2001	85k docs	17 million words
English 2000	104k docs	21 million words
English 1999	99k docs	21 million words
English 1998	97k docs	21 million words
English 1997	89k docs	19 million words
English 1996	88k docs	18 million words
English 1995	80k docs	17 million words
English 1994	64k docs	14 million words
English 1993	64k docs	14 million words
English 1992	75k docs	18 million words
Altogether	845k docs	180 million words

Table 3.3: Xinhua LDC Data

3.2.6 WebBlog Data

The webBlog data is form a variety of different blogs over a time of only 20 days. The file names indicate the day. Though there was already substantial work done to clean the data, the data is still much nosier than regular language model data. Some blogs are about advertisement, some blogs are in different languages and some blogs just say a thousand times 'bored'. The whole data set is rather large, about 1,5 billion word tokens.

name	#documents	#words
480 docs	480 docs	85k words
1000 docs	1000 docs	140k words
04.file	-	70 million words
05.file	408k docs	80 million words
06.file	433k docs	77 million words
07.file	486k docs	76 million words
20 files altogether	-	about 1,5 billion words

Table 3.4: WebBlog Data

3.3 Implementation Details

3.3.1 Word Cache Language Model

The original idea was to use the cache language model of the SRILM toolkit (Stolcke, 2002). This cache model implementation basically keeps track of the *history length* number of words and calculates a maximum likelihood estimation for new words.

$$P_c(w) = \frac{\#\, word\, in\, history}{\#\, words\ in\, history}$$

This probability is then interpolated with a background regular n-gram model. The different λ's are weighting factors for the specific models.

$$P(w) = (1 - \lambda_c) \times P_{ngram}(w) + \lambda_c \times P_c(w)$$

However, it soon became clear that the cache model from the SRILM is very basic, and some expected functionality is not provided or not publicly accessible. For example the method to flush the cache was only available from inside the cache class, and there was no method to retrieve probabilities without updating the cache. Yet, this is crucial functionality for some of the experiments. Hence the cache model implementation has been changed to incorporate this functionality. This changed implementation is given in appendix B.

3.3.2 Bi- and Trigram Cache Language Model

As literature indicated this to produce better results, the word cache model was also extended to keep bigram and trigram histories as well. The bigram probability was estimated as follows:

$$P_c(w) = \frac{\#\, bigram\, in\, history}{\#\, bigrams\ in\, history}$$

This of course is not the usual estimation for bigram probabilities, however, the bigrams are collected on very limited data and, even worse, the history itself might be wrong. So this approach was chosen to slightly prefer bigrams seen in history, without eliminating the possibility to prefer different translations. The same is done for trigrams. After all, the probability of the cache model is now calculated by the following formula

$$P_c(w) = (1 - \lambda_b - \lambda_t) \times P_{word}(w) + \lambda_b \times P_{bigram}(w) + \lambda_t \times P_{trigram}(w)$$

Due to readability $P_c(w)$ was written instead of $P_C(w_i|w_1...w_{i-1})$, which should be obvious. To leave no doubt about how certain things have been implemented, the source code of the cache model is given in appendix B.

Cache Minimum Error Rate Training

This short section describes how the cache parameters have been integrated in the *Minimum Error Rate Training* (Och, 2003) of the STTK CMU toolkit. Unfortunately, much of this information relies heavily on the STTK toolkit, however, it is tried to give an as general as possible description.

One design issue was that it would greatly help to have completely independent parameters to minimize possible side effects from dependencies. As a consequence, the main translation engine is now collecting scores from all four language models and evaluating its hypotheses with respect to these scores. The for language models are the regular ngram background model, word cache, bigram cache and trigram cache model.

A single cache model without interpolation is quite often returning a zero probability, or as the actual calculation is performed with logarithmic probabilities, an negative infinite logarithmic probability. Such a score however would make the whole sentence completely impossible to chose for the translation engine. As a solution, if the word should have a zero language model probability its probability is actually set to one divided by the vocabulary size.

Another issue to be dealt with was that the cache model depended in its code quite heavily on the connection to the background ngram model. For example the translation engine kept only integer hashes to word strings for the word history. This mapping is performed by the vocabulary of the ngram model. As this word history information in the translation engine is also used for internal pruning, it can not be easily extended to store multiple histories. The problem is that the original vocabulary of the ngram model is limited, which causes *unknown* word signs in the cache history. The solution was to extend the vocabulary of the ngram model with every new word. This guarantees that there are no *unknown* word signs in the cache history. If the translation engine is querying the background model for such a word, similar to above a backoff probability of one divided by the vocabulary size is returned instead of a zero probability.

The optimization itself is performed on nbest lists. The scores from every model (this means from the four language models as well as other translation models) are stored together with the generated hypothesis. The optimizer now tries to find weights for these models that their combination is giving high scores in an optimization metric. The score used in this work was always the BLEU score. This is only possible if the optmizer knows the reference, so this can not be performed on real test data.

3.3.3 N-best list Cache Agreement Model

A paper from Zens and Ney (2006) reported very recently improvements from n-best list agreement rescoring. Not only that the cache model already provides a framework for this, it also offers the possibility to perform similar operations in the decoding itself. Hence the cache model was altered to include all hypotheses the decoder is generating, weighted by their translation probability. The following list sums up the step which had to be performed.

- Put all hypotheses for a sentence in the cache for the last 10 sentences, weight them by their translation probability
- Add fields to hash table where information for each sentence is stored
- Change window to work over sentence ID
- Experiment with adding all n-best list information from current sentence before rescoring

One additional advantage is that this approach reduces the data sparsity problem inherent to the regular cache language model. Thus, as data sparsity is no longer an acute problem, bigrams can now be estimated by the better formula

$$P(w_1 w_2) = \frac{\# w_1 w_2 \, in\, history}{\# w_1 \, in\, history}$$

The same was done for the probability calculation of trigrams. In combination with this restructuring the cache model code was reviewed and its correctness was validated by a memory checking tool to find memory leaks.

3.3.4 Class Based Language Model

It was tempting to use the clustering algorithm of the SRI toolkit, as its output is immediately in the correct form to produce class based n-gram language models. First experiments however where disappointing, both in expectations of runtime, possibility to process large data and quality of clustering. For example, one example on a 16 million word corpus finished reasonably fast, however it clustered 65000 of 65314 words in one of the desired 50 classes.

This experience lead to the use of the other clustering toolkit mkcls (Och, 1995). The approach to build a language model was to replace each word in the original corpus by its class and then build a regular n-gram language model on this corpus. Of course, this class language model has to be interpolated with a regular language model. In the following formula W is the word class word w is belonging to, CBLM is referring to Class Based Language Model and BLM is the Background Language Model.

$$P(w_1...w_i) = (1-\lambda)P_{BLM}(w_1...w_i) + \lambda P_{CBLM}(W_1...W_i)$$

Four different sizes of classes have been used for clustering (50, 200, 500 and 1000). The following command was used to generate the clusters

```
mkcls -p inputText -c X -V outputFile -s 3600 - n 50
```

where X is replaced by the desired number of classes. The -n parameter controls the number of iterations, similarly -s sets the maximum execution time in seconds. Actually, always the -s parameter stopped the clustering algorithm, thus to perform 50 optimization iterations the time limit need to be increased. However, for initial experiments it seemed appropriate to cluster every class system in one hour.

As the runtime of clustering algorithms is usually dependent on the number of words pre-tokenized training data should be used. The only token used here was a number token, but this is already reducing the vocabulary of training data significantly. The actual training data was a subset the English part of the Arabic English GALE training data. This subset contained 500k sentences, roughly 16 million words and 65k vocabulary. The whole Arabic English training data has 3414k sentences, 112 million words and a vocabulary size of 205k.

The experiments were run on the Arabic English MT03 test set.

3.3.5 Sentence Mixture Model

One prerequisite of sentence mixture models is to have a clustering algorithm for sentences. Simple approaches to this would be *Term Frequency Inverse Document Frequency* (TFIDF) or a Naive Bayes Classifier. However, when looking at the 27 million word initial experiment data form the 2003 xinhua news, it becomes clear that TFIDF would already penalize some words which certainly are relevant for the type of a sentence like Iraq or government.

During the search, there was also no toolkit available which deliberately stated to be used for sentence clustering. There are, however, some toolkits for document clustering like BOW (McCallum, 1996) and LEMUR (Callan and Croft, 2006). It seemed reasonable to use documents as initial starting clusters, as this should both make it easier to extract relevant words and at the same time speed up the whole process.

For two reasons, the LEMUR toolkit was used rather than the BOW toolkit. First, it is still being developed and makes a much more mature impression. Secondly, the BOW toolkit is not offering any documentation at all for it's clustering component.

When analyzing the LEMUR toolkit, however, it became clear that clustering is none of it's main areas of interests. Thus the otherwise very good documentation was not as good for the clustering components. To make it even worse, there where several clustering tools in the LEMUR toolkit, each offering a certain functionality. Yet there was no possibility to store outputs of one tool, to use it as input for another. The following list shows the initial tools and their abilities. All of the tools work on indexes build on a text file, rather than on the text file itself.

- **Cluster**: takes a text with documents, looks at each document once and compares it to all previous documents, it is not possible to adjust the number of generated clusters
- **Offline Cluster**: does essentially what is needed for sentence mixture models, however, it has no possibility to save its clusters and is restricted to work with 100 documents or less. Uses algorithms from Steinbach et al. (2000).
- **Probabilistic Latent Semantic Analysis (PLSA)**: takes a text with documents and performs a latent semantic analysis with a given number of latent variables, outputs a table with probabilities for the latent variables. Uses algorithms from Schein et al. (2002).

After this analysis, there seemed to be two possible solutions to use the LEMUR toolkit for the needed task. The first option was to change the programs offline cluster and cluster so that they can save and load their results. The other solution would have been to parse the output tables from the PLSA and use them to classify the documents.

Yet, is was hard to make sense out of some initial PLSA results, and reading the paper from Schein et al. (2002) did not help. There are no descriptions of the algorithms and the issue analyzed there seemed far different from the task of text document clustering. Hence solution one was implemented. As the clustering itself was only a needed tool for sentence mixture models, some changes are probably not very nice fixes to make the desired functionality work. The changed code is given Appendix C. The next subsections first depict the intended design, whereas the second subsection describes the actual used design, as there were problems to implement the intended design.

Original Design Intention

- **New parameters for Offline Cluster**
  - clusterIndex: this index will contain the clustering information after the clustering
  - clusterdb_type: type of the cluster database, has to be flatfile at the moment
  - clusterAlg: kmeans or bkMeans (for bisecting kMeans, Steinbach et al. (2000))
  - bkIters : has been removed, use maxIters instead
- **New parameters for Cluster**
  - update: if 1 documents from the index will be added to the existing cluster database
  - clusterDatabaseIndex: refers to the original index which was used by Offline Cluster, should be clear when looking at 3.1

After all this is done, figure 3.1 shows the overall processing which has to be performed to do clustering with the LEMUR toolkit. As can be seen, it is just a push button system where 20 buttons have to be pushed at the right time in the right order. The TREC file format refers to the Text Retrieval Conference (TREC) file format.

Some remarks about this process. First, the cluster program has a parameter threshold. When this parameter is set to zero, all new documents will be clustered to the most likeliest cluster in the cluster database, no matter how low this likelihood is. In the case that threshold is set to nonzero, all new documents whose maximum correlation with an existing cluster is less than threshold will be assigned to a new cluster. Together with an appropriate post processing script that takes a parameter how many intended clusters there are this can be used to ignore documents which have very low similarity to one of the clusters in the cluster database. As mentioned, even after the whole process illustrated in 3.1, there still has to be a post-processing step. For the work conducted here a perl script was used that marked each sentence in the *Text to Cluster* with its corresponding cluster number, or deleted it, if it was not assigned to any of the desired clusters.

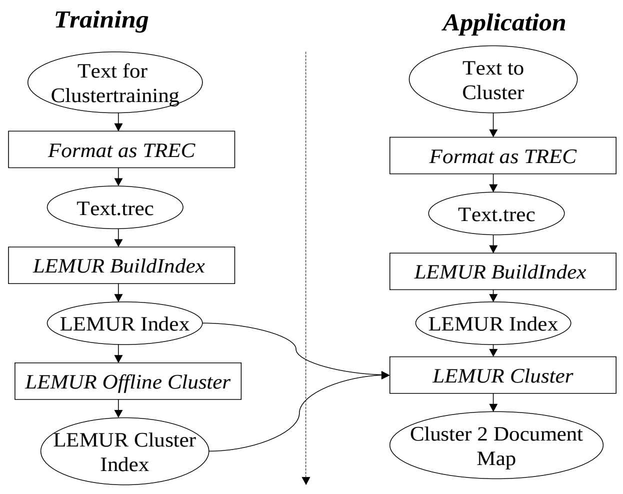


Figure 3.1: Clustering with the LEMUR Toolkit

Changes to the Intended Design

After initial experiments, it became clear that the index managers of the LEMUR toolkit internally worked only with integers to refer to different documents. As a consequence, one index manager is certainly not able two manage two different texts (respectively indexes). Introducing two index managers however makes everything more complex, as all components are designed to work with only one index manager.

As the clustering is not in the main interest, the following simple solution was used. The actual data to cluster was appended to the training documents. The new work flow is shown in 3.2. The disadvantage is obvious, there seems to be no separation between training and application of the clustering. Yet, it is still possible to use one training run for different clustering applications. In order to achieve this, the training data has to be prefixed to the new application data, a index has to build on the whole text, and the already existing cluster database index has to be linked.

These changes also affected some parameters, so the new list of parameters is given here.

- **New parameters for Offline Cluster**
  - clusterIndex: this index will contain the clustering information after the clustering
  - clusterdb_type: type of the cluster database, has to be flatfile at the moment
  - clusterAlg: kmeans or bkMeans (for bisecting kMeans, Steinbach et al. (2000))
  - bkIters : has been removed, use maxIters instead

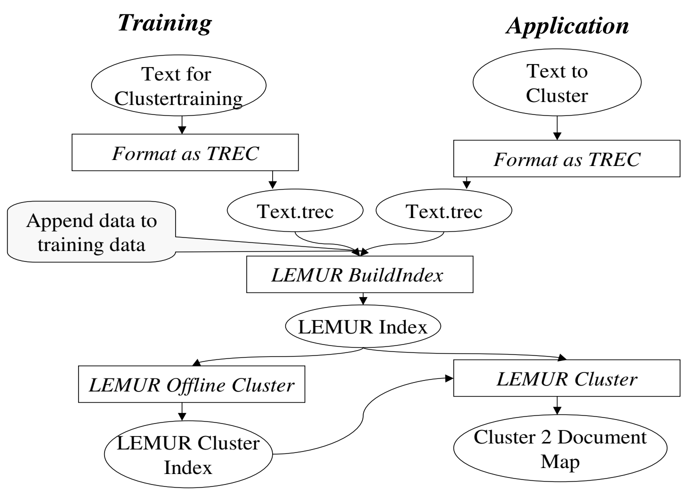


Figure 3.2: Clustering with the LEMUR Toolkit

  - numToCluster: how many documents of the the index are used for the training

- **New parameters for Cluster**
  - threshold: set to non zero to ignore completely different documents

In addition, the parameter *index* has been changed for both tools to *docIndex* to indicate that this is the index which actually contains the documents.

3.3.6 Language Model Adaptation

Essentially there are already a lot of different techniques tested for language model adaptation Rosenfeld (2000) Zhao et al. (2004). Actually some other techniques already used in this thesis can also be counted to adaptive language modeling, for example cache and sentence mixture language models. Basically the believe is out there that adaptive language modeling should help, however most work only reported small benefits.

The idea of adaptive language modeling in this section is basically the same as in Zhao et al. (2004). The motivation why these similar experiments are performed is first of all to verify the conclusions of them. Secondly, previous experiments for the sentence mixture language model already indicated impressive improvements of perplexity (table 4.29). Finally, these experiments can be performed with the tools and programs already designed for the sentence mixture models.

For information about the actual implementation it is referred to the section about sentence mixture models. The only difference is, that the number of clusters was set to one, and the threshold which deletes documents was raised. This leads to one data set which contains all language modeling data which has similarities with the test set.

Chapter 4

Experimental Results

4.1 Class based Language Model

4.1.1 Perplexity

The perplexity numbers reported here are produced on the 16 million words training data by the mkcls clustering tool. Further information about the training data is given in the section with implementation details (3.3.4). The clustering tool was set to start with random initial clusters. The perplexity of this random clustering on the training text is referred to as Start-PP in table 4.1. Starting from this initial clustering, a deterministic algorithm is used for the refinement of the clustering. The algorithm first selects a word according to its frequency in the text and swaps it to the cluster where it produces the highest drop in perplexity. End-PP finally shows the result of the perplexity driven clustering. As the result depends on the number of optimization runs, this information is also given in table 4.1. The actual numbers of optimizations runs was not set before, rather a maximum number of iterations in combination with a time limit for maximum duration was given. More details about the mkcls tool and the clustering performed is given by Och (1995).

Number of Classes	Start-PP	End-PP	Optimization Runs
50	584	235	14
200	433	166	4
500	315	133	2
1000	230	112	1

Table 4.1: Mkcls Training Information

The results are clearly in favor of more classes, however these numbers are only reported on the training data. Hence a second experiment is measuring perplexity numbers on a different test (table 4.2). This time a lower number of classes was used compared to the first experiments, to keep the necessary language models smaller and faster. The test set consisted of 4k sentences of the Arabic GALE data. It has 122k words, among them 9641 different words. Words which have not been in the training data are treated as usual unknown words by the language model.

Number of Classes	SRI 3-gram	SRI 6-gram
50	17.61	17.87
100	26.51	27.03
200	39.80	40.46

Table 4.2: Class LM Perplexity

The effect that more classes result in an increase of the preplexity is expected, as the more classes mean having a larger vocabulary. As a consequence, these numbers can not give final answers, for that the translations of the next section are needed.

4.1.2 Translation

The translation experiments where performed on the Arabic English MT03 TIDES test set (see 3.2.3). First experiments were run with many parameters set to optimize speed rather than quality of the translations. For example the n-best list was only a 10-best list and the beam size for the search was only set to a very small value. This is mentioned here as it might affect the usefulness of the class language model. The results can be seen in table 4.3. The class language model used was trained on the subset of the Arabic English training data. Both the regular n-gram and the class language model considered histories up to a length of 6-grams. The difference between run one and two is that run one uses exactly the

#Classes	no	10	50	100	200
run1 BLEU	45.70	45.22	45.91	45.37	45.19
run1 Scale	0.0	0.008	0.244	0.163	0.116
run2 BLEU	45.70	44.96	45.44	45.78	45.45
run2 Scale	0.0	0.0	0.106	0.062	0.167

Table 4.3: Class Language Model Translations

classes produced by mkcls, whereas run two uses the classes from mkcls in general, but adds additional classes for sentence marks. The scales are automatically determined by the MER training of the STTK decoder.

As a consequence of this experiment, in further experiments it was switched back to use the clustering of mkcls without any changes for sentence marks. In the next experiment, some parameters have been set to more reasonable values which should improve the overall score. First of all, the clustering itself was given much more time, ten hours instead of one hour. The class LM was set to evaluate histories up to a length of 10-grams. Also a 500-best list and a bigger beam size in the actual decoding was used, and last but not least the number of MER iterations was doubled. As the first experiment showed best results for 50 classes, and this experiment is taking much longer, table 4.4 gives only a result for 50 classes compared to a baseline with similar parameters.

#Classes	no	50
BLEU	47.53	47.52
Class LM Scale	0.0	0.004

Table 4.4: Improved Class Language Model Translations

4.2 Word Cache Language Model

4.2.1 Perplexity

First an experiment was made to estimate a good size of the cache. The experiments were performed with the SRI toolkit. The experiment was done on the 370 million word corpus for Gale by IBM (Gale-IBM). However, the trigram language model was only build on every 6th sentence of the 370 million word corpora which resulted in a 43 million word corpus. The 4000 first sentences of the 370 million word corpus were used to for the perplexity benchmark (110k words). The results can be seen in table 4.5. Obviously the cache model enhanced the

Cache Size	no	100	150	180	200	400	600	800
Perplexity	179.4	149.0	147.9	147.7	147.8	149.4	151.6	153.2

Table 4.5: Influence of Cache Size

performance of the language model, the best perplexity reduction of 17,6% was reached with a cache size of 180 words. Due to this positive result, a further test was made, measuring how different interpolation weights of the cache model influence the perplexity. As can be seen in 4.6, heigher weights give an additional gain, though not significant. The relative reduction from the standard cache model weight of 0.05 to 0.15 is 2.2%. After the initial experiments have been quite motivating, the next experiment tries to reduce the possibility of coincidence by testing the influence of cache models on a number of different language models. For better comparison to established results, the perplexity was measured on the Chinese-English (107k words) and Arabic-English (91k words) MT02 reference texts. Table 4.7 gives description about the different language models used. Results are given in table 4.8.

Cache Model Weight	0.00	0.05	0.10	0.15
Perplexity	179.4	147.7	144.5	144.4

Table 4.6: Weight of Cache Model

Name	#word	Corpora
Xinhua_M220_95-04	220 Million	Xinhua from Giga Word Corpus
xin_all.gpp.220M	220 Million	Xinhua from Giga Word Corpus, different preprocessing
IBM_All.370M	370 Million	Gale-IBM

Table 4.7: Different Language Models

4.2.2 N-Best List Rescoring

The n-best list rescoring was performed on the TIDES MT 03 data with a 1000 best list from the Hiero2 systems. The BLEU score was evaluated with four references. The SRI Toolkit cache model was used, which is a simple word cache model. The rescoring was purely done on LM score, without considering the rank in the n-best list. It seems reasonable that considering this would give higher BLEU scores, however, the aim of these experiments was to show improvements from the cache model.

Results of rescoring a 1000bestList are given in table 4.9. The language model used was again the 220 million Xinhua from the Gigaword Corpus. The first rescoring results show a slight drop, as there was no other feature to balance the language model and the language model prefered the short translations. To overcome this, the arithmetic mean of the probability was calculated for further experiments (indicated by AM in the table). Unfortunately the cache model hurts the performance in all cases. After analyzing the data, a possible reason seemed that document boundaries are not considered. Hence some more experiments were performed to evaluate the effect of flushing the cache for each new document. For speed reasons, these experiments have been performed only on a 99bestList of the same translation task. The results in table 4.10 are motivating, as the cache model which considered the document boundaries (DB) could get a small gain. Obviously, the next step was to transfer this gain to the 1000bestList. Yet, this experiment did not show similar benefits, the rescoring

	ch-en.mt02.ref	ch-en.mt02.ref +CACHE	ar-en-mt02.ref	ar-en.mt02.ref +CACHE
Xinhua_M220_95-04	184.5	140.0	144.8	117.8
xin_all.gpp.220M	138.2	114.5	121.0	103.5
IBM_All.370M	203.0	154.9	150.2	123.4

Table 4.8: Multiple LMs and Cache Model

System	no Rescoring	xin220m	xin220 +AM	xin220 +Cache	xin220 +Cache +AM
BLEU	31.44	29.15	31.88	28.60	29.64

Table 4.9: 1000best List Rescoring

System	xin220m +AM	xin220m +Cache +AM	xin220m +Cache +DB +AM
BLEU	30.09	30.02	30.39

Table 4.10: 99List Rescoring

result with xin220m + Cache + AM + DB was worse than simple rescoring with the xin220m language model (BLEU 0.3031). When analyzing this strange behavior, it became clear that a bug in the cache implementation by the SRI Toolkit caused the problem. Even though the SRI Toolkit offered a method running(), which is intended to switch of updating dynamic language models, this did not work with the cache model. After fixing this issue, the cache model clearly improved the performance (table 4.11). All systems had the arithmetic mean and all cache models had in addition the document boundary check. The last experiment in this table with a cache size of 2000 words has been made to confirm the cache size of 180, even though literature pointed out different sizes of cache. The next few experiments

System	noRescoring	xin220m	xin220m +Cache180	xin220m +Cache2000
BLEU	31.44	31.88	31.97	31.95

Table 4.11: 1000bestList correct Rescoring

were performed to evaluate both the influence of document boundaries and if the cache can give additional benefits when interpolated with higher weight (0.1 instead of 0.05). As table 4.12 shows, the influence is not really important. It is interesting to see that the document boundaries are not crucial, which is probably a nice attribute for implementations in complicated toolkits/decoders. The reason for this is probably that the cache is empty whenever a new document gets processed. It seems reasonable to assume that the influence of document boundaries is depending on the evaluated data.

4.3 Bi- and Trigram Cache Language Model

4.3.1 Initial Experiments

The literature review indicated that the additional use of bigrams or trigrams can enhance the performance of the cache model. However, the experiments conducted here did not benefit

System	xin220m +Cache	xin220m +Cache(0.1)	xin220m +Cache +noDB	xin220 +Cache(0.1) +noDB
BLEU	31.97	31.98	31.80	32.02

Table 4.12: Cache Weight and Document Boundaries

from using an extended cache model. The following simple interpolation method was used for the cache model

- If trigram seen in history,
  $p_{Cache}(w_1...w_n) = p_{word}(w_n) + (3 * p_{bigram}(w_{n-1}w_n)) + (5 * p_{trigram}(w_{n-2}w_{n-1}w_n)))$
- If bigram seen in history,
  $p_{Cache}(w_1...w_n) = p_{word}(w_n) + (3 * p_{bigram}(w_{n-1}w_n))$
- If word seen in history,
  $p_{Cache}(w_1...w_n) = p_{word}(w_n)$

Of course, this interpolation gurantees no longer that probabilities sum to one, but it is having the advantage of not reducing the overall weight of the cache compared to the weight of the ngram language model. The partial probabilities like for a word are just the number of occurences in the cache history divided by the number of total units in the cache history. The cache histories were kept and organized completely separately for words, bigrams and trigrams. The backup step in the list performs the same behavior as the cache model which was used for the previous word cache experiments. All experiments have been performed with the 220m Xinhua language model. Some results are given in table 4.13. After the first result in this table was not as good as expected, the next experiments have been tried with a bigger history, as this might help to get better statistics about the trigrams. However, as the results show, none of the experiments gave an improvement. As a consequence for further

System	Cache180(0.1) + noDB	Cache180(0.1) + noDB +Tri +Bi	Cache2000 +DB +Tri + Bi	Cache2000 +DB +Tri
BLEU	32.02	31.97	31.98	31.97

Table 4.13: Bi- and Trigram Cache Model

experiments the following interpolation method was used

$$p_{Cache}(w_1...w_n) = (1-\lambda_b-\lambda_t)\times p_{word}(w_n)+\lambda_b\times p_{bigram}(w_{n-1}w_n)+\lambda_t\times p_{trigram}(w_{n-2}w_{n-1}w_n)$$

to give better chances that probabilities sum to one

4.3.2 N-best List Rescoring

The initial experiments did not show any improvements from bi- or trigram cache models, yet this might be due to wrong guessing of parameter weights. The following list shows all (performance) parameters the n-best list rescoring script is taking.

- **cacheWeight**: interpolation weight of the cache model respectively to the ngram model
- **cacheLength**: how many tokens are kept in the cache
- **unkIsWord**: are tokens with an unknown word assigend a probability?
- **bigram**: weight of the bigram respectively to word and trigram cache
- **trigram**: weight of the trigram respectively to word and bigram cache
- **docBound**: are document boundaries used to flush the cache?
- **nBestOpt**: weight how much the rank of the hypothesis is considered during rescoring

An extensive grid search with all this parameters would have been quite expensive. Thus to reduce the amount of calculation needed, it was nice to fix as many parameters as possible. It seemed quite reasonable to assume that the *nBestOpt* parameter is not affecting any of the other parameters, and as it was not used in previous word cache experiments, it was set to zero. Similar is true for *cacheWeight*. Actually a little bit too high weight for the cache model (0.2) was chosen as fixed, as the author hoped that this might help to accentuate small differences of different cache language models more. Finally the *docBounds* were ignored, as previous experiments showed that this does not hurt the performance. In addition, this freed these experiments from having to trace impacts of the document size inherent to the development data and the actual *cacheLength*.

Thus the grid search still had to deal with different parameters for *cacheLength*, *unkIsWord*, *bigram* and *trigram*. Table 4.14 shows the results of the performed 64 experiments. In the leftmost column, the weight for the bigram- and trigram parts of the cache model is indicated. For example bi3tri1 means *bigram* is set to 0.3, and *trigram* is set to 0.1. For some reason the bi3tri1 did not deliver any scores, when rerunning these experiments it became obvious that for some strange reason the c++ program stopped with the error, that (1.0 - bigram + trigram) + bigram + trigram is not equal to 1.0?! To stop this problem to occur in further experiments, instead of aborting, the program is now only issuing a warning when the probabilities are not summing up to one. However, as the missing experiments are not crucial, they have not been rerun with the new version.

	Cache180		Cache500	
	unkIsWord	unkNoWord	unkIsWord	unkNoWord
bi0tri0	31.91	31.91	32.00	31.98
bi0tri1	31.91	31.91	32.00	31.98
bi0tri3	31.94	31.93	32.01	31.99
bi0tri5	32.00	31.96	32.00	31.99
bi1tri0	31.91	31.90	32.00	31.98
bi1tri1	31.93	31.92	32.01	31.99
bi1tri3	31.96	31.95	32.00	32.00
bi1tri5	32.02	31.98	32.02	32.00
bi3tri0	31.95	31.94	32.01	31.99
bi3tri1	?	?	?	?
bi3tri3	32.02	31.98	32.03	32.00
bi3tri5	32.00	31.94	31.96	31.92
bi5tri0	31.96	31.94	32.00	31.98
bi5tri1	32.02	31.98	32.04	32.01
bi5tri3	32.00	31.95	31.96	31.92
bi5tri5	31.89	31.89	31.89	31.88

Table 4.14: Grid Search with Cache Weight 0.2

Two conclusions can be drawn from these results. Firstly, in contradiction to previous experiments, a history length of 500 seems to provide slightly better results. Secondly, to regard the unknown word as a regular word seems to improve the performance slightly, too. Of course, both factors are not providing any significant differences. Still, for the next grid search (still with the abort for some probabilities), which tried to evaluate the impact of different cache weights, the history is fixed to 500 words, and the unknown word is always treated as regular word. Results of this experiment are given in table 4.15.

4.3.3 Translation

Although the previous experiments have not been too successful, experiments are performed to measure the real translation effect of a cache language model. In order to do this, the cache model has been integrated directly into different versions of the CMU STTK Toolkit. A first set of experiments was performed on the Spanish to English medical BTEC translation task (500 sentences, 7,5k words). The background language model was again the 220m Xinhua language model. For the experiments the best parameter values from the previous n-best list

	cw = 0.1	cw = 0.2	cw = 0.3	cw = 0.4	cw = 0.5
bi0tri0	32.00	32.00	31.96	31.91	31.90
bi0tri1	32.02	32.00	32.02	31.92	31.89
bi0tri3	32.01	32.01	32.06	31.98	31.91
bi0tri5	31.98	32.00	32.00	32.04	31.98
bi1tri0	32.01	32.00	32.01	31.92	31.89
bi1tri1	32.02	32.01	32.04	31.93	31.89
bi1tri3	32.02	32.00	32.02	31.98	31.94
bi1tri5	31.99	32.02	32.01	32.04	31.97
bi3tri0	32.01	32.01	32.02	31.97	31.90
bi3tri1	?	?	?	?	?
bi3tri3	31.98	32.03	32.01	32.00	31.98
bi3tri5	31.97	31.96	32.02	32.03	32.01
bi5tri0	31.96	32.00	31.99	32.01	31.92
bi5tri1	31.98	32.04	32.01	32.00	31.98
bi5tri3	31.97	31.96	32.05	32.01	31.96
bi5tri5	31.91	31.89	31.88	31.90	31.88

Table 4.15: Grid Search with changing Cache Weight

experiments in 4.15 have been chosen. As those parameters give the cache model a strong influence, one additional set of experiments was carried out with a cache weight of 0.05. The cache length was always 500 words. Table 4.16 lists the results.

One aspect which seems unreasonable is that the base scores are both times 18.90, no matter if the unknown word is treated as regular word or not. However, this is due to the fact that the CMU STTK toolkit is blocking the language model to give a probability for the unknown word, and actually assigning it to a fixed value depending on the vocabulary size. Having resolved this, it is still confusing that so many scores are identical between treating the unknown word as regular word or not. Yet, for the best parameter setting where a score of 19.07 is 0.9 % better than the baseline, it is pretty important to treat the unknown word as regular word. Luckily, the intuition that the parameters from the n-best list rescoring are too strong became proven, as the far best score was generated with the lowest cache weight.

When analyzing the results more in detail, it seems that the best cache weight is probably lying between 0.05 and 0.1. This conclusion is based on the fact that the cache model outputs usually higher probabilities the stronger its own probability is based on the word cache model. After saying this, it can be seen that for the cache weight of 0.05 the best score is achieved for a very high 0.9 value for the word cache model, whereas for the cache weight of 0.1 the best score is achieved by a cache model with only 0.6 influence of the word cache model.

However, the results seem quite unstable, and even the best result is not significant. Therefore, before trying to optimize the parameters further in the area of interest, the next

cacheWeight	word	bigram	trigram	BLEU -NoUnk	BLEU -Unk
0.0	-	-	-	18.90	18.90
0.05	0.9	0.0	0.1	18.79	19.07
	0.8	0.1	0.1	18.81	18.75
	0.6	0.1	0.3	18.81	18.81
0.1	0.9	0.0	0.1	18.75	18.75
	0.8	0.1	0.1	18.73	18.73
	0.6	0.1	0.3	18.96	18.92
0.2	0.4	0.1	0.5	18.66	18.66
	0.4	0.3	0.3	18.66	18.66
	0.4	0.5	0.1	18.67	18.56
0.3	0.7	0.0	0.3	18.07	18.07
	0.8	0.1	0.1	18.08	18.08
	0.2	0.5	0.3	18.61	18.61
0.4	0.5	0.0	0.5	18.15	18.15
	0.4	0.1	0.5	18.34	18.34
	0.2	0.3	0.5	18.48	18.48

Table 4.16: Cache BTEC Translations

experiment is evaluating the generalization ability by using similar parameters on a state of the art baseline of the current CMU MT03 Arabic to English machine translation system. Results are given in table 4.17. As the table shows, the same cache parameters which helped

cacheWeight	word	bigram	trigram	BLEU
no	-	-	-	43.21
0.1	0.8	0.1	0.1	42.21
0.05	0.9	0.0	0.1	42.40

Table 4.17: Cache Arabic CMU MT03

for the BTEC task hurt severely. Yet this might be because of previous optimization , which does not fit the changed decisions from the language model. Additionally, the overall scores are not that good, all of them, including the baseline should be higher. Thus, the same experiments have been rerun with better parameter values and with five iterations of the integrated minimum error rate training of the CMU STTK decoder. The new results are shown in 4.18. Fortunately this results are much better. In line with the previous BTEC

cacheWeight	word	bigram	trigram	BLEU
no	-	-	-	47.06
0.1	0.8	0.1	0.1	46.46
0.05	0.9	0.0	0.1	47.37

Table 4.18: Cache Arabic CMU MT03 with Optimization

results (4.16), they indicate that the cache model can improve over a baseline. But, as in the BTEC experiment, the improvement is not significant, and slight changes lead to a drop in performance.

4.3.4 MER Translation

The last step was to integrate the cache parameters itself into the *Minimum Error Rate Training* (MER). As mentioned in the implementation section 3.3.2, the interpolation between cache models and background ngram model is now performed in the translation engine, and can therefore be tuned. However, due to this visibility of the single models, there is no longer a guarantee that the probabilities will sum to one, as each model is tuned separately. Two experiments have been performed. The first experiment (A) is working with the original vocabulary of the ngram model, thus it is possible that the cache stores unknown word signs in its history. The second experiment(B) is also using the original ngram vocabulary, but if it sees an unknown word this is added to the vocabulary of the ngram model. The ngram model has been modified so that it returns the probability one divided by vocabulary size whenever its queried with one this words which have been added to the vocab. Table 4.19 gives the results.

System	baseline	previous highscore	cache +original Vocab	cache +extended Vocab
wordCache	-	0.08421	0.04736	0.02089
bigramCache	-	0.00000	0.00002	0.05367
trigramCache	-	0.00935	0.06055	0.00375
ngram	-	1.77770	1.84187	1.84751
BLEU	47.06	47.37	47.44	47.37

Table 4.19: Cache Arabic CMU MT03 with Optimization 2

As can be seen, there is only very small improvement over the previous highscore, where the optimization was performed without regarding the cache parameters itself. The previous highscore is from table 4.18, note that the weights appear to be different, as the previous table shows relative weights to other language models, whereas this table presents relative weights compared to all other models.

An unexpected result is that the experiment with the extended vocabulary performed worse than the one which kept unknown words in the cache history. This is probably due to an effect mentioned before. The decoder does not ask the language model for the probability of the unknown word, rather it assigns one divided by the vocabulary size *of the decoder vocab*. In experiment B however the decoder is now no longer aware that the new words are actually unknown words, hence the ngram model assigns one divided by the vocabulary size *of the language model vocab.* As the language model is a very important part of the translation process, it seems reasonable that a slight change to its probabilities can hurt the performance as shown here.

A more or less expected result however is that sometimes the bigram, and sometimes the trigram cache weight is nearly set to zero. This is in line with the literature review, which claimed that a trigram cache model gives no additional improvements over a bigram cache model. The results here prove this, as it seems to make no difference how much weight is put on bigram or trigram cache, as long as the weight for bigram + trigram is essentially similar.

4.3.5 Cache Language Model Test Results

All of the previous results have been development results, unless stated otherwise. This section analyzes real benefits on unseen test data. The unseen data is the MT05 Arabic translation task. The parameters for all models are the same as derived on the MT03 MER tuning.

The initial intention to use parameters from the results in table 4.19 did not work. This is due to the fact that the previous results have been obtained with a phrase table from IBM for MT03. For MT05 however, no such phrase table was available. Hence, similar experiments as in table 4.19 had to be made, this time however with a phrase table with 10 parameters generated with Phrase Extraction with Log-Linear Features (PELLF). More information about PELLF can be found in Zhao and Vogel (2005) and Zhao and Waibel (2005), essentially it is an extension to the GIZA++ toolkit (Och and Ney, 2003). Such a phrase table was available for both MT03 and MT05 (table 4.20).

	baseline	cache
wordCache	0	0.014
bigramCache	0	0.044
trigramCache	0	0.065
ngram	1.46588	1.296
MT03 BLEU	44.13	42.46
MT05 BLEU	41.18	-
drop %	-	-

Table 4.20: Cache MT05 Results

The table shows quite disappointing and incomplete results. This is because the new phrase table introduced issues. First of all, it increased the memory usage severely. Previous experiments were able to run in ten hours on six gigabyte machines, with the new phrase table swapping occurred both on six and eight gigabyte machines. The one result retrieved for the cache model took actually three days. The second issue was that the MER seems to have problems with correctly optimizing with so many parameters, or to say in other words depends on the use of well chosen initial parameters. This is indicated by the BLEU length penalty for the systems, which should be close to one and is 0.93 for the cache model. Due to the long runtime however, it was displeasing to run several different setups.

The other issue was the severe BLEU length penalty. To solve these problems, a smaller phrase table was build and a new new approach was taken to tackle the length penalty issue. First, the baseline system was trained with 15 optimization runs on the MT03 data. After that, the optimized parameters where used again on MT03, but this time the cache parameters where allowed to other values than zero. After these two runs have been performed, the two sets of parameters where used on the unseen MT05 test set. The results are given in table 4.21.

	baseline	cache
wordCache	0	0.081
bigramCache	0	0.032
trigramCache	0	0.084
ngram	1.041	0.716
MT03 BLEU	42.10	36.99
MT05 BLEU	-	-
drop %	-	-

Table 4.21: Cache MT05 Results Try 2

Obviously, this results looks bad. However, this is due to some malfunction of the optimization. It remained unclear if this malfunction was due to some models used here or the cache model code itself. Hence another cache model test set was evaluated in the next section. No problems occurred there, hinting that the problem here is likely not to be caused by the cache model code. As the next section also reported no improvements, the issues of this section are not traced further.

4.3.6 BTEC Japanese English Translation

As the experiments on the MT05 Darpa test set where hampered severely, additional experiments on the Japanese to English BTEC corpus where performed (see 3.2.2). Fortunately this experiments ran much faster, and without strange behavior. Indicating that the MT05 test set problems might really be due to problems with the phrase table. However, the results did not fulfill the expectations (table 4.22). A discussion of these results including translation examples can be found in the analysis section (5.1).

	baseline	cache
wordCache	0	0.039
bigramCache	0	0.008
trigramCache	0	0.164
ngram	1	1
DEV BLEU	55.85	55.78
IWSLT_05 BLEU	50.43	50.06
drop	10.7 %	11.4 %

Table 4.22: BTEC Japanese English Results

4.4 N-best List Cache Agreement Model

4.4.1 N-best List Rescoring

In order to get comparable results to previous sections, at first the MT03 TIDES n-best list from the Hiero2 system is rescored. The cache interpolation weight has been set to 0.3 and the cache history to 10 sentences, each sentence adds 1000 hypotheses to the cache with an equal weight of 1/1000. The equal weight is used as there have been no translation model scores available for this n-best list. In addition, as in previous experiments the unknown word is regarded as word by the cache model. The baseline listed in the leftmost column here is actually the center column from table 4.15. The new results are given in table 4.23. Regular cache model refers to the old approach where only the first best hypothesis was written in the cache, n-best cache is different as it adds all hypotheses from the n-best list in the cache. Finally, n-best agreement cache extends the n-best cache by adding all hypotheses of the *current* sentence and then rescores the current sentence. Another interesting number when regarding this table is 31.88, the score which is achieved when rescoring this n-best list just with the Xinhua language model without any cache model.

	regular cache	n-best cache	n-best agreement cache
bi0tri0	31.96	31.91	31.43
bi0tri1	32.02	31.90	31.47
bi0tri3	32.06	31.94	31.50
bi0tri5	32.00	32.02	31.57
bi3tri0	32.02	31.72	31.61
bi3tri1	?	31.73	31.61
bi3tri3	32.01	31.74	31.56
bi3tri5	32.02	31.75	31.59

Table 4.23: Cache Agreement compared to old Cache Model

The next step is then obviously to rescore an n-best list where translation model scores are available. The n-best list is from the current CMU system on the MT03 Arabic task, including translation model scores.

The baseline for the arabic n-best list is 43.67. At first some rank experiments are made. For example, always considering the 10th rank gives 42.57 whereas always considering the 50th rank gives only 41.80. It has to be mentioned that this n-best list is unique, which means that no 1000th rank score can be collected as some sentences have only less than 100 different hypotheses.

Initial experiments have been disappointing as the highest score retrieved was 41.45, which meant this new approach hurts severely. However, after reasoning about this unexpected behavior it became clear that the drop in the result might also be due to bad influence of the rescoring with the Xinhua language model. An experiment with the new n-best agreement cache weight set to zero, meaning to performs language model rescoring alone resulted in a BLEU score of 40.81. This actually means that there might be the chance that the new model gave an improvement.

The obvious solution to this, and to be admitted the better experiment which should have been performed in the first step, is just to use the full original score which caused the n-best ranking and modify this score directly with probabilities from the n-best cache agreement model. This modification was done with linear interpolation with the cache probability. The changed approach yields one further advantage, as it is no longer depending on the vocabulary of the language model used. Instead each word is added to the vocab, making an unknown word redundant.

First experiments were performed with the n-best cache agreement method. For these experiments from each of the last 200 sentences the 50 best hypotheses were added to the cache, and the cache interpolation parameters have been 0.6 for word and equally 0.2 for both the bigram and trigram component. Results are given in table 4.24. The results can be interpreted as that the agreement is unlikely to add any further information helping to select better hypotheses. This is indicated by the fact that the low cache weights have no effect as nearly always the best ranked hypothesis is chosen. This can be seen in column two of the table. Column three than shows that even for a high cache weight, still most of the time the first and second ranked hypothesis are picked.

cache Weight	# rank1	# rank1 or rank 2	BLEU
0	663	663	43.67
0.1	642	662	43.59
0.3	614	652	43.51
0.5	567	631	43.48

Table 4.24: N-best Cache Agreement on Arabic

The next experiment is evaluating the effect of pure n-best cache, meaning not to add the current hypotheses before rescoring them. Other parameters are changed as well to become more similar to the regular cache model. So the last four sentences have been added to the cache, each with its ten best hypotheses. The bi- and trigram interpolation weights remained at 0.2. The results shown in table 4.25 did again not show any improvements.

cache Weight	# rank1	# rank1 or rank 2	BLEU
0	663	663	43.67
0.1	644	662	43.61
0.3	614	654	43.63
0.5	555	617	43.42

Table 4.25: N-best Cache (no Agreement) on Arabic

4.5 Sentence Mixture Language Model

4.5.1 Training Clustering Duration

In order to apply sentence mixture models, there has to be a clustering of according documents before they can be applied. Essentially this is the functionality of the *OfflineCluster* program in 3.1. It seems reasonable that the number of documents and the number of iterations effect the duration severely. Therefore this first set of experiments has the goal to establish a feeling for time duration for different combinations of number of iterations and number of documents. The exact durations of a new training is given in table 4.26.

Documents	100	1,000	10,000	100,000
10 iterations	0:11	16:13	24:22:57	> 72 hours
100 iterations	0:11	24:15	24:33:10	-
1000 iterations	0:11	19:51	-	-

Table 4.26: Training Clustering Duration in Hours:Minutes:Seconds

Some results look strange, however, it has to be said that the machine used for this experiment was running other processes, too. This is certainly the reason why for example the 1000 iterations on 1000 documents were faster than 100 iterations. Additionally, the given number of iterations is only indicating the *maximum* number of iterations, if the clusters become too stable before this number is reached the clustering is stopped.

Overall the time duration is very high. Several of the experiments where aborted due their long duration. This thesis is not analyzing what exactly is the reason, as this, is in a way only a preprocessing step, and not of major interest in this work. A second reason is that the clustering itself is treated as black box system from the LEMUR toolkit, and the only way to trace the time issue would be to extract the implementation of the algorithms from the source code.

However, the time is an issue, so the next experiment performs the same training, but this time with centroid clustering rather than agglomerative clustering. The results are listed in table 4.27.

Documents	100	1,000	10,000	100,000
10 iterations	0:03	4:16	3:39:20	-
100 iterations	0:03	5:17	7:22:22	-
1000 iterations	0:03	5:17	-	-

Table 4.27: Faster Training Clustering Duration in Hours:Minutes:Seconds

4.5.2 Using Clustering Duration

This section analyzes how much time is needed to cluster large amounts of data with an already trained set of clusters. Basically this is the functionality of the *Cluster* program in 3.1. A first experiment tried to cluster a 233 million words, 1327 documents (1.3 gigabyte) of webblog data for language modeling. A first run was very disappointing. Trying to build a *keyfile* index on this data did not finish in three days. It has to be said that the machine was quite under heavy load during this time, but still three days is too long, especially as the LEMUR toolkit advertises to work with terabytes of data.

Yet, a *keyfile* index might be too much overhead, so instead it was tried to build an *inverted* index, which is according to the LEMUR documentation a format with less overhead. From the experience of the previous subsection, the clustering method was also set to centroid mode. Again the processing took very long and was aborted.

When tracing this unexpected behavior, it finally became clear that a bug in a parameter file caused the long processing time. The parameter file was looking for a parameter memory with the value *4000m*, however the used parameter files contained the value *4000mb*, which caused the toolkit to build incredible amounts of swap files. So, this was actually not a fault of the LEMUR toolkit, still, it would be nice to get a warning about such a wrong parameter setting.

Surprisingly, at the end it became clear, that though it is the index with the most overhead, the *indri* index was built much faster than all other indices. It worked very fast, about 10-15 minutes for an indexing of 450 megabytes, and roughly 40 minutes for a 1.3 gigabyte index. This may seem counter-intuitive, however, the author believes that the *indri* index is the heart of the lemur toolkit, and consequently much better tuned and maintained. There is still a disadvantage of the *indri* index, as it needs more filespace than all other indices.

4.5.3 Perplexity

Weblog

For this experiment the first 1000 docs of the 04 dataset were taken for training the clustering with 100 iterations of bisecting k-means. This data was then used to cluster the language model training data which consisted out of the 05, 06 and 07 WebBlog data sets, more details about the data in section 3.2.6. Altogether this training data consisted out of 233 million words and 1327k documents.

After the training was done, seven different SRI 3-gram language models were build. One with the full 233 million words, the other six only with sentences which were clustered to the special cluster.

Then the perplexity for each of the different language models was measured on the first 1000 docs of the 08 datasets.

The author wants to note that here the perplexity results of different language models are not comparable, as they can have different vocabulary sizes. However, each language model is tested on each test set, thus the performance of one language model on different test sets can be compared. Ideally, each language model would produce its best performance on the test set which belongs to the same cluster the language model was trained on. The following results therefore can not really be treated as ordinary perplexity improvement, nevertheless, if the prediction holds true, this indicates that the clustering is performing well.

The first table shows a list how the 1327k documents were distributed among the clusters (4.28). The threshold for cluster similarity was actually set to 0.001, rather than to zero, which allowed the possibility not to assign a cluster for some few blogs.

Cluster	#documents train 04	#documents LM	#documents 08
1	83	34k, 3.0 million words	49
2	206	268k, 34.9 million words	161
3	162	760k, 160.5 million words	249
4	99	45k, 2.6 million words	77
5	378	149k, 13.5 million words	428
6	72	54k, 5.0 million words	36

Table 4.28: Distribution of Documents to Clusters I

The actual results are given in 4.29. Keep in mind that **different language models cannot really be compared** here, as they have different vocabularies. But if the numbers in one row are compared, it is clear that each language model performs best on its corresponding test set. The last two lines of the table give the performance of the general language model with all data and the best relative improvement to this by the clustered language models. Altogether the 1000 documents of the perplexity test set contain 95k words.

	08.cluster1	08.cluster2	08.cluster3	08.cluster4	08.cluster5	08.cluster6
LM 1	**116**	2040	1114	1133	920	2075
LM 2	2717	**445**	968	1291	1133	1664
LM 3	1742	2102	**425**	1955	1570	2180
LM 4	1562	2079	1345	**92**	146	1900
LM 5	1268	1690	1039	178	**59**	1633
LM 6	1584	2273	1067	1385	1132	**813**
All	175	565	499	255	166	894
gain	33.7%	21.2%	14.82%	63.92%	64.45%	9.06%

Table 4.29: WebBlog Perplexity Results on 08 dataset

Xinhua LDC Data

This section basically performs the same steps as the previous one on the Xinhua data. The training docs are the first 2000 documents from the bilingual Xinhua news story part. The 1000 documents test set are the last 1000 documents from the 20001 Xinhua LDC data. Information about the distribution of documents is given in 4.30.

Cluster	training docs(EN+CH)	180m Corpus EN	1000 test docs
1	290	81 k	98
2	428	305 k	359
3	407	139 k	161
4	190	113 k	127
5	446	169 k	212
6	239	57 k	41

Table 4.30: Distribution of Documents to Clusters II

The results (table 4.31) still show perplexity improvement by using the corresponding language model compared to using the full language model. The gains are not that big as in the previous webBlog perplexity (table 4.29). The reason for this is likely to be that the Xinhua data is already much more uniformly distributed, whereas the webBlog data covers completely different topics and styles.

	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6
LM 1	**39**	512	690	476	486	501
LM 2	270	**88**	426	248	199	236
LM 3	634	466	**82**	330	323	352
LM 4	744	327	325	**100**	216	271
LM 5	1113	368	467	311	**74**	220
LM 6	1394	449	538	381	232	**86**
All	59	96	105	111	83	107
gain	33.89%	8.33%	21.90%	9.90%	8.43%	19.62%

Table 4.31: Xinhua Perplexity Results on 1000 documents

4.5.4 Translation with Monolingual Training Data

As there is no bilingual training data, a full translation with one language model had to be done first. There was no special translation system available for webBlogs, hence the CMU GALE system with a language model with 233 million words on the 05 till 07 data sets (see 3.2.6) was used. The first pass translation has a score of 7.73 BLEU.

Table 4.32 shows the distribution of the documents to clusters when working with the same cluster seed as for the perplexity experiments 4.5.3. As can be seen, the distribution of the actual data is quite in favor of cluster three, which is likely to be a mismatch between the seed for the clustering and the actual data.

Cluster	train 04	220m Corpus EN	250 checksum2 CH
1	83	34k	38
2	206	268k	3
3	162	760k	151
4	99	45k	17
5	378	149k	32
6	72	54k	9

Table 4.32: Distribution of Documents to Clusters III

As the language models with the clustering according to the 04 seed have already been built for perplexity measurements, the translation experiment is based on the clustering given in 4.32. The results are shown in table 4.33.

The baseline of these experiments is a translation of all 250 sentences with the full language model (*All baseline*). Using only the clustered language models to translate clusters of the development set resulted in a 5.4% improvement. This improvement could even be topped when translating the six clusters separately with the full language model (improvement 11.2%, *All clustered*). The obvious approach to perform an interpolation between the specific language model and the full language model was only tried out at the Xinhua news

	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6	250 checksum2 WL
LM 1	9.86	-	-	-	-	-	-
LM 2	-	10.03	-	-	-	-	-
LM 3	-	-	7.47	-	-	-	-
LM 4	-	-	-	9.70	-	-	-
LM 5	-	-	-	-	8.29	-	-
LM 6	-	-	-	-	-	13.73	-
All baseline	-	-	-	-	-	-	7.73
LM1-6	9.86	10.03	7.47	9.70	8.29	13.73	8.15
All clustered	12.87	13.94	7.62	11.81	9.53	12.88	8.86

Table 4.33: WebBlog 250 sentences Clustering Translation BLEU scores

clustering.

Though this numbers look very good at first sight, it has to be said that most of the improvement is likely to be caused by high overfitting. How strong this overfitting impact is, and if there is any real improvement at all, is shown in section 4.5.6.

4.5.5 Translation with Bilingual Training Data

For this experiments the Xinhua LDC data, see 3.2.5, was used, both its bilingual and its monolingual components. From the available 20k bilingual documents, 2000 documents were used to train six clusters on the English side. The retrieved clusters where used as seed to cluster the monolingual English 845k documents of the Xinhua LDC data. Note that it was sufficient to train the clusters in English, as there was a one to one alignment from English to Chinese documents, hence the retrieved clusters can be used for Chinese documents as well.

For each of the six collected clusters on the English side a SRILM language model was build. The actual distribution of documents and words on the English side is given in table 4.34. Please note that the clustering of the full 180 million word Chinese corpus is not used in the further experiments, yet it is interesting to see how different the clusters grow in the different languages.

Some engineering was needed to transfer the clustering from one language to another. First of all, a small program was written which takes a TREC file and document to cluster mapping table and generates a cluster index according to that mapping. This step replaces the offline clustering, as described in 3.3.5. This can be done, as the document to cluster mapping was already generated on the other language. After this is done, the other clustering steps can be performed as usual.

However, the translation itself is still a little tricky. The source text has also to be

Cluster	training docs(EN+CH)	180m Corpus EN	180m Corpus CH	191 checksum1 CH
1	290	81 k	10 k	7
2	428	305 k	103 k	31
3	407	139 k	290 k	93
4	190	113 k	114 k	31
5	446	169 k	130 k	20
6	239	57 k	77 k	9

Table 4.34: Distribution of Documents to Clusters IV

clustered, and each of these clusters is translated separately. As the translation is using MER training with four references, this reference has also to be split up in according clusters. Each translation was run with ten iterations of MER training. The data translated are 191 sentences from Xinhua news stories which were also used in the GALE project as checksum1 data.

Table 4.35 shows the translation results. Cluster one through six refers to the clustering of the 191 test sentences. *All baseline* in the first column refers to a language model build on the full 180 million words, whereas *LM1-6* refers to the combined output of the six translations with the six partial language models. Another set of experiments is translating with both the specific cluster language model and the full language model (*LM1-6 & All*). The last experiment is translating each cluster with the full language model (*All clustered*).

	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6	191 checksum
LM 1	24.98	-	-	-	-	-	-
LM 2	-	29.80	-	-	-	-	-
LM 3	-	-	28.04	-	-	-	-
LM 4	-	-	-	31.79	-	-	-
LM 5	-	-	-	-	25.83	-	-
LM 6	-	-	-	-	-	23.62	-
All baseline	-	-	-	-	-	-	27.64
LM1-6	24.98	29.80	28.04	31.79	25.83	23.62	28.13
LM1-6 & All	26.89	30.42	29.30	32.62	25.60	28.50	29.31
All clustered	30.07	30.04	29.96	32.22	26.86	26.70	29.66

Table 4.35: Xinhua Bilingual Clustering Translation BLEU scores

The *LM1-6* experiment show an improvement of 1.8% over the baseline. This is topped by both the interpolation (6.0%) and the full language model alone (7.3%) when translating each cluster separately. As mentioned already in the previous section, most of the improvements are very likely due to overfitting.

A quick experiment was performed to give a feeling of how much overfitting is included here. This quick check is splitting the test set into six clusters of the same size as in table

4.34. However, each cluster just contains a consecutive chunk of the test set. This means cluster one contains the first seven sentences of the test set, cluster two the next 31 sentences and so on. The results of this experiment gives a score of 29.24, which has to be compared to the 29.66. The 29.66 is still a 1,4% improvement which might be due to splitting the test set in a good way. The next section is analyzing if this improvement can be transfered to an unseen test set.

4.5.6 Sentence Mixture Model Test Results

Due to the lack of reference translations for webBlogs, the tests were performed only on the Chinese English news translation task. For these experiments the optimized parameter values from section 4.5.5 were used. The test set was the TIDES Chinese MT05 data set with 1082 sentences.

As the development results showed the best improvement when using the full language model on the clustered test set, only this experiment is conducted here. The baseline is a translation with the same language model, however, with one set of parameters for all sentences instead of different parameters for each cluster. Results are shown in table 4.36. The distribution of the test set to clusters is given in table 4.37.

	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6	Chinese MT05
All baseline	-	-	-	-	-	-	11.55
All clustered	-	-	-	-	-	-	11.83

Table 4.36: Sentence Mixture Model Test Results

Cluster	training docs(EN+CH)	180m Corpus EN	191 checksum1 CH	1082 mt05
1	290	81 k	7	53
2	428	305 k	31	152
3	407	139 k	93	516
4	190	113 k	31	148
5	446	169 k	20	116
6	239	57 k	9	97

Table 4.37: Distribution of Documents to Clusters IV

The results is an improvement of 2.4%, but the scores are all very low. The results should be at least above 20 BLEU points. After analyzing it became clear that similar to other earlier problems a segmentation difference of the phrase table and the test data was the issue.

As a consequence a new phrase table had to be build, and the development translations had to be rerun with this new phrase table. The scores of these translation runs can be seen in 4.38.

	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6	191 checksum
All baseline	-	-	-	-	-	-	26.93
All clustered	25.82	28.90	25.83	30.94	25.18	21.60	26.99

Table 4.38: Bilingual Clustering BLEU scores, new Phrasetable 191 checksum

The interesting results are given in table 4.39. These are results on the unseen MT05 data with the new phrasetable.

	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6	191 checksum
All baseline	-	-	-	-	-	-	22.20
All clustered	-	-	-	-	-	-	21.25

Table 4.39: Bilingual Clustering BLEU scores, new Phrasetable 1082 MT05

4.5.7 Combining Sentence Mixture Model and more Data

A key issue in language modeling is data. In general all natural language processing tasks improve when more appropriate training data is used for the language model. However, there is also a practical issue. As soon as there is too much data, many reseach institutes are no longer able to process the large amount of data due to limited computing resources. Most institutes will only be able to build language models on about 400 million words, but there are already corpora with billions of words available.

The sentence mixture model approach as used here is designed to process up to 100 billion words and more (Callan and Croft, 2006). The result of the appliance of the sentence mixture model will be several smaller data sets that fit to certain parts of the test data. As a consequence, a several billion word corpus could be used and split up into feasible amounts of data. Making use of this larger data should give additional benefits.

Still the processing of such large amounts of data is time consuming. Hence, as a first test, the 4.5.5 experiment is repeated, but this time only with 120 million of word for the complete language model. The clusters however remain the same, derived from the full 180 million word corpus. The data size is different, but the effect should be similar, when the usefulness of 120 million words to 180 million words or 300 million words to 600 million words data is compared, for example.

As a reminder, the previous result on development data was 29.66 when using the clustered test set. Results when translating the full 191 checksum1 data with smaller language models are given in table 4.40.

	cluster1	cluster2	cluster3	cluster4	cluster5	cluster6	191 checksum
77m baseline	-	-	-	-	-	-	27.51
108m baseline	-	-	-	-	-	-	27.57
180m baseline	-	-	-	-	-	-	27.64
180m clustered	30.07	30.04	29.96	32.22	26.86	26.70	29.66

Table 4.40: Sentence Mixture Model & more Data

4.6 Language Model Adaptation

4.6.1 Perplexity

It has to be noted that the language model adaptation here is specifically designed to *adapt* to the current data of interest. Hence it is only a logical consequence to train the adaptation and measure its perplexity on the same piece of data. For both the webBlog and the Xinhua data the perplexity is measured on the same test sets as in the sentence mixture model perplexity section (4.5.3).

WebBlog

As mentioned the perplexity on webBlogs is measured on the same test set as before in section 4.5.3. Different thresholds for the clustering algorithm were set to retrieve different amounts of data. The results are shown in table 4.41. The adaptation of the language model data does look acceptable. For example the 71 million word language model has only a 3% OOV rate, whereas the more than three times bigger full language model has already a 2% OOV rate. However, the small improvements in perplexity do not make improvements in real translations likely.

Threshold	0	0.03	0.05	0.07	0.1
#words	220m	174m	88m	71m	21m
Perplexity	237.4	235.0	231.6	238.5	263.2
# OOV	1903	2089	2399	2847	4070

Table 4.41: WebBlog Perplexity for Different Thresholds

Chapter 5

Analysis

5.1 Class Language Model

To analyze the clusters generated by mkcls (see 3.3.4), the vocabulary coverage of the clusters was evaluated on typical test set references and hypotheses. The Arabic MT03 references contain an average of 17,503 words, and the MT05 references an average of 32,028 words. The results from these experiments showed, that there must be a mistake somewhere, as the Out Of Vocabulary (OOV) rate of over 10% was much too high.

The most likely reason for this is the presence of numbers: these were not tagged in the same way as in the training data. However, it is appropriate to make number tagging before using the class language model, as this reduces the vocabulary greatly even before the class language model is applied. In addition, the language model can make no predictions which will be the right number in most cases.

To further trace this effect, the numbers in the references and hypotheses were counted and subtracted from the OOV counts collected before. The new out of vocabulary numbers were almost 10%, still too high. Finally it turned out that the wrong training corpus was linked, with upper case instead of lower case.

A new preprocessing with similar number tagging and everything lowercased produced OOV rates of 391 OOV tokens on the MT05 hypothesis, among them 236 different word types. On MT03 there are 193 OOV tokens, among them 106 different word types. Most of the OOV words are proper names, actually both hypotheses have 47 times the word Xinhua, which was not in the training data. Basically this means that for these last experiments the OOV rate is about 1% for both hypotheses, which is a number in the expected range.

Actually, there were before further experiments with a correct training corpus for an evaluation. These experiments did not give better results, and were an additional reason why no further work was spent on class language models. To show here that it does not give better results when using a training corpus with correct preprocessing, some of the development results (4.1.2) are repeated. The new results are given in table 5.1. It seemed more reasonable to have different classes for sentences marks, consequently only these experiments were rerun. The new results are named *run2FIX*, the previous results *run2ORG*.

#Classes	no	10	50	100	200
run2ORG BLEU	45.70	44.96	45.44	45.78	45.45
run2ORG Scale	0.0	0.0	0.106	0.062	0.167
run2FIX BLEU	45.70	45.31	45.23	44.21	44.74
run2FIX Scale	0.0	0.034	0.017	0.250	0.0

Table 5.1: Class Language Model Translations Fixed

5.2 Cache Language Model

A variety of experiments has been preformed to evaluate the impact of cache language models on machine translation. In line with previous literature, perplexity results showed nice improvements of up to 17,6%. Yet, comparing all the experiments it can not be said that the cache model provided a constant improvement. Some experiments showed a little bit of improvement, some a slight drop of performance. Due to the number of experiments, it is not possible to analyze all in detail here. One annoying issue was the length penalty issues that prevented the experiments on the MT05 test. Hence some experiments become analyzed regarding their effect on the overall length.

The cache development results on the MT03 were presented in table 4.19, the best result was a 0.8% gain. From the 663 only 169 where identical between the baseline and the translation with cache. The baseline translation contained 19,843 words, whereas the cache translation contained only 19,531. This is a drop of 1,6% in the number of words. However, the first reference contained only 16,756 words. All four references contain 70,012 words, which makes an average of 17,503.

To analyze this length issue further consider the following tables which give information about the number of words in translation hypotheses and references. The first table summarizes what was just said about MT03 (5.2), the second table collects the same information for the Japanese English BTEC task (5.3).

When looking at the MT03 table, it seems possible that the cache model favors short translations. However, on the Japanese English BTEC corpus the effect is exactly the opposite, and the cache model is slightly increasing the length of the translations. In fact, when ignoring the length results with the PELLF phrasetable, the cache model is always moving

System	averaged references	first reference	baseline	cache
IBM Phrasetable	17,503	16,756	19,843	19,531
PELLF Phrasetable			18,050	16,289

Table 5.2: Cache Influence on Translation Length - MT03

System	averaged references	first reference	baseline	cache
Dev Set	3584	3604	3301	3401
Test Set	3630	3617	3408	3458

Table 5.3: Cache Influence on Translation Length - Japanese BTEC

the length of the translation closer to the averaged reference length. So it is likely that the length issue which has occurred in the MT03 experiment is not a problem of the cache model.

Some example translations are given in table 5.4 and table 5.5. The examples are from consecutive sentences, as showing anything else would make it completely impossible to tell anything about cache influences. Obviously it is rather difficult to tell which differences might be due to the cache language model.

5.3 Sentence Mixture Language Model

This section tries to analyze why some of the sentence mixture model experiments did not work. Before actual experiments are discussed, one comment about why six clusters have been used for most experiments.

There are actually (theoretical) straightforward approaches to automatically determine the number of needed clusters. First of all one could think about using the TFIDF distance between clusters, starting out with a high number of clusters and then merging them until a threshold is reached. The other approach would be to calculate clusters for different number of clusters and then calculate perplexities on a test set for different number of clusters. This second approach however would create a high computational demand.

The reason why none of the two approaches was tried is the indication from the literature review (2.6) that the more sentence mixture models there are the better. In the reviewed papers there were usually four or eight different models used. Hence there was the justified hope the six models already should show an improvement. The initial idea was also that once the six clusters have been used successfully, to perform experiments with more clusters to see if the improvement can be increased further. But if six models show no improvement, then there is little hope that more models will create better scores.

After this general comment, it is at first reasoned about if the clustering is working at all as desired. The perplexity experiments were conducted for that reason. Especially the

baseline	cache	reference
iraqi opposition meeting is held in mid january in kurdistan	the iraqi opposition meeting was held in mid january in kurdistan	Iraqi Opposition Prepares for mid-January Meeting in Kurdistan
ahmed said al-bayati is a member of the committee told agence france presse that the meeting which was held yesterday in london on the conference which was organized by seven movements of iraqi opposition in london between 14 and 17 of last december which was agreed on the principle of this meeting be held in iraqi kurdistan .	ahmed said is a member of the committee told agence france presse that the meeting which was held yesterday in london on the conference which was organized by movements of iraqi opposition in london between 14 and 17 of last december which was agreed on the principle of this meeting be held in iraqi kurdistan .	Ahmed al-Bayati, a member of the Committee, told France Presse that last night's meeting in London was connected with the conference organized by 7 Iraqi opposition movements in London last December 14-17 in the course of which it was agreed that this meeting should take place in Iraqi Kurdistan.
he added that the agenda of the meeting last friday was " research committee meeting to be held in mid january in iraq 's kurdistan " specifically with the organization and logistics .	he added that the agenda of the meeting on friday night was " research committee meeting to be held in mid january in iraq 's kurdistan " in particular with the organization and logistics .	He added that the last night's agenda was to "discuss the Committee's meeting to be held mid-January in Iraqi Kurdistan" defining organizational and logistical preparations.

Table 5.4: Cache MT03 Translation Examples

baseline	cache	reference
will the flight depart on time on this train	this train will the flight leave on time	will this train leave on time
what are you commute by	what are you commute by	how do you get to work
i am seat a 510	i am seat a 510	my seat number is a fifteen
italian restaurant please	could you recommend a nice italian please	please recommend a good italian restaurant
medium coke please	i would like a medium coke please	a medium coke please
good afternoon	good afternoon	hello

Table 5.5: Japanese BTEC Translation Examples

webBlog perplexity training set contained highly different texts. The keywords for each cluster are shown in table 5.6. It might be hard to assign topics to each cluster, but the

Cluster 1	s com tips anti aging skin information american formulas antiaging care time like t comments home new people great news 2005 michigan beauty best health info make resources just life
Cluster 2	com s free toolong 2005 new online buy t software posted 0 news download home time price like information best satellite just july resources site internet business games size make
Cluster 3	s 2005 t july news new home day ago time like hours think people m 4th just blog today don read night ve jun world got going make david com
Cluster 4	info time information related subject resources matter site web material research internet available read net best business searching desire like news page advice convenient quite disc new ll reliable strong
Cluster 5	information web time resources site matter news subject research material looking t great internet advice help related topic available 2005 health generated website sites quite library best content strong deal
Cluster 6	s new pet 2005 food like just right t com toolong time service natural click good hours web ca make site ago ve healthy toys wireless cellular help prepaid supplies

Table 5.6: WebBlog 04 Perplexity Cluster Keywords

perplexity results certainly indicate that the clustering was working right.

After the analysis of the clustering, the actual translation experiments are analyzed. The following table 5.7 shows the found clusters for the webBlog translation.

After reading the test set, it becomes clear that the 250 webBlog translation test set is split into two webBlogs by one female author. The first blog is talking about her time and experience with school, being a child from an oil field. The second blog is talking about quarrels in her own family when a new car was bought. The daughter is an environmentalist and does want no car at all, or at least no SUV, whereas the father is in favor of a SUV.

When looking with this knowledge again at the clusters, cluster one and four can clearly be assigned to the blog about school time and oil field childhood. On the other hand, cluster two and three are referring to the argument about the new car. Interestingly cluster five and six cannot be assigned easily to one of the two topics. But when cluster five becomes investigated further, it is obvious that it is having keywords which occur frequently in both blogs. Examples for this are daughter, time, home, school, family. Finally Cluster six seems to have more generic words like soon, fact,new, think, going as most frequent words. Of course, these words also occur in both contexts, and it seems that cluster six is more or less capturing anything else which did not fit in one of the other clusters before.

The other translation experiment was performed on 191 sentences from news stories.

Cluster 1	people time thinking way asked like great p eyes early mother feelings students tears prince friends el teacher buss m think going want right life long came information school days
Cluster 2	heard door car years played mr support fight quickly lines response husband barely balance garage slash amputated diagonal
Cluster 3	parents sense energy environmental like people new home did took understand power future uncle responsibility protection told vehicle prince emissions make need say great let came school car thought news
Cluster 4	looking heart school car set warm heavy small cotton like went new home way work life come years hard high point moment word inside understand pain special states return basically
Cluster 5	daughter car time home husband school years like year family students high day university new say come end want marriage life alma know really good today days able feeling buy
Cluster 6	soon fact new think going work girls come look hours hard phone high trying general schools learning block teaching solid teachers basis environment efforts studies junior atmosphere purity signature adapt

Table 5.7: WebBlog Monolingual Cluster Keywords

Though the news are from about 2003, they still read very similar to today news. There are reports about American scientists inspecting North Korea, American soldiers in Iraq, presidents and revolutions in small countries, reform politics in Japan, mars exploration with Beagle 2 and car of the year awards. The only story which would be unlikely to be found in a todays newspaper might be a report about china blocking meat imports due to the mad cow disease.

Due to the bilingual approach tried for the news test set the test set itself can not be used as seed for the clustering. The disappointing results might hence be indicated in a mismatch between the test set and the seed. Indeed, the seed only contains some similar topics like North Korean inspections, space travel and US military related articles. The vast majority however is much more concerned about China, especially there are quite some reports which are of more local interest, like Chinese education system reforms or Shanghai worker arrangements.

However just using more similar training data is not as easy as it might seem. There are two key constraints in addition to matching the test set as good as possible. The first one is that the initial training relies on document boundaries. The second and much more restraining is that the training corpus must at least contain a considerable amount of documents which are a translation of each other.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis was started with the belief that it should not be too complicated to gain benefits by using better language modeling techniques in SMT. Most SMT systems still use a plain trigram model, and there was a lot of work done in language modeling for speech recognition that has not yet been applied to SMT. It seemed rather straightforward to analyze the existing literature for speech recognition and pick the most promising techniques to apply them to SMT. This selection was done with the general aim to focus on language modeling techniques that allow the integration of long term context dependencies in the language model probability generation.

However, already when analyzing the literature it became clear that this approach might not be as easy as it had seemed initially. The work that had most influence on this thesis stated

> "We point out that trigrams remain the de facto standard not because we don't know how to beat them, but because no improvements justify the cost" (Goodman, 2001)

Still the approach of this thesis was valuable, as it was interesting to analyze the effects of some of the better language modeling techniques in SMT. The approaches chosen were the class based language model, the cache language model and the sentence mixture model. Yet none of the three approaches could really improve a current state of the art system.

Regarding the class based model, no experiments ever showed any improvement. This might be due to the simple concept of the class language model applied here, which was regarding the class based language model probability as a separate feature in the log likelihood feature space. However, the clustering concept itself was sound, thanks to Och (1995) and none of the experiments showed any hints that this might help. Thus no effort was spent to evaluate more sophisticated techniques to integrate class based language model probabilities in the overall translation process.

The cache language model showed some minor gains on some data, but some minor drops on other data. Consequently, it seems reasonable to assume that the cache language model depends on the current test set. However, it is an open question to determine whether a cache language model can help for the current test set. To make more analysis of different test sets and results is actually one of the suggestions for future work that is described more elaborately in section 6.2.1.

The sentence mixture language models approach followed in this thesis was in many ways different than the previous two approaches. First of all, it was not just the application of a concept that is already used for speech recognition. In addition to the existing idea of sentence mixture models two refinements specifically suitable for the task of SMT were made. First, the clustering for some test sets was done on the source side, giving the opportunity to use completely correct information for the clustering. Second, the clustering was not only used to change one model with clustering information, but also to cluster the test set to translate. Thus there is a possibility for all models to be adjusted to a certain style and topic of the test set. Enhanced with these new ideas, sentence mixture models seemed very promising for SMT, and first experiments showed motivating results. Large improvements in perplexity were reported on different test sets, and translation experiments on development data also showed significant gains. However, in the end it became clear that these good results were due to overfitting.

Overall this thesis can conclude with a similar conclusion as Goodman, cited above. As for speech recognition, plain n-gram models remain the standard for SMT. This thesis has shown further proof that though intuitive assumptions might suggest it should be rather easy to build better language models than n-grams, this is not the case. Several approaches were tried in this thesis, none of them could produce an improvement on an unseen test set compared against a trigram trained with enough data. Hence it can be stated, that it is at least very hard to get improvements by simple word dependency approaches. This means approaches where no further linguistic knowledge is applied. It remains an open research question, if language models with more linguistic knowledge like structured language models (Chelba, 1997)(Chelba and Jelinek, 1998) can help.

6.2 Future Work

6.2.1 Analysis Toolkit

Several strategies to analyze the effect of the suggested approaches were applied in this thesis. Translation lengths, out of vocabulary rates and manual comparisons of translations were used to do this. However, it is still hard to trace influences of models in the iterative MER Training with the huge amount of numbers that have to be compared to generate the translation of only one sentence.

This is a common problem for all machine translation issues, hence it would be nice to have a toolkit which would not only facilitate the process of analysis as it is currently, but also offer more sophisticated techniques to do this. For example a visualization of model influences on sentences and/or words could be generated. This would have the advantage that humans are in general more capable to retrieve information from color and pictures then from a large amount of numbers. This skill of humans is also summarized in the proverb "A picture says more than 1000 words".

Furthermore metrics for analyzing multiple translation systems and maybe multiple references can be integrated. Thus a human judge would only need to look at sentences that have for example significant different translations by two systems. The combination with the previously visualization component would allow a far more detailed analysis of SMT system output as it is currently possible.

6.2.2 Development Set Adaptation

The sentence mixture model approach followed in this thesis was slightly different than the existing idea about sentence mixture models (Iyer and Ostendorf, 1999). The additions, which are geared towards the application in machine translation are first the clustering on the source side and second the clustering of the test set itself to train different model parameters for each of these clusters. This second idea could be further amplified by actually adapting the development set.

An adaptation of the development set can be realized as follows. For each sentence in the test set, a number of similar sentences from an existing bilingual training corpus is extracted. With the sentences retrieved a development translation is executed with the aim to get optimized parameters for this set of sentences.

This adaptation of the development set can maybe give better parameters for this specific sentence than the usual averaging of the best parameters over all sentences.

6.2.3 Back Translation Future Prediction

When looking again at the initial motivation for this thesis 1.4 and the proposed improvements 1.5, 1.6 and 1.7 it becomes clear, that their is yet no language modeling technique to integrate words from the right side of the word under investigation in the probability estimation.

Of course, in the general setting of language modeling this is not really possible, as knowing the right side of the word means looking in the future. In the special case of machine translation however, the future is known, but only on the source side. The following listing shows an example of a German sentence and three English translation hypotheses.

- **Source:** Heute halte ich eine Presentation über maschinelle Übersetzung.
- **Hypothesis 1:** Today I give a presentation about machine translation.
- **Hypothesis 2:** Today I clamp a presentation ...
- **Hypothesis 3:** Today I last for a presentation ...

The correct translation of the source sentence is given by hypothesis one. Yet, when translating the German word "halte", which should be translated to "give", the SMT system does not know if "give", "clamp" or "last for" is the correct translation of "halte", as this can not be decided from the context "Today I". Yet the future context would make clear that "give" is the correct translation.

As a solution to this problem, the system could generate word hypotheses for each of the English words "give", "clamp" or "last for". Possible translations of give would all fit more or less in the context "Heute ... ich eine Präsentation über maschinelle Übersetzung". However, many possible word translations for "clamp" or "last for" would not fit in this context. Therefore the back translation could help in this case to make the correct decision.

Appendix A

Summary of the Thesis in German

Analyse und Anwendung von Techniken für die Integration weitreichender kontextabhängigkeiten in der maschinellen Übersetzung

A.1 Abstrakt

Die maschinelle Übersetzung zwischen zwei Sprachen ist eine komplizierte Thematik mit der sich bereits viele Leute beschäftigt haben. Bis heute gibt es keine vollständige Lösung für dieses Problem, aber seit Brown et al. (1993) gilt die statistische Übersetzung als vielversprechender Kandidat und ist seitdem von primärem Forschungsinteresse. Viel Arbeit wurde im Bereich des Übersetzungsmodells und dessen Teilbereichen geleistet. Weniger Arbeit wurde dahingegen im Bereich der Sprachmodelle für die maschinelle Übersetzung getan, die wesentlich sind um flüssige Sprache in der Zielsprache zu gewährleisten.

Als Folge dessen ist die zentrale Idee dieser Diplomarbeit die Integration und Evaluation von Sprachmodellierungstechniken für weitreichende Kontextabhängigkeiten. Hervorragende Literatur zu diesem Thema findet man im Bereich der Spracherkennung. Der Grund dafür ist, dass Sprachmodelle, die eine ähnliche Rolle in der Spracherkennung spielen, für diese bereits gründlich analysiert wurden. Bis heute wurden jedoch wenige der zahlreichen Sprachmodellierungstechniken bezüglich ihrer Nützlichkeit für die maschinelle Übersetzung untersucht. Nach einer Analyse der vorhandenen Literatur erschienen folgende Modelle vielversprechend: Klassenbasierte Modelle, Cache Modelle und Satz Mixtur Modelle. Jede dieser Techniken wurde daraufhin weiter untersucht, und zumindest bezüglich ihrer Perplexität und den Übersetzungs Ergebnissen evaluiert.

Der Rest der Diplomarbeit hat folgenden Aufbau. Zuerst wird eine Einleitung zum Thema der maschinellen Übersetzung gegeben. Danach werden verschiedene Sprachmodellierungstechniken im Literatur Überblick diskutiert. In den darauffolgenden drei Kapiteln werden

die durchgeführten Experimente beschrieben. Nach einigen Details zu Implementierung und Daten werden die eigentlichen Ergebnisse vorgestellt, welche im letzten dieser drei Kapitel analysiert werden. Am Ende wird eine Zusammenfassung und Vorschläge für zukünftige Untersuchungen gegeben.

A.2 Einleitung

Diese Einleitung wird den Kontext sowie die Motivation für diese Diplomarbeit beschreiben. Dabei werden zuerst einige Grundkonzepte der statistischen maschinellen Übersetzung (SMT, für Statistical Machine Translation) erläutert. Danach werden die Ansätze dieser Diplomarbeit zusammen mit ihrer Motivation eingeführt. Zum Abschluss wird kurz die Struktur der restlichen Diplomarbeit beschrieben.

SMT

SMT ist gegenwärtig der meist verfolgte Ansatz um maschinelle Übersetzungen zu erzeugen. Eine wichtige Eigenschaft von SMT ist, dass Wahrscheinlichkeiten benutzt werden und dementsprechend nur schwache Entscheidungen (soft decisions) gemacht werden, die später revidiert werden können. Diese Eigenschaft erlaubt die Kombination von vielen Komponenten bzw. Modellen, die zusammen eine Entscheidung treffen welche Übersetzung die beste ist. Eine Übersicht über diese Komponenten der SMT wird in A.1 dargestellt. Es kann durchaus noch weitere Modelle geben, allerdings sollten die Modelle der Abbildung in den meisten SMT Systemen vorhanden sein. Das Zusammenspiel der verschieden Modelle wird in den nächsten Sätzen beschrieben.

Der Decoder, der sich in der unteren Hälfte der Abbildung befindet ist die zentrale Schnittstelle einer maschinellen Übersetzung. Er erzeugt die eigentliche Übersetzung und kombiniert die Ausgaben der verschiedenen Teilmodelle. Das wichtigste dieser Teilmodelle ist das Übersetzungsmodell, welches seine Entscheidungen anhand verschiedener Wahrscheinlichkeitsverteilungen bestimmt. Als Antwort auf die Anfragen des Decoders mit Wörtern der Quellsprache wird eine Menge von Übersetzungsmöglichkeiten zurückgeliefert. Der Decoder kombiniert diese Übersetzungsmöglichkeiten (=Hypothesen) für verschiedene Wörter und erzeugt daraus einen Graphen der alle möglichen Übersetzungen eines Satzes enthält. In Englisch bezeichnet man einen solchen Graphen als Translation Lattice. Eine solche Translation Lattice ist in A.2 dargestellt. Jede der Hypothesen in einer Translation Lattice hat eine Übersetzungswahrscheinlichkeit. Allerdings enthält sie typischerweise auch noch Hypothesen die nicht der Grammatik der Zielsprache entsprechen. Solche Hypothesen versucht man

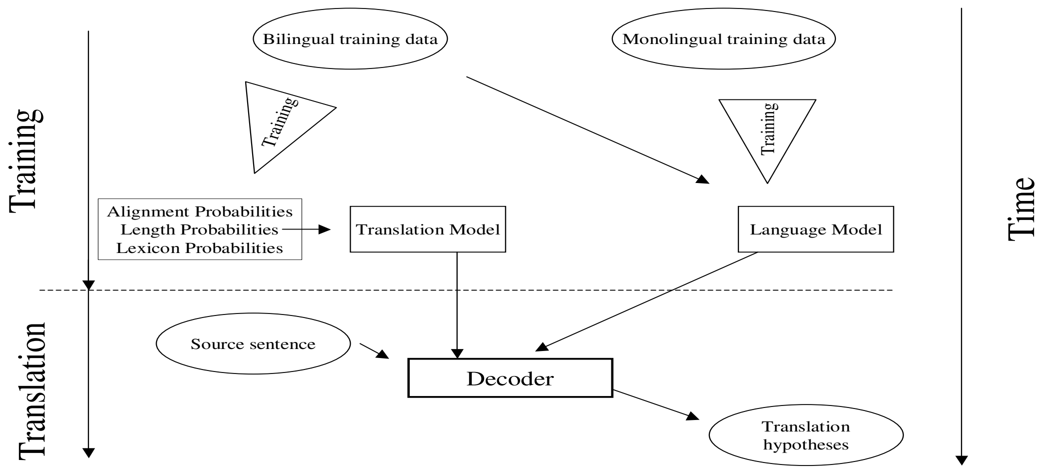


Figure A.1: Modelle der SMT

durch die Verwendung eines Sprachmodells geringer zu gewichten, bzw. Auszusortieren. Auf welche Weise ein Sprachmodell diese Aufgabe erledigt wird noch in diesem Kapitel genauer beschrieben. Nachdem man die Wahrscheinlichkeit des Sprachmodells hinzugefügt hat wird die Hypothese mit der insgesamt größten Wahrscheinlichkeit als die Übersetzung ausgewählt.

Eine Evaluierung der maschinellen Übersetzung wird typischerweise durch den Vergleich der maschinellen Übersetzung mit mehreren menschlichen Übersetzungen durchgeführt. Zwei verbreitete Metriken um diesen Vergleich zu realisieren sind BLEU und NIST. Da im Rahmen dieser Diplomarbeit BLEU scores verwendet werden, einige Details mehr zu BLEU. BLEU ist stark beeinflusst durch lange, übereinstimmende Passagen in der maschinellen Übersetzung und den Referenzen. Dieser Effekt basiert darauf, das BLEU scores im wesentlichen das geometrische Mittel von n-grammen bestimmt, die in beiden Dateien auftreten. Zu kurze Übersetzungen werden darüber hinaus mit zusätzlichen Abzügen versehen, auch bekannt als Length Penalty. Dieser Length Penalty wird durch den Vergleich der kompletten Länge der Hypothesen und der Referenzen für alle Sätze bestimmt. Durch diese Beziehung zu allen Sätzen kann man auch BLEU scores nur für längere Texte ($>$ 100 Sätze) korrekt angeben. Weitere Informationen zu BLEU und NIST kann man in Papineni et al. (2001) und NIST (2003) finden.

Bevor man Übersetzen kann muss man natürlich die verwendeten Modelle trainieren.

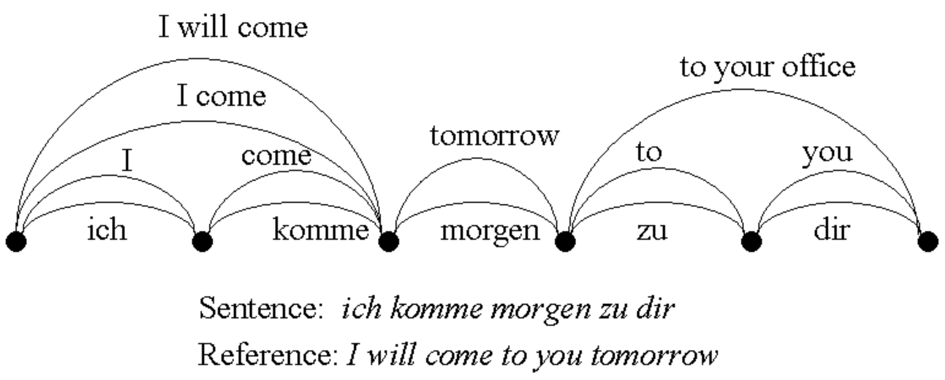

I will come
I come
I
come
tomorrow
to your office
to
you
ich
komme
morgen
zu
dir
Sentence: *ich komme morgen zu dir*
Reference: *I will come to you tomorrow*

Figure A.2: Translation Lattice Beispiel

Meistens verwendet man die fünf IBM Modelle die von Brown et al. (1993) eingeführt wurden. Heutzutage verwendet man oft noch das einfachste IBM Modell 1. Dieses Modell verwendet Wort-zu-Wort Übersetzungswahrscheinlichkeiten, die unabhängig sind von der Satzlänge und den Positionen innerhalb des Satzes. Oft wird dieses Modell mit weiteren Modellen kombiniert die direkt Wahrscheinlichkeiten für bestimmte Wortfolgen angeben. Die grundlegende Idee dabei ist, Quell- und Zielsatz in Phrasen zu unterteilen und dann Wahrscheinlichkeiten für diese Phrasen anzugeben. Trainingstechniken für diese Modelle sind zum Beispiel Integrated Segmentation und Alignment oder Phrase Extraction via Sentence Alignment. Der interessierte Leser sei für weitere Informationen auf Brown et al. (1993), Zhang et al. (2003) und Vogel (2005) verwiesen. Für eine mathematische Formulierung des SMT Problems sei auf die englische Einleitung verwiesen.

Die Aufgabe eines Sprachmodells ist die Bewertung von Sätzen bezüglich ihrer grammatikalischen Korrektheit. Der verbreitete Standard um dies umzusetzen sind n-gram Sprachmodelle. N-gram Modelle bestimmen Wahrscheinlichkeiten anhand der Häufigkeit von Wörtern in großen Trainingsdaten.

Um Sprachmodelle ohne den Einfluss anderer Modelle zu bewerten wird normalerweise die Perplexität herangezogen. In mathematischer Notation ist Perplexität das geometrische Mittel der umgekehrten Wortwahrscheinlichkeit eines Testsets.

$$\sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

Weniger formell repräsentiert die Perplexität die durchschnittliche Anzahl an Wörtern die einem Wort folgen kann. Im schlimmsten Fall kann die Perplexität daher die Vokabulargröße sein.

Motivation und Ansätze

Die folgenden Visualisierungen der Motivation und der Ansätze werden alle folgende Abstraktion benutzen. Einzelne Wörter werden durch Linien dargestellt, Wörter die von besonderem Interesse für den jeweiligen Ansatz sind werden durch Rechtecke hervorgehoben. Je schwärzer die Farbe eines Rechtecks ist, desto mehr Einfluss hat dieses Rechteck auf den gegenwärtigen Ansatz.

Nach dieser Beschreibung der Abstraktion zu den eigentlichen Konzepten. Der aktuelle Ansatz für Sprachmodellierung ist auf A.3 abbgebildet. Das am weitesten rechts stehende Rechteck ist die Wortposition die momentan untersucht wird. Der Übersetzungsprozess hat verschiedene Wörter für diese Position vorgeschlagen. Das Sprachmodell muss jetzt entscheiden, welche dieser Wörter an diese Position passen. Typische n-gram modelle (tri-gramme) würden nur die beiden Wörter rechts von aktuellen Position rechts in die Entscheidungsfindung miteinbeziehen. Verbesserung in den Algorithmen und der Computerleistungsfähigkeit ermöglichen auch immer häufiger das Miteinbeziehen der letzen vier Wörter. Dennoch, wenn man sich die Abbildung anschaut, wird deutlich, dass große Teile des Kontexts nicht in die Entscheidungsfindung einfließen. Diese Diplomarbeit untersucht daher Ansätze wie man weitreichendere Kontexte in diesen Prozess integrieren kann.

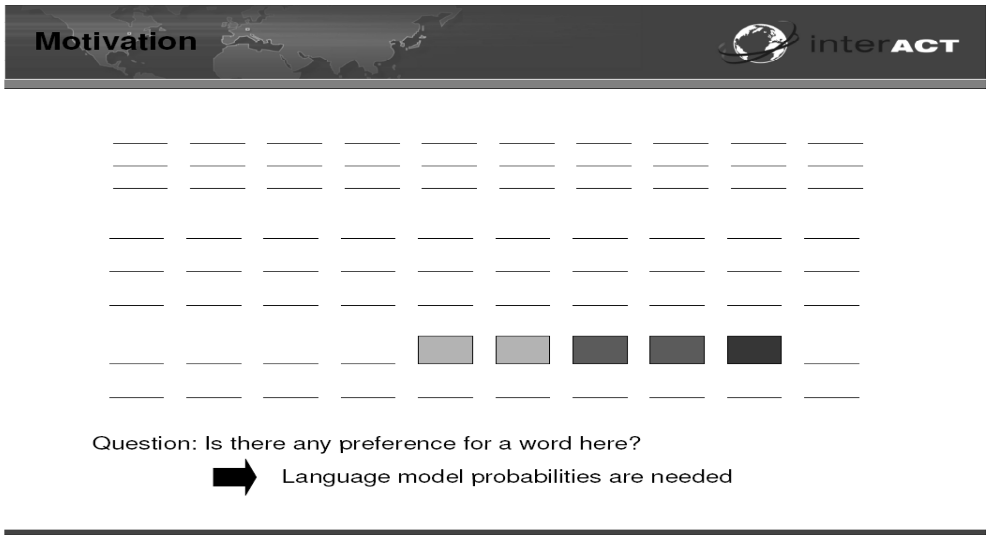

Motivation
interACT
Question: Is there any preference for a word here?
Language model probabilities are needed

Figure A.3: Motivation

Der Grund, warum man nicht ohne weiteres mehr als vier Wörter des Kontexts bei der n-gram Technik benutzen kann liegt in der Datenknappheit. Selbst sehr große Datenmengen reichen nicht aus, da n-gramme exponentiell mehr Daten brauchen wenn man längere Kontexte betrachten will. Ein Ansatz dieses Problem zu umgehen ist klassenbasierte Sprachmodelle zu benutzen (siehe A.4). Bei diesen Modellen wird durch die Abbildung von Wörtern auf

Wortklassen das Vokabular stark verringert. Dieses Verringern des Vokabulars sorgt dafür, dass man wesentlich längere Kontexte benutzen kann.

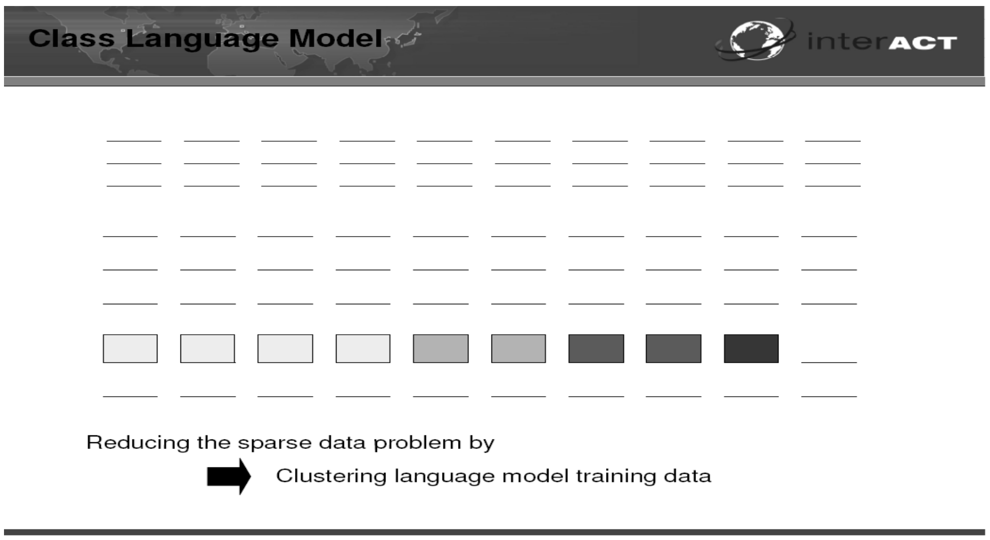

Class Language Model
interACT
Reducing the sparse data problem by
Clustering language model training data

Figure A.4: Idee eines klassenbasierten Sprachmodells

Aber selbst nach der Anwendung des klassenbasierten Sprachmodells enthält der Kontext noch viele Wörter, die nicht berücksichtigt werden. Als weiteren Ansatz kann man deshalb das Cache Sprachmodell (siehe A.5) benutzen um Einflüsse von Wörtern aus den vorhergehenden Sätzen modellieren zu können. Dabei erhöht das Cache Sprachmodell die Wahrscheinlichkeiten für Wörter die man in den letzten Sätzen gesehen hat.

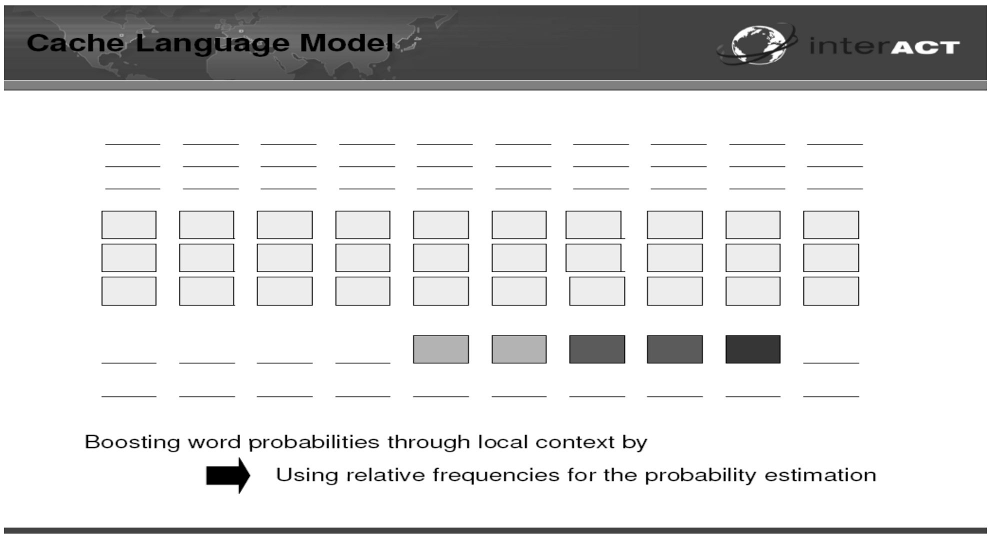

Cache Language Model
interACT
Boosting word probabilities through local context by
Using relative frequencies for the probability estimation

Figure A.5: Idee eines Cache Sprachmodells

Der letzte Ansatz der in dieser Diplomarbeit untersucht wird sind sogenannte Satz Mixtur Modelle. Satz Mixture Modelle versuchen Gemeinsamkeiten in Stil und Thema zwischen

Sätzen zu finden und dieses Wissen für die Bildung besserer Modelle zu benutzen. Daher wird der ganze Satz in dem sich das aktuelle Wort befindet benutzt um Aussagen zu treffen, anstatt sich wie in den anderen Modellen nur auf eine begrenzte Menge an Kontextwörtern zu konzentrieren. Der Satz Mixtur Modell Ansatz ist in Abbildung A.6 zu sehen. In dem grau hinterlegten Bereich ist ein Trainingskorpus abgebildet. Ähnliche Sätze aus diesem Trainingskorpus fließen stärker in die Bildung von Sprachmodell Wahrscheinlichkeiten ein.

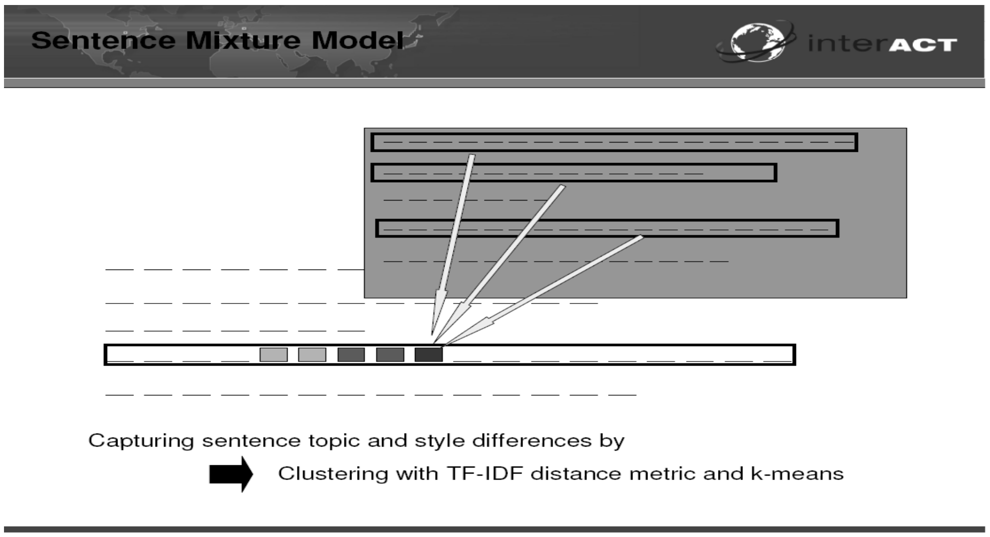

Sentence Mixture Model
interACT
Capturing sentence topic and style differences by
Clustering with TF-IDF distance metric and k-means

Figure A.6: Idee von Satz Mixtur Modellen

Es muss gesagt werden, dass alle drei Ansätze für die Sprachmodellierung bereits bekannt sind, jedoch wurden sie bisher nur im Bereich der Spracherkennung gründlich untersucht und bewertet. Für die maschinelle Übersetzung gibt es bis jetzt aber keine oder kaum Berichte über die Effekte dieser Modelle. Diese Lücke versucht diese Diplomarbeit zu schließen, indem sie diese Techniken in die maschinelle Übersetzung integriert und evaluiert.

Ausblick

Das Ziel dieser Einleitung war den Leser mit der SMT vertraut zu machen und die Ansätze dieser Diplomarbeit zu beschreiben und zu motivieren. Das restliche Dokument ist folgendermaßen aufgebaut. Das nächste Kapitel diskutiert Literatur zu den verfolgten Ansätzen der Sprachmodellierung. Im Rahmen dieser Zusammenfassung wird im folgenden Kapitel dann nur kurz auf verwendete Daten und Implementierungsdetails eingegangen. Die weiteren Kapitel beschreiben dann einige der durchgeführten Experimente und diskutieren diese. Am Ende werden die Ergebnisse dieser Arbeit zusammengefasst und darauf aufbauend Vorschläge für weitergehende Arbeit gemacht.

A.3 Literatur Überblick

In der englischen Ausarbeitung werden im Literature Review verschiedene Techniken der Sprachmodellierung analysiert. Das englischsprachige Literature Review orientiert sich sehr stark an der ausgezeichneten Arbeit von Goodman (2001). In dieser deutschen Fassung werden nur Aspekte der in weiteren Diplomarbeit bearbeiteten Ansätze betrachtet.

N-gram Sprachmodelle

Ein Sprachmodell schätzt Wahrscheinlichkeiten für Wortfolgen. N-gram Sprachmodelle berechnen diese anhand der n letzten Wörter im Kontext.

$$P(w_1...w_i) = P(w_1) \times P(w_2|w_1) \times ... \times P(w_i|w_1...w_{i-1})$$

Aus praktischen Gründen, die noch weiter erläutert werden, wird normalerweise eine Trigram-Annahme gemacht, und nur die letzten beiden Wörter des Kontexts für die Berechnung herangezogen.

$$P(w_1...w_i) \approx P(w_i) \times P(w_i|w_{i-1}) \times P(w_i|w_{i-2}w_{i-1})$$

Die benötigten Wahrscheinlichkeiten werden automatisch anhand von Trainingsdaten bestimmt. Der primitive Ansatz einfach alle Trigramme in den Trainingsdaten zu zählen erzeugt allerdings Probleme. Zum einem kommen manche Trigramme dort nicht vor, und zum anderen kommen viele Trigramme nur sehr selten vor. Beides führt zur Schätzung von eher schlechten Wahrscheinlichkeiten. Um diese Probleme zu beheben wird normalerweise eine Glättung durchgeführt (engl. Smoothing). Smoothing Techniken verbessern die Bestimmung der Wahrscheinlichkeiten häufig in dem sie Wahrscheinlichkeitsmasse von der normalen Berechnung von Trigrammen verändern, und auch Trigramme zulassen, die nicht in den Trainingsdaten vorhanden waren. Mehr Details findet man im englischen Orginal und in Chen and Goodman (1998). Die heutzutage am meisten verwendete Smoothing Technik ist das Kneyser Ney Smoothing.

Nun zu den praktischen Gründen, warum man in den meisten Anwendungen bisher die Trigram Annahme gemacht hat anstatt mehr Wörter des Kontexts mit einzubeziehen. Das zentrale Problem, ist das mit der Vergrößerung des betrachteten Kontexts die Anzahl der Parameter von n-gram Modellen exponentiell ansteigt. Als Beispiel sei gesagt, dass ein 5-gram Sprachmodell 10 Milliarden mal so viele Parameter wie ein Trigram Sprachmodell hat bei einem Vokabular von 100.000 Wörtern. Diese Menge an Parametern ruft jedoch zwei praktische Probleme hervor.

Zum Einen braucht man riesige Mengen an Trainingstext um diese Anzahl an Parametern korrekt zu bestimmen. Zum Anderen müssen die Algorithmen auch in der Lage sein, diese riesigen Mengen von Trainingstext zu verarbeiten. Wie gesagt sind beides nur praktische Probleme, und in den Zeiten des Internet steigt die Menge an Trainingsdaten sowieso mit nie für möglich gehaltener Geschwindigkeit. Mit der Suffix Array Sprachmodellierungstechnik (siehe englisches Literature Review) gibt es zudem auch eine Möglichkeit mit weniger Rechenpower die benötigten Parameter zu bestimmen. Mehr über n-gram Sprachmodelle kann man in Knight (1999) und Stolcke (2002) nachlesen.

Klassenbasierte Sprachmodelle

Grundsätzlich gibt es zwei verschiedene Ansätze ein klassenbasiertes Sprachmodell zu erzeugen. Der eine Ansatz geht von handgemachten Klassen aus, während der andere versucht automatisch Klassen auf einem Trainingskorpus zu bestimmen. Der erste Ansatz führt häufig zu linguistisch motivierten Klassen. Bei dem zweiten Ansatz fällt es Menschen eher schwer Gründe für die Klassifizierung nachzuvollziehen.

Nachdem man die Klassen gefunden hat, gibt es immer noch verschiedene Möglichkeiten wie das klassenbasierte Sprachmodell seine Wahrscheinlichkeiten bestimmen kann. Oft wird folgende Formel angewandt

$$P(w_1...w_i) = P(w_i|W_i) \times P(W_i|W_{i-2}W_{i-1})$$

Wobei die W sich auf die Klassen des jeweiligen Wortes beziehen. In einfachen klassenbasierten Sprachmodellen wird oft eine eins zu eins Zuordnung zwischen Wort und Klasse gefordert. Der hier gezeigt Ansatz ignoriert die tatsächlichen Wörter im Kontext. Als Folge dessen muss man diesen Ansatz immer mit einem normalen Sprachmodell interpolieren um insgesamt gute Sprachmodellwahrscheinlichkeiten zu bekommen.

Der Vorteil von klassenbasierten Sprachmodellen ist die Reduktion der Datenknappheit. Durch die Verringerung des Vokabulars wird automatisch auch die Anzahl der benötigten Parameter stark vermindert. Ein 5-gram würde jetzt nur noch 1000 mal so viele Parameter benötigen wie ein klassenbasiertes 3-gram Modell.

Als ein Beispiel für die Anwendung eines klassenbasierten Sprachmodells betrachte man "'Ich wurde in Berlin geboren"'. In einem normalen n-gram Sprachmodell wäre dieses Beispiel eher nutzlos für die Schätzung von 5-gram Wahrscheinlichkeiten, da es selten in einem Trainingstext vorkommt. Würde man allerdings eine (handgemachte) Klasse <STADT> haben, würde der Satz folgendermaßen lauten "'Ich wurde in <STADT> geboren"'. Offensichtlich ist diese Wortfolge viel häufiger, und eine Wahrscheinlichkeit kann dementsprechend robuster geschätzt werden.

Frühere Arbeiten haben gezeigt das Sprachmodelle mit automatisch erzeugten Klassen besser arbeiten als Sprachmodelle mit handgemachten Klassen (Niesler et al., 1998). Goodman (2001) berichtet, dass klassenbasierte Sprachmodelle die Perplexität um 7% für große Datenmengen und um bis zu 13% für kleinere Datenmengen verringern können.

Eine weitere interessante Idee im Zusammenhang mit klassenbasierten Sprachmodellen wurde von Och (1999) vorgestellt. In dieser Arbeit werden bilinguale Wortklassen gebildet. Die gefundenen Ergebnisse sind vielversprechend, allerdings wurden sie mit einer bestimmten Idee für den gesamten Übersetzungsprozess gekoppelt, der speziell von diesen Wortklassen profitiert.

Es gibt eine Reihe von Toolkits die die automatische Erzeugung von Klassen unterstützen (Och, 1995) (Stolcke, 2002).

Cache Sprachmodelle

Cache Sprachmodelle versprechen die größte Verbesserung im Vergleich zu normalen Trigram Sprachmodellen. Goodman (2001) zeigt, dass Verbesserung der Perplexität um 35% auf kleinen Trainingsdaten möglich sind, und immer noch 15% für größere Mengen an Trainingsdaten. Bi- und Trigram Cache Sprachmodelle haben bessere Leistungen als nur wortbasierte Cache Sprachmodelle gezeigt.

Die Idee bei Cache Sprachmodellen ist lokale Kontextabhängigkeiten zu modellieren. Typischerweise meint man mit lokal die letzten 2000 bis 5000 Wörter (Nepveu et al., 2004). Die Wahrscheinlichkeit von Wörtern wird entsprechend ihrer relativen Häufigkeit innerhalb dieses Fensters angehoben. Dabei kann es variieren, wie dieses Anheben im Endeffekt aussieht. Man könnte z.B. eine Interpolation zwischen Cache- und n-gram Sprachmodell durchführen, oder auch eine eine Multiplikation mit eins plus der Wahrscheinlichkeit des Cache Sprachmodells ist möglich.

Eine anspruchsvollere Variante lokale Wortabhängigkeiten zu modellieren wurde von Kuhn and de Mori (1990) untersucht. Dabei wurden für 19 unterschiedliche Part of Speech Klassen 19 unterschiedliche Cache Geschichten gespeichert. Der passende Cache wird ausgewählt nachdem der Part of Speech des aktuellen Wortes bestimmt ist. Diese Cache Wahrscheinlichkeit wurde dann mit einem 3g-gram Model interpoliert, welches auch in Kuhn and de Mori (1990) beschrieben wird. Der Nachteil für dieses Model ist der Bedarf an Part of Speech getagten Trainingsdaten.

Ein Problem bei Cache Sprachmodellen ist, dass man nie weiß, ob die Wortgeschichte korrekt ist oder nicht. Dennoch werden Cache Modelle in den letzten Jahren wieder populärer.

Zhang et al. (2004) und Vaiciunas and Raskinis (2006) sind zwei Beispiele, die beide jedoch nur Perplexitätsverbesserungen berichten. Nepveu et al. (2004) berichtet über Verbesserungen im Bereich der interaktiven maschinellen Übersetzung.

Insgesamt erscheint es äußerst interessant die Effekte von Cache Modellen auf die Maschinelle Übersetzung zu untersuchen.

Satz Mixtur Sprachmodelle

Innerhalb eines Textes können sich Sätze zum Beispiel durch Stil oder Inhalt unterscheiden. Satz Mixtur Modelle versuchen diese Gemeinsamkeiten auszunutzen, indem für Gruppen von Sätzen spezielle Sprachmodelle gebaut werden. Dadurch werden versteckte Abhängigkeiten zwischen Wörtern in ähnlichen Sätzen modelliert. Goodman (2001) hat Verbesserungen von 19% mit 284 Millionen Wörtern für das Spravchmodell berichtet. Iyer and Ostendorf (1999) konnte mit weniger Daten (38 Millionen Wörter), aber einer besseren Clustering Technik bessere Ergebnisse erzielen. Bei einem vergleichbaren Test erreicht Iyer and Ostendorf (1999) in etwa eine doppelt so gute Perplexitätsverbesserung.

Soweit dem Autor bekannt ist sind Satz Mixtur Modelle bis jetzt nicht für die maschinelle Übersetzung untersucht worden. Dabei könnten sie außerordentlich gut geeignet sein für die Anwendung auf die maschinelle Übersetzung. Der Grund dafür ist, dass sie in der bisherigen Anwendung in der Spracherkennung erst angewendet werden konnten, nachdem man eine komplette Übersetzung des Satzes hatte. Das machte sie wertlos für one pass Erkenner, bei denen man Satz Mixtur Sprachmodelle im besten Fall für n-besten Listen Rescoring einsetzen kann. In der maschinelle Übersetzung kann man dieses Problem jedoch umgehen in dem man das Clustering in der Quellsprache macht. Da man dabei vollkommen korrekte Sätze für das Clustering verwendet könnten Satz Mixtur Modelle für die maschinelle Übersetzung sogar bessere Ergebnisse liefern als für die Spracherkennung.

Es gibt noch weitere Arbeiten, die mit ähnlichen Ansätzen Verbesserungen bekommen haben. Wu and Khudanpur (1999) berichtet mit einem sehr ähnlichen Ansatz 12% Perplexitätsverbesserung, sagt aber auch, dass man bessere Ergebnisse bekommt wenn man nur Content Wörter berücksichtigt.

A.4 Implementierungsdetails

Nachvollziehbarkeit ist ein wesentlicher Aspekt von Forschung, weshalb dieser Abschnitt im englischen Original eine wichtige Rolle spielt. Im Rahmen dieser Zusammenfassung allerdings sind die meisten Implementierungsdetails eher unwichtige Details. Daher werden diese hier in der deutschen Zusammenfassung nicht weiter ausgeführt. Ebenso wird für genauere Angaben zu verwendeten Daten auf das englische Original verwiesen.

A.5 Experimente

Klassenbasierte Sprachmodelle

Um klassenbasierte Sprachmodelle zu realisieren braucht man zuerst einmal eine Zuordnung von Wörtern zu Klassen, oder ein Program das diese erzeugen kann. Für die im weiteren verwendeten Klassen wurde das Programm mkcls von F.J. Och verwendet (Och, 1995). Die erste Tabelle A.1 zeigt die Ausgabe dieses Programms auf einem 16 Millionen Wort Trainingstext. Die Anzahl der Klassen war dabei immer fest vorgegeben und variierte zwischen 50 und 1000 Klassen. Die Startplexität ist die Perplexität die eine zufällige Zuordnung erreicht, die Endperplexität wird erreicht durch die Optimierungen des mkcls Programms.

Klassenanzahl	Start-PP	End-PP	Optimierungsiterationen
50	584	235	14
200	433	166	4
500	315	133	2
1000	230	112	1

Table A.1: Mkcls Training Information

Die Übersetzungsexperimente sind auf den arabisch englischen MT03 TIDES Test Daten gemacht worden. Tabelle A.2 zeigt die Ergebnisse. Die Klassenzuordnung war auf 65 Millionen Wörter trainiert worden. Wie die Ergebnisse zeigen, führt das klassenbasierte Sprachmodell fast immer zu einer Verschlechterung.

#Klassen	no	10	50	100	200
run1 BLEU	45,70	45,22	45,91	45,37	45,19
run1 Scale	0,0	0,008	0,244	0,163	0,116
run2 BLEU	45,70	44,96	45,44	45,78	45,45
run2 Scale	0,0	0,0	0,106	0,062	0,167

Table A.2: Übersetzungen mit Klassenbasiertem Sprachmodell

Der Unterschied zwischen run1 und run2 ist, dass run2 separate Klassen hat für Satzzeichen. Die Gewichtungsfaktoren für das Modell werden automatisch durch minimales Fehlertraining bestimmt. Auch ein weiteres Experiment konnte keine Verbesserung zeigen.

Cache Sprachmodelle

Im englischen Original wurden Experimente mit drei unterschiedlichen Cache Modellen gemacht. Zuallererst mit dem Wort Cache Modell, welches seine Wahrscheinlichkeit nur anhand einzelner Worte trifft. Desweiteren das Bi- und Trigram Cache Modell, welches zudem noch Wortfolgen von zwei oder drei Wörter beachtet. Schließlich noch das N-best Cache Agreement Cache Modell, welches auf einer Idee eines aktuellen Papers von Zens und Ney beruht. Da letzteres aber immer Verschlechterungen lieferte, wird in dieser Zusammenfassung nur auf Experimente mit den ersten beiden Modellen eingegangen, im Wesentlichen dabei auf das Bi- und Trigram Cache Sprachmodell.

Das erste Wort Cache Modell Experiment diente dazu eine geeignete Größe für die Cache Geschichte zu wählen. Die Ergebnisse sind in Tabelle A.3 zu sehen. Die Speicherung von 180 Wörter erscheint demnach ausreichend, da damit die maximale Verbesserung von 17,6% erreicht wird.

Cache Size	no	100	150	180	200	400	600	800
Perplexity	179,4	149,0	147,9	147,7	147,8	149,4	151,6	153,2

Table A.3: Länge der Cache Geschichte

Weitere Experimente wurden mit dem Wort Cache Model für das n-besten Listen Rescoring gemacht. Von dem jetzigen Standpunkt aus, sind diese allerdings nur ein Spezialfall für das n-besten Listen Rescoring bei Bi- und Trigram Sprachmodellen. Deshalb werden hier gleich diese Ergebnisse vorgestellt.

Bei diesen Experimenten gibt es eine größere Anzahl an Parametern die zu bestimmen sind. Unter anderem Mixturgewichte für die vier Sprachmodelle(Wort, Bi- und Trigram Cache, normales n-gram), Länge der Cache Geschichte usw. Zusätzlich gibt es noch die Möglichkeit den Cache zu leeren, wenn man die Grenze eines Dokuments überschreitet. Um geeignete Werte für alle diese Parameter zu finden wurde eine Gittersuche durchgeführt (Tabelle A.4). Dabei wurden alle Cache Modelle zuerst miteinander interpoliert, bevor das Ergebnis dieser Interpolation mit dem normalen Trigram Sprachmodell interpoliert wird. Im Ersten Fall werden die Interpolationsgewichte hier interne Gewichte genannt und mit bi bzw. tri in der Tabelle abgekürzt. Im zweiten Fall wird die Gewichtung des Cache Modells einfach Cache Gewicht genannt. In der vorliegenden Tabelle war dieses erstmal fest auf 0,2 gesetzt.

Aus dieser Tabelle kann man folgendes schließen. Erstens ist es besser das unbekannte Wort als normales Wort zu modellieren, und zweitens scheint hier eine Cache Geschichte mit 500 Wörtern bessere Ergebnisse zu liefern. Allerdings basieren beide Aussagen auf nicht signifikanten Ergebnissen. Es wurde noch eine weitere Gittersuche durchgeführt, bei der diesmal das Cache Gewicht variiert wurde, dafür aber immer mit 500 Wörtern in der Geschichte und immer mit dem unbekannten Wort als normales Wort. Diese Experimente lieferten keine bedeutenden neuen Ergebnisse.

Übersetzungsergebnisse wurden auf drei unterschiedlichen Test Daten erzeugt. Zuerst wurde mit dem spanisch englischen BTEC Korpus gearbeitet. Da zu diesem Zeitpunkt das

	Cache Geschichte 180		Cache Geschichte 500	
	unkIsWord	unkNoWord	unkIsWord	unkNoWord
bi0tri0	31,91	31,91	32,00	31,98
bi0tri1	31,91	31,91	32,00	31,98
bi0tri3	31,94	31,93	32,01	31,99
bi0tri5	32,00	31,96	32,00	31,99
bi1tri0	31,91	31,90	32,00	31,98
bi1tri1	31,93	31,92	32,01	31,99
bi1tri3	31,96	31,95	32,00	32,00
bi1tri5	32,02	31,98	32,02	32,00
bi3tri0	31,95	31,94	32,01	31,99
bi3tri1	?	?	?	?
bi3tri3	32,02	31,98	32,03	32,00
bi3tri5	32,00	31,94	31,96	31,92
bi5tri0	31,96	31,94	32,00	31,98
bi5tri1	32,02	31,98	32,04	32,01
bi5tri3	32,00	31,95	31,96	31,92
bi5tri5	31,89	31,89	31,89	31,88

Table A.4: Gittersuche mit Cache Gewicht 0,2

Cache Modell noch nicht in das minimale Fehlertraining integriert war, musste hier wieder eine Gittersuche nach den optimalen Parametern durchgeführt werden. Dabei wurden im Cache die letzten 500 Wörter gespeichert. Die Ergebnisse sieht man in Tabelle A.5.

Die Baseline ist immer 18,90, egal ob man das unbekannte Wort als normales Wort oder betrachtet oder nicht. Der Grund hierfür ist das Übersetzungsprogramm, welches unbekannten Wörtern automatisch eine Grundwahrscheinlichkeit zuweist anstatt bei dem n-gram Sprachmodell anzufragen. Die Größte Verbesserung gegenüber der Baseline beträgt 0,9%. Danach wurden weitere Experimente gemacht, schnell wurde jedoch klar, dass es äußerst unpraktisch ist jedesmal eine manuelle Gittersuche durchführen zu müssen.

Dementsprechend wurden die Cache Parameter in das vorhandene minimale Fehlertraining integriert. Nachdem diese Integration abgeschlossen war, wurden Experimente auf dem arabisch englischen MT03 Testset durchgeführt. Die Ergebnisse sind in Tabelle A.6 zu sehen.

Wie zuvor bringt das Cache Modell eine kleine Verbesserung. Allerdings waren alle Experimente bisher nur auf Entwicklungsdaten, auf denen die Parameter optimiert werden können. Die nächsten Experimente auf dem japanisch englischen BTEC Korpus untersuchen inwieweit sich dieser Gewinn auf ein ungesehenes Testset übertragen lässt.

Leider zeigen die Experimente A.7, dass sich dieser Gewinn nicht auf ein ungesehenes Testset übertragen lässt.

cache Gewicht	word	bigram	trigram	BLEU -NoUnk	BLEU -Unk
0,0	-	-	-	18,90	18,90
0,05	0,9	0,0	0,1	18,79	19,07
	0,8	0,1	0,1	18,81	18,75
	0,6	0,1	0,3	18,81	18,81
0,1	0,9	0,0	0,1	18,75	18,75
	0,8	0,1	0,1	18,73	18,73
	0,6	0,1	0,3	18,96	18,92
0,2	0,4	0,1	0,5	18,66	18,66
	0,4	0,3	0,3	18,66	18,66
	0,4	0,5	0,1	18,67	18,56
0,3	0,7	0,0	0,3	18,07	18,07
	0,8	0,1	0,1	18,08	18,08
	0,2	0,5	0,3	18,61	18,61
0,4	0,5	0,0	0,5	18,15	18,15
	0,4	0,1	0,5	18,34	18,34
	0,2	0,3	0,5	18,48	18,48

Table A.5: Cache BTEC Translations

System	Baseline	bestes Ergebnis
wordCache	-	0,04736
bigramCache	-	0,00002
trigramCache	-	0,06055
ngram	-	1,84751
BLEU	47,06	47,44

Table A.6: Cache Arabic CMU MT03 mit Optimierung

	baseline	cache
wordCache	0	0,039
bigramCache	0	0,008
trigramCache	0	0,164
ngram	1	1
DEV BLEU	55,85	55,78
IWSLT_05 BLEU	50,43	50,06
drop	10.7 %	11.4 %

Table A.7: BTEC Japanisch Englisch Ergebnisse

Satz Mixtur Modelle

In diesem Abschnitt wird zwischen monolingualen und bilingualen Satz Mixtur Modellen unterschieden. Dabei sind monolinguale Satz Mixtur Modelle im Wesentlichen das, was von Iyer and Ostendorf (1999) beschrieben wurde. Anhand einer first pass Erkennung wird ein Sprachmodell adaptiert, und mit dieser Adaption wird dann eine erneute Übersetzung durchgeführt. Bilinguale Satz Mixtur Modelle dagegen führen keine first pass Erkennung durch, sondern klassifizieren das Testset in der Quellsprache und die Sprachmodell Daten in der Zielsprache. Beide hier beschriebenen Ansätze unterscheiden sich auch dadurch von der ursprünglichen Idee, dass eine Klassifikation des Testsets durchgeführt wird. Alle Parameter, nicht nur die Sprachmodell Parameter werden dann für diese Klassen trainiert.

In dem englischen Original wurden zuerst einige Experimente gemacht, die sich mit dem Zeitfaktor beschäftigen. Der Sinn dabei war, geeignete Größen für Parameter zu finden. Da diese für das weitere Verständnis nicht wichtig sind, wird hier direkt mit der Beschreibung von Perplexitäts Ergebnissen begonnen.

Für alle folgenden Experimente gilt: monolinguale Satz Mixtur Modelle wurden im Zusammenhang mit chinesisch englischen WebBlog Daten verwendet, bilinguale Satz Mixtur Modelle mit dem chinesisch englischen Xinhua Korpus. Für das trainieren des Klassifikators wurden immer 1000 Dokumente verwendet, die Sprachmodelldaten bestanden aus 230 Millionen Wörtern für die Webdaten und 180 Millionen Wörtern für die Xinhua Aufgabe. Es wurden immer 6 Klassen verwendet (Begründung siehe Analyse Abschnitt) .

Entsprechend dem Ansatz gab es sechs verschiedene Teile jedes Testsets (*testkl[asse]1-6*), und 7 verschiedene Sprachmodelle, je eins für jede Klasse (*SM1-6*) und das komplette Sprachmodell (*KSM*). Die folgenden Tabellen geben die Perplexität immer für jede Klasse des Testsets mit jedem Sprachmodell an. Fairerweise muss gesagt werden, dass Perplexitäten zwischen verschiedenen Sprachmodellen hier nicht direkt verglichen werden können, da sie unterschiedliche Vokabulargrößen haben.

Das Perplexitäts Testset für die monolingualen Satz Mixtur Modelle bestand aus 1000 Dokumenten mit 95.000 Wörtern. Das Ergebnis kann man in Tabelle A.8 sehen. Ein Sprachmodell der passenden Klasse erzeugt jeweils die besten Ergebnisse, auch besser als das volle Sprachmodell.

Im Wesentlichen das gleiche Experiment wurde für die bilingualen Satz Mixtur Modelle durchgeführt. Im Rahmen dieser Perplexitätsexperimente wurde dabei nur mit englischen Daten gearbeitet, dennoch sind die Zahlen interessant, da Zeitungsartikel viel ähnlicher zueinander sind als verschiedene WebBlogs. Tabelle A.9 zeigt die Ergebnisse.

Wie erwartet sind die Perplexitätsgewinne kleiner als bei den WebBlog Daten, da die Zeitungsartikel nicht so unterschiedlich sind wie verschieden WebBlogs.

	testkl1	testkl2	testkl3	testkl4	testkl5	testkl6
SM 1	**116**	2040	1114	1133	920	2075
SM 2	2717	**445**	968	1291	1133	1664
SM 3	1742	2102	**425**	1955	1570	2180
SM 4	1562	2079	1345	**92**	146	1900
SM 5	1268	1690	1039	178	**59**	1633
SM 6	1584	2273	1067	1385	1132	**813**
KSM	175	565	499	255	166	894

Table A.8: WebBlog Perplexitätsergebnisse

	testkl1	testkl2	testkl3	testkl4	testkl5	testkl6
SM 1	**39**	512	690	476	486	501
SM 2	270	**88**	426	248	199	236
SM 3	634	466	**82**	330	323	352
SM 4	744	327	325	**100**	216	271
SM 5	1113	368	467	311	**74**	220
SM 6	1394	449	538	381	232	**86**
KSM	59	96	105	111	83	107

Table A.9: Xinhua Perplexitätsergebnisse

Nach diesen Ergebnisse, die zeigen das das Klassifizieren funktioniert, nun zu den eigentlichen Übersetzungsergebnissen. Zuerst werden einige Ergebnisse auf Entwicklungsdaten vorgestellt.

Bei den WebBlog Daten hatte ein Baseline System einen BLEU Score von 7,73. Die Ergebnisse stammen von einem Testset mit 250 Sätzen, bekannt als checksum2 der GALE Evaluation. Wie Tabelle A.10 zeigt, kann ein Satz Mixtur Modell eine Score von bis zu 8,86 erreichen. Das beste Ergebnis *KSM clustered* wird erreicht, wenn man die einzelnen Teile des Testsets mit dem vollen Sprachmodell übersetzt, aber alle Parametergewichte für jede Klasse separat anpasst.

	testkl1	testkl2	testkl3	testkl4	testkl5	testkl6	250 checksum2
SM 1	9,86	-	-	-	-	-	-
SM 2	-	10,03	-	-	-	-	-
SM 3	-	-	7,47	-	-	-	-
SM 4	-	-	-	9,70	-	-	-
SM 5	-	-	-	-	8,29	-	-
SM 6	-	-	-	-	-	13,73	-
Baseline	-	-	-	-	-	-	7,73
LM1-6	9,86	10,03	7,47	9,70	8,29	13,73	8,15
KSM clustered	12,87	13,94	7,62	11,81	9,53	12,88	8,86

Table A.10: WebBlog 250 Sätze Entwicklungsdaten

Wie sich später herausstellen wird, sind diese Verbesserungen allerdings auf eine Überanpassung an die Entwicklungsdaten zurückzuführen. Als nächstes wieder das vergleichbare Experiment mit den bilingualen Satz Mixtur Modellen. Diese wurden auf den 191 Sätzen des checksum1 Testsets der GALE Evaluation durchgeführt.

Tabelle A.11 zeigt die Ergebnisse. Wieder wird das beste Ergebnis durch *KSM clustered* erreicht. Zusätzlich zu der Übersetzung mit dem jeweils passenden Sprachmodell wurde hier noch eine Interpolation zwischen dem passenden Sprachmodell und dem vollen Sprachmodell getestet (*SM1-6 & KSM*). Diese erreichte jedoch eine schlechtere Score als die Übersetzung nur mit dem kompletten Sprachmodell.

	testkl1	testkl2	testkl3	testkl4	testkl5	testkl6	191 checksum
SM 1	24,98	-	-	-	-	-	-
SM 2	-	29,80	-	-	-	-	-
SM 3	-	-	28,04	-	-	-	-
SM 4	-	-	-	31,79	-	-	-
SM 5	-	-	-	-	25,83	-	-
SM 6	-	-	-	-	-	23,62	-
Baseline	-	-	-	-	-	-	27,64
SM1-6	24,98	29,80	28,04	31,79	25,83	23,62	28,13
SM1-6 & KSM	26,89	30,42	29,30	32,62	25,60	28,50	29,31
KSM clustered	30,07	30,04	29,96	32,22	26,86	26,70	29,66

Table A.11: Xinhua 191 Sätze Entwicklungsdaten

Die Verbesserungen im Überblick: eine 1,8% Verbesserung, wenn man nur mit dem passenden Sprachmodell übersetzt, eine 6,0% Verbesserung wenn man mit den interpolierten Sprachmodellen übersetzt, und eine 7,3% Verbesserung wenn man nur mit dem vollen Sprachmodell übersetzt.

Das nächste Experiment auf ungesehenen Testdaten zeigt leider, dass diese Verbesserungen aufgrund von Überanpassung an die Trainingsdaten entstehen.Das echte Testset ist das TIDES chinesisch englische MT05 Testset mit 1082 Sätzen. Die Baseline ist wieder eine Übersetzung aller Daten mit dem vollen Sprachmodell. Dieses wird mit einer Übersetzung wiederum mit dem vollem Sprachmodell, aber mit angepassten Parametern für jedes Klasse des Testsets verglichen. Dabei kann keine Verbesserung erzeugt werden. Im Gegensatz, man sieht sogar eine Verschlechterung, die darauf hinweist das man hier eine Überanpassung an die Entwicklungsdaten hatte.

	testkl1	testkl2	testkl3	testkl4	testkl5	testkl6	191 checksum
Baseline	-	-	-	-	-	-	22,20
KSM clustered	-	-	-	-	-	-	21,25

Table A.12: Bilinguale Satz Mixtur Modelle Testergebniss

A.6 Analyse

Klassenbasiertes Sprachmodell

Um die Abdeckung des Testsets durch die Trainingsdaten zu untersuchen wurde ein Out Of Vocabulary (OOV) Wort Check gemacht. Erste Ergebnisse ergaben viel zu hohe OOV Raten. Im Endeffekt stellte sich heraus, dass in den in der Diplomarbeit vorgestellten Ergebnissen ein Trainingskorpus mit Groß- und Kleinschreibung verwendet worden war, während das Übersetzungssystem eine Ausgabe ohne großgeschriebene Wörter produzierte. Nachdem dieses Problem behoben war, nahm die OOV Rate erwartete Werte um die 1% an.

Bereits zuvor waren weitere Experimente für eine Evaluation durchgeführt worden, die nicht in dieser Diplomarbeit erwähnt wurden. Diese Experimente waren mit korrekten Daten durchgeführt worden, ohne andere Ergebnisse zu produzieren. Der Vollständigkeit wegen werden hier nochmal die in der Diplomarbeit vorgestellten Ergebnisse durchgeführt, diesmal mit korrekten Daten. Run2Org bezieht sich auf die ursprünglichen Ergebnisse mit separaten Klassen für Satzzeichen, run2Fix zeigt die neuen Ergebnisse mit korrekten Trainingsdaten. Der Einfluss des klassenbasierten Sprachmodells ist einfach zu klein, um nennenswerte Unterschiede zu erzeugen, wie die Ergebnisse in Tabelle A.13 zeigen.

#Klassen	no	10	50	100	200
run2ORG BLEU	45,70	44,96	45,44	45,78	45,45
run2FIX BLEU	45,70	45,31	45,23	44,21	44,74

Table A.13: Neue Ergebnisse mit klassenbasiertem Sprachmodell

Cache Sprachmodell

Eine Vielzahl an Experimenten wurde durchgeführt um den Einfluß von Cache Sprachmodellen zu untersuchen. Wie erwartet konnte das Cache Sprachmodell schöne Verbesserungen in der Perplexität von bis zu 17,6% erzeugen. Hinsichtlich Übersetzungsergebnissen konnte jedoch keine konstante Verbesserung gezeigt werden. Einige Experimente zeigten eine kleine Verbesserung, einige eine kleine Verschlechterung. Es erschien seltsam, dass einige Ergebnisse relativ hohe Strafen für zu kurze Übersetzungen bekommen hatten wenn man das Cache Modell benutzte. Folglich wird dieser Aspekt hier näher untersucht.

Eine Zusammenfassung von insgesamten Übersetzungslängen bei verschiedenen Experimenten kann man in den beiden Tabellen A.14 und A.15 sehen. Die Experimente mit Phrase Extraction with Log-Linear Features (PELLF) Phrasentabelle wurden in dieser Zusammenfassung bisher nicht erwähnt, da es hier trotz einiger Versuche nicht gelang das Längenproblem zu beseitigen. Wie die folgende Analyse zeigen wird, lag dieses Problem jedoch nicht am Cache Sprachmodell.

System	durchsch. Referenz	erste Referenz	baseline	cache
IBM Phrasentabelle	17.503	16.756	19.843	19.531
PELLF Phrasetabelle			18.050	16.289

Table A.14: Cache Einfluß auf die Länge der Übersetzung - MT03

System	durchsch. Referenz	erste Referenz	baseline	cache
Dev Set	3584	3604	3301	3401
Test Set	3630	3617	3408	3458

Table A.15: Cache Einfluß auf die Länge der Übersetzung - Japanese BTEC

Wenn man sich die erste Tabelle zu dem MT03 Testset anschaut, kann man glauben, dass ein Cache Modell eher kürzere Übersetzungen bevorzugt. Auf dem japanisch englischen Testset hat man den genau umgekehrten Effekt. Hier werden die Übersetzungen eher länger, wenn man das Cache Sprachmodell benutzt. Tatsache ist, dass wenn man das Ergebnis mit der PELLF Phrasentabelle ignoriert, das Cache Sprachmodell die Übersetzungslänge immer in die gewünschte Richtung verändert hat. Dabei ist die gewünschte Richtung die durchschnittliche Referenzlänge. Daher erscheint es wahrscheinlich, dass das Längenproblem bei der PELLF Übersetzung nicht am Cache Sprachmodell lag.

In der englischen Analyse werden hier noch einige Beispielübersetzungen gezeigt. Der wesentliche Grund dafür ist, zu zeigen wie schwer man den Einfluss eines Cache Modells auf eine Übersetzung nachvollziehen kann. In Sätzen mit 50 Wörtern gibt es einfach zu viele Abhängigkeiten, die man als Mensch nur schwer erfassen kann.

Satz Mixtur Sprachmodelle

Zuerst hier eine kurze Erklärung, warum immer sechs Klassen verwendet wurden. Es ist theoretisch nicht allzu schwer die beste Klassenanzahl automatisch zu bestimmen. Der Grund warum keine dieser Möglichkeiten benutzt wurde, ist ersten die Information aus dem Literatur Review, dass mehr Klassen besser sind. In der vorhandenen Literatur wurde meistens vier oder acht Klassen verwendet. Daher war die Hoffnung, dass sechs Klassen bereits eine

Verbesserung zeigen. Nachdem man mit sechs Klassen erfolgreich gearbeitet hat, kann man immer noch die Anzahl der Klassen varieren um bessere Ergebnisse zu bekommen. Wenn sechs Klassen aber keine Verbesserung zeigen, ist es wenig vielversprechend diese durch mehr Klassen erreichen zu können.

Die Perplexitätsergebnisse haben eindeutig gezeigt, dass die Klassifikation korrekt funktioniert. Sprachmodelle haben auf korrespondierenden Testsets deutlich bessere Ergebnisse erzielt. Hier wird nur auf die Klassen bei der WebBlog Übersetzung eingegangen. Die Schlüsselwörter der Klassen kann man in Tabelle A.16 sehen.

Cluster 1	people time thinking way asked like great p eyes early mother feelings students tears prince friends el teacher buss m think going want right life long came information school days
Cluster 2	heard door car years played mr support fight quickly lines response husband barely balance garage slash amputated diagonal
Cluster 3	parents sense energy environmental like people new home did took understand power future uncle responsibility protection told vehicle prince emissions make need say great let came school car thought news
Cluster 4	looking heart school car set warm heavy small cotton like went new home way work life come years hard high point moment word inside understand pain special states return basically
Cluster 5	daughter car time home husband school years like year family students high day university new say come end want marriage life alma know really good today days able feeling buy
Cluster 6	soon fact new think going work girls come look hours hard phone high trying general schools learning block teaching solid teachers basis environment efforts studies junior atmosphere purity signature adapt

Table A.16: WebBlog Monolinguale Schlüsselwörter

Nachdem Lesen des Testsets wird klar, dass die 250 Sätze des Testsets sich auf zwei Blogs eine weiblichen Autorin verteilen. Der erste Blog handelt über ihre Kindheit auf einem Ölfeld und ihrer Erfahrung mit der Schule. Der zweite Blog handelt über ihre eigene Familie, und den Streit der ausbricht als ein neues Auto gekauft werden soll. Ihre umweltbewusste Tochter möchte kein neues Auto, und schon gar kein Sports Utility Vehicle (SUV).

Wenn man sich die verschiedenen Klassen des Testsets anschaut nachdem man die Referenzübersetzung gelesen hat, kann man diese größtenteils sauber zu einem der beiden Blogs zuordnen. Dies trifft zumindest auf vier der sechs Klassen zu. Von den letzten beiden Klassen scheint eine eher die Schlüsselwörter zu haben, die in beiden Blogs häufig vorkamen. Die andere scheint dann die Auffangklasse zu sein, die alles auffängt was in keine der anderen Klassen gepasst hat. Insgesamt hat man also durchaus das Gefühl, das die Klassifizierung gut funktioniert hat.

Bei der anderen Übersetzung auf den Zeitungsartikeln, tut sich schon eher ein Problem auf. Die Zeitungsartikel der Trainingsdaten sind einfach zu sehr auf China fokussiert, während die Testdaten eher die typischen Weltnachrichten abdecken. Da wären z.B. Artikel über amerikanische Wissenschaftler die Nord Korea untersuchen, amerikanische Soldaten im Irak, Präsidenten und Revolutionen in kleinen Ländern und die Beagle 2 Marserkundung. Die Suche nach besseren Trainingsdaten ist allerdings nicht so einfach. Zum Einen braucht man Trainingsdaten, die Dokumentinformation beinhalten, und zum Anderen eine eindeutige Zuordnung von Dokumenten zwischen den Sprachen.

A.7 Fazit & Zukünftige Arbeit

Diese Diplomarbeit wurde mit der Einstellung begonnen, dass es eher einfach sein sollte Verbesserungen in der SMT durch bessere Sprachmodellierung zu bekommen. Die meisten SMT Systeme benutzen immer noch ein einfaches Trigram Sprachmodell, und gibt soviel Arbeit zu Sprachmodellen in der Spracherkennung, die bis jetzt nicht auf die SMT angewendet wurde.

Nach einer Analyse der vorhandenen Sprachmodellierungstechniken wurden drei Techniken ausgewählt, um in dieser Diplomarbeit hinsichtlich ihrer Nützlichkeit für die maschinelle Übersetzung zu untersuchen. Diese Ansätze sind das klassenbasierte Sprachmodell, das Cache Sprachmodell und das Satz Mixtur Sprachmodell. Bei der Auswahl wurde inbesondere auf die Einbeziehung weitreichender Kontextbeziehungen in die Sprachmodellierung geachtet.

Bereits bei der Literatur Analyse wurde klar, dass es nicht einfach werden würde, mit Sprachmodellierungstechniken Verbesserungen für die maschinellen Übersetzung zu erzeugen. Die Arbeit die am meisten Einfluss auf diese Diplomarbeit hatte kommt zu dem Schluß, dass

> "We point out that trigrams remain the de facto standard not because we don't know how to beat them, but because no improvements justify the cost" (Goodman, 2001)
> "Wir zeigen dass Trigramme der de facto Standard bleiben, nicht weil wir nicht wissen wie wir besser werden könnten, aber weil keine Verbesserungen den Aufwand wert sind" Übersetzung des Verfassers

Dennoch war die Untersuchung weiterer Sprachmodellierungstechniken für die SMT immer noch eine interessante Forschungsfrage. Im Endeffekt konnte jedoch keiner der drei verfolgten Ansätze eine Verbesserung erzeugen.

Das klassenbasierte Sprachmodell konnte in keinem Experiment eine Verbesserung erzielen. Dies mag auf den eher einfachen Ansatz des hier realisierten klassenbasierten Sprachmodells zurückzuführen sein. Die damit erreichten Ergebnisse ließen allerdings auch bessere Umsetzungen wenig aussichtsreich erscheinen.

Das Cache Sprachmodell erzielte manchmal kleine Verbesserungen, oft verschlechterte es aber auch die Leistung. Daher erscheint es vernünftig anzunehmen, dass das Cache Modell nur für bestimmte Testsets nützlich ist. Es ist allerdings eine offene Frage, wann ein Cache Modell für ein Testset sinnvoll ist.

Der Satz Mixtur Modell Ansatz ist in mancherlei Hinsicht anders als die anderen beiden Ansätze. Zuallererst war es nicht einfach die Anwendung eines Konzepts, wie es bereits für die Spracherkennung verwendet wurde. Zusätzlich zu dem vorhandenen Konzept, wurden hier noch zwei Veränderungen vorgenommen, die speziell auf die SMT angepasst sind. Zum Einen wurde die Klassifikation von manchen Testsets in der Quellsprache vorgenommen. Das hat den Vorteil, dass vollkommen korrekte Information für die Klassifikation verwendet wird, im Gegensatz zu einer fehlerbehafteten first pass Übersetzung. Zum Zweiten wurde nicht nur das Sprachmodell durch die Klassifikation angepasst, sondern das komplette Testset wurde klassifiziert, und für jede dieser Klassen wurden alle Übersetzungsparameter optimiert. Mit diesen beiden Veränderungen erschienen Satz Mixtur Modelle äußerst vielversprechend für die maschinelle Übersetzung. Erste Ergebnisse auf Entwicklungsdaten konnten das noch bestätigen, am Ende wurde jedoch klar, dass alle Verbesserungen nur durch Überanpassung auf die Entwicklungsdaten erzeugt wurde.

Zusammenfassend kommt diese Diplomarbeit zu einem ähnlichen Schluss wie Goodman, dessen Arbeit oben zitiert wurde. Wie für die Spracherkennung, bleiben auch für die SMT n-gram Sprachmodelle der Standard. Diese Arbeit hat weitere Beweise dafür geliefert, dass auch wenn intuitive Annahmen vermuten lassen würden, dass es eher einfach sein sollte bessere Sprachmodelle als n-gramme zu bekommen, dies nicht der Fall ist. Mehrere Ansätze wurden in dieser Diplomarbeit verfolgt, keiner konnte eine Verbesserung auf ungesehenen Testdaten erzeugen. Die Schlussfolgerung ist dementsprechend, dass es zumindest schwer werden dürfte mit einfachen Wortabhängigkeitsmodellen eine Verbesserung zu erzielen. Damit sind Ansätze gemeint, die kein weiteres linguistisches Wissen in die Entscheidungsfindung einfließen lassen.

Es bleibt eine offene Frage, ob Sprachmodelle mit mehr linguistischem Wissen wie zum Beispiel strukturierte Sprachmodelle (Chelba, 1997) helfen können.

Appendix B

Changes to the SRI Toolkit

B.1 CacheLM.h

```
/*
 * CacheLM.h
 * Unigram cache language model.
 *
 * The model keeps track of the last N words (across sentences)
 * and computes a maximum likelihood unigram distribution from them.
 * (This is typically used in interpolating with other LMs.)
 *
 * Copyright (c) 1995, SRI International.  All Rights Reserved.
 *
 * @(#)$Header: /export/d/stolcke/project/srilm/src/RCS/CacheLM.h,v 1.1 1995/08/23 21:57:27 stolcke Exp $
 *
 */

#ifndef _CacheLM_h_
#define _CacheLM_h_

#include "LM.h"
#include "LHash.h"
#include "Array.h"
#include <map>

class BigramIndex{

    public:
    int a;
    int b;
    BigramIndex(){
    }
    BigramIndex(int x, int y){
        a = x;
        b = y;
    }
    void set(int x, int y){
        a = x;
        b = y;
    }
    void get(int *words){
       words[0] = a;
       words[1] = b;
    }
    bool operator== (BigramIndex two) const {
        if( (a == two.a) && ( b == two.b ) ) return 1;
        else return 0;
    }
    bool operator= (BigramIndex two) {
        a = two.a;
        b = two.b;
        return 1;
    }
    bool operator< (BigramIndex two) const {
        if ( (a < two.a ) ) return 1;
        if ( (a > two.a ) ) return 0;
        if ( (a == two.a ) && ( b < two.b ) ) return 1;
        if ( (a == two.a ) && ( b > two.b ) ) return 0;
      return 0;
    }
};

class TrigramIndex{

    public:
    int a;
    int b;
    int c;
    TrigramIndex(){
    }
    TrigramIndex(int x, int y, int z){
        a = x;
        b = y;
        c = z;
    }
    void set(int x, int y, int z){
        a = x;
        b = y;
        c = z;
    }
   void get(int *words){
        words[0] = a;
        words[1] = b;
        words[2] = c;
   }
    bool operator== (TrigramIndex two) const {
        if( (a == two.a) && ( b == two.b ) && ( c == two.c ) ) return 1;
        else return 0;
    }
    bool operator= (TrigramIndex two) {
        a = two.a;
        b = two.b;
        c = two.c;
        return 1;
    }
    bool operator< (TrigramIndex two) const {
        if ( (a < two.a ) ) return 1;
        if ( (a > two.a ) ) return 0;
        if ( (a == two.a ) && ( b < two.b ) ) return 1;
        if ( (a == two.a ) && ( b > two.b ) ) return 0;
        if ( ( a== two.a ) && ( b == two.b ) && ( c < two.c ) ) return 1;
        return 0;
    }
};


class CacheLM: public LM
{
    public:
    CacheLM(Vocab &vocab, unsigned historyLength);
    ~CacheLM();

    unsigned historyLength; /* context length used for cacheLength */

    /*
     * LM interface
     */
    virtual LogP wordProb(VocabIndex word, const VocabIndex *context);
    void flushCache(); /* forget history */
    void setWeights(double word, double bi, double tri);    /*sum should be one */
    virtual Boolean running(Boolean newstate);
    virtual Boolean running();

protected:

    double lambda_w; /* weight of unigram cache */
    double lambda_b; /* weight of bigram cache */
    double lambda_t; /* weight of trigram cache */
    double totalCount; /* total number of words in history */
    unsigned historyEnd; /* index into wordHistory */
    Array<VocabIndex> wordHistory; /* the last historyLength words */
    LHash<VocabIndex,double> wordCounts;/* (fractional) word counts */
    Array<BigramIndex> biHistory; /* the last history length bigrams */
    map<BigramIndex,double> biCounts;   /* count the bigram occurences */
    Array<TrigramIndex> triHistory; /* the last history length trigrams */
    map<TrigramIndex,double> triCounts;   /* count the trigram occurences */
    bool update; /* writing to cache yes or no */
};

#endif /* _CacheLM_h_ */
```

B.2 CacheLM.cc

```
/*
 * CacheLM.cc --
 * Unigram cache language model
 *
 */

#ifndef lint
static char Copyright[] = "Copyright (c) 1995, SRI International.  All Rights Reserved.";
static char RcsId[] = "@(#)$Header: /home/srilm/devel/lm/src/RCS/CacheLM.cc,v 1.7 2006/01/05 20:21:27 stolcke Exp $";
#endif

#include <iostream>
using namespace std;
#include <stdlib.h>
#include <math.h>
#include <map>

#include "CacheLM.h"

#include "Array.cc"
#ifdef INSTANTIATE_TEMPLATES
// INSTANTIATE_ARRAY(VocabIndex);
#endif

#include "LHash.cc"
#ifdef INSTANTIATE_TEMPLATES
INSTANTIATE_LHASH(VocabIndex,double);
#endif

/*
 * Debug levels used
 */
#define DEBUG_CACHE_HITS 2

CacheLM::CacheLM(Vocab &vocab, unsigned historyLength)
    : LM(vocab), historyLength(historyLength),
      wordHistory(0, historyLength), wordCounts(0)
{
    flushCache();
    update = 1;
    lambda_w = 1.0;
    lambda_b = 0.0;
    lambda_t = 0.0;
}

~CacheLM::CacheLM()
{
flushCache();
}

/*
 * Forget all that is in the cache
 */
void
CacheLM::flushCache()
{

    /*
```

```
     * Initialize word history.
     */
    for (unsigned i = 0; i < historyLength; i++) {
       wordHistory[i] = Vocab_None;
    }
    historyEnd = 0;
    totalCount = 0.0;

    /*
     * Reset word counts to zero
     */
    LHashIter<VocabIndex,double> wordIter(wordCounts);
    VocabIndex word;
    double *wordCount;

    while (wordCount = wordIter.next(word)) {
      *wordCount = 0.0;
    }
    /*
     * Reset bigram counts to zero
     */
    map<BigramIndex, double>::iterator biIter = biCounts.begin();
    map<BigramIndex, double>::iterator biLast = biCounts.end();
    BigramIndex bi;
    double *biCount;

    while (biIter != biLast) {
      biCount = &biCounts[(*biIter).first];
      *biCount = 0.0;
      biIter++;
    }
    /*
     * Reset trigram counts to zero
     */
    map<TrigramIndex, double>::iterator triIter = triCounts.begin();
    map<TrigramIndex, double>::iterator triLast = triCounts.end();
    TrigramIndex tri;
    double *triCount;

    while (triIter != triLast) {
      triCount = &triCounts[(*triIter).first];
      *triCount = 0.0;
      triIter++;
    }
}

LogP
CacheLM::wordProb(VocabIndex word, const VocabIndex *context)
{
    // how much context is given
    unsigned int clen = Vocab::length(context);
    int cWords = 0;
    if(clen > 2) cWords = 2;
    else cWords = clen;
```

```
// generate trigram string
/*char trigramString[500];
sprintf(trigramString, "%s ", vocab.getWord(word));
for(int i = 0; i < cWords; i++) {
  sprintf(trigramString, "%s %s ", trigramString, vocab.getWord(context[i]));
}
if(update) cout << "Trigram updated " <<trigramString << endl;*/

// generate bigram index class
BigramIndex bigram;
bigram.set(word,context[0]);

// generate trigram index class
TrigramIndex trigram;
if(cWords < 2) trigram.set(word,context[0],0);
else trigram.set(word, context[0], context[1]);

/*
 * We don't cache unknown words unless <unk> is treated as a regular word.
 */
if ( ( word ==  vocab.unkIndex()) && !(vocab.unkIsWord()) ) {
  return LogP_Zero;
}

/*
 * Return the maximum likelihood estimate based on all words
 * in the history.  Return prob 0 for the very first word.
 */
double *wordCount = wordCounts.insert(word);

if( biCounts.find(bigram) == biCounts.end() ) biCounts[bigram] = 0.0;
double *biCount = &biCounts[bigram];

if( triCounts.find(trigram) == triCounts.end() ) triCounts[trigram] = 0.0;
double *triCount = &triCounts[trigram];

// calculate the probability
Prob allProb;
Prob wordProb = 0.0;
Prob biProb = 0.0;
Prob triProb = 0.0;
if(totalCount == 0.0) allProb = 0.0;
else {
  wordProb    = lambda_w * (*wordCount / totalCount);
  biProb     = lambda_b * (*biCount / totalCount);
  triProb     = lambda_t * (*triCount / totalCount);

  // perform interpolation for allProb here
  if(triProb != 0.0) allProb =  wordProb + triProb + biProb;
  else {
    if(biProb != 0.0) allProb = wordProb + biProb;
    else allProb = wordProb;
  }
  // ensure no probability > 1
  if(allProb > 1) allProb = 1;
```

```
    }

    if (running() && debug(DEBUG_CACHE_HITS)) {
      dout() << "[cache=" << allProb << "]";
      dout() << "[wordCache=" << wordProb << "]";
      dout() << "[biCache=" << biProb << "]";
      dout() << "[triCache=" << triProb << "]";
    }

    /*
     * Update history and counts
     */
    if ((update) && (historyLength > 0)) {
      VocabIndex oldWord = wordHistory[historyEnd];
      BigramIndex oldBigram = biHistory[historyEnd];
      TrigramIndex oldTrigram = triHistory[historyEnd];
      if (oldWord == Vocab_None) {
        totalCount ++;
      } else {
        double *oldWordCount    = wordCounts.find(oldWord);
        double *oldBiCount     = &biCounts[oldBigram];
        double *oldTriCount     = &triCounts[oldTrigram];
        assert(oldWordCount != 0);
        assert(oldBiCount != 0);
        assert(oldTriCount != 0);

        *oldWordCount -= 1.0;
        *oldBiCount -= 1.0;
        *oldTriCount -= 1.0;
      }

      wordHistory[historyEnd] = word;
      biHistory[historyEnd] = bigram;
      triHistory[historyEnd] = trigram;
      *wordCount += 1.0;
      *biCount += 1.0;
      *triCount += 1.0;

      historyEnd = (historyEnd + 1) % historyLength;
    }

    return ProbToLogP(allProb);
}

void
CacheLM::setWeights(double word, double bi, double tri) {
    if ( (word + bi + tri) != 1.0 ) {
      cerr << "Warning : Cache weight factors don't sum to one \n";
      cerr << "Instead they sum to :" << (word + bi + tri) << endl;
    }
    lambda_w = word;
    lambda_b = bi;
    lambda_t = tri;

    cerr << "Cache Weights set to: w " << lambda_w << " b " << lambda_b << " t " << lambda_t << endl;
```

```
}

Boolean
CacheLM::running() {
    return update;
}

Boolean
CacheLM::running(Boolean newstate) {
    update = newstate;
    return update;
}
```

Appendix C

Changes to the LEMUR Toolkit

C.1 Offline Cluster main file

```
/*
  author: dmf
 */

using namespace lemur::api;
using namespace lemur::cluster;

// get application parameters
void GetAppParam() {
  ClusterParam::get();
}

int AppMain(int argc, char * argv[]) {
  if (argc > 2) {
    cerr << "Usage: OfflineCluster <parameter file>" << endl;
    return -1;
  }
  COUNT_T i;
  Index *myIndex;
  try {
    myIndex  = IndexManager::openIndex(ClusterParam::docIndex);
  } catch (Exception &ex) {
    ex.writeMessage();
    throw Exception("OfflineCluster",
                    "Can’t open index, check parameter index");
  }
  // construct cluster method.
  lemur::cluster::OfflineCluster* clusterDB;
  clusterDB = new lemur::cluster::OfflineCluster(*myIndex,
                                               ClusterParam::simType,
                                               ClusterParam::clusterType,
                                               ClusterParam::docMode);
  // crank through the collection
  COUNT_T numDocs = myIndex->docCount();
  if (numDocs > 10000) numDocs = 10000;
  if (numDocs > ClusterParam::numToCluster) numDocs = ClusterParam::numToCluster;
  vector <DOCID_T> toCluster;
  for (i = 1; i <= numDocs; i++) {
    toCluster.push_back(i);
  }

  vector <lemur::cluster::Cluster *> *clusters;
  if(ClusterParam::clusterAlg == "kMeans") {
      // generate k-means cluster
      cout << "Using kmeans on " << numDocs << " documents..." << endl;
      clusters = clusterDB->kMeans(toCluster, ClusterParam::numParts,
   ClusterParam::maxIters);
  }
  else if (ClusterParam::clusterAlg == "bkMeans") {
      // generate bisecting k-means cluster
      cout << "Using bisecting kmeans on " << numDocs << " documents..." << endl;
      clusters = clusterDB->bisecting_kMeans(toCluster, ClusterParam::numParts,
    ClusterParam::numIters,
    ClusterParam::maxIters);
  } else {
      // add additional clusterAlgs here
      cout << "Clustering algorithm not supported\n";
      exit(1);
  }


  // set up storing database MARTIN 8/14/2006
  lemur::api::ClusterDB* clusterSaveDB;
  if (ClusterParam::clusterDBType == "keyfile") {
      /*clusterSaveDB = new lemur::cluster::KeyfileClusterDB(myIndex,
       ClusterParam::clusterIndex,
       ClusterParam::threshold,
       ClusterParam::simType,
       ClusterParam::clusterType,
       ClusterParam::docMode);
      */
      cout << "The keyfile cluster DB is not supported, use flatfile instead" << endl;
      cout << "Problem is control over when information is stored in file" << endl;
      exit(1);
  } else if (ClusterParam::clusterDBType == "flatfile") {
      clusterSaveDB = new lemur::cluster::FlatFileClusterDB(myIndex,
ClusterParam::clusterIndex,
ClusterParam::threshold,
ClusterParam::simType,
ClusterParam::clusterType,
ClusterParam::docMode);
  } else {
      // add additional db types here. Convert to Factory usage.
cout << "Cluster database type '" << ClusterParam::clusterDBType
     << "' is not supported.\n";
exit(1);
  }

  // save - add generated clusters to ClusterDB
  cout << "Writing generated clusters to file " << ClusterParam::clusterIndex << endl;
  for (i = 0; i < clusters->size(); i++) {
      clusterSaveDB->addCluster( (*clusters)[i] );
    //(*clusters)[i]->print();
    delete((*clusters)[i]);
  }
  delete(clusters);

  // write to file and to screen
  ((FlatFileClusterDB*) clusterSaveDB)->writeClusterDB();
  clusterSaveDB->printClusters();
  delete clusterSaveDB;

 // END MARTIN
  delete clusterDB;
  delete myIndex;
  return 0;
}
```

C.2 Cluster main file

```
/*
  author: dmf
 */

// get application parameters
void GetAppParam() {
  ClusterParam::get();
}

int AppMain(int argc, char * argv[]) {
  if (argc > 2) {
    return -1;
  }

  // load index with documents
  Index *docIndex;
  try {
    docIndex  = IndexManager::openIndex(ClusterParam::docIndex);
  } catch (Exception &ex) {
    ex.writeMessage();
    throw Exception("Cluster", "Can’t open docIndex");
  }

  try {
    // construct cluster method.
    lemur::api::ClusterDB* clusterDB;
    if (ClusterParam::clusterDBType == "keyfile") {
      /*clusterDB = new lemur::cluster::KeyfileClusterDB(newDocIndex,
                                      ClusterParam::clusterIndex,
                                      ClusterParam::threshold,
                                      ClusterParam::simType,
                                      ClusterParam::clusterType,
                                      ClusterParam::docMode);
```

```
       */
cout << "KeyFile ClusterDB is not supported at the moment, use flatfile instead\n";
cout << "Problem is control over when writing to files\n";
exit(1);
    } else if (ClusterParam::clusterDBType == "flatfile") {
      clusterDB = new lemur::cluster::FlatFileClusterDB(
docIndex,
                                        ClusterParam::clusterIndex,
                                        ClusterParam::threshold,
                                        ClusterParam::simType,
                                        ClusterParam::clusterType,
                                        ClusterParam::docMode);
    } else {
      // add additional db types here. Convert to Factory usage.
        cout << "Cluster database type '" << ClusterParam::clusterDBType
             << "' is not supported.\n";
        exit(1);
    }

    // check if clusterDB got loaded
    cerr << clusterDB->countClusters() << " Clusters with " << clusterDB->getNumDocs() << " documents loaded successfully from C
    cerr << "Evaluating " << docIndex->docCount() << " documents " <<  endl;

    // output typical words of clusters
    for(int i= 1; i < clusterDB->countClusters()+1; i++) {
       cerr << "Cluster " << i << " KeyWords " << endl;
       cerr << clusterDB->getKeyWords(i, 30) << endl;
    }

    // crank through the collection
    double score;
    for (DOCID_T i = 1; i <= docIndex->docCount(); i++) {
      //if (clusterDB->getDocClusterId(i).size() == 0) {
        // Hasn't been clustered yet.
        int myCluster = clusterDB->cluster(i, score, ClusterParam::update, docIndex);
        cout << docIndex->document(i) << " " << myCluster << " "
             << score << endl;
      //}
    }
    delete clusterDB;
  } catch(lemur::api::ClusterDBError x) {
    cout << "Problem with Clustering Database: " << x.what() << "\n";
  }
  delete docIndex;
  return 0;
}
```

C.3 ClusterDB.cpp

```
int lemur::api::ClusterDB::cluster(lemur::api::DOCID_T docId,
                                   double &finalScore,
   bool update = 0,
   Index* newDocIndex = NULL) {
```

```
  if (newDocIndex == NULL) {
      cout << "There seem to be no new documents" << endl;
      exit(1);
  }
  lemur::cluster::Cluster *cluster;
  int cid, myCluster;
  double score;
  TermInfoList *tList = newDocIndex->termInfoList(docId);
  lemur::cluster::ClusterRep *docRep = new lemur::cluster::ClusterRep(tList,
                                                                       *newDocIndex);
  sim->weigh(docRep);
  int numClusters = countClusters();
  int maxId = maxID();
  IndexedRealVector results(numClusters);
  results.clear();

  for (int i = 1; i <= maxId; i++) {
    cluster = getCluster(i);
    if (cluster)
      results.PushValue(i, cluster->score(docRep));
  }
  if (results.size() == 0) {
    cluster = newCluster();
    score = 1.0;
  } else {
    results.Sort();
    myCluster = results[0].ind;
    score = results[0].val;
    cluster = getCluster(myCluster);
    if (! thresh->threshold(score, cluster->getSize())) {
      // new cluster
      cluster = newCluster();
      score = 1.0;
    }
  }
  // whether to add document to cluster or not
  if(update) {
    addToCluster(docId, cluster, score);
  }
  finalScore = score;
  cid = cluster->getId();
  delete (docRep);
  delete(tList);
  return cid;
}
```

Appendix D

StopWords for Sentence Mixture Model

1 2 3 4 5 6 7 8 9 10 a about above across after afterwards again against all almost alone along already also although always am among amongst amoungst amount an and another any anyhow anyone anything anyway anywhere are around as at back be became because become becomes becoming been before beforehand behind being below beside besides between beyond bill both bottom but by call can cannot cant co computer con could couldnt cry de describe detail do done down due during each eg eight either eleven else elsewhere empty enough etc even ever every everyone everything everywhere except few fifteen fify fill find fire first five for former formerly forty found four from front full further get give go had has hasnt have he hence her here hereafter hereby herein hereupon hers herself him himself his how however hundred i ie if in inc indeed interest into is it its itself keep last latter latterly least less ltd made many may me meanwhile might mill mine more moreover most mostly move much must my myself name namely neither never nevertheless next nine no nobody none noone nor not nothing now nowhere of off often on once one only onto or other others otherwise our ours ourselves out over own part per perhaps please put rather re said same see seem seemed seeming seems serious several she should show side since sincere six sixty so some somehow someone something sometime sometimes somewhere still such system take ten than that the their them themselves then thence there thereafter thereby therefore therein thereupon these they thick thin third this those though three through throughout thru thus to together too top toward towards twelve twenty two un under until up upon us very via was we well were what whatever when whence whenever where whereafter whereas whereby wherein whereupon

wherever whether which while whither who whoever whole whom whose why will with within without would yet you your yours yourself yourselves

List of Figures

List of Tables

Bibliography

Arnold, D., Balkan, L., Meijer, S., Humphreys, R. L., and Sadler, L. (1994). *Machine Translation: an Introductory Guide.* Blackwells NCC, London. 1.1

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: parameter estimation. *Comput. Linguist.*, 19(2):263–311. (document), 1.2.1, A.1, A.2

Callan, J. and Croft, B. (2006). Lemur toolkit. http://www.lemurproject.org. 2.7, 3.3.5, 4.5.7

Chelba, C. (1997). A structured language model. In Cohen, P. R. and Wahlster, W., editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 498–503, Somerset, New Jersey. Association for Computational Linguistics. 6.1, A.7

Chelba, C. and Jelinek, F. (1998). Refinement of a structured language model. 6.1

Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. In Joshi, A. and Palmer, M., editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, San Francisco. Morgan Kaufmann Publishers. 2.1.2, 2.2, A.3

Eck, M., Vogel, S., and Waibel, A. (2004). Language model adaptation for statistical machine translation based on information retrieval. In *Proceedings of LREC 2004*, Lisbon, Portugal. LREC. 2.7

Goodman, J. (2001). A bit of progress in language modeling: Extended version. Technical report, Microsoft Research, 56 Fuchun Peng. 2, 2.1.2, 2.3, 2.4, 2.5, 2.6, 2.8, 6.1, A.3, A.3, A.3, A.3, A.7

Hildebrand, A. S. (2005). Translation model adaptation for statistical machine translation using information retrieval. Master's thesis, Carnegie Mellon University, Pittsburgh. Diplomarbeit. 2.7

Iyer, R. and Ostendorf, M. (1999). Modeling long-distance dependence in language: Topic mixtures versus dynamic cache models. *IEEE Transactions on Acoustics, Speech and Audio Processing*, 7(1):30–37. 2.6, 6.2.2, A.3, A.5

Knight, K. (1999). A statistical mt tutorial workbook. In *Johns Hopkins University Workshop on Language Engineering*. 2.1.3, A.3

Kuhn, R. and de Mori, R. (1990). A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583. 2.4, A.3

Mahajan, M., Beeferman, D., and Huang, X. (1999). Improved topic-dependent language using information retrieval techniques. In *Proc. of the Int. Conf. on Acoustics, Speech, and Signal Processing*, Phoenix. 2.7

Manber, U. and Myers, G. (1993). Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22:935–948. 2.2

McCallum, A. K. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/ mccallum/bow. 3.3.5

Nepveu, L., Lapalme, G., and Foster, G. (2004). Adaptive language and translation models for interactive machine translation. In *Proceedings of the EMNLP, NRC 48083*, Barcelona, Spain. 2.4, 2.4, A.3

Niesler, T., Whittaker, E., and Woodland, P. (1998). Comparison of part-of-speech and automatically derived category-based language models for speech recognition. 2.3, A.3

NIST (2003). Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. Technical report, NIST, http://www.nist.gov/speech/tests/mt/doc/ngram-study.pdf. 1.2.1, A.2

Och, F. J. (1995). Maximum-likelihood-schätzung von wortkategorien mit verfahren der kombinatorischen optimierung. Master's thesis, Univ. of Erlangen-Nürnberg. Studienarbeit. 2.3, 3.3.4, 4.1.1, 6.1, A.3, A.5

Och, F. J. (1999). An efficient method for determining bilingual word classes. In *EACL*, pages 71–76. 2.3, A.3

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *ACL*, pages 160–167. 3.3.2

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51. 4.3.5

Papineni, K., Roukos, S., Ward, T., and Zhu, W. (2001). Bleu: a method for automatic evaluation of machine translation. Technical Report RC22176(W0109-022), IBM Research Division, Thomas J. Watson Research Center. 1.2.1, A.2

Rosenfeld, R. (2000). Two decades of statistical language modeling: Where do we go from here? 2.4, 3.3.6

Schein, A., Popescul, A., Ungar, L., and Pennock, D. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002)*, pages 253–260. 3.3.5

Steinbach, M., Karypis, G., and Kumar, V. (2000). A comparison of document clustering techniques. 3.3.5, 3.3.5, 3.3.5

Stolcke, A. (2002). Srilm – an extensible language modeling toolkit. *Proc. Intl. Conf. on Spoken Language Processing*, 12:901–904. 2.1.3, 2.3, 3.3.1, A.3, A.3

Vaiciunas, A. and Raskinis, G. (2006). Cache-based statistical language models of english and highly inflected lithuanian. *Informatica*, 17(1):111–124. 2.4, A.3

Vogel, S. (2005). Pesa: Phrase pair extraction as sentence splitting. In *Proceedings of MT Summit X*, Phuket, Thailand. 1.2.1, A.2

Weaver, W. (1955). Translation. In *Machine Translation of Languages*, MIT Press. (document)

Wu, J. and Khudanpur, S. (1999). Combining nonlocal, syntactic and N-gram dependencies in language modeling. 2.6, A.3

Zens, R. and Ney, H. (2006). N-gram posterior probabilities for statistical machine translation. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 72–77, New York City. Association for Computational Linguistics. 3.3.3

Zhang, J., Sun, L., Qu, W., Du, L., and Sun, Y. (2004). A three level cache-based adaptive chinese language model. In Su, K.-Y., ichi Tsujii, J., Lee, J.-H., and Kwong, O. Y., editors, *IJCNLP*, volume 3248 of *Lecture Notes in Computer Science*, pages 487–492. Springer. 2.4, A.3

Zhang, Y., Hildebrand, A. S., and Vogel, S. (2006). Distributed language modeling for n-best list re-ranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 216–223, Sydney, Australia. Association for Computational Linguistics. 2.2

Zhang, Y., Vogel, S., and Waibel, A. (2003). Integrated phrase segmentation and alignment algorithm for statistical machine translation. In *Proceedings of International Conference on Natural Language Processing and Knowledge Engineering (NLP-KE'03)*, Beijing, China. 1.2.1, A.2

Zhao, B., Eck, M., and Vogel, S. (2004). Language model adaptation for statistical machine translation via structured query models. In *Proceedings of Coling 2004*, pages 411–417, Geneva, Switzerland. COLING. 3.3.6

Zhao, B. and Vogel, S. (2005). A generalized alignment-free phrase extraction. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, Michigan, USA. ACL. 4.3.5

Zhao, B. and Waibel, A. (2005). Learning a log-linear model with bilingual phrase-pair features for statistical machine translation. In *Proceedings of the SigHan Workshop*, Jeju, Korea. 4.3.5