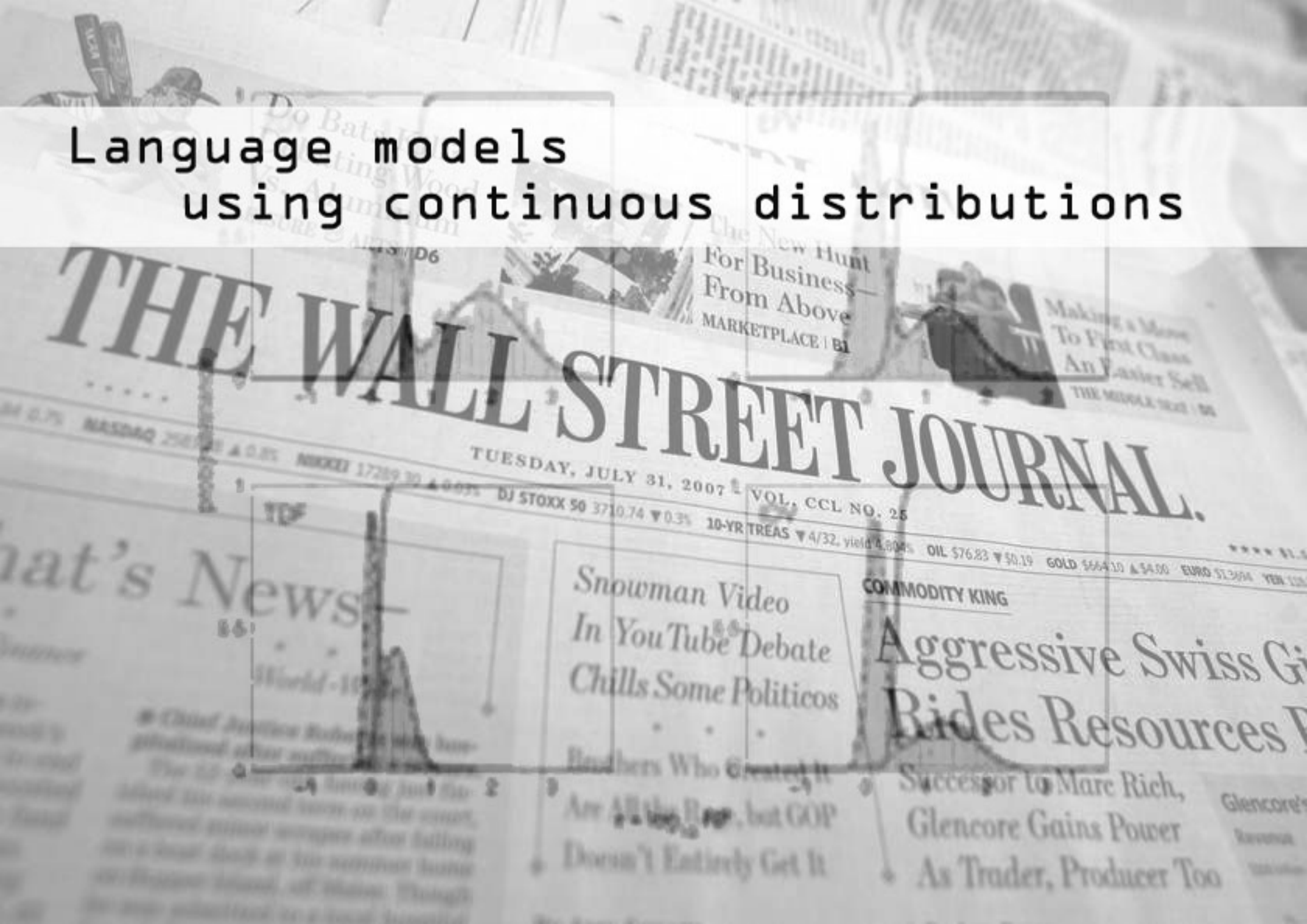
Language models using continuous distributions



Language Models using Continuous Distributions

Tsiakas Konstantinos
Diploma Thesis, Supervisor Professor: Prof.Vassilis Digalakis

Technical University of Crete
Chania, October 2012

Acknowledgments

First and foremost I offer my sincerest gratitude to my supervisor, Dr Vassilis Digalakis, who has supported me throughout my thesis with his patience and knowledge whilst allowing me the room to work in my own way. He introduced me to this specific area of study, which I am sure that I will follow. I would also like to thank Vassilis Diakoloukas, whose support and knowledge was very helpful and essential for this project. I would also like to thank my friends and family for their valuable emotional support. Without them I would not have been able to cope with all this.

Tsiakas Konstantinos

Contents

Table of Figures and Tables

Chapter 1

Introduction

The research on language modeling is referred to computational techniques and structures that describe word sequences as they are produced by humans. Such models can assign probabilities to words of a data and have become necessary tools for many researches.

Two basic approaches of natural language modeling can be defined. The first one refers to syntax and semantic analysis of text to determine the hierarchical structure of sentences. This approach uses a series of rules to parse a sentence is acceptable. Despite that this analysis can describe the structure of natural language at a significant degree, there are continuous changes on the language that occur and make this approach difficult. Moreover, some verbal formalities in a language are not grammatically proper; consequently they cannot be described by this analysis.

On the other hand, another approach, based on statistical techniques, is more robust to grammar abnormalities. Such *statistical language models* assign to each word in a sequence a probability according to the estimated likelihood given the word's context. These probabilities are estimated from a large text set, which we refer to as *train data set*. In this way, such models can model natural language as it is used in practice and not as it is defined by its theoretical grammar. The basic drawback of this approach is that the size of the available train sets, nowadays, fluctuates to hundreds of millions of words.

1.1. Natural Language Processing

Natural language processing (NLP) is a field of computer science, machine learning and linguistics concerned with the interactions between computers and human (natural) languages. Specifically, it is the process of a computer extracting meaningful information from natural language input and/or producing natural language output. In theory, natural language processing is a very attractive method of human–computer interaction.

Modern NLP algorithms are based on machine learning, especially statistical machine learning. Research into modern statistical NLP algorithms requires an understanding of a number of disparate fields, including linguistics, computer science, and statistics. For a discussion of the types of algorithms currently used in NLP, we should refer to pattern recognition.

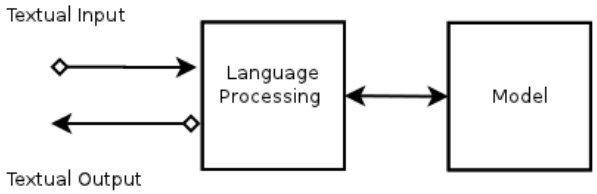


figure 1.1. Language Processing

The paradigm of machine learning is different from that of most prior attempts at language processing. Prior implementations of language-processing tasks typically involved the direct hand coding of large sets of rules. The machine-learning paradigm calls instead for using general learning algorithms — often, although not always, based on statistical inference — to automatically learn such rules through the analysis of large corpora of typical real-world examples. A corpus

(plural, "corpora") is a set of documents (or sometimes, individual sentences) that have been hand-annotated with the correct values to be learned.

Statistical natural-language processing uses stochastic, probabilistic and statistical methods to resolve some of the difficulties discussed above, especially those which arise because longer sentences are highly ambiguous when processed with realistic grammars, yielding thousands or millions of possible analyses. Methods for disambiguation often involve the use of corpora and Markov models. Statistical NLP comprises all quantitative approaches to automated language processing, including *probabilistic modeling*, *information theory*, and *linear algebra*. The technology for statistical NLP comes mainly from machine learning and data mining, both of which are fields of artificial intelligence that involve learning from data.

1.1.1. Statistical Modeling and Statistical Language Modeling

Statistical modeling or model building is an activity aimed at learning rules and restrictions in a set of observed data, proverbially called `laws'. The traditional approach is to imagine or assume that the data have been generated as a sample from a population, originally of a parametrically defined probability distribution and later, more generally, a so-called nonparametric distribution.

Statistical modeling is about finding general laws from observed data, which amounts to extracting information from the data. Despite the creation of information theory half a century ago with its formal measures of information, the entropy and the Kullback-Leibler distance or the relative entropy, there have been serious difficulties in applying them to make exact the idea of information extraction for model building. The main problem is that these measures refer to information in probability distributions rather than in data.

Statistical modeling can be applied in many applications such as natural language processing and language modeling using *language models*. As mentioned earlier, language models assign probabilities to words of a word sequence using stochastic models.

1.1.2. Language Model

Stochastic language models (SLM) take a probabilistic viewpoint of language modeling. We need to accurately estimate the probability P (**W**) for a given word sequence **W**= $w_1\ w_2\ ...\ w_n$. The key goal of SLM is to provide adequate probabilistic information so that the likely word sequences should have a higher probability. The most widely used SLM is the *n*-gram model. A language model can be formulated as a probability distribution P (**W**) over word strings **W** that reflects how frequently a string **W** occurs as a sentence.

Using the chain rule of probability P (**W**) can be decomposed as:

$$
\begin{aligned}
P(\mathbf{W}) &= P(w_1, w_2, \ldots, w_n) \\
&= P(w_1)\,P(w_2 \mid w_1)\,P(w_3 \mid w_1, w_2) \cdots P(w_n \mid w_1, w_2, \ldots, w_{n-1}) \\
&= \prod_{i=1}^{n} P(w_i \mid w_1, w_2, \ldots, w_{i-1})
\end{aligned}
$$

, where $P\ (w_i|w_1,w_2,\ ...\ ,w_{i-1})$ is the probability that w_i will follow, given that the word sequence was presented previously. The choice of w_i thus depends on the entire past history of the input. For a vocabulary of size V there are V^{i-1} different histories and so, to specify $P(w_i|w_1,w_2,\ ...\ ,w_{i-1})$ completely, V^i values would have to be estimated. In reality, the probabilities

$P(w_i|w_1,w_2, \ldots ,w_{i-1})$ are impossible to estimate for even moderate values of i , since most histories are unique or have occurred only a few times. A practical solution to the above problems is to make the Markovian assumption that $P(w_i|w_1, w_2, \ldots, w_{i-1})$ depends only on some equivalence classes. The equivalence class can be simply based on the several previous words. This leads to an N-*gram* language model.

If the word depends on the previous two words, we have a *trigram*: $P(w_i|w_{i-1},w_{i-2})$. Similarly, we can have *unigram:* $P(w_i)$, or *bigram:* $P(w_i|w_{i-1})$ language models.

The trigram is particularly powerful, as most words have a strong dependence on the previous two words, and it can be estimated reasonably well with an attainable corpus.

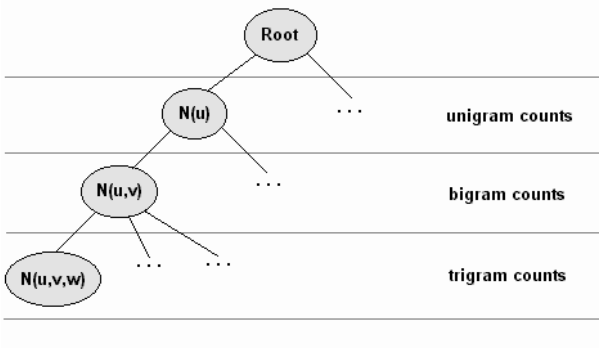


figure 1.2. Various N-grams

1.2. Purpose of the Thesis

In this work, we propose a continuous- space language model using continuous distributions, such as multivariate Gaussian distributions, Gaussian Mixture Models and their implementations, which are tied Gaussian Mixture Models and clustered mixture language models. We compare these continuous models with the discrete models. Two potential benefits of using this model are smoothing unseen events, and ease of adaptation. It is shown how this model can be used alone or interpolated with a conventional N-gram model to calculate word probabilities. An interesting feature of the proposed technique is that many methods developed for acoustic models can be easily ported to continuous- space models.

Firstly, we use the multivariate normal distribution to build our continuous model. We use Gaussian Mixture Models to improve our results. In order to overcome GMM training problems which are the large set of parameters and the potential data over- fitting, we build a language model based on Tied Gaussian Mixture Models, which are models with common set of parameters. Starting with a small vocabulary, we train these model parameters with statistical techniques and algorithms, such as the Expectation Maximization algorithm (EM).

Moreover, we give a suitable mapping method from the discrete word space to a continuous space and show how to model the resulting continuous vectors. Techniques and methods for mapping and dimensionality reduction from the field of linear algebra and Pattern Recognition have been used, such as Singular Value Decomposition and Linear Discriminant Analysis. We show the use of existing tools used for lower- dimensional space and GMM parameters estimation, such as SVDPACKC and HTK toolkit.

1.3. Outline of the Thesis

In the second Chapter, we present the dominant technology of language modeling, N-grams. We refer to their theory and their basic drawbacks and the smoothing methods that are used. We introduce the definition of the model quality presenting measures such as entropy and perplexity, based on the Information Theory. In the third Chapter we analyze the continuous space language models and their theory. We continue with the techniques that have been used for the word mapping and the projection to a lower space and the tools that have been used. In the fourth Chapter, we describe the data corpus and the SRILM toolkit, used for N-gram implementation. Then, there is a full description of the model parameters estimation and the model adaptation along with the continuous space models algorithm. The summary of results and the conclusion are on the fifth chapter with some directions for future work.

Chapter 2

Language Modeling Technology

2.1. Discrete Statistical Language Models

As mentioned before, statistical language models use statistical techniques to estimate models from text data sets by assigning probabilities in each word of the data. One of the most dominant technologies is N-gram models. N-gram models regard each word as a discrete variable. They are significant for many applications such as speech recognition, optical character recognition, machine translation, even dictation correction. Generally, the N-gram model has good results, when there is a satisfying set of data for a specific task.

2.2. N-grams description and drawbacks

Even simple models can affect the performance of application that they are used from (i.e. a speech recognizer). However, language models are very sensitive to train data topic alternation. For example, building a model of daily telephone conversations can be more precise if 2 millions of such recordings have been used than 140 millions of television and radio recordings. This affection is so strong even for changes that are essential for human perspective. A well- built language model that has been trained with the Wall Street Journal corpus will not be precise if it is applied to a scientific or sports corpus.

When train data differs from test data, the quality of the model will not be as satisfying. Even on large data sets, there will be many N-grams with no appearance. It is expected that many accepted N-grams may never appear on a train data set. Also, if relative frequencies are used for probability estimation, insufficient estimations can be obtained for small N-gram appearances. It is necessary to smooth N-gram probabilities in order to have normalized distributions. Many smoothing techniques have been developed to overcome this drawback. These techniques are applied to smooth N-gram probabilities, so as any N-gram that has not appeared does not have a zero probability.

To estimate the above probabilities we use large amounts of text, called training data, and we count the occurrences of words, bigrams and trigrams. That training data must be derived from the domain we want to estimate the model accuracy or to apply speech recognition, in order to reflex the particular style. A standard task for the English language is the Wall Street Journal corpus.

Assuming a bigram model, we estimate $p\ (w_i|w_{i-1})$, by counting the number of occurrences of the bigram w_{i-1}, w_i and normalize

$$p(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} = \frac{c(w_{i-1}, w_i)}{\sum_{w_i} c(w_{i-1}, w_i)}$$

This is called the *maximum likelihood* (ML) estimate, because it maximizes the probability of occurrence of the training data. It is understandable that if a bigram does not appear in our train data the probability will be zero. This is a very undesirable situation, because that means that an acceptable string may never be assigned a non zero probability. To overcome this problem, we have to *smooth* the distributions taken from the maximum likelihood estimate. Smoothing

means to adjust the probabilities, to make them non zero. Smoothing does not affect only zero probabilities. It affects small probabilities as well, for example probabilities of bigrams occurring only once, the so-called *singletons* in the training data.

There are many smoothing techniques that are used to treat zero probabilities. N-gram models can be adjusted by these techniques. Some of the most know techniques are Good- Turing, add-one smoothing, Kneser - Ney technique and Katz's backing -off. Their description and their use in smoothing are well- known, so we will not expand to this topic.

2.3. Information Theory and Language Models quality

Language models are used to estimate word- sequence probabilities. For a word sequence of *N* words, *P (W)* has information about the sequence's probability and accuracy. We can decide on the quality of a model according to *P (W)* of each data sequence. *Entropy* and *perplexity* are two basic measures from the field of Information Theory that are use for evaluating the quality of a language model.

Language can be thought of as an information source whose outputs are words w_i belonging to the vocabulary of the language. The most common metric for evaluating a language model is the word recognition error rate, which requires the participation of a speech recognition system. Alternatively, we can measure the probability that the language model assigns to test word strings without involving speech recognition systems. This is the derivative measure of cross-entropy known as test-set *perplexity*.

2.3.1. Entropy

Given a language model that assigns probability $P(\mathbf{W})$ to a word sequence $\boldsymbol{W}$, we can derive a compression algorithm that encodes the text $\boldsymbol{W}$ using $-\log_2 P(\boldsymbol{W})$ bits. The cross-entropy $H(\boldsymbol{W})$ of a model $P(w_i|w_{i-n+1} \ldots w_{i-1})$ on data $\boldsymbol{W}$, with a sufficiently long word sequence, can be simply approximated as

$$H(\mathbf{W}) = -\frac{1}{N_{\mathbf{W}}} \log_2 P(\mathbf{W})$$

where $N_{\mathbf{W}}$ is the length of the text $\boldsymbol{W}$ measured in words.

2.3.2. Perplexity

The perplexity $PP(\mathbf{W})$ of a language model $P(\mathbf{W})$ is defined as the reciprocal of the (geometric) average probability assigned by the model to each word in the test set $\boldsymbol{W}$. This is a measure, related to cross-entropy, known as test-set perplexity

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

The perplexity can be roughly interpreted as the geometric mean of the branching factor of the text when presented to the language model. The perplexity defined has two key parameters: a language model and a word sequence. The test- set perplexity evaluates the generalization capability of the language model. The training- set perplexity measures how the language model fits the training data, like the likelihood. It is generally true that lower perplexity correlates with better language modeling. This is because the perplexity is essentially a statistically weighted word branching measure on the test set.

For each language model, it is possible to calculate the perplexity for test data. Perplexity minimum value is 1 that would mean that all words of a sequence have probability equal to 1. On the other hand, if a word has zero probability, then the probability of the whole sentence will be zero and perplexity will be infinite. So, we can assume that the challenge of a language model is to avoid zero probabilities. A well- built model should assign small perplexity for large test data sets. Perplexity value is a quality measure of different models for common test data.

Chapter 3

Continuous Space Language Models

3.1. Introduction

As covered before, N-gram models are the dominant technology in Language Modeling. In spite of their success, discrete N-gram models suffer from two basic drawbacks. We can refer to them as *generalization* and *adaptability*. These two problems refer to the N-grams with zero probability and to the parameters of N-gram model. Usually, an N-gram model has a huge number of parameters. Thus, it is very difficult to adapt it using a relatively small amount of data. We propose continuous space language models, in order to overcome these problems and have benefits such as smoothing unseen events, and ease of model adaptation. The basic idea of such models is the similarity of words, as they are projected in a continuous parameter space. That is, some words or N-grams are ”closer” to each other than other words or N-grams.

There are some important issues about these models. At first, we have to make an appropriate projection of each "discrete" word to the new continuous space. Then, we have to deal with the high- dimensional spaces. Modeling in large dimensions, mentioned as curse of dimensionality, is difficult. So, what we need to construct a continuous space LM is: a mapping from the discrete word space to a continuous representation, and a classifier which decides the next word given the mapped history in the resulting space, or equivalently assigns a probability on each word given its history.

3.2. Word mapping

Word mapping is the projection of each word of our data in the new continuous space. Concerning the large amount of words and the difficulty to process parameters in a high- dimensional space, we try to represent the most frequent words from our train data. Firstly, we find the *V* most frequent words, and we include them in our model vocabulary included a class to represent the other words as *unk* words. To determine the mapping, we should take into account the word's frequency and importance, but also its relation with the other words.

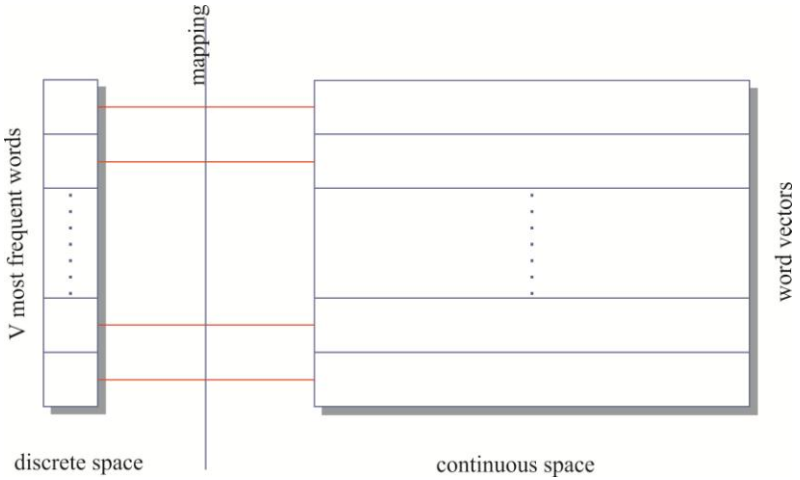


figure 3.1 word mapping

3.2.1. Word feature vectors

Each word can initially be represented by an indicator vector w_i, having one at the i^{th} position and zeros on the rest of the *V-1* positions. So , we have a *V*x*V* matrix, consisted of all the word vectors. This feature vector matrix is sparse and it has the following form:

w_1	1	0	0	0	0	…	0
w_2	0	1	0	0	0	…	0
w_3	0	0	1	0	0	…	0
w_4	0	0	0	1	0	...	0
w_5	0	0	0	0	1	…	0
...	...	...	...	...	...	...	...
w_v	0	0	0	0	0	...	1

figure 3.2 Indicator vectors

This matrix consists of V^2 elements. This means that as the size of the vocabulary rises, the size of the matrix rises exponentially. The next step is to map each vector to a lower dimension, concerning the word's frequency and how each word behaves with the others.

3.2.2. Word co-occurrences

As mentioned before, to make a suitable mapping of each word, we see how each word correlates with the rest. So we form a word co-occurrence matrix E that depicts this relation of the words. Each element e_{ij} is the number of times that word *i* follows word *j*, in the training data

E_{ij}	V_1	V_2	V_3	V_4	V_5	….	V_v
V_1	0	0	0	92	34	….	57
V_2	10	0	0	0	12	….	0
V_3	0	0	0	37	53	….	156
V_4	12	78	0	0	64	….	0
V_5	0	35	118	0	0	….	745
….	….	….	….	….	….	….	….
V_v	45	65	0	64	0	….	0

figure 3.3 Co-occurrence matrix

The co-occurrence matrix is a *V*x*V* matrix that has many zeros, because there are many words that do not appear together on the train data. Especially, in our data, the matrix E consists of 2700x27000 = 729 x 10^4 elements (with a vocabulary of V = 2700) and the 6971326 elements are zero, that is the 95.63**%** of the matrix. The count is then smoothed as $e_{ij} = \log(1+e_{ij})$. Practically, this matrix consists of each word's bigram history. Each row is the word's vector and the columns represent the histories. Concerning the large amount of the elements we perform a method for dimensionality reduction, which is called *Singular Value Decomposition* (SVD).

3.2.3. Singular Value Decomposition

The singular value decomposition of a matrix is one of the most elegant and powerful algorithms in linear algebra, and it has been extensively used for dimensionality reduction in pattern recognition and information retrieval applications.

Singular value decomposition transforms an *m* x *n* matrix A into a $A = Q_1 \Sigma Q_2$

- Q_1 is an m×m matrix with the eigenvalues of AA^T as its columns
- Q_2 is an n×n matrix with the eigenvectors of A^TA as its columns
- Σ is an m×n diagonal matrix consisting of the square roots of the eigenvalues of AA^T and A^TA (both matrices have the same eigenvalues but different eigenvectors)

Graphically:

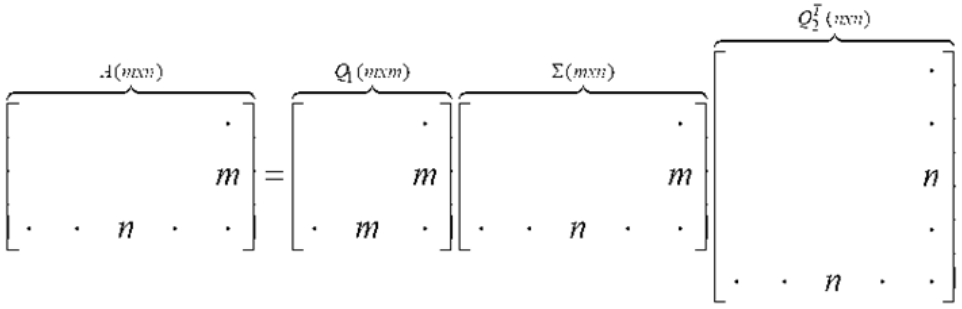


figure 3.4. SVD technique

Something we have to notice is matrix Σ, which is a diagonal matrix, consists of the singular values

$$\Sigma = \begin{bmatrix} \sigma_1 & & & & & & \\ & \sigma_2 & & & & & \\ & & \ddots & & & & 0 \\ & & & \sigma_r & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & 0 & & & & & 0 \end{bmatrix}$$

figure 3.5. singular value matrix

Being part of the matrix, singular values multiply only certain columns of Q_1 and Q_2. The singular values determine how these columns of Q_1 and Q_2 influence the matrix. If a value is small enough then its column or row is not added to the new matrix. Many of the values are zero. That means that the corresponding columns and rows have not useful information. So we can transform the matrix to another one that has the valuable information.

Often, singular values are sorted by the significance of their rows and columns. That is the reason that SVD is used for matrix decomposition and data reduction. If we want to maintain the basic information of a matrix by reducing its dimensionality, we can keep the M greatest singular values and build another matrix Σ', containing only these M singular values. So we can obtain a transformation matrix *M* x *N*.

As mentioned before, the word co- occurrence matrix is sparse, so we need to perform a suitable method of SVD to obtain the largest singular values and the corresponding vectors. We use the C version of SVDPACK. SVDPACK is a collection of C libraries, scripts and executables that comprises four numerical and iterative methods for computing the SVD of large sparse matrices using ANSI C. SVDPACK implements Lanczos and subspace iteration based methods for

determining several of the largest singular triplets, i.e. the singular values and the corresponding left and right singular vectors.

In this work, we represent this kind of sparse matrix in Harwell- Boeing format. In short, the Harwell- Boeing format is a file format designed to represent sparse matrices. Due to the fact that a sparse matrix has many zero entries, it is more efficient not to allocate memory for the whole matrix, but rather to keep track of the location and value of the nonzero entries. Our input for the SVD procedure is the sparse co- occurrence matrix stored in HB format. We compute a 100-factor singular value decomposition using the Block Lanczos Method. We smooth the co- occurrence counts by $\widetilde{e_{ij}} = \log(e_{ij} + 1)$ and leave the zero entries unchanged. The output of the SVD method for the experiment is:

```
    ... HYBRID BLOCK LANCZOS (CYCLIC)
... NO. OF EQUATIONS          =      5400
... MAX. NO. OF ITERATIONS    =       100
... NO. OF ITERATIONS TAKEN   =        52
... NO. OF TRIPLETS SOUGHT    =       100
... NO. OF TRIPLETS FOUND     =       100
... INITIAL BLOCKSIZE         =        10
... FINAL   BLOCKSIZE         =         0
... MAXIMUM SUBSPACE BOUND    =       120
... FINAL   SUBSPACE BOUND    =        20
... NO. MULTIPLICATIONS BY A  =      2737
... NO. MULT. BY TRANSPOSE(A) =      2250
... TOTAL SPARSE MAT-VEC MULT.=      4987
... MEMORY NEEDED IN BYTES    =  23455760
... WANT S-VECTORS ON OUTPUT?           T
... TOLERANCE                 =  1.00E-03

  co- occurrence matrix in hb format                                    ^
                'matrix'
... NO. OF TERMS (ROWS OF A)   =     2700
... NO. OF DOCS  (COLS OF A)   =     2700
                   ...
```

```
...... BLSVD EXECUTION TIME=  1.04E+01
                        ......
......          COMPUTED S-VALUES      (RES. NORMS)
                        ......
......    1     5.47259348241408E+02  (   3.43E-13)
......    2     3.22206791583908E+02  (   3.04E-13)
......    3     1.73932018874499E+02  (   1.73E-13)
......    4     1.65112830750816E+02  (   1.86E-13)
......    5     1.19968244436092E+02  (   2.83E-13)
......    6     1.17511147677450E+02  (   1.58E-13)
......    7     1.08888762495068E+02  (   4.10E-13)
......    8     1.06357847839991E+02  (   2.62E-13)
......    9     8.83332281664914E+01  (   1.82E-13)
......   10     8.60567530808088E+01  (   2.21E-13)

… … … … … … … … … … … … … … … … … … … … … … … … … … …

......   96     2.81183952850477E+01  (   4.61E-05)
......   97     2.79150775531576E+01  (   6.54E-05)
......   98     2.77850731839839E+01  (   1.05E-04)
......   99     2.75700175733032E+01  (   1.06E-04)
...... 100     2.74371789426662E+01  (   1.58E-04)
```

table 3.6. SVDPACK output

Using the 100 largest singular triplets, we can project each word vector to the M- dimensional space, for M = 100. We project each word vector w_i to u_i using the projection $u_i = A \cdot w$, where A is the output of the SVD of the co-occurrence matrix for the M largest singular values. In particular, matrix A is formed by the left singular vectors of the SVD. Let [U,S,V] be the output of the decomposition, our word projection is $u_i = Q_1{}' \cdot w$.

3.3. History mapping

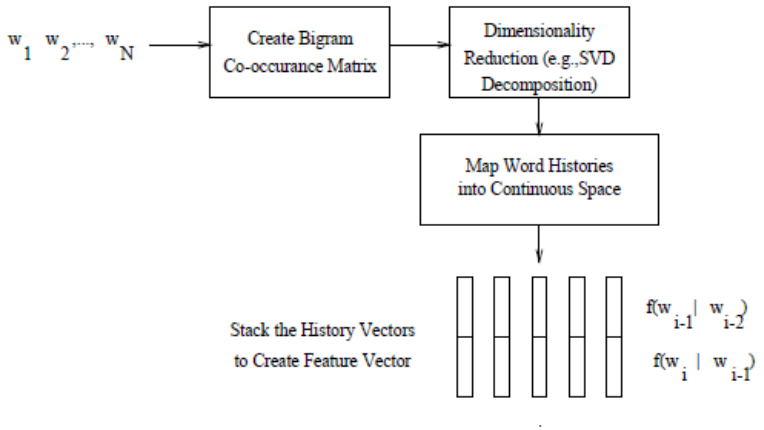


figure 3.7. History mapping

Based on N-gram models, each word's history consists of the previous N-1 words. Choosing the value of N depends on the implementation. If N=1, each word is independent. For N=2, each word depends on the previous one and if N=3, each word depends on the previous 2 words. In natural language, it seems that each word has strong dependence on its previous two, so we train trigram models, for N = 3.

'Whether the fashion designers who license their names will agree to stay on in the new company comma remains a key to the viability of the specific group period'.

We may observe that 'designers' is closely related to its previous two words, 'the fashion designers'. In addition, 'the new company' and 'the specific group' are two more examples showing the dependence of the words as triplets. The next step is to collect all histories for each word. This can be done if we find for each word, the N-1 previous. Using the above mapping of each word, each word's history consists of the concatenation of the appropriate mapped words. Each history is a vector with M(N-1) elements.

Having the history vectors, we could model our parameters, with Gaussian distributions. We must collect each word's history from our training data and then train one model for each word As mentioned before, modeling is difficult in large

dimensions. Assume that we have a trigram model (N=3), a vocabulary of size V = 3K and we perform SVD for M = 100 singular values. That means that each history consists of a 100*(N-1) = 200 elements. This is still a high- dimensional space and we may suffer from the curse of dimensionality. A suitable mapping for history vectors is $y_i = B \cdot h_i$ where y_i is the projection of the history vector h_i in the new- dimensional space. The estimation of the transformation matrix *B* is described below.

3.3.1. Curse of Dimensionality

It is intuitive to think that increasing the dimension of the features should never reduce the model's performance, since we are providing a larger, or at least the same, amount of information. Therefore the worst that could happen should be the performance staying the same. As practice shows, this is unfortunately not the case; the performance can decrease although we feed more data to the system. This behavior is due to the finite amount of training data that can be presented to the model. In theory we normally assume the training data to be infinite and so the model ends up being perfectly trained under all circumstances. In practice this is not possible and if we chose a too complex model then it is unlikely that all of our parameters will be well estimated. On the other hand the model should not be too simplifying either, since this would prevent the system from living up to the complexity of human speech.

Feature reduction uses statistical methods to reduce the dimension of the features, while maximizing the information that is preserved in the reduced feature space. Mathematically we can express this by applying a linear transformation

$$y_i = B \cdot h_i$$

, where y denotes a feature vector in the reduced feature space $y \in \mathcal{R}^L$ and $h \in \mathcal{R}^{2M}$ stands for the original feature vector.

The transformation matrix B is a 2M × L matrix. The goal of all feature reduction techniques is to find the optimal B with respect to some optimization criterion. Linear Discriminant Analysis (LDA) an effective method.

3.3.2. Linear Discriminant Analysis

LDA finds the optimal transformation matrix in terms of preserving most of the information that can be used to discriminate between the different classes. Therefore the analysis requires the data to have appropriate class labels. We associate words with class labels and assign each observed history vector to the corresponding. LDA estimates the between and within class scatters in order to find the optimal transformation matrix.

In order to formulate the optimization procedure mathematically, we have to compute the mean vector and the covariance matrix for each class

$$\bar{x}_v = \frac{1}{N_v}\sum_{i=1}^{N_v} x_i$$

$$W_v = \frac{1}{N_v}\sum_{i=1}^{N_v} (x_i - \bar{x}_v)(x_i - \bar{x}_v)^T$$

and for the complete data set (with all classes pooled together)

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$T = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$$

In the above formulas N denotes the total number of training tokens and N_v stands for the number of training tokens in class *v*. Naturally (there are *V* classes),

$$\sum_{v=1}^{V} N_v = N$$

With these definitions, we can easily formulate the optimization criterion, namely

$$\hat{B} = argmax_{B^L} \frac{|B_L^T \bar{T} B_L|}{|B_L^T W B_L|}$$

, where

$$W = \frac{1}{N}\sum_{v=1}^{V} N_v W_v$$

Although this criterion might look complicated at first glance, it can easily be understood. The numerator represents the covariance of the pooled training data in the transformed feature space. The denominator represents the average covariance within each class in the transformed feature space. Hence, the criterion really tries to maximize the 'distance' between classes while minimizing the 'size' of each of the classes at the same time. This is exactly what we want to achieve because this criterion guarantees that we preserve most of the discriminant information in the transformed feature space.

In our case, we have to encounter the problem of the large size of the history vectors. In order to estimate the projection matrix *B*, we estimate the statistics that LDA uses to compute the scatter matrices. We need to estimate between- class scatter S_B and within- class scatter S_W. At first, we estimate two sufficient statistics for the matrices computation which are:

$$t_{1i} = \sum_{n \in i} x_n$$

$$t_{2i} = \sum_{n \in i} x_n x_n^T$$

, where *i* is the word – class i.e. $i = 1, \ldots, c$ and x_n is the history vector.
After estimating these statistics for all history vectors, we compute the mean vector for each word – class.

$$m_i = \frac{1}{n_i} \sum_{n \in i} x_n = \frac{1}{n_i} \cdot t_{1i}$$

, where n_i is the amount of class *i* history vectors.

Then, we have to estimate the between – scatter matrix, which is computed as follows:

$$S_B = \sum_{i=1}^{c} n_i (m_i - m\ )(m_i - m)^T$$

, where

$$m = \frac{1}{n} \sum_{i=1}^{c} n_i m_i$$

In order to estimate the within – class scatter, we have to estimate S_i for each word – class.

$$S_i = \sum (x - m_i\ )(x - m_i)^T =$$

$$= \sum [xx^T - xm_i^T - m_i x^T + m_i m_i^T] =$$

$$= \sum xx^T - \ (\sum x) m_i^T - \ m_i \sum x^T + \ n_i m_i m_i^T =$$

$$= \sum xx^T - \ n_i m_i m_i^T$$

And finally,

$$S_W = \sum_{i=1}^{c} S_i$$

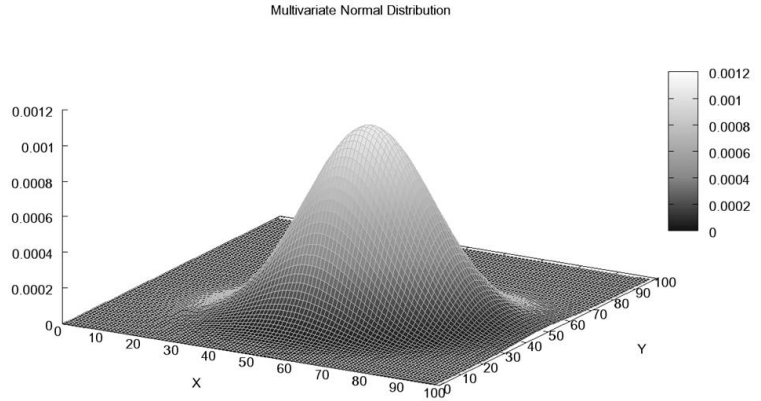
The rest of the transformation matrix B is the same as previous. By estimating the projection matrix B we are able to project each history vector to the new- dimensional space. In our model we project each history vector in the reduced feature space $y \epsilon R^L$ for L = 50.

3.4. Continuous Language Model Statistical methods

In order to build a continuous- space language model we have to use some distributions in the continuous space. The most known and used distribution is the Gaussian distribution. Firstly, we used the multivariate Gaussian distribution for initial parameter estimation and then we used Gaussian Mixture Models and the Expectation- Maximization algorithm. Finally we used parameter tying techniques to build Tied Gaussian Mixture Model.

3.4.1. Multivariate Gaussian distribution

figure 3.8. Multivariate Gaussian distribution



In probability theory and statistics, the multivariate normal distribution or multivariate Gaussian distribution, is a generalization of the one-dimensional (univariate) normal distribution to higher dimensions. One possible definition is that a random vector is said to be p-variate normally distributed if every linear combination of its p components has a univariate normal distribution.

However, its importance derives mainly from the Multivariate central limit theorem. The multivariate normal distribution is often used to describe, at least approximately, any set of (possibly) correlated real-valued random variables each of which clusters around a mean value.

The multivariate normal distribution of a k-dimensional random vector $\mathbf{X} = [X_1, X_2, \ldots, X_k]$ can be written in the following notation:

- $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

with k-dimensional mean vector

- $\boldsymbol{\mu} = [\, E[X_1], E[X_2], \ldots, E[X_k] \,]$

and k x k covariance matrix

- $\boldsymbol{\Sigma} = [\, Cov[X_i, X_j] \,]$, $i = 1,2,\ldots,k$; $j=1,2,\ldots,k$

Multivariate normal distribution describes variables that tend to cluster around their mean value. Based on the Multivariate central limit theorem, any random variable can be described by the normal distribution if it has a large set of observations. That is why Gaussian distributions are often used for statistical modeling and language modeling.

3.4.1.1. Multivariate Gaussian Language Model

After collecting the mapped history data, we use one multivariate Gaussian distribution for each word. Then, we calculate each word's mean vector and covariance matrix. So, we model each word based on its history y: $P(y|w) = N(y;\ \mu_w, \Sigma_w)$ where μ_w and Σ_w are mean vector and covariance matrix.

With this distribution we evaluate the probability of each history given the word. With our model we want to evaluate the probability of the word given its history. Using Bays rule we have:

$$P(w|y) = \frac{P(w)p(y|w)}{p(y)} = \frac{P(w)p(y|w)}{\sum_{v=1}^{V} P(v)p(y|v)}$$

figure 3.9. Bayes rule

, where ***P (w)*** is the unigram probability of the word.

We must consider that $P(w|y)$ must sum up to 1 for each $w \in V$. An appropriate check is performed after the training, for a history set $Y = \{y_1 y_2 \cdots y_k\}$, checking if $\sum_{w \in V} P(w|y_n) = 1$ for each $y_n \in H$.

The model parameters for this approach are the SVD output of the co-occurrence matrix, the LDA projection matrix B and the mean vectors and covariances matrices for each word. After the model estimation, we evaluate the test data logarithmic probability and perplexity.

3.4.2. Gaussian Mixture Models (GMM)

A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities. GMMs are commonly used as a parametric model of the probability distribution of continuous measurements or features in acoustic and language models. GMM parameters are estimated from training data using the iterative Expectation-Maximization (EM) algorithm.

A Gaussian mixture model is a weighted sum of M component Gaussian densities as given by the equation,

$$p(x|\lambda) = \sum_{k=1}^{K} c_k \, g(\mathrm{x} \,|\mu_k \,, \Sigma_k)$$

where x is a *D*-dimensional continuous-valued data vector (i.e. measurement or features), c_k, $k = 1, \ldots, K$, are the mixture weights, and $g(\mathrm{x} \mid \mu_k, \mathbf{\Sigma}_k)$, $k = 1, \ldots, K$, are the component Gaussian densities. Each component density is a *D*-variate Gaussian function of the form,

$$g(\mathrm{x} \,|\mu_k \,, \Sigma_k) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_k|^{\frac{1}{2}}} exp\left\{-\frac{1}{2}(\mathrm{x} - \mu_\kappa)' \Sigma_k^{-1} (\mathrm{x} - \mu_\kappa)\right\}$$

, with mean vector $\boldsymbol{\mu_i}$ and covariance matrix $\mathbf{\Sigma}_i$. The mixture weights satisfy the constraint that their sum is 1.The complete Gaussian mixture model is parameterized by the mean vectors, covariance matrices and mixture weights from all component densities.

These parameters are collectively represented by the notation,

$$\lambda = \{w_k, \mu_k, \Sigma_k\} \quad k = 1, \ldots, K$$

The covariance matrices, $\mathbf{\Sigma}_i$, can be full rank or constrained to be diagonal. Additionally, parameters can be shared, or tied, among the Gaussian components, such as having a common covariance matrix for all components. The choice of model configuration (number of components, full or diagonal covariance matrices, and parameter tying) is often determined by the amount of data available for estimating the GMM parameters and how the GMM is used in a language model application. It is also important to note that because the Gaussian components are acting together to model the overall feature density, full covariance matrices are not necessary even if the features are not statistically independent. The linear combination of diagonal covariance basis Gaussians is capable of modeling the correlations between feature vector elements. The effect of using a set of M full covariance matrix Gaussians can be equally obtained by using a larger set of diagonal covariance Gaussians. GMMs are often used in biometric systems, most notably in continuous- speech recognition systems, due to their capability of representing a large class of sample distributions.

3.4.2.1. Gaussian Mixture Language Model (GMLM)

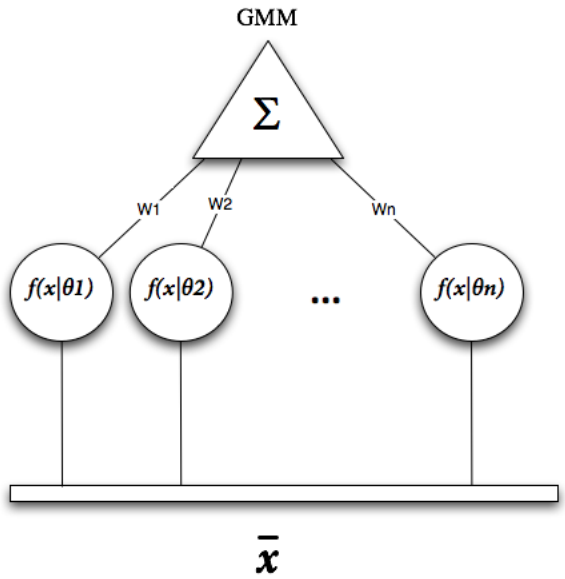


figure 3.10. Gaussian Mixture Model

As mentioned before, each GMM consists of K_w weighted distributions. To build our model, we train one GMM for each word using the Expectation Maximization algorithm. After the word and history mapping, we collect each word's history and we build the mixtures:

$$p(y|w) = \sum_{k=1}^{K_w} c_{w,k} \mathcal{N}(y, \mu_{w,k}, \Sigma_{w,k})$$

, where K_w is the number of the components.

We use different values of K mixtures for each implementation and we train the model parameters. The model parameters for GMLM are the SVD output matrix A of the co-occurrence matrix, the LDA projection matrix B and the mixture parameters, mean vectors and covariances matrices for each word and the priors for each mixture. After the model estimation we can evaluate the test data logarithmic probability and perplexity using the Bays rule as previous.

3.4.3. Tied Gaussian Mixture Models (T-GMM)

GMMs use different set of distributions for each variable and each of one is parameterized by the mean vectors, covariance matrices and mixture weights from all component densities. It is inevitable that as the dimension and the variables are being increased, the GMM parameters are being increased consequently. There is a specialization of GMMs that instead of having separate sets of Gaussian distributions for each word, a common set of distributions can be used for all words with different weights.

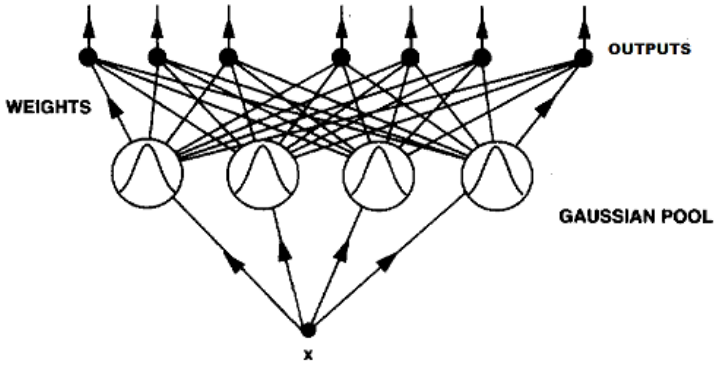


figure 3.11. Gaussian pool

Let's assume that we have a set of distributions, we may refer to it as a Gaussian pool, which is common for all words. Tied Gaussian mixture model is a weighted sum of *J* component Gaussian densities as given by the equation,

$$p\ (y\ |w) = \sum_{k}^{K} c_{w,j} N(h;\ \mu_j \Sigma_j)$$

T -GMM is been used in pattern recognition and statistical modeling applications, such as acoustic modeling. Their advantage is that they use a small set of parameters for large amount of data, and, consequently, the model training is more efficient.

3.4.3.1. Tied Mixture Language Model (TMLM)

GMLM is proposed to overcome N-gram drawbacks, such as generalizability and adaptability. Although, this method has a disadvantage as far as the amount of parameters is concerned. Tied-Mixture Language Model (TMLM) does not have the model parameter estimation problems that GMLM has. TMLM provides a great deal of parameter tying across words, hence achieves robust parameter estimation. As such, TMLM can estimate the probability of any word that has as few as two occurrences in the training data. Also, we must notice that GMM training may suffer from data- overfitting problems. That means that some history data have really small variances, which leads to computational problems. Tying parameters such as variance vectors or the entire mixtures is used to overcome this problem

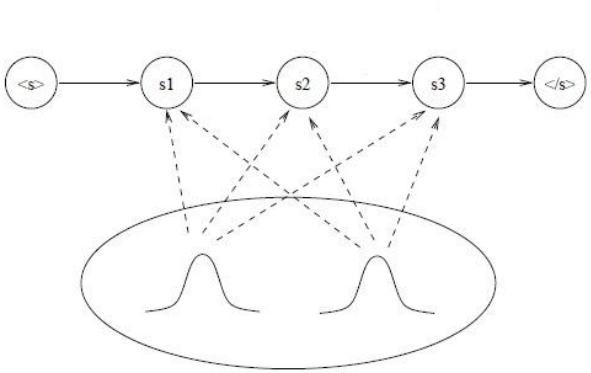


figure 3.12 parameter tying

Using tying techniques for our implementation, we tie the variance vector for each word, in order to train our model. So, all the words use the same variance vector for each distribution and different weights for each mixture.

The model parameters for TMLM are the SVD output of the co-occurrence matrix, the LDA projection matrix B and the mixture parameters, mean vectors and covariances matrices for the common set of distributions and the weights for each word. After the model estimation, we evaluate the test data logarithmic probability and perplexity using the Bays rule as previous.

Chapter 4

Experimental Evaluation

4.1. Data description

In this work, we use data from the Wall Street Journal for our train and test data. In particular, we used articles from 1994. The data files are WS94_*.VPZ and each sentence has the following form:

<p.wsj94_001.0012.3>
<s.wsj94_001.0012.3.2>
D. D. T. is a highly persistent chemical that moves up the food chain, COMMA and it accumulates in the fatty tissue of humans .PERIOD
</s>
</p>

First line means that the sentence comes from document 001.0012, paragraph 3 and is sentence two. Text data is not normalized and is verbalized. That means that punctuation marks are followed by the word of the mark, i.e. “,COMMA”. Moreover, there are many incorrect words so we have to edit the data before splitting them into train and test data.

- At first, we removed all of the headers, such as <p.wsj94_001.0012.3> and <s.wsj94_001.0012.3.2>
- We turned the data into lowercase
- We discarded incorrect words such as «kknow», ”tthey”, ”aare”
- We removed digits and numbers
- Punctuation is only the word, not the mark
- We sorted all sentences according to their length
- We inserted <s> and </s> symbols at the beginning and the end of each sentence, consequently.
- For our small vocabulary, we found the 2700 most frequent words, which are the vocabulary, and replaced any other words with <unk> symbol. For our large vocabulary, we did not replace any of the words, and we used all of the words for our vocabulary.
- According to the vocabulary, we turned our data into index sequences. Each index shows the position of each word. For example, *<s> percentage gains for <unk> ended march thirty first comma nineteen ninety three semicolon assets as of december thirty first comma nineteen ninety two assets a <unk> fee on shares held for a year or less <\s>*turns into 4 691 680 14 0 385 322 69 75 1 44 54 29 90 441 22 6 901 69 75 1 44 54 18 441 8 0 1920 19 118 458 14 8 48 57 295 3
- So, our train and test data are:

WS94	Train data	Test data
No of sentences	160000	6000
No of words	4447740	164574

figure 4.1. Data description

4.2. Baseline experiments – SRILM toolkit

SRILM is a collection of C++ libraries, executable programs, and helper scripts designed to allow both production of and experimentation with statistical language models for speech recognition and other applications. The toolkit supports creation and evaluation of a variety of language model types based on N-gram statistics, as well as several related tasks, such as smoothing and class- based models.

At first it generates the n-gram count file from the corpus, then it trains the language model from the n-gram count file and it calculates the test data perplexity using the trained language model. Also it can perform word clustering.

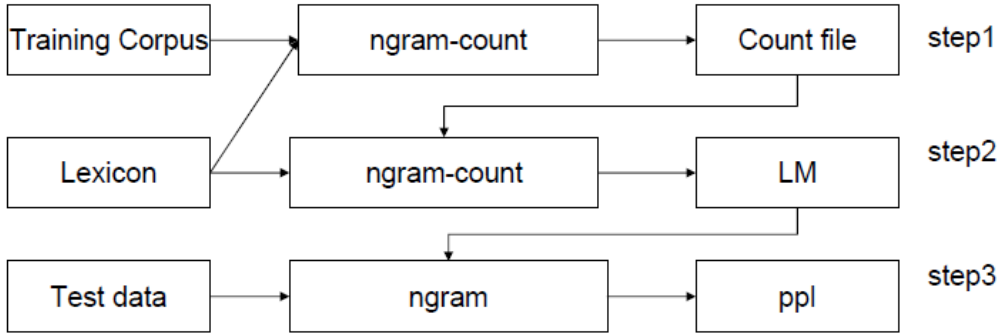


figure 4.2. SRILM toolkit

In particular:

- ngram -count -vocab Lexicon.file
  -text train.txt
  -order 3
  -write train_3gram
  -unk

This command generates and manipulates N-gram counts, and estimates N-gram language models from them.

-vocab *file*: reads a vocabulary from file
-text *filename*: train data set
-order: sets the maximal length of N-grams
-write *filename*: output file that contains N-gram counts
-unk: sets any unknown words with oov (out-of-vocabulary)

-output

<unk> 668381
<unk> <unk> 112108
loan does 1
loan negotiations will 1
from university <unk> 1
from ten 40
from ten million 1
suggested the final 1
suggested the industry 1
suggested the chicago 1
majority stake of 1
majority democrats

- ngram-count -vocab Lexicon.file
  -read train_3gram
  -order 3
  -lm 3gram.train.lm

This program reads count file and creates the language model file

-vocab *file*: vocabulary file
-read *countfile* : N-gram counts file
-order: N-gram length
-lm *lmfile*: language model file

-output

\data\
ngram 1=2001
ngram 2=260999
ngram 3=246775
\1-grams:
-1.354499 </s>
-99 <bos>
-99 <eos>
-99 <s> -1.72626
-4.109913 aircraft -0.6429651
-4.023325 airline -0.5559635
-3.777582 airlines -0.6082685
-1.560787 zero zero two
-1.123479 zero zero zero

\end\

- ngram -ppl test.txt
  -order 3
  -lm 3gram.train.lm

This program evaluates test data perplexities based on the language model we trained.

-ppl *pplfile*: test data perplexities file
-order: N-gram length
-lm *lmfile*: το language model file

-output

brain banks need more depositors period
p(<unk> | <s>) = [OOV] 0 [-inf]
p(banks | <unk> ...) = [1gram] 0.000475788 [-3.32259]
p(need | banks ...) = [1gram] 7.3968e-05 [-4.13096]
p(more | need ...) = [2gram] 0.0191286 [-1.71832]
p(<unk> | more ...) = [OOV] 0 [-inf]
p(period | <unk> ...) = [1gram] 0.0436764 [-1.35975]
p(</s> | period ...) = [2gram] 0.925196 [-0.0337663]
1 sentences, 6 words, 2 OOVs
0 zeroprobs, logprob= -10.5654 ppl= 129.741
ppl1= 437.869
file test.txt: 6000 sentences, 152570 words, 24582 OOVs
0 zeroprobs, logprob= -255134 ppl= 70.2338 ppl1= 76.339

This file contains the logarithmic probability of each sentence and its perplexity. It also evaluates the perplexity of the whole test data, the number of words and the out- of –vocabulary words.

Here, we have the SRILM results for the test and train data.

SRILM results	Test data	Train data
ppl	70.2338	52.2659

4.3. Model training

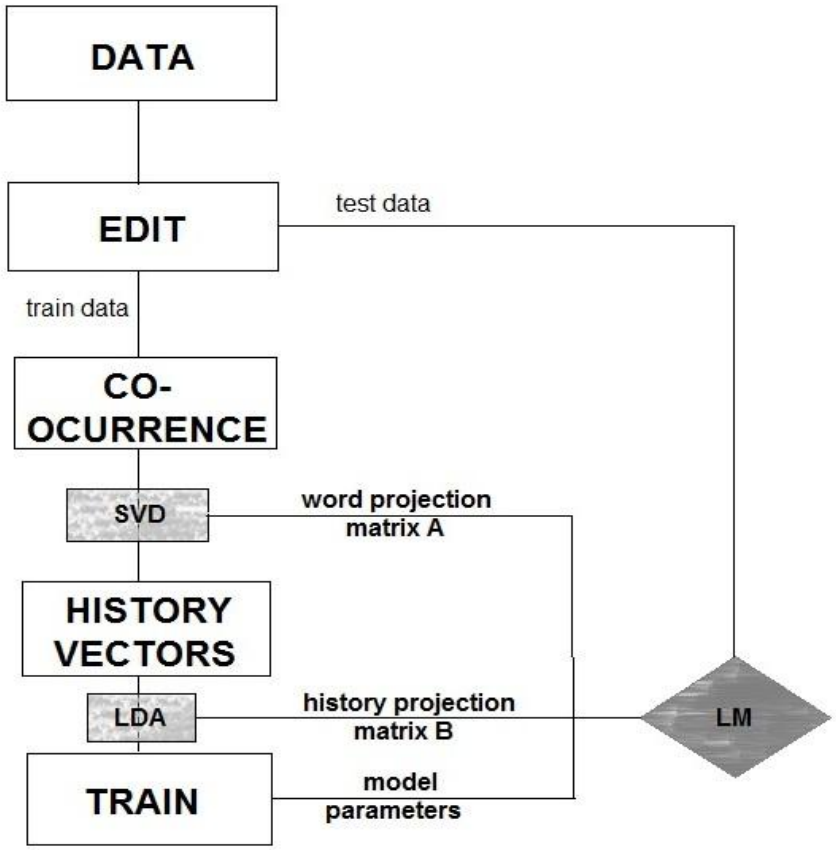


figure 4.3. Model algorithm

As covered before, on model training chapter, for all of the previous models, the implementation is same. We edit and process our data; we split them into train and test sets. Then, we construct the co- occurrence matrix, even if it is bigram - oriented or as it is shown later. We perform Singular Value Decomposition for dimensionality reduction, for the M maximum singular values. So we have our first model parameter, the word projection matrix A. Using this word mapping, we collect our history vectors for each word and project them to a lower dimension using Linear Discriminant Analysis. The projection matrix B, is used for the projection of each history matrix to our new continuous space, with lower dimension.

We use these vectors as training data. The HTK tool, which is described in this section, uses this data to train the mixture models. Having estimated the model we compute the set perplexity. We also use tying techniques and other approaches for the co- occurrence matrices to improve our results.

4.4. Experimental Results

After the training of the models we referred to, we estimated the test data set perplexity for each language model.

$$\log P(test\_data) = \log[P(S_1) \cdot P(S_2) \cdots P(S_T)]$$

For each test data sentence, we estimate its log- probability.

$$\log P(S_k) = \log[P(w_1| < s >< s >) \cdot P(w_2| < s > w_1) \cdots P(w_n|w_{n-2}w_{n-1})] = \\ = \log[P(w_1| < s >< s >)] + \log[P(w_2| < s > w_1)] + \cdots + \log[P(w_n|w_{n-2}w_{n-1})]$$

Using the Bayes rule we estimate each trigram log- probability.

$$\log[P(w_k|w_{k-2}w_{k-1})] = \log\frac{P(w_k)P(w_{k-2}w_{k-1}\,|w_k)}{\sum_V P(u)P(w_{k-2}w_{k-1}\,|u)}$$

Then, we estimate the test data perplexity:

$$PPL = e^{\frac{-logprob}{(T+W-OOVS)}}$$

, where T is the number of test data sentences, W the number of the words and OOVS the number of out –of –vocabulary words.

The output of each experiment has the following format:

```
Test data set perplexity estimation
Estimating test data sentences perplexity
logP(a|<s>,<s>) = -2.545684
logP(leading|<s>,a) = -1.209133
logP(member|a,leading) = -1.232993
logP(of|leading,member) = -1.239600
logP(the|member,of) = -1.135314
logP(party|<unk>,<unk>) = -1.095569
logP(said|<unk>,party) = -1.384926
logP(that|party,said) = -1.185744
……..
logP(a|i,r) = -1.736261
logP(response|r,a) = -0.692497
logP(period|a,response) = -1.699502
logP(</s>|response,period) = -Inf
Sentence No. 1 has 32 words with 5 oovs.
logprob = -168.689786 ppl = 413.490780
```

The statistics printed in the last two lines describe the number of the sentence, the number of the words of the sentences. Also, the number of out-of vocabulary words is referred. This is the number of unknown word tokens, i.e. tokens that appear in **test data** but not in **train data** from which language model was generated. Logprob gives us the total logprob ignoring the 5 unknown word tokens. The logprob does include the probabilities assigned to </s> tokens which are introduced in the beginning of the training. Thus the total number of tokens which this logprob is based on is: NoOfWords – OOVs + sentences = 32 – 5 + 1 = 28, for the specific sentence.

Perplexity is the geometric average of 1/probability of each token, i.e., perplexity. The exact expression is:

$$ppl = e \wedge (-logprob / (words - OOVs + sentences))$$

After the estimation of the model parameters, which are the SVD and LDA output matrices, we train our model using HTK. We have to decide how we will train our models. There are two ways. Firstly, we train one multivariate Gaussian distribution for each word. Then we can use the HHEd tool to split the Gaussian into more components. We also trained word GMMs with direct number of components, without using the split procedure.

Concerning the data overfitting problem we set a stable variance floor with v = 0.1. We referred to this problem as the computational problem that occurs when data and Gaussians have incredibly small variance values. The value of the variance floor is large so there are many variances that are floored. We use this value to estimate the two methods of training to decide which we are about to use. We use Hinit tool to initialize the GMLM models and then we estimate the model parameters using HREst tool. We used K = {1,2,4,8,16,32} components for each implementation

We estimated the test data perplexity using this method on a small test data set of T = 100 sentences

	K = 1	K = 2	K = 4	K= 8	K = 16
logprob	-13176	-13155	-13030	-12783	-12701
ppl	343.18	340.003	321.589	288.286	278.021

HHEd splitting	K = 1	K = 2	K = 4	K= 8	K = 16
logprob	-13176	-14101	-13568	-13154	-12735
ppl	343.18	516.86	408.183	339.736	282.17

tables 4.4,4.5. small test set perplexity with direct number of components and HHEd splitting

We can see that training the GMMs with direct number of components gives us better results, especially for small number of components. So we use this method for the rest of the experiments. Our next step is to use another value of variance floor (vfloor), so for a certain number of components we estimate the perplexity for different values of vfloor.

K = 8	0.1	0.05	0.04	0.01
logprob	-12783	-13799	-13904	-20421
ppl	288.28	327.22	342.07	5268

table 4.6. small test set perplexity using different vfloor values

It is obvious that vfloor affects the results. We have to concern that if we choose a large value of vfloor the results are better, but there are more variances that are floored. We choose vfloor to be 0.05 in order to compare the perplexity values for different number of components. We still estimate the small test set perplexity (T = 100).

	K = 1	K = 2	K = 4	K= 8	K = 16	K = 32
logprob	-14189	-14174	-14033	-13799	-13603	-13546
ppl	385.43	383.11	361.098	327.221	301.493	294.317

table 4.7. small test set perplexity for different mixture components

Whereas, the test set perplexity for the whole set (T = 6000) is:

	K = 1	K = 2	K = 4	K= 8	K = 16	K = 32
logprob	-788595	-787443	-775338	-763566	-761124	-770041
ppl	301.12	298.62	273.57	251.22	246.79	263.21

table 4.8. test set perplexity for different mixture components

Our next experiment is about setting a value of vfloor which is not pre- defined but being depended on the global variance of the data. We use the HCompV tool to estimate the global variance of its word training data and set a vfloor vector as a percent of the global variance. By default, HCompV sets the f * global_variance as variance floor, where f = 0.01.

We also use the HHEd tool to tie our mixture variances. Concerning that all mixtures will share the same variance; we used more components for this experiment.

	K = 32	K = 64	K = 128
logprob	-13717	-13662	-13575
ppl	352.45	308.95	297.9

table 4.9. small test set perplexity with tied variances

Something else that we tested is to increase the variance vector by 5 or 10 percent, so as the variance vector is var’ = a* var.

K = 128	a = 1	a = 1.1	a = 1.05
logprob	-13575	-13680	-13629
ppl	297.9	311.32	304.8

Table 4.10. small test set perplexity with variance increase

We can conclude that all of the above experiments have similar perplexity results. For that reason, we followed a different approach. Something we must take into consideration is the initial word mapping. We constructed the word vectors using the SVD output of the word co- occurrence matrix. This matrix can be regarded as a bigram counts matrix, keeping trace of the counts of each vocabulary bigram. The first alternative of this matrix is a co- occurrence matrix which counts the occurrences of each word with the M most frequent history words. So we construct a word co- occurrence matrix H (V x M), using the same procedure as with the 'original' co- occurrence matrix. For example:

	H_1	H_2	H_3	H_4	…	H_{500}
V_1	223	143	414	87	⋯	43
V_2	0	423	95	0	⋯	12
V_3	62	0	43	0	⋯	0
…	…	…	…	…	…	…
V_v	13	0	5	0	…	9

table 4.11. co – occurrence matrix H with fewer history words

The rest of the implementation is the same. The SVD output of this matrix is used for the word vectors and LDA for the history vectors projection to the new- dimensional space. We trained the GMLM model with K = 64 mixtures by tying the variance vector and estimated the test set perplexity:

K = 64	bigram co- occurrence	co- occurrence H
logprob	-13662	-13547
ppl	308.95	294.39

table 4.12. small test set perplexity with co- occurrence matrix H

We observe a slight improvement of the results. Our next step is to try another vector representation for the vocabulary words. Instead of using the 'raw' counts of the bigrams for each matrix entry, we use the probability of each bigram. So, matrix P consists of the bigrams probabilities:

	V_1	V_2	V_3	V_4	…	V_V
V_1	0.004	0.013	0	0.087	⋯	0.043
V_2	0	0.4	0.095	0	⋯	0.12
V_3	0.054	0	0.013	0.078	⋯	0
…	…	…	…	…	…	…
V_v	0.09	0	0.0015	0	…	0.0019

table 4.13. probability co-occurrence matrix P

After the SVD and LDA projection we estimated the test data perplexity:

K = 64	bigram co- occurrence	probability co- occurrence
logprob	-13662	-14636
ppl	308.95	464.9

table 4.14. small test set perplexity with co- occurrence matrix P

There is no improvement on our results with this projection. We can assume that this may happen due to the fact that these co- occurrence matrices refer to bigram counts while we build a trigram model. For this reason, we constructed a 'trigram' co- occurrence matrix T, where T_{ij} is the amount of times that word *i* appears after the bigram history *j*.

The number of the possible history bigrams is large enough, so we use the N top history bigrams that are seen in the training set. The size of the trigram co- occurrence matrix is VxN, where N is the number of the top bigrams that we use. We used several values of N.

The form of this co- occurrence matrix is:

	H_1	H_2	H_3	H_4	…	H_V
V_1	457	332	230	87	⋯	143
V_2	230	104	95	0	⋯	112
V_3	54	0	113	78	⋯	0
…	…	…	…	…	…	…
V_v	9	0	15	0	…	19

table 4.15. co- occurrence matrix T

We used the N = 50, 100, 200, 300, 500, 1000 and 2500 top bigrams to construct the co- occurrence matrix. Using these matrices for the word representation we build our GMLM models with K = 64 mixtures and estimated the test data perplexity:

K = 64	bigram co- occurrence	50	100	200	300	500	1000	2500
logprob	-13662	-13328	-13490	-13570	-13651	-13538	-13693	-13681
ppl	308.95	268.58	287.4	297.25	307.63	293.36	313.06	311.46

table 4.16. small test set perplexity with co- occurrence matrix T for N top bigrams histories

We observe that using the 50 top bigrams for the co- occurrence matrix gives us the best results compared to the others. We estimated the entire test set and training set perplexity using this implementation, as so as the implementation of the original co- occurrence matrix representation using K = 64 mixtures.

	Training data		Test data	
	Bigram co- occurrence	Trigram co-occurrence	Bigram co- occurrence	Trigram co-occurrence
Logprob	-66230	-58069	-764152	-735380
ppl	460.54	216.33	252.25	204.83

table 4.17. test and train set perplexity

Chapter 5

Conclusion

5.1. Summary of Results

In this work, we proposed language models to continuous space. Based on the main drawbacks of N-gram discrete language models, adaptability and generalization, we used appropriate mapping methods to project words as variables in the continuous space. In the beginning we used a vocabulary of the V = 2700 most frequent words of our train data. We used Singular Value Decomposition and Linear Discriminant Analysis for feature dimensionality reduction and the EM algorithm for the training of the mixture models. We used Gaussian mixture models for each vocabulary word. We applied tying methods in these models and we proposed other word mapping approaches with different coocurence matrix estimation.

After the training of each model, we estimate the logarithmic probability for the test data set $\log P(test\ set)$:
$\log P(test\ set) = \log P(sentence_1) + \log P(sentence_2) + \log P(sentence_3) + \ldots \log P(sentence_n)$,
where $P(sentence_k) = P(w_1| < s >)P(w_2| < s > w_1) \ldots P(w_n|w_{n-2}w_{n-1})$. Each one of the word probabilities is estimated form the Bayes rule. Then, we compute the test data perplexity, based on the SRILM formula:

$$PPL(test\ set) = e^{-\log P(test\ set)/(\#Sentences+\#Words-\#oovs)}$$

5.2. Future work

We proposed continuous- space language models to cope with the main problems of the traditional discrete N-gram language models. We evaluated several continuous- space language models and compared them to the SRILM N-gram models in the WS94 corpus. Comparing our best results with the SRILM results, we see that our implementation is slightly worse. A basic problem that we encountered is data over fitting. This may explain our model accuracy.

Generally, language modeling in continuous space is more adaptable and smooth for words with few occurrences. More experiments with different parameters can be repeated. These parameters are the M largest singular values for the SVD technique, the initial word projection based on the coocurence matrix type, the dimensions L of the projected history vectors which LDA deals with.

Furthermore, there can be clustering methods to split words into clusters and train cluster based mixture models. Tying model parameters of words that belong to the same cluster can be implemented in order to improve model accuracy. A future work could integrate these models with a continuous- space recognizer and evaluate word accuracy. Moreover, a language model that can adapt with topic alternation can be built based on these models.

Bibliography

1. M.Afify, O. Siohan and R. Sarikaya. 2007. Gaussian Mixture Language Models for Speech Recognition, ICASSP, Honolulu, Hawaii.
2. Xuedong Huang, Alex Acero, Hsiao- Wuen Hon, 2001. Spoken Language Processing, Prentige Hall PTR
3. Wei Chen, Sanjeev Khudanpur, 2007. Building Language models on continuous space using gaussian mixture models.
4. Berlin Chein, Introduction to SRILM Toolkit, 2007. Department of Computer Science & Information Engineering National Taiwan Normal University.
5. V. Digalakis, D. Oikonomidis, 2002. Language Models for Speech Recognition. Technical University of Crete.
6. R. Duda, P.Hart, and D. Stork, Pattern Classification (Second Edition). Wiley-Interscience, October 2000.
7. A. Stolcke," SRILM – an extensible language modeling toolkit," Proc. ICSLP'02, Denver, Colorado, Sept., 2002.
8. R. Sarikaya, M.Afify, Brian Kingsbury. 2007. Tied–Mixture Language Modeling in Continuous Space.
9. P. Brown, P. V. DeSouza, R. L. Mercer, V. J. Della Pietra, J. C. Lai. Class- Based n-gram Models of Natural Language, IBM T. J. Watson research Center.
10. T. Matzuzaki, Y. Miyao, J. Tsujii. An Efficient Clustering Algorithm for Class- Based Language Models, Tokyo, Japan.
11. S. Geirhofer, 2004. Feature Reduction with Linear Discriminant Analysis and its Performance on Phoneme Recognition, University of Illinois at Urbana-Champaign Department of Electrical and Computer Engineering
12. I. T. Nabney, 2004. Netlab: Algorithms for Pattern Recognition. APR, Springer.
13. S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev P. Woodland, 2006, The HTK Book, Cambridge University Engineering Department
14. S. Theodoridis, K. Koutroumbas, 2009. Pattern Recognition, 4th edition. Elsevier Inc. AP.
15. L. van den Maaten, E. Postma, J. van den Herik, 2009. Dimensionality Reduction: A Comparative Review. TiCC.