TECHNICAL UNIVERSITY OF CRETE
ELECTRONICS AND COMPUTER ENGINEERING

Discriminative Training of Language Models

by

Nikolaos Fitopoulos

ELECTRONICS AND COMPUTER ENGINEERING

2014

Abstract

The present thesis investigates the use of discriminative training on continuous Language Models. The main motivation for dealing with continuous language models was that by construction they overcome the limits of N-gram based models. N-gram models have been widely used in Language Modeling, but suffer from lack of generalizability and contain a very large number of parameters that are hard to adapt. Another flaw of N-gram models is the need for a large amount of training data, in order to cover as many N-grams as possible. Continuous Gaussian Mixture Language Models (GMLMs) for Speech Recognition have proven to be effective in terms of smoothing unseen events and adapting efficiently while using relatively small amount of data when compared to N-gram models.

The training and testing data were extracted from the Wall Street Journal Corpus. Although the size of the vocabulary used in the corpus is large, the actual number of words being used in the present thesis is restricted. Data has the form of a continuous-space vector and consists of the history of each word in the corpus. The dimensions of these vectors were reduced by using SVD and LDA techniques.

As far as the main objective of the thesis is concerned, attempts focus on improving the performance of GMLMs that have been previously trained by using the ML criterion on Language Models by adapting and using the Maximum Mutual Information(MMI) Estimation Method previously deployed in training HMMs for acoustic models. MMI acoustic models have proven to perform better than ML models, therefore MMI training gave a strong incentive in order to apply it on continuous language models. In addition, other discriminative criteria such as Minimum Phone Error (MPE) or Minimum Classification Error(MCE) are also theoretically investigated. Perplexity is the metric being used to measure the effectiveness of the presented method. The experiments of the thesis focus on testing MMI models that are smoothed with their correspondent baseline ML model and MMI models that are unsmoothed, with mixed results. The desired improvement is achieved in the case of unsmoothed MMI models against ML models.

Acknowledgements

I would like to thank the supervisor of the present thesis, Professor Vassilis Digalakis, for the valueable knowledge and support he provided and Dr.Vassilis Diakoloukas for his advice and of course his patience during the implementation of the project. Special thanks to all of my friends and fellow students for making my studying experience in Chania a very pleasant and unforgettable journey. Last but not least, I am grateful to my parents and my sister for their unconditional support during my studies.

Table of Contents

Chapter 1

Introduction

1.1 Speech Recognition

In general, speech recognition processes involve building *Acoustic Models* and *Language Models*. Acoustic Models deal with matching utterances to a sequence of words. This matching process uses the maximization of the posterior probability of a specific word sequence S given a certain utterance (observation) O. This probability is the following:

$$P(S|O) = \frac{P(O|S)P(S)}{P(O)} \tag{1.1}$$

The maximization of the probability above can be achieved by maximizing the numerator term *P(O|S)P(S)*. Acoustic modeling estimates likelihoods similar to *P(O|S)*. These likelihoods follow a distribution that models spoken language [1]. Term *P(S)* represents the probability of a particular word sequence. Language modeling estimates these probabilities.

After having estimated an acoustic model along with a language model, speech recognition tasks can take place for a given segment of speech: the acoustic model can decide upon which word sequence was uttered and, subsequently, the language model calculates how probable this word sequence is and gives alternate and maybe more probable choices, thus enhancing the accuracy of the recognition process.

1.2 What is a Language Model?

Language modeling constitutes a very important part of speech recognition; it provides awareness of grammar, syntax rules and vocabulary of a spoken language. In other words, a Language Model can be considered as the "glue" that sticks speech segments together in a coherent way [1]. Along with these rules, the language model is supplemented by stochastic processes that facilitate fast search within the rules list and cover any possible

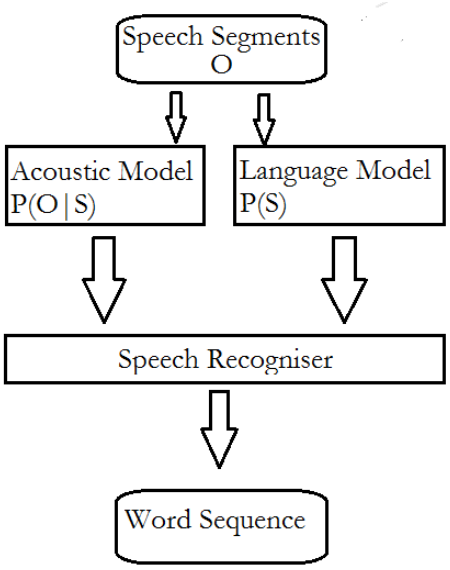

Figure 1.1: Simplified depiction of a speech recognition task

gaps.

By definition, a stochastic Language Model calculates the probability of potential word sequences in a text, utilizing a specific probability distribution. This means that the word sequence $<w_1\ w_2\ ...\ w_i>$ occurrence has the following probability:

$$P(w_1 \cdots w_i) = P(w_1)P(w_2|w_1)P(w_3|w_1w_2) \cdots P(w_i|w_1 \cdots w_{i-2}w_{i-1}) \tag{1.2}$$

according to the chain rule of probability[2].

1.3 Usage of Language Models

Word sequence probabilities provided by a Language Model have proven to be a valuable tool in *word prediction*[2]. In such prediction processes, a recognizer scans a speech segment and attempts to predict the word that follows the previously read sequence. This prediction is accommodated by the probabilities included in the language model used by the recogniser.

Word prediction procedures are useful in speech recognition or hand-writing recognition systems, in which input can be very noisy and ambiguous. Therefore, previous words during the word-scanning process can provide a strong hint on which word is the next one.

Predicting the next word in a context can also be useful in communication systems that help the disabled in communication. For instance, people who are unable to speak or use sign-language to communicate use systems that speak for them, letting them choose words with simple hand movements by selecting from a menu of possible words. Having all possible words of a language available in a list is rather strenuous, so predicting which words are the most possible to be used next is extremely important for this task.

Real-word spelling errors can also be drastically reduced by using word prediction techniques. These errors result in false but existing words, thus their detection is difficult than this of conventional spelling errors that lead to non-existing words. The elimination of these errors is made by estimating the probability of the examined word sequence. The challenge is that sentences of low probability may also occur in the context.However, context-sensitive spelling error correction may be used to alleviate this problem[2].

1.4 N-gram models

If the equation (1.2) is used for all the potential word sequences of any size, we will need a great deal of corpora and plenty of time to calculate these probabilities. This problem is solved by using an approximation: the word depends on its previous n-1 words. This assumption is the basis of N-gram models, in which the probability P(w_1 ... w_i) would be calculated as:

$$P(w_1 \cdots w_i) \approx \prod_{k=1}^{i} P(w_k | w_{k-n+1} \cdots w_{k-1}) \tag{1.3}$$

In this work, a Language Model has been built in order to calculate the probability of a word occurrence when having observed the previous two words (this is referred to as *history* of the current word), thus the *trigram scheme* is being followed. According to the trigram model, the probability P(w_1 ... w_i) is evaluated as:

$$P(w_1 \cdots w_i) \approx \prod_{k=1}^{i} P(w_k | w_{k-2} w_{k-1}) \tag{1.4}$$

The training process of N-gram models involves counting N-grams within the training corpus. Subsequently, this count is normalized by dividing with the count of the N-grams

that share the same N-1 words, as shown below:

$$P(w_n|w_{n-N+1}\cdots w_{n-1}) = \frac{C(w_{n-N+1}\cdots w_{n-1}w_n)}{C(w_{n-N+1}\cdots w_{n-1})} \tag{1.5}$$

The above equation estimates the probability of occurrence of the word w_n given the word sequence $<w_{n-N+1}$ ... $w_{n-1}>$ [2]. In the case of a trigram model, the equation (2.3) is simplified as following:

$$P(w_n|w_{n-2}w_{n-1}) = \frac{C(w_{n-2}w_{n-1}w_n)}{C(w_{n-2}w_{n-1})} \tag{1.6}$$

1.5 N-gram smoothing

The sparseness of data is a major problem in building a N-gram language model; several word sequences may not occur in a training corpus and, as a result, are given zero probability [3]. This inefficiency in modeling unseen events requires the development of techniques that smooth the models.

A smoothing technique was introduced by Katz [4] in 1987. This technique, the "n-gram back-off model", estimates the probability P($w_n|w_{n-N+1}$ ... w_{n-1}) for a generally unseen sequence w by backing-off to probabilities of lower order N-grams. For instance, the estimation of the probability of a scarce trigram can be reduced to the estimation of the respective bigram. The equation of this back-off model is the following:

$$P(w_n|w_{n-N+1}\cdots w_{n-1}) = d\frac{C(w_{n-N+1}\cdots w_{n-1}w_n)}{C(w_{n-N+1}\cdots w_{n-1})} \tag{1.7}$$

if C(w_{n-N+1} ... w_n) is higher than a positive constant k, or:

$$P(w_n|w_{n-N+1}\cdots w_{n-1}) = aP(w_n|w_{n-N+2}\cdots w_{n-1}) \tag{1.8}$$

if it is lower than k. Term d is a discount factor and a is the back-off weight. Intuitively, if the count of the N-gram is rather high, the respective probability is scaled down by d. If it is low, this probability is boosted by a.

Another N-gram smoothing process described in [1] is the *Interpolation Smoothing*, where a N-gram of high order is linearly interpolated with a N-gram of a lower order. This interpolation enables the incorporation of additional useful information concerning a less frequent N-gram. The resulting probability P_I is expressed as follows:

$$P_I(w_n|w_{n-N+1}\cdots w_{n-1}) = \lambda P(w_n|w_{n-N+1}\cdots w_{n-1}) + (1-\lambda)P_I(w_n|w_{n-N+2}\cdots w_{n-1}) \tag{1.9}$$

where λ is a constant between 0 and 1. High values of λ are advised for N-grams that are ubiquitous within the training corpus, while lower values are applied to N-grams of low occurrence. This happens because highly observed sequences produce a more reliable model, while scarce sequences derive a more 'overfitted' and less generalized model.

1.6 N-gram drawbacks and Continuous Models

Although the above N-gram smoothing techniques alleviate the problem of zero probabilities, there are some other issues that persist even after smoothing; N-gram models may contain a very large number of parameters, considering that each different N-gram within the corpus has to be kept in memory. This fact also referred to as *curse of dimensionality* [5]. As a result, N-grams face extreme computational difficulties, especially in large-vocabulary tasks [11]. The probabilities shown in Part 1.4 reflect both problems of generalizability and adaptability; many possible N-grams are technically left out of the model because of limited data and the number of parameters is very high considering a medium-sized vocabulary.

Another problem of N-grams is the lack of semantic and syntactic correlation; similarity between sentences is not taken into account by N-grams and any similarity is restricted within the boundaries of each N-gram. Examining similar sentences may prove very effective in combating the lack of generalizability of a language model [5].

Therefore, the two main challenges for N-gram models are *generalizability* and *adaptability* of the parameters [11]. An alternative solution to the problems mentioned above is the projection of the words onto the continuous space. This method was initially proposed in [5], where a neural continuous language model is created. The main idea of continuous language models is to represent each word by using continuous-space vectors which follow a certain probability distribution.[11]. A continuous representation of a language model

offers the opportunity to work with larger vocabulary sizes, adapt the model parameters faster and incorporate the notion of similarity among text with highly correlated context. Results in [5] and [6] indicate improvement in perplexity figures where continuous language models are used against back-off N-gram models.

1.7 Gaussian Mixture Models

In [11], *Gaussian Mixture Models (GMMs)* have been used in order to model likelihoods in a continuous-space matter. The major advantage of modeling with GMMs is the efficient parameter adaptation even with limited data available.

A Gaussian Mixture Model defines a (previously unknown) probability density function *p(x)* of the random variable x as the linear combination of J Gaussian probability density functions and has the following form:

$$p(x) = \sum_{j=1}^{J} p(x|j)P_j \tag{1.10}$$

where

$$\sum_{j=1}^{J} P_j = 1, \int_X p(x|j)dx = 1 \tag{1.11}$$

In equation (1.10), each Gaussian density function *p(x|j)* is scaled by the mixture coefficients, *Pj* [9]. The density functions *p(x|j)* are called *components* of the mixture.

1.8 Thesis Objectives and Structure

The present thesis attempts to apply the *Maximum Mutual Information (MMI)* discriminative training technique to continuous Gaussian Mixture Language Models (GMLMs) which are initially built according to the technique described in [11]. The MMI training procedure being used in this thesis was applied to [13], where discriminative training is investigated against *Maximum Likelihood Estimation (MLE)* in HMM parameter estimation for speech recognition tasks. The main incentive of this work is the improvement of

GMLM performance in the same way it was achieved in the training of the acoustic models in [13].

As far as the structure of the thesis is concerned, the second chapter analyses the continuous space models and describes the data preparation procedure as well as dimensionality issues and how they are resolved by using *Linear Discriminant Analysis (LDA)* and *Singular Value Decomposition (SVD)*, the third chapter refers to discriminative training in general, analyses the derivation of the MMI estimation technique, compares it to MLE and refers to additional smoothing techniques, the fourth chapter elaborates on the implementation of MMI training in the GMLMs, presents the *perplexity* metric which is used in measuring the effectiveness of a language model and finally gives the experimental results of the presented method. The final chapter is about future work that can be done in discriminative training of language model parameters.

Chapter 2

Continuous Language Models

As mentioned in the above chapter, Continuous Language Models have showed hopeful improvements when compared to their respective N-gram based models. Previous work on continuous space models as well as a more thorough investigation on such models follows in the present chapter. Subsequently, the formation of data that is used from these models in the current work will be analysed and finally, there will be a reference on Maximum Likelihood Parameter Estimation applied to continuous LMs.

2.1 Previous Work

Initial ventures to express a language model into continuous space were made in [5], where a language model was built by using the structures of a neural network. The words of the vocabulary were expressed as *feature vectors* in $\mathbb{R}^m$, which are vectors of m dimensions and have real values. Subsequently, the probabilities $\mathrm{p}(w_n \mid w_{n-N+1} \ldots w_{n-1})$ of the language model were transformed into functions of the correspondent feature vectors of the words involved. The parameters of these functions were then iteratively estimated by using a neural network.Work on Continuous Language Models has also been done in [8], where a continuous language model was used onto a previously built telephone speech recognition system. The desired model would perform a one-pass recognition and help in creating a lattice that would reduce time complexity. The model parameters in [8] were also estimated by a neural network structure. The results gave improved Word Error Rate figures when compared to back-off n-gram models. The paper of M.Afify et al.[11], which builds a languagel model for speech recognition of the Arabic language, steps on the research made in both [8] and [5] and proceeds in proposing the use of *Gaussian Mixture Models (GMMs)* for the formation of the probability functions of the continuous feature vectors initially proposed in [5]. The GMMs are more adaptive than the neural networks that still have a large number of parameters to be trained [11]. The approach of [11] is used in the current thesis because it seems to overcome both generalizability issues of unseen events and adaptability issues of the model parameters.

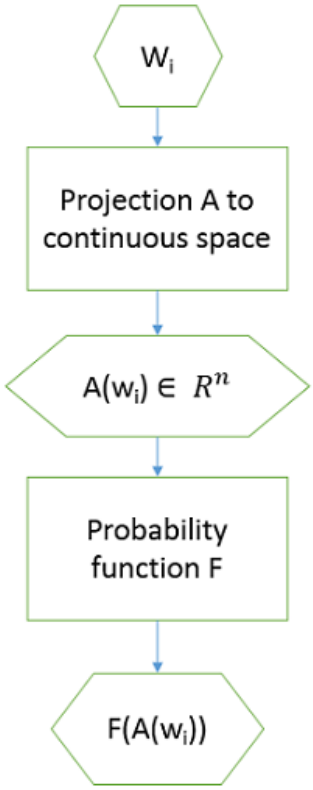

Figure 2.1: Incorporation of discrete variables into a continuous probability function

2.2 Description of a Continuous Language Model

Although there is no formal definition for a Continuous Language Model, we could refer to certain common qualities that each Continuous LM has. The most basic quality of a continuous LM is the continuous-space form of the data of the model. This data is represented by discrete random variables (the words of the vocabulary) that are projected onto continuous-space feature vectors [5].

A second quality of a continuous LM is that there is a function that models each word probability within the LM; the probabilities p($w_n|w_{n-N+1}$ ... w_{n-1}) of each n-th word to be preceded by the word sequence w_{n-N+1}^{n-1} are transformed into functions that map the word feature vectors to the probability space over the words of the vocabulary.

2.3 Data Formation

The above part described the general form of data and models used in continuous-space language modeling. The present section refers to the exact process of shaping the data so as to fit into certain continuous language models. In particular, describes the way in which the data, in form of plain text, is transformed into continuous-space vectors. The involved methods are the *Singular Value Decomposition (SVD)* and the *Linear Discriminant Analysis (LDA)*, which will be analysed further below.

2.3.1 Text Corpus

The text that is used as training and testing data is extracted from the Wall Street Journal '94 (WSJ94) corpus. The punctuation marks (e.g. period, comma, hyphen, doublequote) are written as the word that describes it and not as the respective mark, for example the sentence "The result was a social and economic disaster." is written as "the result was a social and economic disaster period" . The beginning and the end of a sentence are denoted by the special symbols <s> and </s> respectively. The sentence ends either with a period or a questionmark. As a result, the previous sentence will be written in the corpus as follows: "<s> the result was a social and economic disaster period </s>" .

The training corpus of WSJ94 consists of 150000 sentences, whereas the test set contains 6000 sentences.

2.3.2 Initial Word Mapping

The number of different words used in the corpus is rather high. This fact impedes the implementation of a language model analysed above because of the great number of parameters to be estimated for each GMM that corresponds to a single word. Therefore, a reduction of the number of different words, or else the size of the vocabulary, is needed. In order to decrease the vocabulary size from X to V, the frequency of each word in the corpus is estimated and subsequently the X - V less frequent words are notated as an unknown word, or <unk>. In the present work, the words considered as "known" are 2699 plus the unknown word.

What follows is the representation of each word into a vector, which is initially done by assigning an indicator vector to each word. The indicator vector w_i of word i has V dimensions and consists of zeros, except from the i-th dimension, which is equal to one

[11]. As a result, the indicator vector is a $V \times 1$ matrix. This vector will now be referred to as *word vector*.

2.3.3 History Vectors

The language model built in the present work is a trigram, which means that each word in the corpus depends only on its previous two words. These two words constitute the *history* of the word. As a result, the two previous word vectors can be concatenated in order to form the *history vector* of the current word. These history vectors are the data of each word. For instance, the word w_i has a history vector that is a concatenation of the word vectors of w_{i-2} and w_{i-1}:

$$h_{ij} = w_{i-2}w_{i-1} \tag{2.1}$$

where h_{ij} is the j^{th} history vector of the i^{th} word.

Word and history vectors have an excessive number of dimensions, that technically prohibit any computation needed for the language model estimations. The following parts describe the ways in which the word vectors of 2700 dimensions are transformed into vectors of significantly lower number of dimensions.

2.4 Dimensionality Reduction

It is obvious that handling a vector of V dimensions is rather difficult. Thus, mapping indicator vectors to a lower dimension is important to the procedure of building the desired language models. In order to do so, a matrix A of dimensions $M \times V$ is needed, where $M < V$. If we assume that the resulting vector after the reduction of each w_i is u_i, we have:

$$u_i = Aw_i \tag{2.2}$$

2.4.1 Singular Value Decomposition

The Singular Value Decomposition is a matrix factorization method that is used in dimensionality reduction. The SVD method states that every $m \times n$ matrix A is factorized as

follows:

$$A = Q_1 \Sigma Q_2 \tag{2.3}$$

where Q_1 is an orthogonal $m \times m$ matrix, the columns of which are the eigenvectors of the matrix AA^T. Q_2 is an orthogonal $n \times n$ matrix the columns of which are the eigenvectors of the matrix $A^T A$. The matrix Σ is diagonal, has $m \times n$ dimensions and its diagonal consists of the square roots of non-zero eigenvalues (singular values) of both AA^T and $A^T A$ [20].

The dimensionality reduction of A can be done by processing the singular values of Σ; we can reject the singular values that are lower than an arbitrary floor and keep the rest. This rejection is safe because the columns that contain a low singular value do not bear significant information. As a result, only the columns that have a singular value higher than the threshold are kept. For example, we want to maintain 100 singular values. Thus, the Σ matrix will now be a 100×100 matrix, while Q_1 and Q_2 will be m $\times$ 100 and 100 $\times$ 100 matrices respectively. The procedure is shown below:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} q1_{11} & \cdots & q1_{1m} \\ \vdots & \ddots & \vdots \\ q1_{m1} & \cdots & q1_{mm} \end{bmatrix} \begin{bmatrix} \sigma_{11} & \cdots & 0 \\ 0 & \sigma_{22} & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{mm} \end{bmatrix} \begin{bmatrix} q2_{11} & \cdots & q2_{1n} \\ \vdots & \ddots & \vdots \\ q2_{n1} & \cdots & q2_{nn} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & \cdots & a_{1,100} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{m,100} \end{bmatrix} = \begin{bmatrix} q1_{11} & \cdots & q1_{1,100} \\ \vdots & \ddots & \vdots \\ q1_{m1} & \cdots & q1_{m,100} \end{bmatrix} \begin{bmatrix} \sigma_{1,1} & \cdots & 0 \\ 0 & \sigma_{2,2} & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_{100,100} \end{bmatrix} \begin{bmatrix} q2_{11} & \cdots & q2_{1,100} \\ \vdots & \ddots & \vdots \\ q2_{100,1} & \cdots & q2_{100,100} \end{bmatrix}$$

The result is the new m $\times$ 100 matrix A. We should also note that the elements of the diagonal in matrix Σ are already sorted in descending order, hence the retainment of the 100 larger values is a trivial process.

2.4.2 Linear Discriminant Analysis

A dimension reduction can also be possible by using a linear transorfmation of the following form:

$$y = \theta^T x \tag{2.4}$$

where x is the initial matrix that needs the reduction, θ^T is the matrix that performs the linear transformation and y is the resulting matrix with reduced number of dimensions.

Moreover, LDA attempts to reduce dimensionality while preserving the information needed for the discrimination between the classes [21]. Therefore the analysis requires that data has appropriate class labels. We associate words with class labels and assign each observed history vector to the corresponding word class. LDA estimates the between and within class scatters in order to find the optimal transformation matrix.

The first step of the LDA technique is the estimation of the mean vector for each word class j:

$$\mu_j = \frac{1}{N_j} \sum_{i=1}^{N_j} x_i \tag{2.5}$$

where the word classes are equiprobable and x_n is the n-th data point that belongs to the j-th word class. N_j is the number of data points of the class j. The next step is the estimation of the covariance matrix within the word class j:

$$S_j^W = \frac{1}{N_j} \sum_{i=1}^{N_j} (x_i - \mu_j)(x_i - \mu_j)^T \tag{2.6}$$

The estimation of the mean vector and covariance matrix for the whole data set follows:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{2.7}$$

$$S = \frac{1}{N_j} \sum_{i=1}^{N} (x_i - \mu)(x_i - \mu)^T \tag{2.8}$$

The main idea of LDA is to maximize the ratio of between-class variance to the within-class variance in any data set in order to assure maximum separation [10]. The covariance matrix of (2.8) can be considered as the between-class variance because the difference between all the data points of the set and the general mean vector can be thought of the difference between each class and the mean of the entire data. The mean within-class covariance is expressed as:

$$S^W_{mean} = \frac{1}{N} \sum_{i=1}^{N} N_j S^W_j \tag{2.9}$$

The criterion that is maximized by LDA is as follows:

$$\hat{\theta} = \arg\max_{\theta_p} \frac{|\theta_p^T S \theta_p|}{|\theta_p^T S^W_{mean} \theta_p|} \tag{2.10}$$

As a result, the θ that satisfies the condition in (2.10) is chosen as the matrix that conducts the transformation of the form in (2.4).

2.4.3 Usage of SVD and LDA in the present work

The estimation of the matrix A of (2.2) can be carried out by applying the SVD method on the *co-occurrence matrix* of the word document. A co-occurrence matrix E consists of the elements e_{ij}, each of which holds the number of times word i follows word j in the document. This co-occurrence matrix approximates the correlation between each word, i.e. how often a word is followed by another [11].

In our case, the size of this matrix is 2700 $\times$ 2700. The result of applying SVD on the co-occurrence matrix using 100 singular values is a matrix A of dimensions 2700 $\times$ 100. Matrix A then transforms each of the 2700 word vectors by using (4.2), which results in having 2700 word vectors of size 100 $\times$ 1, which is a significant reduction when compared to their initial size (2700 $\times$ 1). If we assume that the initial word vector of each i is

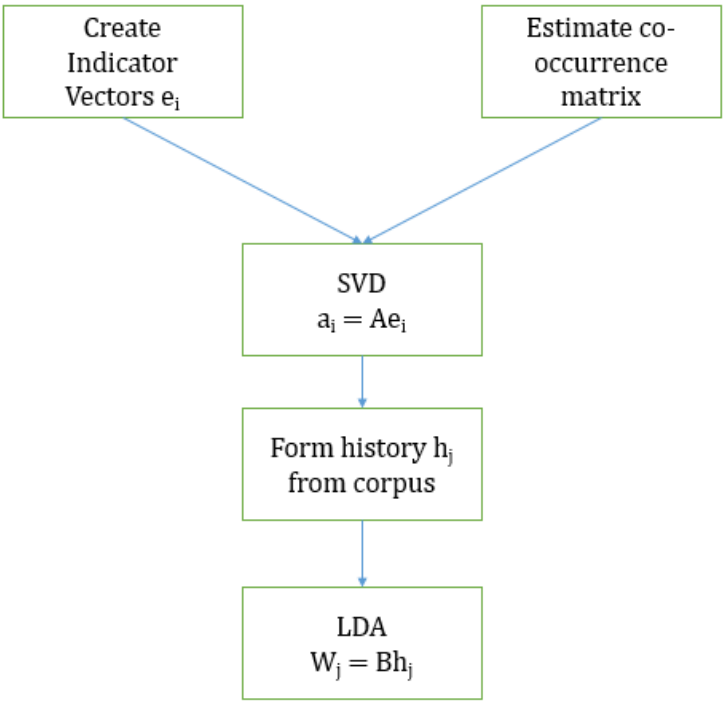

Figure 2.2: The process of data formation

denoted as w_i, the matrix A maps w_i to the new word vectors u_i, according to the linear transformation of (2.2).

A similar dimensionality reduction can be achieved in the history vectors of each word in the corpus. The means to achieve it is by applying LDA; the history vectors, which are now 200×1 vectors (two word vectors concatenated of 100 dimensions each) can be transformed into vectors of lower dimension. As proposed in [11], a linear mapping of the form

$$u_{ij} = Bh_{ij} \tag{2.11}$$

In our case, the above mapping can decrease each of the history vectors h_{ij} dimensions from 200 down to, say, 50. B is a 50×200 matrix, which implies that the resulting mapped history vector u_{ij} will have 50×1 dimensions.

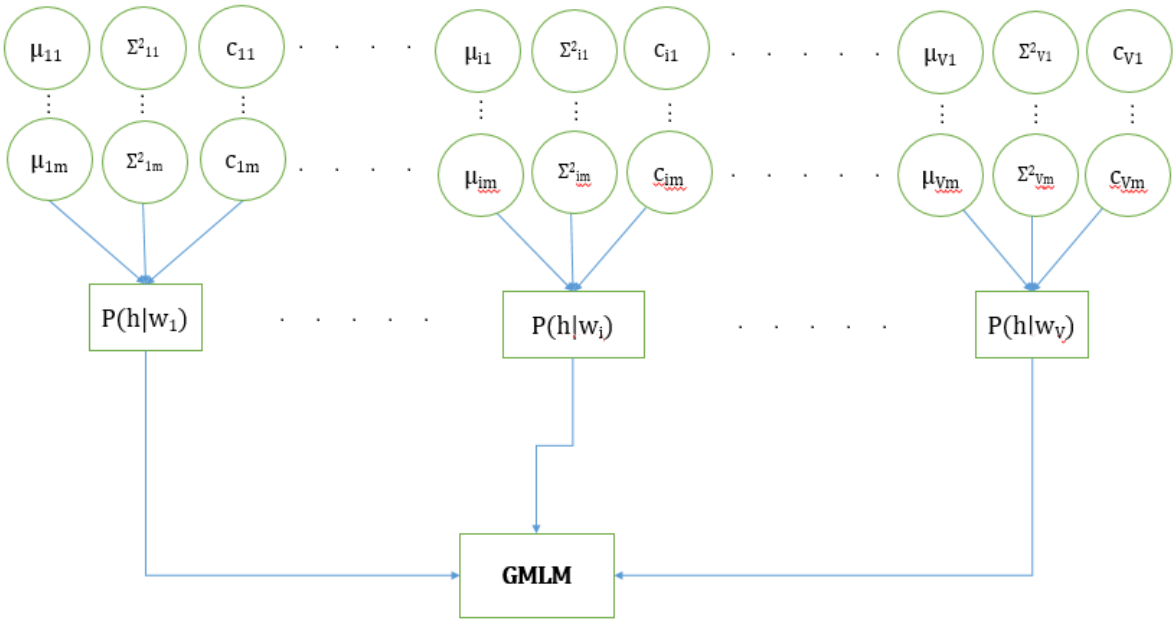

Figure 2.3: Depiction of a GMLM and its components

2.5 Gaussian Mixture Language Models (GMLMs)

The Gaussian distribution is widely used in stochastic processes because it has proven to model effectively the majority of random variables in a very wide spectrum of scientific disciplines. In particular, Gaussian Mixture Models (GMMs), which have been mentioned in part 1.7 were used in continuous-space language models in [11] and [12] with satisfactory results.

The current work uses GMMs in order to model every p(h|w_i) likelihood, where h is the *history vector* that was decribed above and w_i is the i-th word of our vocabulary. There are V different mixtures, where V is the size of the vocabulary, one for each word. The value of p(h|w_i) is then used in order to estimate the posterior p(w_i|h) through the rule of Bayes:

$$p(w_i|h) = \frac{p(w_i)p(h|w_i)}{\sum_j p(w_j)p(h|w_j)} \tag{2.12}$$

This posterior is used in recognition of the following word for a given history (two words in the case of trigram) within the speech segment to be processed.

2.5.1 Maximum Likelihood Estimation of GMLM parameters

The GMLMs of the present thesis, both baseline and newly created, consist of 2700 mixtures, where 2700 = V. Each mixture has m Gaussians each of which j includes a mean vector, μ_{jm}, a covariance matrix σ^2_{jm} and priors c_{jm} that sum to 1 and constitute the weights of each Gaussian in the mixture.

Baseline Language Models train the GMLM parameters with the *Maximum Likelihood Estimation* criterion, utilising the *Expectation-Maximization (EM)* algorithm, which is described in its general form in [18]. EM algorithm for calculating maximum likelihood estimates contains two steps; the *E-step*, where the posterior probabilities p($w_j|h_n$) for each data point n are estimated by using the Bayes rule and the *M-step*, where the likelihoods p(h|w_j) are maximized with respect to means, covariances and weights and, by using the posterior probabilities estimated in E-step, yield the following update equations [14]:

$$\mu_{jm} = \frac{1}{N_{jm}} \sum_{n=1}^{N_j} p(w_j|h_n) h_n \tag{2.13}$$

$$\sigma^2_{jm} = \frac{1}{N_{jm}} \sum_{n=1}^{N_j} p(w_j|h_n) h_n (h_n - \mu_{jm})(h_n - \mu_{jm})^T \tag{2.14}$$

$$c_{jm} = \frac{N_{jm}}{N_j} \tag{2.15}$$

where N_{jm} is the total count of data points of the m-th component of the j-th mixture, while N_j is the total count of the data points of the j-th mixture. During a pass of the algorithm, the first step is the evaluation of p($w_j|h_n$), which is then fitted into the equations of (2.13) to (2.15). The process is repeated until the value of log-likelihood ln p(H|w_j) = $\sum_n$ ln p($h_n|w_j$) converges.

Chapter 3

Discriminative Training

3.1 Introduction

In general, a Discriminative training technique tries to make the correct hypothesis more probable and, at the same time, tries to penalize any incorrect competing hypotheses. Thus, the method aims at finding a suitable objective function to be optimized according to specific requirements being set from this method. Such objective functions depend on the parameters that are to be trained and are also called *training criteria*. *Maximum Mutual Information (MMI)* and *Minimum Classification Error (MCE)* have been the most used discriminative training criteria [13]. This chapter deals with elaborating on the MMI training method.

3.2 Discriminative criteria

The training criterion being used in the present work is the MMI, but there are some other proposed criteria which are worth to be mentioned, such as the *Minimum Classification Error (MCE)* [22] and the *Minimum Phone Error (MPE)* along with its extension, the *Minimum Word Error (MWE)*, which are introduced in [13].

3.2.1 Minimum Classification Error

The task on which MCE focuses is the isolated word recognition. The main idea of MCE objective function is that each utterance (observation) O that occurs during the speech recognition process corresponds to one of a fixed number of classes $i = 1 \ldots M$. The conditional likelihood $g_i(O;\lambda)$ of each class i is defined as follows:

$$g_i(O;\lambda) = \log p(O|M_i,\lambda) \tag{3.1}$$

where M_i is the model that corresponds to the i-th class and λ represents the model parameters. In addition, a misclassification measure is also defined:

$$d_i(O) = -g_i(O;\lambda) + \log(\frac{1}{M-1}\sum_{j,j\neq i}\exp g_j(O;\lambda)\eta)^{1/\eta} \tag{3.2}$$

The above measure gets positive values when the observation O is not classified as belonging to the i-th class and negative values when being classified in i-th class. These $d_i(O)$ values are incorporated into sigmoid functions, as shown below:

$$l_i(O) = \frac{1}{1+\exp(-\gamma d_i(O))} \tag{3.3}$$

The factors η and γ are positive constants. The objective function of MCE is formed by the sum of $l_i(O)$ over all the correct classes. By definition, the sigmoid factors $l_i(O)$ of (3.3) are equal to zero when $d_i(O)$ values are negative and equal to one when they are positive. Therefore, the objective function for an utterance that was recognised correctly is equal to zero, while misclassified utterances set the objective function value to one. It is obvious that the minimisation of the objective function is required in order to increase the recognition accuracy of the system.

3.2.2 Minimum Phone Error

The Minimum Phone Error (MPE) criterion, which was originally proposed in [13], involves maximising the phone transcription accuracy of a speech recognition system. The objective function of MPE is the following:

$$F_{MPE}(\lambda) = \sum_{r=1}^{R}\sum_{s} P_\lambda^\kappa(s|O_r)A(s,s_r) \tag{3.4}$$

where λ represents the model parameters, s is the current sentence(observation) and $A(s,s_r)$ is the raw phone transcription accuracy of the sentence s given the reference sentence s_r, which equals the number of reference phones minus the number of errors. κ is a constant that scales down the posterior probability of the sentence s being the correct one.

The objective function of (3.4) can be expanded as follows:

$$F_{MPE}(\lambda) = \sum_{r=1}^{R} \frac{\sum_s p_\lambda(O_r|s)^\kappa P(s)^\kappa A(s, s_r)}{\sum_u p_\lambda(O_r|u)^\kappa P(u)^\kappa} \tag{3.5}$$

From the above, it is obvious that the MPE objective function is an average of the raw phone accuracy *A(s,s$_r$)* over all possible sentences s weighted by the $p_\lambda(O_r|\mathrm{s})$ terms, which express the likelihood of observation O_r given the model of sentence s.

In general, the maximisation of the MPE criterion tries to make the more accurate transcriptions more probable than their competitors. Specifically, the term *A(s,s$_r$)*, which is in the numerator of the criterion of (3.5), is the main target of this maximisation; sentences that have higher values of *A(s,s$_r$)* are given higher probability than sentences of lower phone accuracy.

As far as the likelihood scaling is concerned, the scaling factor κ accomodates the smoothing of the MPE criterion, in order to make it more differentiable. In this way, less probable sentences also contribute to the MPE objective function along with the sentences that have been given higher probability. In general, high values of κ tend to exclude less likely sentences from the objective function, while lower values tend to incorporate them more into the criterion, thus rendering the system more flexible to unseen events (generalisation).

3.2.3 Minimum Word Error

The Minimum Word Error criterion is identical to the Minimum Phone Error objective function, with the only difference being the transcription accuracy *A(s,s$_r$)*; in MWE, it is measured in words instead of phones. MWE is expected to be more effective in terms of word recognition because it maximizes the word accuracy and, as a result, is a better approximation to the word error [13].

3.3 MMI Training

The MMI theoretical basis can be traced back to Information Theory. The measure of uncertainty of a random event, also called *entropy* [15], is as follows:

$$H(X) = -\sum_{X} P(X) log P(X) \tag{3.6}$$

if the random variable that models the event is assumed to be X. In the same way, the *conditional entropy* of a random event X given the observed event Y is defined in [16] as:

$$H(X|Y) = -\sum_{X,Y} P(X,Y) log P(X|Y) \tag{3.7}$$

The average amount of information that Y provides for X could be empirically quantified as the difference between the entropy *H(X)* and the conditional entropy *H(X| Y)*. Let this difference be notated as *I(X;Y)*:

$$I(X;Y) = H(X) - H(X|Y) \tag{3.8}$$

The above equation can be expressed in probability and log-likelihood terms as follows:

$$\begin{aligned}
I(X;Y) &= H(X) - H(X|Y) \\
&= -\sum_{X} P(X) log P(X) - (-\sum_{X,Y} P(X,Y) log P(X|Y)) \\
&= -\sum_{X,Y} P(X,Y) log P(X) + \sum_{X,Y} P(X,Y) log P(X|Y) \\
&= -\sum_{X,Y} P(X,Y)(log P(X) - log P(X|Y)) \\
&= \sum_{X,Y} P(X,Y) log \frac{P(X|Y)}{P(X)} \\
&= \sum_{X,Y} P(X,Y) log \frac{P(Y|X)}{P(Y)}
\end{aligned}$$

3.3.1 MMI Objective function

In general, an objective function is a criterion that needs to be maximized or minimized in order to train HMM parameters optimally. [13]. The objective function of the MMI

training procedure could be derived from the form of *I(X;Y)* which was described in the previous section, because the maximization of this term is equivalent to the minimization of the entropy *H(X|Y)*. This fact means that the uncertainty of variable X given the observed variable Y is reduced.

Thus, the objective function for MMI Estimation used in [13] is produced by making the assumption that the $\log\frac{P(Y|X)}{P(Y)}$ term is sufficient for parameter estimation, since *P(X,Y)* is unknown [16].

The random variable Y mentioned above is the variable being observed, whereas the random variable X is the one that models the "actual event". This means that X represents the correct transcriptions in speech recognition tasks. If O are the observations, s_r are the correct transcriptions and λ is the parameter set of the HMM to be estimated, the MMI objective function is as follows:

$$F_{MMI}(\lambda) = \sum_{r=1}^{R} log\frac{P_\lambda(O_r|s_r)P(s_r)}{\sum_s P_\lambda(O_r|s)P(s)} \tag{3.9}$$

where $P_\lambda(O_r|s_r)$ is the likelihood of the observations O_r given the HMM that derives the correct transcriptions s_r and consists of the parameter set λ. The $P(s_r)$ term is the probability of the sentence s_r, provided by a language model. The denominator in equation (3.9) was derived from *P(O_r)* by applying the law of total probability over all possible transcriptions s.

3.3.2 Why MMI?

While MPE has reportedly better results than MMI in acoustic models built in [13], an MPE implementation of the present thesis would require the incorporation of phoneme structure into the model. This fact makes the MPE method a less strong 'candidate' for the desired task. MCE is closer to MMI in terms of competing hypotheses distinction, however there has been less work that would allow a precise implementation of a language model.

3.3.3 Comparison with Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) has been the most used training procedure for HMM parameters so far. MLE attempts to maximize the likelihood of the observations,

which was referred to as $P_\lambda(O_r|s_r)$ above. As a result, the MLE objective function is:

$$F_{MLE}(\lambda) = \sum_{r=1}^{R} logP_\lambda(O_r|s_r) \tag{3.10}$$

MLE derives models that can be easily trained by using the globally convergent Baum-Welch algorithm [17]. This fact has made it the predominant method for HMM parameter estimation. However, MLE assumes that the training data follow a certain probability distribution. In the (realistic) case where the data do not generally follow this distribution, MLE will not perform as desired [16]. Additionally, in order to truly achieve full compliance with a specific distribution, a virtually infinite amount of data will be needed.

The assumption of a false distribution in MLE could be alleviated by the simultaneous maximization of the mutual information between the observations and the actual transcriptions [16]. Hence, MMIE is a way to theoretically improve the performance of a recognition system trained by using MLE methods. One obvious fact about the two estimation methods is that their objective functions have a common numerator, the $P_\lambda(O_r|s_r)$ term (if we assume that the $P(s_r)$ term does not affect the optimization of the objective function). The difference between the two lies in the denominator of the MMI objective function, which consists of the sum of the common probabilties between the current observation and any possible transcription s. As a result, the maximization of $F_{MMI}(\lambda)$ implies the minimization of $P_\lambda(O_r|\mathrm{s})$ for every false transcription s, which means that the wrong hypotheses will become less probable against the correct hypothesis s_r [16].

3.4 Optimization

The maximization of the MMI objective function is the key to the optimal parameter estimation. The general method having been used for ML HMM parameter estimation is the *Expectation-Maximization (EM) algorithm,* which consists of two steps; the E-Step, where statistics are gathered in order to estimate the unknown distributions and the M-Step, where the likelihoods of the previously estimated distributions are maximized in order to find the optimal parameters for the distribution [18]. In most cases, this method is convergent for MLE tasks.

However, the optimization of discriminative criteria is quite different. A rational function, the discriminative objective function, is to be maximized. Among the proposed

solutions to the problem of optimizing discriminative criteria is the *Extended Baum-Welch (EB)* algorithm, the update equations of which are easy to adapt to the parameter estimation [13]. The process of building the update equations according to the EB algorithm relies heavily on the use of *auxiliary functions*, which help in optimizing the discriminative criteria, as the one in Equation (3.9).

3.4.1 Auxiliary Functions

The smooth function *G(λ, λ')* is called a *strong-sense auxiliary function* around λ' of the *F(λ)* function that is to be maximized if the following inequality holds:

$$G(\lambda, \lambda') - G(\lambda', \lambda') \leq F(\lambda) - F(\lambda') \tag{3.11}$$

If there is a maximum of the auxiliary function *G(λ, λ')* with respect to λ, then it is ensured that this maximum will also increase the "main" function *F(λ)*, as implied by the inequality (3.11). Therefore, applying iterative maximization on the auxiliary function leads to finding a local maximum of the function *F(λ)* [13].

A more "loose" version of auxiliary function is the *weak-sense auxiliary function*. Such functions are useful in cases of inability to optimize some terms of a strong-sense auxiliary function. According to [13], a weak-sense auxiliary function *G(λ, λ')* for function *F(λ)* around λ' has the following property:

$$\frac{\partial}{\partial\lambda} G(\lambda, \lambda') = \frac{\partial}{\partial\lambda} F(\lambda) \tag{3.12}$$

around the point $\lambda = \lambda'$. Although the maximization of the auxiliary function does not now imply that *F(λ)* increases, the covergence (if achieved) to a particular λ after a number of iterations guarrantees that *F(λ)* has reached a local maximum.

3.4.2 Smoothing functions

A *smoothing function G(λ,λ')* has the following property:

$$G(\lambda, \lambda') \leq G(\lambda', \lambda') \tag{3.13}$$

This kind of function reaches its maximum point in $\lambda = \lambda$'. This property is useful in creating strong-sense auxiliary functions. For example, if a smoothing function is added to an objective function, then an auxiliary function is generated, because the maximum of the smoothing function is at point λ and the inequality of (3.11) holds. Adding a smoothing function to a weak-sense auxiliary function can also help in the optimization process, as it will be explained in the following part.

3.4.3 MLE auxiliary functions

The MLE objective function (Eq. 3.10) is closely related to the form of the MMIE objective function (Eq. 3.9) because $F_{MMI}(\lambda)$ is a difference of log likelihoods, whose form is similar to the log likelihood of $F_{MLE}(\lambda)$. Hence, the derivation of the MLE auxiliary function can be useful for the derivation of the respective MMI auxiliary function.

As mentioned above, the objective function of Maximum Likelihood Estimation is a sum of log-likelihoods of the observation given the correct transcription, symbolized as $P(O|s_r,\lambda)$. Assuming that the HMM to be estimated is denoted as M and that each HMM corresponds to each correct transcription, $P(O|s_r,\lambda)$ can be expressed as $P(O|M,\lambda)$, which is equal to the sum $\sum_x P(O|M,\lambda,x)$, where x is any potential state sequence within the HMM. If we denote the likelihood $P(O|M,\lambda,x)$ as $f_x(\lambda)$, a log-likelihood term within the sum of the MLE objective function will now be:

$$F(\lambda) = log \sum_x f_x(\lambda) \tag{3.14}$$

which is the function to be maximized.

A strong-sense auxiliary function $G(\lambda,\lambda')$ for function (2.9) around the point $\lambda = \lambda$' is appropriate if the inequality of (3.11) holds. The auxiliary function used in [13] is the posterior probability of the state sequence x and has the following form:

$$G(\lambda, \lambda') = \sum_x \frac{f_x(\lambda')}{\sum_y f_y(\lambda')} \log f_x(\lambda) \tag{3.15}$$

The auxiliary function of (3.15) can be written as:

$$G(\lambda,\lambda') = \sum_x \frac{f_x(\lambda')}{\sum_y f_y(\lambda')} \log \sum_y f_y(\lambda) + \sum_x \frac{f_x(\lambda')}{\sum_y f_y(\lambda')} \log \frac{f_x(\lambda)}{\sum_y f_y(\lambda)} \tag{3.16}$$

$$= \log \sum_y f_y(\lambda) + \sum_x \frac{f_x(\lambda')}{\sum_y f_y(\lambda')} \log \frac{f_x(\lambda)}{\sum_y f_y(\lambda)} \tag{3.17}$$

Given the expression of (3.17), the inequality of (3.11) reduces to:

$$G(\lambda,\lambda') - G(\lambda',\lambda') \leq F(\lambda,\lambda') - F(\lambda',\lambda') \tag{3.18}$$

$$\Rightarrow \log \sum_y f_y(\lambda) - \log \sum_y f_y(\lambda') + \sum_x \frac{f_x(\lambda')}{\sum_y f_y(\lambda')} (\log \frac{f_x(\lambda)}{\sum_y f_y(\lambda)} - \log \frac{f_x(\lambda')}{\sum_y f_y(\lambda')}) \tag{3.19}$$

$$\leq \log \sum_x f_x(\lambda) - \log \sum_x f_x(\lambda') \tag{3.20}$$

$$\Rightarrow \sum_x \frac{f_x(\lambda')}{\sum_y f_y(\lambda')} (\log \frac{f_x(\lambda)}{\sum_y f_y(\lambda)} - \log \frac{f_x(\lambda')}{\sum_y f_y(\lambda')}) \leq 0 \tag{3.21}$$

The inequality of (3.21) has the form: $\sum_x$ p(x) log $\frac{q(x)}{p(x)} \leq 0$, from which we get: $\sum_x$ p(x) ($\frac{q(x)}{p(x)}$ - 1) $\leq 0 \Rightarrow \sum_x$ (p(x) - q(x)) $\leq 0 \Rightarrow 0 \leq 0$, because *p(x)* and *q(x)* sum to one [13]. As a result, the function of (3.15) is a strong-sense auxiliary function for $F(\lambda)$.

For convenience, the above auxiliary function can be expressed as the sum over all the Gaussians in the HMM. If the sum over the posterior probabilities $\frac{f_x(\lambda')}{\sum_y f_y(\lambda')}$ of every state sequence x that contains state j at time t is denoted as $\gamma_j(t)$, the auxiliary function now becomes:

$$G(\lambda,\lambda') = \sum_{j=1}^{J} \sum_{t=1}^{T} \gamma_j(t) \log \mathcal{N}(o(t)|\mu_j,\sigma_j^2) \tag{3.22}$$

The term *o(t)* is the data observed at time *t*. The above case assumes that one Gaussian per state is used. A different case is the use of a GMM, as it occurs in the present thesis. For the sake of simplicity though, we will stick to the case of a single Gaussian per state for the time being.

By replacing the term $\mathcal{N}$ with the full expression of the Gaussian pdf, the auxiliary function of (3.22) becomes:

$$G(\lambda, \lambda') = \sum_{j=1}^{J} -\frac{1}{2}(\gamma_j \log(2\pi\sigma^2) + \frac{\theta_j(O^2) - 2\theta_j(O)\mu_j + \gamma_j\mu_j^2}{\sigma_j^2}) \tag{3.23}$$

$$= \sum_{j=1}^{J} Q(\gamma_j, \theta_j(O), \theta_j(O^2)|\mu_j, \sigma_j^2) \tag{3.24}$$

where:

$$\theta_j(O) = \sum_{t=1}^{T} \gamma_j(t)o(t) \tag{3.25}$$

$$\theta_j(O^2) = \sum_{t=1}^{T} \gamma_j(t)o(t)^2 \tag{3.26}$$

$$\gamma_j = \sum_{t=1}^{T} \gamma_j(t) \tag{3.27}$$

and Q($\cdots$) has the form:

$$Q(t, X, S|\mu, \sigma^2) = -\frac{1}{2}(t \log(2\pi\sigma^2) + \frac{S - 2X\mu + t\mu^2}{\sigma^2}) \tag{3.28}$$

where t corresponds to γ_j, X corresponds to $\theta_j(O)$ and S is $\theta_j(O^2)$. As a result, the function that will be maximized for the MLE is the one in equation (3.28). Having maximized Q for each state j leads to the maximization of the whole auxiliary function in (3.24).

3.4.4 MMI Estimation auxiliary function

The MMI objective function in Equation (3.9) has a numerator which includes the likelihood of the observed data given the correct transcription, $P(O_r|s_r)$, and a denominator that contains all the respective likelihoods for every possible transcription, $\sum_s P(O_r|s)$. If we assume that the HMM that derives the correct transcription is M^{num} and M^{den} is the HMM that produces all possible transcriptions, the MMI objective function will be: $F(\lambda) = logP(O|M^{num},\lambda) - logP(O|M^{den},\lambda)$. It is obvious that both likelihoods resemble the likelihood of the MLE objective function. Therefore, the same auxiliary function can

be used for both the numerator and the denominator likelihood. As a result, we get two strong-sense auxiliary functions, $G_{num}(\lambda,\lambda')$ and $G_{den}(\lambda,\lambda')$ which have the form of the (3.15) equation.

Thus, the difference $G_{num}(\lambda,\lambda')$ - $G_{den}(\lambda,\lambda')$ could serve as an auxiliary function of MMI objective function. However, it is no longer strong-sense because the condition of (3.13) is not satisfied, but it still is a weak-sense auxiliary function [13]. Another issue that arises is the convexity of the "candidate" auxiliary function $G_{num}(\lambda,\lambda')$ - $G_{den}(\lambda,\lambda')$; the areas where γ_j^{num} is smaller than γ_j^{den} are generally concave and, as a result, difficult to maximize. The solution to this problem is the addition of a smoothing function $G^{sm}(\lambda,\lambda')$ which has a differential equal to zero around λ' [13]. In this way, the position of the local optima will not be affected and the outcome will still be a weak-sense auxiliary function as follows:

$$G(\lambda,\lambda') = G^{num}(\lambda,\lambda') - G^{den}(\lambda,\lambda') + G^{sm}(\lambda,\lambda') \tag{3.29}$$

The smoothing function that could be used might have the form:

$$G^{sm}(\lambda,\lambda') = \sum_{j}^{J} Q(D_j, D_j\mu_j', D_j(\mu_j'^2 + \sigma_j'^2)|\mu_j,\sigma_j^2) \tag{3.30}$$

The function in (3.30) is of the same form of the one in (3.28). The D_j terms are positive constants that scale the contribution of the smoothing function G^{sm} to the total auxiliary function. It reaches a maximum as every log- Gaussian pdf does and as a result it has a zero differential with respect to both μ_j' and $\sigma_j'^2$, which satisfies the condition of the weak-sense auxiliary function.

The form of the auxiliary function for the MMI objective function around $\lambda = \lambda$' will eventually be:

$$\begin{aligned} G(\lambda,\lambda') = \sum_{j=1}^{J} (&Q(\gamma_j^{num}, \theta_j^{num}(O), \theta_j^{num}(O^2)|\mu_j,\sigma_j^2) - Q(\gamma_j^{den}, \theta_j^{den}(O), \theta_j^{den}(O^2)|\mu_j,\sigma_j^2) \\ &+ Q(D_j, D_j\mu_j', D_j(\mu_j'^2 + \sigma_j'^2)|\mu_j,\sigma_j^2)) \end{aligned}$$

The procedure described above assumed a single Gaussian per HMM state. The current

thesis is though using Gaussian Mixture Models. This fact does not change the above analysis because every function that refers to the state j of the HMM can in the same way refer to the GMM of the state j and the component m of this GMM. Thus, γ_j^{num}, γ_j^{den}, $\theta_j^{num}(\mathrm{O})$, $\theta_j^{den}(\mathrm{O})$, $\theta_j^{num}(O^2)$ and $\theta_j^{den}(O^2)$ will now be denoted as: γ_{jm}^{num}, γ_{jm}^{den}, $\theta_{jm}^{num}(\mathrm{O})$, $\theta_{jm}^{den}(\mathrm{O})$, $\theta_{jm}^{num}(O^2)$ and $\theta_{jm}^{den}(O^2)$.

3.4.5 Update equations for MMI Training

The means and the covariance matrices can now be estimated by maximizing the auxiliary function *G(λ,λ')*. This maximization leads to the Extended Baum-Welch update equations below:

$$\mu_{jm} = \frac{\theta_{jm}^{num}(O) - \theta_{jm}^{den}(O) + D_{jm}\mu'_{jm}}{\gamma_{jm}^{num} - \gamma_{jm}^{den} + D_{jm}} \tag{3.31}$$

$$\sigma_{jm}^2 = \frac{\theta_{jm}^{num}(O^2) - \theta_{jm}^{den}(O^2) + D_{jm}(\sigma_{jm}^{'2} + \mu_{jm}^{'2})}{\gamma_{jm}^{num} - \gamma_{jm}^{den} + D_{jm}} - \mu_{jm}^2 \tag{3.32}$$

where μ'_{jm} is the old means vector and $\sigma_{jm}^{'2}$ is the old covariance matrix of the distribution of m-th component of the j-th state. (3.31) and (3.32) equations are applied iteratively until they converge.

3.4.6 I-smoothing

In [13], a variation concerning the incorporation of priors over the parameters of each Gaussian into the auxiliary function for the MMI objective function is also proposed. The main idea is the use of prior log-likelihoods over means and covariances of the numerator term which have the following form:

$$\log p(\mu, \sigma^2) = Q(\tau, \tau\mu_{prior}, \tau(\sigma_{prior}^2 + \mu_{prior}^2)|\mu, \sigma^2) \tag{3.33}$$

The above equation has the same form as the $\mathrm{Q}(\cdots)$ function defined in (3.28). The constant τ adjusts the contribution of this prior distribution to the total auxiliary function.

In particular, I-smoothing, which is the variation being proposed in [13], uses the (3.33) equation as follows:

$$\log p(\mu, \sigma^2) = Q(\tau^I, \tau^I \frac{\theta_{jm}^{num}(O)}{\gamma_{jm}^{num}}, \tau^I \frac{\theta_{jm}^{num}(O^2)}{\gamma_{jm}^{num}} | \mu_{jm}, \sigma_{jm}^2) \tag{3.34}$$

If the prior log likelihood of (3.34) is added to the auxiliary function *G(λ,λ')*, the γ and θ terms for the numerator will be:

$$\gamma_{jm}^{num'} = \gamma_{jm}^{num} + \tau^I \tag{3.35}$$

$$\theta_{jm}^{num}(O)' = \theta_{jm}^{num}(O) \frac{\gamma_{jm}^{num} + \tau^I}{\gamma_{jm}^{num}} \tag{3.36}$$

$$\theta_{jm}^{num}(O^2)' = \theta_{jm}^{num}(O^2) \frac{\gamma_{jm}^{num} + \tau^I}{\gamma_{jm}^{num}} \tag{3.37}$$

It is obvious that higher values of τ^I mean that the priors have more influence on the target models. In practice, these prior estimates of means and covariances can be obtained by using the corresponding ML parameters for the j-th state and the m-th component. The update equations for the MMI training can now be expressed as [19]:

$$\mu_{jm} = \frac{\theta_{jm}^{num} - \theta_{jm}^{den} + D_{jm}\mu'_{jm} + \tau^I \mu_{jm}^P}{\gamma_{jm}^{num} - \gamma_{jm}^{den} + D_{jm} + \tau^I} \tag{3.38}$$

$$\sigma_{jm}^2 = \frac{\theta_{jm}^{num}(O^2) - \theta_{jm}^{den}(O^2) + D_{jm}(\sigma'^2_{jm} + \mu'^2_{jm}) + \tau^I({\sigma_{jm}^2}^P + {\mu_{jm}^2}^P)}{\gamma_{jm}^{num} - \gamma_{jm}^{den} + D_{jm} + \tau^I} \tag{3.39}$$

The terms μ_{jm}^P and ${\sigma_{jm}^2}^P$ are the prior ML estimates of the means and covariance matrices of the Gaussian of the j-th state and the m-th component. These estimates are theoretically drawn from the a distribution of the form in (3.34). In the experiments of the present work, the statistics of the correspondent baseline GMMs will be used, but this is to be analysed further in the respective chapter.

3.4.7 Weights update equation

As previously mentioned, the intuitive weak-sense auxiliary function for the MMI objective function is the $G^{num}(\lambda,\lambda')$ - $G^{den}(\lambda,\lambda')$. In the case of a Gaussian Mixture Model instead of a single Gaussian per state, the term (both for numerator and denominator) $G(\lambda,\lambda')$ as previously expressed in (3.22) will now be:

$$G(\lambda,\lambda') = \sum_{j=1}^{J}\sum_{t=1}^{T}\sum_{m=1}^{M}\gamma_{jm}(t)\log(\mathcal{N}(o(t)|\mu_{jm},\sigma_{jm}^2)c_{jm}) \tag{3.40}$$

$$= \sum_{j=1}^{J}\sum_{t=1}^{T}\sum_{m=1}^{M}\gamma_{jm}(t)(\log(\mathcal{N}(o(t)|\mu_{jm},\sigma_{jm}^2) + \log c_{jm}) \tag{3.41}$$

The c_{jm} terms are the m weights of each Gaussian in the j-th GMM, being subject to the sum-to-one constraint, which means that $\sum_{m=1}^{M} c_{jm} = 1$ for each j. Considering the form of $G(\lambda,\lambda')$ in (3.41), the auxiliary function for the MMI objective function now becomes:

$$G_{MMI}(\lambda,\lambda') = \sum_{j=1}^{J}\sum_{t=1}^{T}\sum_{m=1}^{M}(\gamma_{jm}^{num}(t)(\log(\mathcal{N}(o(t)|\mu_{jm},\sigma_{jm}^2) + \log c_{jm}) - \gamma_{jm}^{den}(t)(\log(\mathcal{N}(o(t)|\mu_{jm},\sigma_{jm}^2) + \log c_{jm})$$

Hence, the terms including the weights in the above auxiliary function are:

$$\sum_{m=1}^{M}(\gamma_{jm}^{num} - \gamma_{jm}^{den})\log c_{jm} \tag{3.42}$$

In order to estimate the weights c_{jm}, it is enough to maximize the term in (3.42). However, we need an iterative update equation which uses the values previously estimated so as to calculate the values of the current iteration. (3.42) has the same differential with respect to c_{jm} at point $c_{jm} = c_{jm}^{'}$ with the following function [13]:

$$G(\lambda, \lambda') = \sum_{m=1}^{M} \gamma_{jm}^{num} \log c_{jm} - \frac{\gamma_{jm}^{den}}{c_{jm}^{'}} c_{jm} \tag{3.43}$$

The terms $c_{jm}^{'}$ are the original values of weights. As a result, the auxiliary function in (3.43) is a weak-sense auxiliary function for the function in (3.42). Nevertheless, it is necessary to add a smoothing function to the auxiliary function of (3.44) so as to make it differentiable in areas that may be concave, in a similar fashion to the addition of a smoothing function in section 3.4.4. A smoothing function that can be used is the following:

$$G_{sm} = \sum_{m=1}^{M} k_{jm} (c_{jm}^{(p)} \log c_{jm}^{(p+1)} - c_{jm}^{(p+1)}) \tag{3.44}$$

The terms $c_{jm}^{(p)}$ are the weights of the current (p) iteration, while the terms $c_{jm}^{(p+1)}$ are the weights of the next (p+1) iteration. The k_{jm} terms are positive constants. G_{sm} is a smoothing function, because its differential w.r.t. $c_{jm}^{(p+1)}$ equals to zero at the point $c_{jm}^{(p+1)}$ $= c_{jm}^{(p)}$, which means that it reaches a maximum at the desired point. After the addition of the smoothing function, the auxiliary function becomes:

$$G(\lambda, \lambda') = \sum_{m=1}^{M} \gamma_{jm}^{num} \log c_{jm}^{(p+1)} - \frac{\gamma_{jm}^{den}}{c_{jm}^{'}} c_{jm}^{(p+1)} + k_{jm} (c_{jm}^{(p)} \log c_{jm}^{(p+1)} - c_{jm}^{(p+1)}) \tag{3.45}$$

As stated in [13], constants k_{jm} should result in having all coefficients -$\frac{\gamma_{jm}^{den}}{c_{jm}^{'}}$ - k_{jm} of the term $c_{jm}^{(p+1)}$ equal to a constant value for each m. In this way, the term $\sum_{m=1}^{M}$ (-$\frac{\gamma_{jm}^{den}}{c_{jm}^{'}}$ - k_{jm}) $c_{jm}^{(p+1)}$ is independent of the m component of the mixture and, because of the sum-to-one constraint for c_{jm}, it is a constant value for each m. After applying the Baum-Welch algorithm, the update equation for the weights is proved to be:

$$c_{jm}^{(p+1)} = \frac{\gamma_{jm}^{num} + k_{jm} c_{jm}^{(p)}}{\sum_{m=1}^{M} \gamma_{jm}^{num} + k_{jm} c_{jm}^{(p)}} \tag{3.46}$$

The constant k_{jm} is set to be:

$$k_{jm} = (\max_m \frac{\gamma_{jm}^{den}}{c'_{jm}}) - \frac{\gamma_{jm}^{den}}{c'_{jm}} \tag{3.47}$$

In general,the convergence of (3.46) is slower than the update equations for means and covariances mentioned in part 3.4.5. In some cases, 100 iterations are necessary for the values to converge.

3.4.8 Smoothing weights with priors

Prior information of weights might prove useful in case of insufficient training data; their incorporation into the auxiliary function for the MMI objective function is implemented in the same way that the priors of means and covariances did: An appropriate prior distribution of weights is chosen and subsequently it is added to the log-expressed MMI auxiliary function. As mentioned in [13], this prior distibution is $\sum_m \tau^W \frac{\gamma_{jm}^{num}}{\sum_m \gamma_{jm}^{num}} \log c_{jm}$. This distribution is maximized at the same point where c_{jm} equals the Maximum Likelihood estimate. The constant τ^W sets the width of the prior, serving the same purpose as τ^I for the means and covariance smoothing. The only factor affecting the update equation of weights is γ_{jm}^{num}, as indicated in (3.46). After the addition of the weight prior, this factor will be:

$$\gamma_{jm}^{'num} = \gamma_{jm}^{num} + \frac{\tau^W \gamma_{jm}^{num}}{\sum_m \gamma_{jm}^{num}} \tag{3.48}$$

This results in having the following update equation for weights [19]:

$$c_{jm}^{(p+1)} = \frac{\gamma_{jm}^{num} + k_{jm} c_{jm}^{(p)} + \tau^W c_{jm}^P}{\sum_m \gamma_{jm}^{num} + k_{jm} c_{jm}^{(p)} + \tau^W c_{jm}^P} \tag{3.49}$$

The term c_{jm}^P, which is equal to $\frac{\gamma_{jm}^{num}}{\sum_m \gamma_{jm}^{num}}$, is the prior estimate of weight c_{jm}. In this work, this estimate is obtained by the baseline GMMs.

3.5 Adaptation to the Thesis Objectives

The above analysis referred to a Maximum Mutual Information estimation method which estimates the parameters (means, covariances, weights) of a HMM built for speech recognition tasks. The present work attempts to extend this analysis to non-acoustic data in order to train a continuous-space language model. The baseline language models, to which the discriminatively trained language models are compared, were built by using the HRest tool of the HTK toolkit, which performs Baum-Welch re-estimation of the parameters [19]. The distribution of the baseline language models is a Gaussian Mixture Model, which is an assumption to be preserved in this thesis.

The main idea of this adaptation is the assumption that each word of the language model is modeled by a single-state HMM, which is trained by using the update equations of the MMI training technique for the state *j*. Hence, the term γ_{jm} refers to the sum of the posterior probabilities for each observation, given the model for j-th word. The desired result of this adaptation would be the better discrimination between word models.

Assuming that the m-th component of the Gaussian of the single-state HMM of the j-th word is denoted as M_{jm}, the likelihood of each observation O is symbolized as $P(O|M_{jm})$. The γ_{jm}^{num} term, which is the posterior probability of the correct hypothesis(numerator) for the m-th component of the j-th word given the observation O, can now be expressed as:

$$\gamma_{jm}^{num} = \frac{p(O|M_{jm})c_{jm}}{\sum_m p(O|M_{jm})c_{jm}} \tag{3.50}$$

The γ_{jm}^{den} term, which is the posterior probability of all the hypotheses(denominator) for the m-th component of the j-th word given the observation O, is:

$$\gamma_{jm}^{den} = \frac{p(O|M_{jm})c_{jm}}{\sum_j \sum_m p(O|M_{jm})c_{jm}} \tag{3.51}$$

The difference between γ_{jm}^{num} and γ_{jm}^{den} lies in the denominator term; the denominator of γ_{jm}^{num} only considers the likelihoods of the correct model, whereas the denominator of γ_{jm}^{den} takes into account the likelihoods of all the word models.

Chapter 4

Implementation and Experimental Evaluation

4.1 Implementation Process

4.1.1 Formation of data

The first step of the current project is the conversion of each word of the vocabulary into a manageable continuous vector. As mentioned before, each word is initially represented by a V x 1 indicator vector, where V is the vocabulary size. The *co-ocurrence matrix* that is used for the SVD applied on each indicator vector is calculated by running a simple *perl* script (*cooccurrence.pl*).

The same script also creates the *indexed* training and testing files, in which the words are substituted by their correspondent indexes within the vocabulary. For example, the word "we" is at the 95^{th} place of the vocabulary, so it is substituted by the index 95 at the indexed file.

The following steps involve the reduction of the dimensions of the indicator vectors through SVD. In particular, the co-occurrence matrix is an input to the Matlab function *svds* that estimates the 100 x 100 matrix A of SVD. Each indicator vector is then multiplied by A and, as a result, the 2700 dimensions are cut down to 100.

After the 2700 1x100 word vectors had been created, the perl script *create_ngrams.pl* was run in order to form the trigrams that appear in the whole training corpus. The output of this script is a folder that contains a specific file for each word. This file holds every history vector (made up by the vectors of the two previous words of the word being read) of this word. Therefore, this folder contains 2700 history files.

The next action was the calculation of the A matrix needed for the LDA process. The LDA matrix is calculated by using the Matlab script *lda_statistics.m*. This projection matrix transforms each history vector and, as a result, reduces the dimensions of each from 100 to 50. The transformation has the form described in (2.11). The 1 x 50 history

vectors are now ready to be used within the training process.

4.1.2 Baseline Models

The baseline models that are used as a base for the creation of the MMI-trained models are created by utilising the HTK 3.4.1 toolkit. These baseline models are trained by using the EM algorithm. Instead of performing this training process by starting from scratch, the HTK tool *HRest* was used in order to estimate the parameters of the distribution of each word. HRest is actually designed for estimating the parameters (means, covariances) of HMMs. In our case, the distribution of each word is assumed to be the output of a single-state HMM which is estimated by using HRest. Specifically, the projected history vectors of each word (which are the files created by the *create_ngrams.pl* script) had been set as input to the HRest tool and,after a number of iterations of Baum-Welch re-estimation update formulae, the parameters of each distribution were finally estimated. The baseline models were trained by using Baum-Welch re-estimation update equations and hence the ML criterion was used.

4.1.3 MMI Training Implementation Issues

Applying the MMI Training method onto estimating the parameters of Continuous GMM Language Models in the same fashion it was applied on training Continuous GMLMs in [11] proved to be quite challenging because of the great number of calculations needed in order to estimate γ_{jm}^{den} factors.

Lattices

As stated above, the calculation of posterior probabilities γ_{j}^{den} can be extremely time-consuming: these posteriors need to be evaluated for each observation, while the denominators of these posteriors require the calculation of $P(O|M_{jm})$ terms for every word j in the vocabulary. For example, if a vocabulary of 3000 words in the vocabulary and 200000 different observations in the training set are assumed, the $P(O|M_{jm})$ term will be re-evaluated up to 600.000.000 times, which shows the importance of saving time in the training procedure.

The lattices to be created could have the form that is used in HTK tools, in the same way used in [13]: each word is represented by an arc, while start and end times of each observation (word) are represented by nodes that are connected by the arcs, forming a

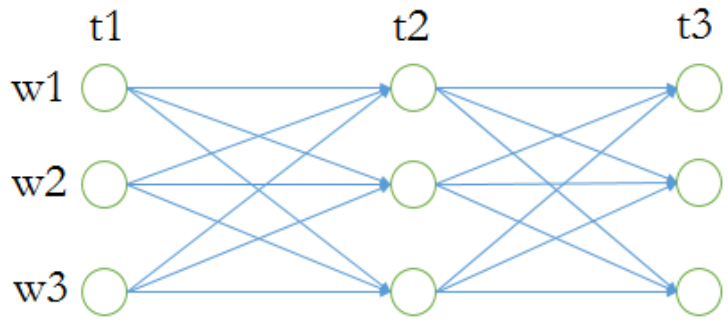

Figure 4.1: A lattice during 3 observation times

graph. Each arc is given a likelihood value, which is calculated by using the baseline language model. The lattice then rejects any likelihood value that is below a specific threshold: this means that the evaluation of $P(O|M_{jm})$ terms will be restricted to the most likely cases, which significantly reduces the training time.

An example of a lattice could be the following: Let the vocabulary of a speech recognition task consist of three words, w_1, w_2 and w_3. The observations are assumed to be three, which means that there are three time snapshots, t_1, t_2 and t_3. The lattice will be a graph consisting of three nodes per time snapshot. The arcs of the graph represent a word given its previous word; for example, the arc that goes from the node (t_1, w_1) to the node (t_2,w_3) represents word w_3 which was preceded by w_1. In this case, the different $P(O|M_{jm})$ terms would be (3^2) (3-1) = 18. Tasks of larger vocabularies and more observations would be much more time-demanding. The main notion behind the usage of lattices is the rejection of any arc that has a relatively low likelihood value $P(O|M_{jm})$.

The HMMIRest Approach

An initial approach was the use of *HMMIRest* HTK tool, which is used in HMM parameter estimation through discriminative training (MPE, MWE or MMI options are available). The main incentive for trying HMMIRest was the great saving of time. HMMIRest requires the usage of lattices, whose importance was analysed in the above section.

In particular, two kinds of lattices have to be created in order to make HMMIRest work: the numerator lattice and the denominator lattice. The numerator lattice represents the correct word sequence throughout the whole training set. The denominator lattice contains confusable hypotheses for each observation (history) through a one-pass recognition process of the training corpus.

The main problem concerning the usage of HMMIRest was the construction of the lattices; the lattices in HTK are built by using a special recognition tool, *HDecode*. This tool performs a one-pass recognition through the training data and then constructs both the numerator and the denominator lattice. The construction of these lattices required phone-marked data and an exact phoneme analysis for each word of the vocabulary. This analysis was hard to obtain, hence after having made several unsuccessful efforts to achieve such compatibility, the plan to use HMMIRest was finally abandoned.

4.1.4 MATLAB MMI Implementation

The eventual decision about the implementation of MMI was to use MATLAB for the construction of the required MMI-trained language models. The main challenge in this decision was the extreme time complexity.

The steps for implementing MMI training on the existing baseline language models are:

- Start reading the history vectors files for each word, starting from the first word in the vocabulary.
- Estimate γ_{jm}^{num} and γ_{jm}^{den} . This estimation requires the calculation of the Gaussian multivariate distribution for every observation(history vector) of every word. The calculation of γ_{jm}^{den} is the main bottleneck of the process, because the value of the Gaussian distribution for the specific observation has to be calculated for every word in the vocabulary.
- Estimate θ_{jm}^{num} and θ_{jm}^{den}.
- Estimate $\theta_{jm}^{num}(O^2)$ and $\theta_{jm}^{den}(O^2)$.
- Apply the MMI update equations for 100 iterations in order to calculate means, covariance matrices and weights.

The key to reduce time at the above process was the way of reading the history vectors; a serial way is definitely more expensive than a parallel one. Therefore, the history vectors

of a word were loaded simultaneously into the MATLAB code that implemented the MMI training process, hence making use of the so-called 'vectorization' of the MATLAB code.

4.2 Experimental Evaluation

4.3 Summary

The present chapter describes the experiments that tested the newly created MMI language models in terms of perplexity, a metric that is widely used in language model evaluation tasks. At first, the experiments involve the testing of unsmoothed models, which show only a slight improvement compared to the baseline ML models. Subsequently, models with smoothed mean vectors and covariance matrices are tested. Such models further improve perplexity figures. The final experiments involve component weight smoothing, which gives the best results in the thesis. However, variances within the covariance matrices had to be floored to certain values in order to produce better perplexity figures, a fact that certainly affects the accuracy of the generated models.

4.3.1 Perplexity as a metric

In the introduction of the current chapter, it was noted that perplexity is the metric that measures the performance of the investigated language model. The notion of perplexity is ultimately derived from *entropy*, which was mentioned as the measure of the average uncertainty of a random event. In other words, entropy measures the amount of information in a random variable. It is usually measured in bits, which explains the fact that logarithms of base 2 are normally used, but using any other base leads to a linear scaling of results that is acceptable [7].

A simple example that demonstrates the notion of entropy is the following: Suppose we want to flip a fair coin; this experiment has two potential equiprobable results, *heads* or *tails*. Thus, the entropy of the random variable X that represents the outcome of the experiment is evaluated as:

$$H(X) = -\sum_{X} P(X) \log P(X)$$
$$= -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = 1.$$

The entropy that equals to 1 means that 1 bit is the amount of information needed in order to represent random variable X, which is correct according to the intuition that 2 different kinds of results need one bit to represent them.

Assuming that the entropy *H(X)* has a 2-based logarithm, the term

$$ppl_2(X) = 2^{H(X)} \tag{4.1}$$

is called the *perplexity* of the random variable X. The value of perplexity is equivalent to the size of an imaginary equivalent list, whose words are equally probable[1]. In other words, perplexity expresses the average number of different guesses that could be made through an experiment that is modeled by X. For example, the above experiment of tossing the fair coin has an entopy that equals to one. This implies that the perplexity is equal to $2^1 = 2$, which means that there are two different guesses that can be made for this experiment.

In the experiments of this work, the perplexity is evaluated by using natural logarithm-based entropy, which makes calculations with natural log-probabilities easier. As a result, the perplexity is calculated as follows:

$$ppl_{exp}(X) = e^{H(X)} \tag{4.2}$$

The shift from 2 base to exponential base does not incur any change in the perplexity figures, considering that

$$2^{\log_2 X} = e^{\log_e X} \tag{4.3}$$

The only difference in using natural logarithms is that entropy is now measured in nats and not in bits.

However, in the present work, the perplexity should involve cross-entropy instead of the initial definition of entropy:

$$ppl_{exp}(X, q) = e^{-\sum_X P(X) \log_e q(X)} \tag{4.4}$$

where q(X) is the probability which follows a distribution other than the prior P(X).

4.3.2 Testing Process

The test set that was used for the testing process of the generated language models was extracted from the WSJ94, like the training set. It consists of 6000 sentences which are different from the sentences used for training.

The exact process is the following: Each word in the test corpus is substituted by its correspondent index in the vocabulary list that contains the 2700 words. For example, the word "out" is at the 100^{th} position of the vocabulary list, hence it is substituted by "100". After creating this index file, the history h of each index (the previous two words) in every sentence is serially read. When the history of the word of index i is read, the probability p(w_i|h) is evaluated as follows:

$$p(w_i|h) = \frac{p(w_i)p(h|w_i)}{\sum_j p(w_j)p(h|w_j)} \tag{4.5}$$

according to Bayes' theorem. The p(h|w_i) term is the likelihood of the current history given the Gaussian Mixture Model of the i^{th} word, while p(w_i) is the prior unigram probability of the i^{th} word. The denominator in (4.5) is derived from the law of total probability: p(h) = $\sum_j$ p(h,w_j) = $\sum_j$p(h|w_j), where j is the index of every word in the vocabulary. The posterior probabilities of (4.5) are then expressed in log-probabilities, which makes the calculation of perplexity more comfortable.

After the log-probabilities are estimated for each word in the sentence, the perplexity of this sentence is evaluated as:

$$ppl(sentence) = \exp\{\frac{-\sum_j \log p(w_j|h)}{words - oovs + 1}\} \tag{4.6}$$

However, not all of the N words should be taken into account, because <unk> words, which are out of the vocabulary (oovs), cannot be in the total word count. The log-probability of the </s> token is also included in the sum over all log-probabilities in the sentence. As a result, the tokens that actually contribute to the estimation of the sentence perplexity are the words that exist in the vocabulary plus the </s> token, that is *words+oovs+1*.

4.4 Results

The experimental process involves testing the new MMI-trained language models for various number of components per mixture. Apart from the number of components, another significant factor is the restriction in covariance values; extremely low covariances lead to excessive values of perplexity, which is an aspect of data overfitting, which occurs when there is a small number of observations for some classes. This is the main reason for the need of a variance floor, which sets a minimum value that a covariance may have. Throughout the experimental process, it was found that the higher the variance floor, the lower the perplexity values. However, a high variance floor causes distortion of the actual model. Therefore, the increase of variance floor improves the perplexity figures, though sacrificing model accuracy.

The first experiments involved a comparison between the baseline GMM models and the produced MMI models over the first 1000 sentences of the test set of WSJ94 corpus. At first, the number of components per mixture was set to 2, then to 4 and eventually to 8. The covariance values of each model and mixture were floored to 0.05. The MMI models are smoothed with the correspondent baseline (ML) models. In particular, the mean vectors and covariance matrices of each mixture are smoothed with the means and covariances of the respective ML baseline models. The smoothing constant is initially set to 100. The results are given below:

Components	Baseline Models	Smoothed MMI Models
2	355.11	338.52
4	331.47	327.43
8	306.34	312.36
16	287.69	310.25
32	280.85	311.09

Table 4.1: Performance of Baseline and smoothed MMI Models ($\tau^I = 100$)

The experiment above shows that there is a slight improvement of perplexity for low

number of Gaussians per mixture, while higher number puts baseline models on the lead; this might be explained by the fact that when I-smoothing is applied to a large number of parameters, the distortion due to the incorporation of the priors is more visible than in MMI models of 2 or 4 Gaussians per mixture.

The following experiment sets the smoothing constant to 25, with slightly better results:

Components	Baseline Models	Smoothed MMI Models
2	355.11	314.65
4	331.47	303.11

Table 4.2: Performance of Baseline and smoothed MMI Models ($\tau^I = 25$)

The next course of experiments includes comparison of baseline models to unsmoothed MMI models, which are expected to surpass baseline models performance for high numbers of Gaussians per mixture (8, 16 or 32). The results of this comparison are below:

Components	Baseline Models	MMI Models
2	355.11	303.64
4	331.47	287.15
8	306.34	270.04
16	287.69	260.47
32	280.85	256.61

Table 4.3: Performance of Baseline and MMI Models

The results of the experimental part of the thesis showed mixed performance of MMI models. Smoothed MMI models for number of Gaussians per mixture equal to 2 and 4 narrowly outperform the respective baseline models. For 8 or 16 Gaussians per mixture, baseline models perform better.

This mixed behavior is not the same in non-smoothed MMI models: for any number of Gaussians per mixture, MMI models perform better than the correspondent baseline ML models, while the gap between the perplexity figures has now widened.

The following experiments examine models that use history vectors of lower dimensions. This means that the LDA matrix projects the vectors to dimensions less than 50. Initially, vectors of 40 dimensions are used. The results are shown in the table below:

Components	Baseline Models	MMI Models
2	255.45	281.67
4	207.29	244.51

Table 4.4: Performance of Baseline and unsmoothed MMI Models for 40-dimensional vectors

The respective results for history vectors of 30 dimensions are given below:

Components	Baseline Models	MMI Models
2	234.91	283.24
4	208.40	258.52

Table 4.5: Performance of Baseline and unsmoothed MMI Models for 30-dimensional vectors

The case of 20-dimensional vectors is examined next, with the results given in the following table:

Components	Baseline Models	MMI Models
2	235.54	292.18
4	226.04	268.24

Table 4.6: Performance of Baseline and unsmoothed MMI Models for 20-dimensional vectors

The above results do not encourage any further experimentation with MMI models, as baseline models give better perplexity figures. Although baseline models may give better results in terms of perplexity, this reduction in dimensions may lead to a distortion of the model.

Chapter 5

Conclusions and Future Work

5.1 Conclusions of the present work

The current thesis involved the study of continuous-space language modeling and focused on the MMI training of continuous Language Models. The MMI models were built upon baseline ML-estimated models. There were implementation issues that involved time complexity, because lattices were not used. The experiments involved the comparison of MMI models and baseline ML models in terms of perplexity. The results showed that unsmoothed MMI models outperformed baseline models, while MMI models that were smoothed with their respective baseline model showed mixed results; for small number of components per mixture, MMI GMLMs showed slightly better figures, while for higher number of components per mixture they performed worse than their baseline counterparts. In the case of less dimensions (40, 30 or 20 per history vector), baseline models perform better that their respective MMI models

5.2 Future Work

5.2.1 MPE Implementation

The present thesis utilised Maximum Mutual Information Estimation methods in order to train language models in a discriminative matter. The results could be further improved by using alternate techniques. Such technique is the previously mentioned Minimum Phone Error (MPE) method which reportedly gives better results than the MMI training method when applied during the training process of HMMs of acoustic models. However, there are some cases where the amount of the training data is small and, as a result, the generalisation of MMI is more effective than that of MPE. Another challenge to be beaten is the formation of data so as to fit in the MPE method; MPE uses phone accuracy, thus data should be 'pseudo-analysed' into phones in order to be trained by MPE.

5.2.2 Lattice adjustment

As mentioned in 4.6.3 the generation of lattices is very important in saving training time. Implementations over acoustic models in [13] and in [23] use lattices and give very encouraging results. Lattice-based MMI can essentially reduce the training time while approximating the actual MMI. In general, the lattice structure in a MMI training scheme stores the most probable hypotheses for a given utterance and at the same time rejects hypotheses of lower probability.

The main challenge of building lattices for a language model is, as in MPE, the adjustment of the available data. Lattices refer to start and end times of phone speech segments and are more close to an acoustic implementation than a word-level implementation. Nevertheless, a lattice-based MMI language model would be really effective in terms of both performance and time complexity.

Bibliography

[1] X.Huang, A.Acero, H.W. Hon, Spoken Language Processing: a guide to theory, algorithm and system development, Prentice Hall, 2001.

[2] D.Jurafsky, J.H.Martin, Speech and Language Processing, Prentice Hall, September 1999.

[3] S.F.Chen, J.Goodman,"An Empirical Study of Smoothing Techniques for Language Modeling", Harvard University, June 1996.

[4] S.M.Katz, "Estimation of probabilities from sparse data for the language model component of a speech recogniser", IEEE Transactions on Acoustics, Speech, and Signal Processing, March 1987.

[5] Y.Bengio, R.Ducharme, P.Vincent, "A neural Probabilistic Language Model", Departement Informatique et Recherche Operationnelle, Universite de Montreal, 2003.

[6] H.Schwenk, "Continuous space language models", 2006.

[7] C.D. Manning, H. Schütze, Foundations of Statistical Natural Language Processing, MIT Press, February 1999.

[8] H.Schwenk, J.L. Gauvain, "Using Continuous Space language Models for Conversational Speech Recognition", April 2003.

[9] S.Theodoridis, K.Koutroumbas, Pattern Recognition, Fourth Edition, Elsevier, 2009.

[10] S. Balakrishnama, A. Ganapathiraju, Linear Discriminant Analysis - A Brief Tutorial, Institute for Signal and Information Processing of Mississippi State University.

[11] M.Afify, O.Siohan, R.Sarikaya, "Gaussian Mixture Language Models for Speech Recognition", ICASSP, 2007.

[12] R.Sarikaya, M.Afify, B. Kingsbury, "Tied-Mixture Language Modeling in Continuous Space", 2007.

[13] D.Povey, Discriminative Training for Large Vocabulary Speech Recognition, University of Cambridge, March 2003.

[14] C.M. Bishop, Pattern Recognition and Machine Learning, Springer Science+Business Media, 2006.

[15] C.E.Shannon, A Mathematical Theory of Communication, Bell System Technical Journal, 1948.

[16] P.Brown, The Acoustic-Modeling Problem in Automatic Speech Recognition, Carnegie-Mellon University, May 1987.

[17] S.Kapadia, Discriminative Training of Hidden Markov Models, University of Cambridge, 1998.

[18] A.P.Dempster, N.M. Laird and D.B. Rubin, "Maximum Likelihood Estimation from Incomplete Data", Harvard University and Educational Testing Service, December 1976.

[19] S.Young,G.Evermann,M.Gales,T.Hain,D.Kershaw,X.Liu,G.Moore,J.Odell, D.Ollason, D.Povey, V.Valtchev,P.Woodland, The HTK Book Version 3.4, Microsoft Corporation, Cambridge University Engineering Department, March 2009.

[20] G.Strang, Linear Algebra and its Applications, Harcourt Brace Jovanovich, 1988.

[21] S.Geirhofer, Feature Reduction with Linear Discriminant Analysis and its Performance on Phoneme Recognition, University of Illinois at Urbana-Champaign, May 2004.

[22] B.H.Juang, W.Chou, C.H.Lee, "Minimum Classification Error Rate Methods for Speech Recognition", IEEE, May 1997.

[23] V.Valtchev, J.J. Odell, P.C. Woodland, S.J. Young, "Lattice-based Discriminative Training for Large Vocabulary Speech Recognition", Cambridge University Engineering Department, 1996.