Automatic Acoustic Modeling using Tied-Mixture Successive State Split Algorithm*

Alexandre Girardi

Abstract

This thesis describes an automatic acoustic model generation algorithm. The new algorithm extends the work of state-of-the-art algorithms, like Maximum Likelihood Successive State Splitting (ML-SSS) algorithm and Phonetic Decision Trees (PDT) algorithm.

ML-SSS and PDT use top-down clustering algorithms to produce their acoustic model topologies. Initially in this approach a set of states representing a set of triphones (or contexts in a more generic case) are divided successively until an acoustic model topology is produced.

The problem with ML-SSS and PDT is that the cost function they use to take the decision on how and where to split a state needs to be restricted to states modeled by single Gaussians. The more natural and complete representation, namely, a continuous density (Gaussian Mixture - GM) representation of the probability density function (pdf) is simply too expensive computationally.

Attempts have been done to solve this problem in GM representation too, but without success. A tied-mixture (TM) representation is also a rich representation for the pdf of a candidate state for splitting, but in its original format this is also computationally expensive. In this thesis we propose a simplification to the TM approach which yields a computationally less intensive algorithm.

The main focus of this thesis is therefore this simplification that makes TM top-down clustering feasible. In this thesis we also present speech recognition experiments which compare state-of-the-art algorithms, such as ML-SSS and PDT,

*Doctor's Thesis, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9661204, February 10, 2001.

with the proposed Tied-Mixture Successive State Splitting (TM-SSS) algorithm. It is expected that, despite the simplification, a richer representation of the pdf of candidate state splits will improve the final acoustic model topology. The experiments presented in this thesis confirm this hypothesis with a significant increase of the recognition rate.

The thesis is divided in two parts. The two parts combined result in the proposed Fast Tied-Mixture Successive State Splitting (FTM-SSS) algorithm. In the first part it is proved that the new algorithm results in a significant increase in accuracy as compared to ML-SSS. However, the algorithm is still slow and hence is demonstrated with speaker-dependent experiments only. In the second part, which is the final version, a few more simplifications lead to an algorithm which is fast enough for large dictation tasks. This is then compared to PDT on a speaker-independent large vocabulary task. Results for the final version show that for equal conditions a better accuracy is achieved.

Keywords:

Speech Recognition, Acoustic Modeling, HMM, Tied Mixture, Clustering

Contents

List of Figures

List of Tables

1. Introduction

Automatic Speech Recognition (ASR) systems are becoming a reality in daily life with applications that range from personal computers (command & control to dictation), telephony services, mobile phones to even toys. Its importance is self-evident.

The goals of a speech recognition engine are as diverse as its applications. Some ASR systems are used in applications where accuracy is the most important factor (i.e., in personal computers with dictation or in telephony services with credit card transactions), others where speed or conversely minimum hardware is the dominant factor, such as in toys and voice dialing systems for mobile phones. ASR system goals are then speech recognition accuracy or speed (minimum hardware).

The goals are clear. And ASR systems are out there working. But is the problem solved? The answer is no. Most systems are full of wrong assumptions, theoretical problems and practical problems. Theoretical problems are in the center of most performance degradations of such systems, e.g., word error rate versus speed characteristic curve degradation. Practical problems are the reasons for many unused theory and hence constrained utilization of ASR systems.

On the theoretical problems we can point out that most commercial ASR systems use Hidden Markov Models (HMM's), despite the theoretical problems that it carries. A few examples of problems with HMM's are:

- HMM's assume independent states, but states as they are trained in commercial applications are not independent (they are trained with data that is too correlated sample by sample).
- Most systems are trained based on the criterion of Maximum Likelihood. This works reasonably, but is known not to provide the best answer.
- States in HMM's are poorly represented by assuming that each dimension of the input data is independent of each other by using diagonal covariance matrices to represent the underlying model.
- When constructing the HMM topology, using a clustering algorithm, single Gaussians are used to represent states. This is both a conceptual error and

a mismatch with the HMM's models representation eventually used.

On the practical or commercial aspect we are interested in minimizing the resources needed for a certain task to be effectively accomplished. This translates to minimum hardware or energy requirements for a certain task size and necessary accuracy. In the case of hardware, constraints are a consequence of memory or chip size. In the case of energy it is related to CPU speed, since energy consumption grows with the speed of the CPU clock. We can further divide the practical issues as follows:

- Memory: buffers (as synchronous as possible, also as short response delay as possible), acoustic and language model sizes (as small as possible, provided that there is no significant loss in accuracy).
- CPU speed: number of parameters to evaluate, representativeness of parameters (only relevant data should be addressed).
- Accuracy: should be as high as possible, but there is always a trade-off with memory and recognition speed. Most applications will lie in between the extreme points. As pointed out before, there are systems where accuracy has more importance, such as dictation systems, and others where recognition speed is more important, such as data retrieval systems.

A solution to the theoretical problem mentioned before exists (using Gaussian Mixtures during the HMM topology construction), but it is too computational intensive to be practically feasible.

A partial solution to this problem is the construction of HMM topologies based on states represented by single Gaussians. However this approach leads to a non optimal HMM topology. It is important to notice that the initial HMM topology deals with the initialization of HMM's which is usually immutable during the acoustic model training process and any error in the HMM topology used is propagated and it is difficult if not impossible to compensate later in most speech recognition engines. This partial solution needs therefore be improved.

Concluding, there are a number of arguments for focussing efforts on the construction of the initial HMM topology:

- HMM states are insufficiently represented by single Gaussians and a richer representation must be used.
- It is crucial to construct the HMM topology in a limited time frame. Moreover it is a procedure that is reproduced many times during the development and across multiple dialects and languages.

Because of these arguments this thesis aims at finding a solution for the construction of initial HMM topologies that solves the theoretical problems and it is practically feasible at the same time. It can be seen as speeding up recognition time, increasing accuracy or even reducing hardware requirements. The focus will be on acoustic model generation, more specifically in the HMM topology generation, since it is important to be sure we are evaluating the relevant data and only that.

1.1 HMM Topology Generation

Creation of precise and robust acoustic models for speech recognition, given a limited amount of training data, has been an intensively studied field of research over the last 10 years [16, 19]. Acoustic modeling is a important component of the development of a recognition engine. Its implementation is done in three main steps. We can name them as a first rougth estimate of context dependent models, followed by HMM topology creation (where context dependent models are estimated) and finally a training procedure.

HMM topology generation is the second of the three steps needed to create acoustic models. It is the subject of this thesis and will be further explained in more details in Sec. 3.1 and Sec. 4.

In order the three steps of acoustic model generation are:

- Context independent modeling. First we produce context independent acoustic models based in a uniform segmentation. The output of this step is a first rough segmentation in terms of context independent acoustic models. This segmentation is used to create the codebook used in the next step.

  A typical list of phonemes for Japanese, that is also used in this thesis and that is modeled in this step is:

N, Q, a, a-, aN, aa, ai, ao, b, by ,ch ,d ,e ,eN ,ee ,ei ,f ,g ,gy ,h ,hy ,i ,i+ ,iN ,ii ,j ,k ,ky ,m ,my ,n ,ny ,o ,o- ,oN ,oo ,ou ,p ,pau ,r ,ry ,s ,sh ,t ,ts ,u ,u+ ,u- ,ue ,ui ,uu ,w ,y ,z.

Short silence 'sp' and silence models 'pau' are usually trained in separate just after we have trained the rest of the phonemes, because its presence is usually not known before hand.

Those models are useful also to help on identifying the acoustic space and as initialization for the later Vector Quantization steps.

- HMM topology generation. At this level we try to find context dependent models, usually triphones (also the choice of this work). As the number of triphones is large and normally we do not have training data for all possible triphones we need to model sets of triphones (allophones) that are close together as if they were a single model.

  That was in fact the direction followed in the first algorithms for HMM topology generation [16, 19, 9]. More generically in HMM's we can model together parts of some phonemes as if the were one, i.e. tying states, Gaussians, Gaussian weights and even parts of its topologies (transitions). This way, if a set of phonemes start with statistically close frames we can model those phonemes with the same state. A Similarly can be used also with the center and the final states of phonemes.

  To define how those states should be shared is a very complex task to be reliably done by hand, so we need an automatic tool to produce those state tying. Successive State Splitting (SSS) algorithm is one of the first and successful attempts to do so [8]. Later this algorithm was consolidated by using uniquely maximum likelihood (ML) as a criteria for HMM topology generation. Also in the same line is Phonetic Decision Tree (PDT) algorithm using phonetic questions [18, 14].

  The result of this step is a HMM topology, were the unseen contexts (in our case triphones) are modeled by sharing states, transition probabilities and mixture weights.

- Retraining. retraining is necessary, because usually after the HMM topology generation we have HMM states with a single Gaussian, due to limitation on current automatic HMM topology algorithm generation.

The number of Gaussian used to represent a state is usually augmented by duplicating a Gaussian (however changing slightly its values) and normalizing the mixture weights so that the HMM re-estimation equations stay valid. The number of Gaussians used to represent a state is usually augmented to something between 4 and 32 Gaussians, depending on the amount of training data available.

This process then requires some EM (estimation maximization) iterations to converge the HMM models. Later on even discriminative training [13] or adaptation can be used on top of the final HMM acoustic model in order to improve recognition rate.

1.2 Thesis Structure

This thesis is divided as follows: in Sec. 2 we shortly describe speech recognition using Hidden Markov Models (HMM's). The importance of feature extraction is discussed in Sec. 2.1, the relationship between acoustic scores and language models in Sec. 2.2, the importance and our choice for the language model used in speech recognition in Sec. 2.3. Hidden Markov Models are explained in Sec. 2.4, the way we represent our observation probabilities is also described and will play a key role during the training procedure as we will see in Sec. 2.4.1.

Sec. 2.4.2 discusses the forward algorithm which is used for efficiently evaluating the probability of an observation given a model. Its reverse engineering Viterbi algorithm which is used to recover the best state sequence for a given model is discussed in Sec. 2.4.3. The backward algorithm with which we can derive the re-estimation formulas needed to estimate the acoustic models is described in Sec. 2.4.4.

This section concludes with an introduction to the recognition engine Julius Sec. 2.5 used in this thesis.

After this HMM introduction we introduce the reader to the first part of the thesis in Sec. 3.

Sec. 3.1 points out the problems addressed by the proposed automatic HMM topology training Tied-Mixture Successive State Split Algorithm. It is then showed how we improve the base Maximum Likelihood Successive State Split Algorithm Sec. 3.2. The algorithm is summarized in Sec. 3.2.1.

The theoretical development of its extension to a contextual split is discussed in Sec. 3.2.2. The quantitative preliminary results in speaker dependent data is shown in Sec. 3.3, Sec. 3.3.1 and also the qualitative results in Sec. 3.3.3. This is followed by a summary of this first part in Sec. 3.3.2 and Sec. 3.4.

The description of the second part of the training algorithm starts in Sec. 4. Some additional assumptions are described in Sec. 4.1. These are explored in Sec. 4.1.1, leading to a faster TM-SSS algorithm, the so called Fast Tied-Mixture Successive State Split Algorithm. Due to the assumption the codebook do not need to be updated during state split in TM-SSS the new algorithm is now faster enough to be tested in large amounts of training data (in a speaker independent experiment). The data itself is discussed in Sec. 4.2, and the experiments carried out and their results in Sec. 4.2.1. Finally the partial results from the second part are summarized in Sec. 4.3.

Finally Sec. 5 concludes the thesis and Sec. 6 gives directions for future work.

2. Speech Recognition using Hidden Markov Models

In this section, we will briefly describe speech recognition using Hidden Markov Models or simply HMM's. In the first part we will describe how a speech signal is modeled Sec. 2.2. This contains a description of language modeling Sec. 2.3. Next we describe how observations are modeled Sec. 2.4.1. The next three parts are the three basic problems efficiently handled by the Estimation Maximization (EM) algorithm, namely the forward (Sec. 2.4.2), the Viterbi (Sec. 2.4.3) and the backward (Sec. 2.4.4) problems. Those three algorithms are the answer respectively to how to effectively compute $P(Y|W)$ (the probability of the observation sequence Y given the model W), how to choose a state sequence that is optimal in some sense (Maximum Likelihood in our case), given the observation sequence $Y = \{y_1, y_2, ..., y_F\}$ and finally how to adjust the model parameters W to maximize $P(Y|W)$. The last subsection describes the Julius Recognition Engine used in this thesis Sec. 2.5.

2.1 Sampling and Feature Vectors

Speech has much more information than a speech recognition engine can handle, but we can reduce significantly the amount of data we need to process, without loosing a significant part of the information that we are looking for (for example if we can reconstruct the speech signal in such a way that a human being can still recognize what has being said, as much as in the original data).

Among those data reduction techniques we can mention:

- sampling the speech signal at a bit more than twice its Niquist frequency is enough to ensure that we can reproduce the original signal.
- working with a Cepstral representation of the signal (the signal viewed in its log frequency domain) is quite independent of the exact point in time when the window of the signal to be analyzed is extracted. In the time domain we would need to know when the pitch is available and take windows (set of samples) always in the same position. Besides it would be only possible

at points in time when the pitch information. In practice the time domain approach results in unreliable recognition engines.

- also it is important to notice that the human ear is more sensitive to smaller frequency ranges for low frequencies than it is for high frequencies. A well know method is to consider frequency bands, depending on how our human ear processes speech, in other words to consider a MEL-scale frequency range for each band.

On top of that we could use Cepstral mean normalization, because it compensates channel distortions like microphone or even telephone line distortions. Cepstral mean normalization estimates the Cepstral frequency of the noise from the silence part (non-speech) and subtracting that from the Cepstral representation of the speech part. That can be done because in this domain noise can be considered additive.

After processing the signal in the time domain X, with a preprocessor similar to one on Fig. 1 we end up with the feature vectors Y.

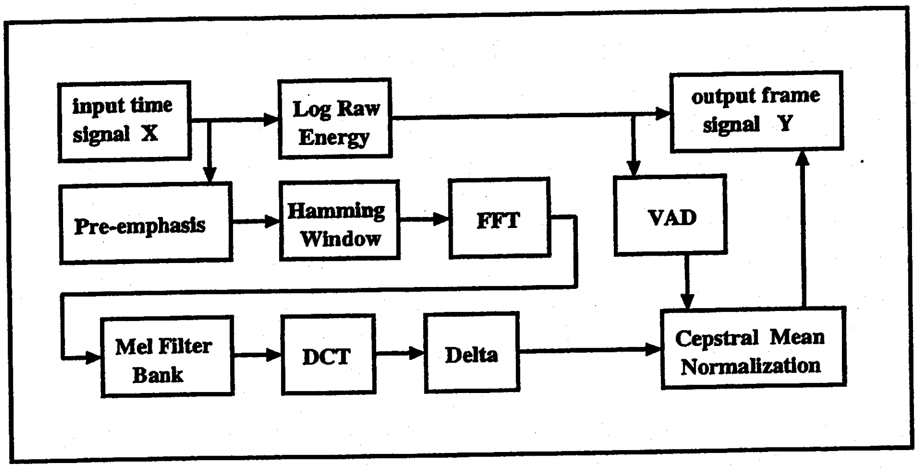

Figure 1. Signal preprossessing: sampling, windowing, Cepstral transformation, normalization and channel compensation for efficient speech frame extraction.

In Fig. 1, FFT stands for Fast Fourier Transform, DCT stands for Discret Cossine Transform and VAD stands for Voice Activity Detection.

From this point on we will not refer directly to the input signal $X = x_1, x_2, ..., x_T$ in the time domain, but to it's equivalent in the frame domain $Y = y_1, y_2, ..., y_F$.

2.2 Modeling the Speech Signal

The goal of speech recognition is to find the most likely word sequence W for a given speech frame sequence Y. This is the word sequence for which

$$P(W|Y) = \frac{P(W)P(Y|W)}{P(Y)} \qquad (1)$$

is largest.

$P(W)$ gives the probability of a particular word sequence, and is independent of the actual input signal.

$P(Y)$ is the probability of observing the frame sequence Y.

$P(Y|W)$ is the probability of observing the frame sequence Y given that the actual word sequence is W.

Recognition is then performed comparing $P(W|Y)$ with all possible word sequences. As $P(Y)$ is the same for all hypotheses we can consider this term constant and the largest $P(W|Y)$ reduces to simply looking for the largest $P(W)P(Y|W)$.

$P(Y|W)$ can be statistically represented by Hidden Markov Models (HMM's), see Sec. 2.4. Its evaluation is a problem to be solved by the Estimation Maximization (EM) algorithm, described in the forward (Sec. 2.4.2), backward (Sec. 2.4.4) and Viterbi (Sec. 2.4.3) sections.

The next section will describe how to estimate the $P(W)$.

2.3 Language Models

The language model probability $P(W)$ is the probability of a given word sequence W which is commonly represented by n-grams such as bigrams and trigrams [3]. $P(W)$ is estimated using large amounts of text data, where usually units, in particular words, have not only been identified but also annotated with their syntactical classification.

In general an n-gram is used to compute $P(W)$ as

$$P_{trigram} = \Pi_{i=1}^{N} P(w_i|w_{i-1}, w_{i-2}, \ldots, w_{i-n+1}) \qquad (2)$$

There is never enough text material to estimate all possible n-grams, and even if there was, we would have too many parameters to store.

For the problem of insufficient text material a common practice is to employ back-off smoothing techniques, such as the discounting in the Witten-Bell [6] method.

To alleviate the second problem of large language model file sizes we used an n-gram language model with the smallest set of probability parameters that minimizes the training set perplexity [12].

2.4 Hidden Markov Models

The probability $P(Y|W)$ of observing the frame sequence Y given that the actual word is W can be statistically represented by a state machine or Markov chain with states and transitions. Each state s can produce an observed frame y with a certain probability $p(y|s)$. Using threse probabilities, the probability $P(Y|W)$ of a frame sequence Y produced by the Markov chain corresponding to the word sequence W can be evaluated. A graphical representation of such a Markov Chain can be seen in Fig. 2.

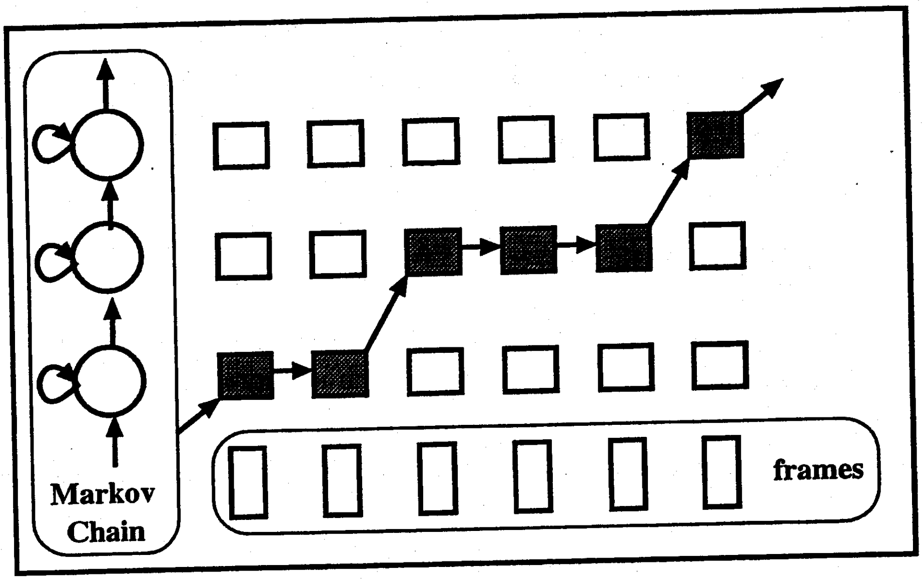

Figure 2. Set of Markov Chains with 3 states left to right, input frames in the bottom and in shaded boxes the best state/frame sequence for that particular model.

In speech recognition a Markov Chains represents a phoneme (context dependent case) or a triphone set (context independent case). Sets of Markov Chains are aligned to form each transcription of a word, as given by a dictionary.

A speech recognition system works by comparing the probability that the observed sequence of frames was produced by the Markov Chain representing a given word, against other words in the dictionary.

2.4.1 Observation Probabilities

The computation of $P(Y|W)$ for the observation sequence $Y = y_1, y_2, , ..., y_f, ..., y_F$, considering all possible state sequences $\forall s_1^F = s_1, s_2, ..., s_f, ..., s_F$, is given by:

$$P(Y|W) = \Sigma_{\forall s_1^F} p(s_1^F|W) p(y_1^F|s_1^F, W) \quad (3)$$

$$(4)$$

In equation 4 we assume statistical independence of observations frames y_f. Also, because it is modeled by a Markov chain, the state s_f depends only on its predecessor state s_{f-1}. Equation 4 therefore becomes

$$P(Y|W) = \Sigma_{\forall s_1^F} \Pi_{f=1}^F p(s_f|s_{f-1}, W) p(y_f|s_f, W) \quad (5)$$

It is also clear from equation 5 that as part of the computation of $P(Y|W)$ it is necessary to compute the probability density function $p(y_f|s_f, W)$ of a given feature vector y_f being generated by a single state s_f. When modeling speech with Hidden Markov Models (HMM), it is assumed that $p(y_f|s_f, W)$ only depends on the current feature vector and state. The probability density function can be approximated by a superposition of L Gaussian distributions. If the coefficients in the feature vectors are not correlated, this probability density is computed as a Tied-Mixture system:

$$P(Y|s, W) = \Pi_f^F \Sigma_l^L b_l(s_f) N(y_f|\mu_l, \Sigma_l) \quad (6)$$

or as a Gaussian Mixture:

$$P(Y|s, W) = \Pi_f^F \Sigma_l^L b_l(s_f) N(y_f|\mu(s_f)_l, \Sigma(s_f)_l) \quad (7)$$

The difference between a Tied-Mixture system and a Gaussian Mixture system is that in the first case the Gaussians are shared among states, while in the Gaussian Mixture there is a set of Gaussians per state.

In both equations the Gaussian $N(y|\mu,\Sigma)$ is normally evaluated as a multivariate Gaussian distribution as in equation 8 below, where D is the number of dimensions of the frame vector y.

$$N(y|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^D|\Sigma|}}e^{-\frac{1}{2}(y-\mu)^T\Sigma^{-1}(y-\mu)} \tag{8}$$

The equation 8 is normally responsible for much more than half of the computation required from a speech recognizer. To reduce the computational complexity, the covariance matrix Σ is normally assumed to be a diagonal one. This approximation is normally reasonable, and can be even reinforced by some LDA transformation of the input frames y. In such a case equation 8 simplifies to

$$N(y|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^D\Pi_{d=1}^D\sigma_d^2}}e^{-\frac{1}{2}\Sigma_{d=1}^D\frac{(y_d-\mu_d)^2}{\sigma_d^2}} \tag{9}$$

It is also important to notice that by using a full covariance matrix we increase the number of free parameters we need to estimate. This will then require a larger training database (in other words, it is inefficient).

2.4.2 Forward Algorithm

With a little effort we can see that equation 5 has many redundancies and we can be more efficiently evaluated using the following algorithm

Forward Algorithm

- initialize ($f = 0$)

  $\alpha_1(s) = p(y_0|s, W)\pi_s, 1 \leq s \leq S$

- iterate for $f = 1, 2, \ldots, F-1$

  $\alpha_{f+1}(s) = p(y_{f+1}|s, W)\Sigma_{s'=1}^{S}\alpha_f(s')p(s|s', W)$,

  $s = s_f, s' = s_{f-1}$

- terminate

  $P(Y|W) = \Sigma_{s=1}^{S}\alpha_F(s)$

where S is total number of states in the model W and π_s is the initial transition probability for the first state s, which is 1 for the initial state and 0 otherwise.

With this algorithm we efficiently evaluate the probability of a certain model W, given an observation frame y_1^F. $\alpha_f(s)$ is therefore the probability of the partial observation y_1^f given the model W. However we also need to know the best state sequence in some sense (e.g. in the Maximum Likelihood sense) that best explains the observation. The next algorithm Sec. 2.4.3 shows how to do this.

2.4.3 Viterbi Algorithm

In order to obtain the state sequence which in some sense best explains the observation a common solution is to take the path that maximizes $\alpha_f(s)$. This is known as the Viterbi algorithm. This algorithm is in fact the Forward algorithm, but replacing the summation by a maximization step, as shown by the description of the Viterbi algorithm below.

Viterbi Algorithm

- **initialize** ($f = 0$)

  $\alpha_1(s) = p(y_0|s, W)\pi_s, 1 \leq s \leq S$

- **iterate** for $f = 1, 2, \ldots, F-1$

  $\alpha_{f+1}(s) = p(y_{f+1}|s, W) \operatorname{argmax}_{s'=1}^{S} \alpha_f(s')p(s|s', W)$, $s = s_f, s' = s_{f-1}$

- **terminate**

  $P(Y|W) = \operatorname{argmax}_{s=1}^{S} \alpha_F(s)$

2.4.4 Backward Algorithm

We still need to address the problem of re-estimating the model parameters $W = \{a_(s, s'), b_l(s), \nu_l\}$. Here $a_(s, s') = p(s_f|s_{f+1}, W)$ is the transition probability from state s_f to frame s_{f+1}, and $b_l(s) = p(\nu_l|s_f, W)$ is the probability of Gaussian l or simply the Gaussian weight for state s_f, mixand l and model W, and $\nu_l = \{\mu_l, \sigma_l\}$ is the set of means and variances of mixand l.

In order to solve this problem we must first define a few new expressions:

- the backward probability $\beta_f(s_s)$ that is the partial probability of the observation sequence Y from frame f until the last frame F.

The backward probability can be solved efficiently in a way analogous to the forward probability by using the following algorithm

Backward Algorithm

- initialize ($f = F$)

  $\beta_F(s_s) = 1, 1 \leq s \leq S$

- iterate for $f = F-1, F-2, \ldots, 1$

  $\beta_f(s) = \Sigma_{s'=1}^{S} p(s|s', W)p(y_{f+1}|s', W)\beta_{f+1}(s')$

We also need to look for $P(y_f|s_f, \nu_l, W)$, the probability of an observation y_f given the state s_f, the mixand $\nu_l = \{\mu_l, \sigma_l\}$ and the model W, and $P(y_f|s_f, s_{f+1}, W)$, the probability of an observation sequence during the transition from state s_f to the next state s_{f+1}. This can be calculated as follows:

$$\begin{aligned}
p(y_f|s_f, \nu_l, W) &= \gamma_f(s_f, l) && (10)\\
\gamma_f(s_f, l) &= \alpha_f(s_f)\frac{p(y_f|\nu_l, W)p(\nu_l|s_f, W)}{p(y_f|s_f, W)}\beta_f(s_f) && (11)\\
\gamma_f(s_f, l) &= \alpha_f(s_f)b_l(s_f)N(y_f|\mu_l, \Sigma_l)\beta_f(s_f) && (12)\\
P(y_t|s_f, s_{f+1}, W) &= \xi_f(s_f, s_{f+1}) && (13)\\
\xi_f(s_f, s_{f+1}) &= \alpha_f(s_f)p(s_{f+1}|s_f, W)p(y_{f+1}|s_{f+1}, W)\beta_{f+1}(s_{f+1}) && (14)\\
p(y_f|s_f, W) &= \Sigma_l^L \gamma_f(s_f, l) && (15)
\end{aligned}$$

where

$$\begin{aligned}
b_l(s_f) &= p(\nu_l|s_f, W) && (16)\\
a(s_f, s_{f+1}) &= p(s_{f+1}|s_f, W) && (17)
\end{aligned}$$

We can than, from the model parameters $W = \{a(s_f, s_{f+1}), b_l(s_f), \nu_l\}$ estimate the new models $\mathbf{a}(s_f, s_{f+1}), \mathbf{b}_l(s_f), \nu_l\}$ as a function of $\gamma_f(s_f, l)$ and $\xi_f(s_f, s_{f+1})$.

$$\begin{aligned}
\mathbf{a}(s, s') &= \frac{\Sigma_{f=1}^{F-1}\xi_f(s_f, s'_{f+1})}{\Sigma_{f=1}^{F}\Sigma_{s'=1}^{S}\xi_f(s_f, s'_{f+1})} && (18)\\
\mathbf{b}_l(s) &= \frac{\Sigma_{f=1}^{F}\gamma_f(s_f, l)}{\Sigma_{f=1}^{F}\Sigma_l^L\gamma_f(s_f, l)} && (19)\\
\mu_l &= \frac{\Sigma_{f=1}^{F}\Sigma_{s=1}^{S}\gamma_f(s_f, l)y_f}{\Sigma_{f=1}^{F}\Sigma_{s=1}^{S}\gamma_f(s_f, l)} && (20)\\
\sigma_l &= \frac{\Sigma_{f=1}^{F}\Sigma_{s=1}^{S}\gamma_f(s_f, l)(y_t - \mu_l)(y_t - \mu_l)^T}{\Sigma_{f=1}^{F}\Sigma_{s=1}^{S}\gamma_f(s_f, l)} && (21)
\end{aligned}$$

The symbols in bold are used to indicate the estimated values that will be used for the next iteration.

2.5 Julius Recognition Engine

This work makes use of the Julius recognition engine. The Julius recognition engine is a two pass recognition engine that uses the concept of a word trellis index [2]. A word trellis index consists of a first pass where paths of Viterbi scores of word-end nodes within a predefined beam are kept. In a second pass word boundaries are determined using stack decoder, in which the trellis from the first pass is connected as backward heuristic, since also the word beginning frame corresponding to each end node of a survived word (word not prunned) is stored.

3. Acoustic Modeling by Tied-Mixture Successive State Split Algorithm

In this section we will describe the proposed Fast Tied-Mixture Successive State Split algorithm (FTM-SSS) and point out the differences in respect to the original TM-SSS algorithm, during the description.

This section shows how a divisive state clustering algorithm that generates acoustic Hidden Markov Models (HMM) can benefit from a tied-mixture representation of the probability density function (pdf) of a state and increase the recognition performance.

Popular decision tree based clustering algorithms, like for example the Successive State Splitting algorithm (SSS) make use of a simplification when clustering data. They represent a state using a single Gaussian pdf. We show that this approximation of the true pdf of a state by a single Gaussian is too coarse, for example a single Gaussian cannot represent the differences in the symmetric parts of the pdf's of the new hypothetical states generated when evaluating the state split gain (which will determine the state split).

The use of more sophisticated representations lead to intractable computational problems that we solve by using a tied-mixture pdf representation of a state. Additionally, we constrain the codebook to be immutable during the split. Between state splits, this constraint is relaxed (the codebook is updated).

The proposed algorithm is then an extension to the SSS algorithm, the so-called Tied-mixture Successive State Splitting algorithm (TM-SSS).

3.1 Tied-Mixture Successive State Splitting (TM-SSS)

Creation of precise and robust acoustic models for speech recognition, given a limited amount of training data, has been an intensive field of research over the last 10 years [16, 19]. The approach in most successful speech recognition systems has been based on some type of decision tree based splitting or clustering to automatically find an efficient parameter sharing structure, i.e. tying states, sharing transition probabilities, tying mixtures and so on. Examples of successful algorithms for this type of automatic acoustic modeling are decision tree clustering using phonetic questions [18, 14] and ML-SSS [5].

Of particular interest in this thesis is the robustness and degree of representation that a tied-mixture representation yields in an HMM, in particular at the clustering phase. During the clustering phase all states are hypothetically splited in two. The process consists in matching the pdf of the data against the hypothetical pdf's, evaluating a split (match) gain for each state that we consider for split. The data is assigned to the hypothetical state that best matches its pdf and the state to split is the one that offers the highest split gain.

The problem is the error in the clustering process we may be committing when we ignore the Gaussian indistinguishable component that any pdf has. The previous algorithms work as if the pdf of the split state were well behaved as in Fig. 3. In this figure we show a triphone set whose center phonemes b, d, g are considered for a split (state S). In this example the phonemes b, d (state S_1) have quite different mean values if compared to phoneme g (state S_2). The two new hypothetical states can thus be distinguished by single Gaussians.

However Fig. 4 shows an extreme case where the resulting two new Gaussians are not distinguishable by a single Gaussian. In Fig. 4 the triphone set to be split represents a state with center phonemes m, n, preceded by phoneme a and succeeded by phoneme e. When this triphone set is split the two center phonemes m and n are more and less influenced, respectively, from the neighboring phonemes a, e, resulting in two new states with same mean and variance.

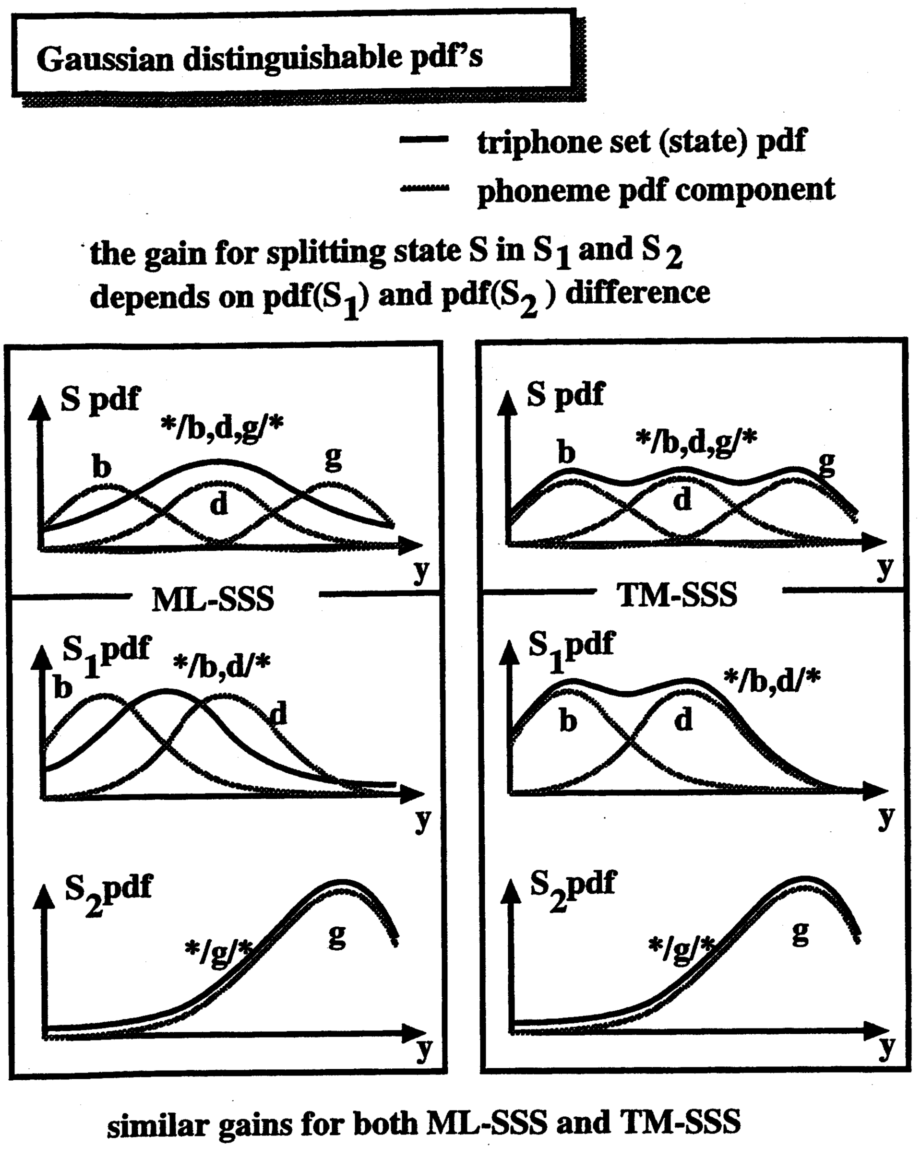
Gaussian distinguishable pdf's
— triphone set (state) pdf
— phoneme pdf component
the gain for splitting state S in S1 and S2 depends on pdf(S1) and pdf(S2) difference
S pdf
/b,d,g/
b
d
g
y
ML-SSS
TM-SSS
S1pdf
/b,d/
S2pdf
/g/
similar gains for both ML-SSS and TM-SSS

Figure 3. State split with only Gaussian distinguishable pdf's. Here a single Gaussian is enough to distinguish the two new hypothetical states. Here, the R dimensional observation vector y is plotted as a single dimension to simplify the plot.

Despite their different true pdf's, they are indistinguishable from the point of view of a single Gaussian distribution representation and thus the split gain will be zero. However, if we could calculate the gain using the true pdf's and not the approximation using a single Gaussian, this state split might give us the maximal gain. In other words, the state split order might be different.

ML-SSS uses a single Gaussian during the state split gain evaluation and hence its split gain is not affected by the differences that are Gaussian indistinguishable in the resulting pdf's of the two new states. Obviously, using a Continuous Density (CD) mixture representation with a sufficiently large number of mixands (Gaussians of a state) would result in a more realistic gain calculation. However, the CD implementation would be computationally non-tractable as the necessary sufficient statistics would have to be recalculated from the data samples for each hypothetical split. In this thesis, we therefore propose a tied-mixture (TM) representation. As we will show in later sections, the TM representation makes it computationally feasible to evaluate the split hypotheses.

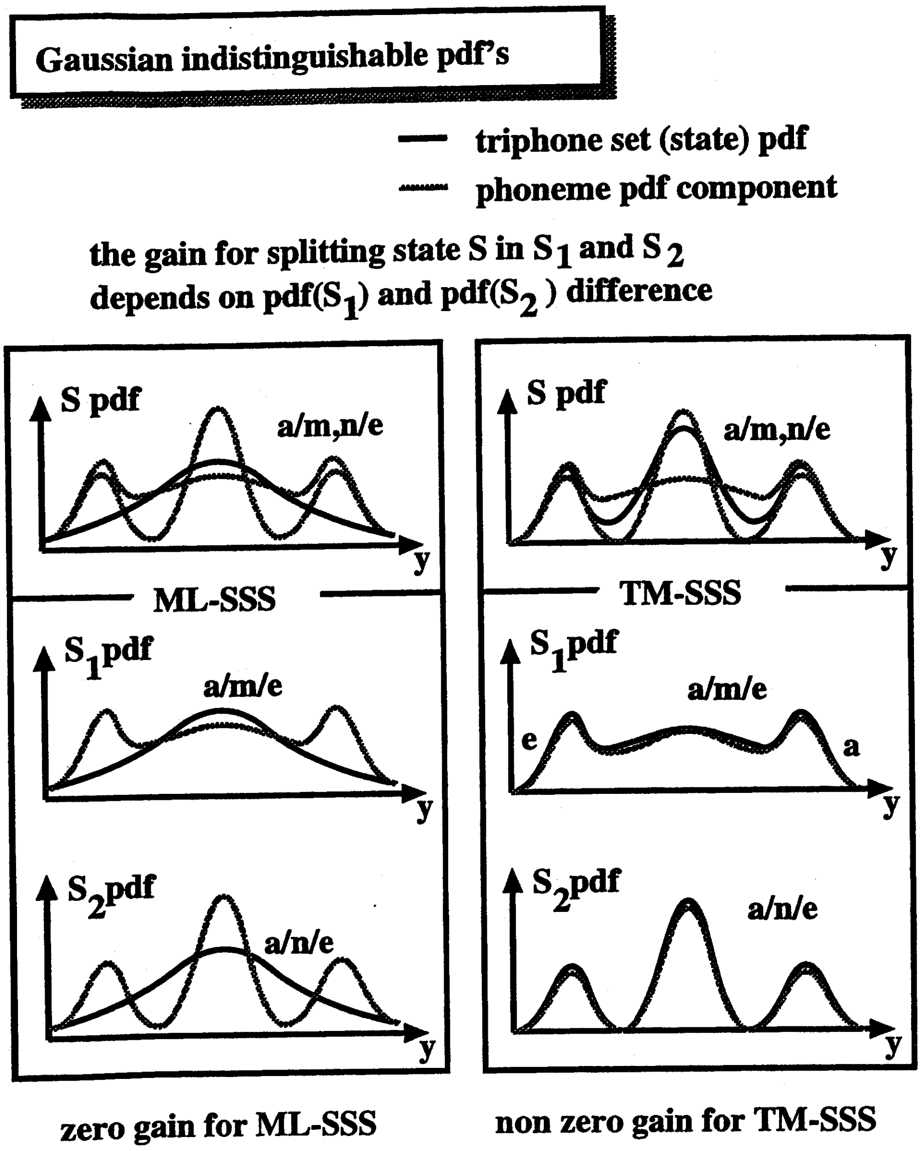

Figure 4. State split with only Gaussian indistinguishable pdf's. Here we see an example of resulting new hypothetical pdf's that a single Gaussian representation is not capable to distinguish. Here the R dimensional observation vector y is plotted as a single dimension to simplify the plot.

The rest of this section is divided as follows: in Sec. 3.2.1 we describe the TM-SSS algorithm to solve the pdf representation problem and the clustering process in more detail. In Sec. 3.3.1 we apply the new algorithm to a word recognition task in order to verify its usefulness and discuss the results. Finally, in Sec. 5 we draw conclusions.

3.2 Reformulation of ML-SSS

In order to represent the pdf of a state during the split phase by a tied-mixture representation, the proposed algorithm, from now on called Tied-Mixture Successive State Splitting algorithm (TM-SSS), introduces several changes to the basic algorithm (ML-SSS).

3.2.1 TM-SSS algorithm

The TM-SSS algorithm starts with a simple HMM constructed to represent all the phonemes in the database. In our case a three state left-to-right HMM is used. The Baum-Welch algorithm is applied to the initial HMM to get the state occurrence counts and to update the HMM parameters.

From the statistics of the re-estimation, a split gain is evaluated simulating a state split. For each candidate state split s^c, a gain $G(s^c)$ is obtained that is dependent on the pdf that this state represents. The state that gives the maximum expected likelihood gain $s^* = \text{argmax}_s^c\, G(s^c)$ is split and all the affected states are re-estimated with the Baum-Welch algorithm. Affected states are all states in HMM's which contain the split state. The process is repeated, i.e. the HMnet is grown until it reaches a pre-defined state number, the increase in gain is negligible or there are no more states to split. Detailed figures and formulas will be shown in Sec. 3.2.2.

Finally, triphone contexts that are lost during the split, the so-called unseen triphones, are placed in the resulting HMnet structure. This is performed using a tree of non overlapping phoneme space environments [19]) that is produced as a result of the TM-SSS training. The unseen triphones (gaps and holes of the phoneme space environment) are placed in the leaves of this tree, based on the co-occurrence counts of phoneme labels collected from the split history.

The proposed algorithm is as follows:

```
                         TM-SSS Algorithm

1. create a codebook
    (a) flat start (estimate means and variances)
    (b) vector quantization to get the codebook
    (c) Baum-Welch (BW) of initial HMM
2. iterate over the candidate states:
    (a) get state split information for each domain and factor
    (b) find best split for each domain and factor
    (c) split the state with the highest expected likelihood gain
    (d) if update codebook (state split pertain to affected states since last
        codebook update)
          i. run BW, train means, variances, weights and transitions for all
             states
         ii. use only the most significant $L$ mixands
        else
          i. run BW over affected states (states in HMM's which contain the
             split state), only train weights and transitions
3. repeat last step until number of states reached or increase in gain is negli-
   gible
4. recover unseen triphones contexts based on split history
```

Get State Split Information

```
1. iterate over the domains and factors
   (preceding, center or succeeding phonemes):
   (a) from original state create two new hypothetical states
   (b) iterate until convergence is met
       i. for all data
          A. assign data to the more likely state
       ii. estimate new states and split gain
```

For a given data sample, there is a unique path through the HMnet. It means that when we split a state we are assigning the data that is passing that state to either of the two new states. As the number of possible combinations of splits may be too large, a recursive algorithm [15, 22] is used to find the optimal split: the initial state is copied to provide two new states to start the recursion, one which is a copy of the original state and a second that is a perturbed copy of it.

All the data that passed through the original state is assigned one by one to the most likely state to represent it. From the two sets of data the pdf parameters for the new states are re-estimated and the process is repeated until a convergence criterion is met.

What simplifies the use of a tied-mixture system is that we constrain the codebook to be immutable during the estimation of the gain $G(s^c)$ of the candidate state split s^c. Hence just adjusting the mixture weights is enough to evaluate the state split. This constraint may be considered well satisfied when we have a large number of states in the HMnet, but might be too strong a constraint when the state split process is still at the beginning.

More often we update the codebook, better the final HMnet is, however for computational reasons the codebook is updated only if the current state split s^* pertains to one of the affected states since the last codebook update. By doing so, we carry out a full BW (Baum-Welch) more often when we have a small number of states (when a state split affects more the codebook) and less vice-versa.

The training algorithm follows the classical tied-mixture BW algorithm [7]. It is used after a state split to accommodate small differences that may arise from the fixed codebook constraint during the split and the influence of the split in neighboring states.

In order to derive the gain function we define the observed data y_1^T as $y_1^T = \{y_1, y_2, \ldots, y_T\}$ and the related hidden states s_1^T as $s_1^T = \{s_1, s_2, \ldots, s_T\}$.

Requiring that means and variances of the codebook are constant during the split, as stated above, the gain function is expressed as a difference of the expected log likelihood before and after the split. For each state split iteration k the expected log likelihood becomes:

$$\begin{aligned}
&Q(\theta^{(k+1)}|\theta^{(k)}) = E[\log \mathrm{P}(y_1^T, s_1^T|\theta^{(k+1)})|y_1^T, \theta^{(k)}] \\
&= \sum_{s_1^T} \mathrm{P}(s_1^T|y_1^T, \theta^{(k)}) \log \mathrm{P}(y_1^T, s_1^T|\theta^{(k+1)}) \\
&= \sum_{s:s=s_t, s'=s_{t-1}}^{S} \sum_t \xi_t(s, s') \log a(s, s') \\
&+ \sum_{s:s=s_t}^{S} \sum_t \sum_l \gamma_t(s, l) \log b_l(s) \\
&+ \sum_{s:s=s_t}^{S} \sum_t \sum_l \gamma_t(s, l) \log N(y_t|\mu_l, \Sigma_l)
\end{aligned} \tag{22}$$

where

$$\begin{aligned}
\gamma_t(s, l) &= \mathrm{P}(s_t = s, \nu_l|y_1^T, \theta^{(k)}) \\
\xi_t(s, s') &= \mathrm{P}(s_t = s, s_{t-1} = s'|y_1^T, \theta^{(k)}) \\
a(s, s') &= \mathrm{P}(s_t|s_{t-1}, \theta^{(k+1)}) \\
b_l(s) &= \mathrm{P}(\nu_l|s_t, \theta^{(k+1)}) \\
N(y_t|\mu_l, \Sigma_l) &= \mathrm{P}(y_t|\nu_l, \theta^{(k+1)})
\end{aligned}$$

and $a(s, s')$ is the transition probability from the state $s = s_{t-1}$ to state $s' = s_t$, $b_l(s)$ is the weight of the codebook vector ν_l and $N(y_t|\mu_l, \Sigma_l)$ is the Gaussian mixture of the observation y_t with respect to ν_l. θ are the HMnet parameters.

Note that this equation allows us, assuming that the transition probabilities are fixed as in ML-SSS[5], to consider only the contribution to $Q(\theta^{(k+1)}|\theta^{(k)})$ that comes from the candidate split state s^c. Hence, our gain $G(s^c)$ resumes to:

$$\begin{aligned} G(s^c) = \;& Q(\theta^{(k+1)}|\theta^{(k)})(s_1, s_2) - Q(\theta^{(k+1)}|\theta^{(k)})(s^c) \\ = \;& \textstyle\sum_{s,s'} M_1(s,s') \log a(s,s') - M_1(s^c,s^c) \log a(s^c,s^c) \\ + \;& \textstyle\sum_l [\sum_s M_2(s,f,l) \log b_l(s) - M_2(s^c,f,l) \log b_l(s^c)] \\ + \;& \textstyle\sum_{j(s^c,f)} [M_3(s_1,j) + M_3(s_2,j) - M_3(s^c,j)] \end{aligned} \tag{23}$$

where

$$\begin{aligned} M_1(s,s') &= \sum_{j(s,f)} \xi_{j(s,f)}(s,s') \\ M_2(s,f,l) &= \sum_{j(s,f)} \sum_{t:x_t=x_{j(s,f)}} \gamma_t(s,l) \\ M_3(s,j) &= \sum_{t:x_t=x_{j(s,f)}} \sum_l \gamma_t(s,l) \log N(y_t|\mu_l,\Sigma_l) \end{aligned}$$

and j represents a phoneme, with $j \in j(s,f)$, where $j(s,f)$ represents all the generalized allophones [19] in a certain factor $f \in \{preceding, center, succeeding\}$ phonemes for state s. With $(s,s') \in \{(s_1,s_1)\}$ for the contextual domain split and $(s,s') \in \{(s_1,s_1),(s_1,s_2),(s_2,s_2)\}$ for the temporal domain split (see Fig. 5).

The first term $M_1()$, in equation 23 is independent of the chosen factor f and constant due to the constraint of fixed transition probabilities during the split. Additionally due to the constraint of constant codebook during the state split, the last term $M_3()$ is constant too. These constraints reduce the calculation of the gain to the evaluation of the term $M_2()$.

3.2.2 Contextual Split

In the contextual split as used in TM-SSS, the aim is to split a state s^c in two new states s_1 and s_2 based on the observation data y associated with the split state s^c. Associated with the split state is the phoneme space A. Accordingly the two new phoneme subspaces A_1 and A_2 with $A_{s^c} = A_1 \cup A_2$ refer to s_1 and s_2, respectively.

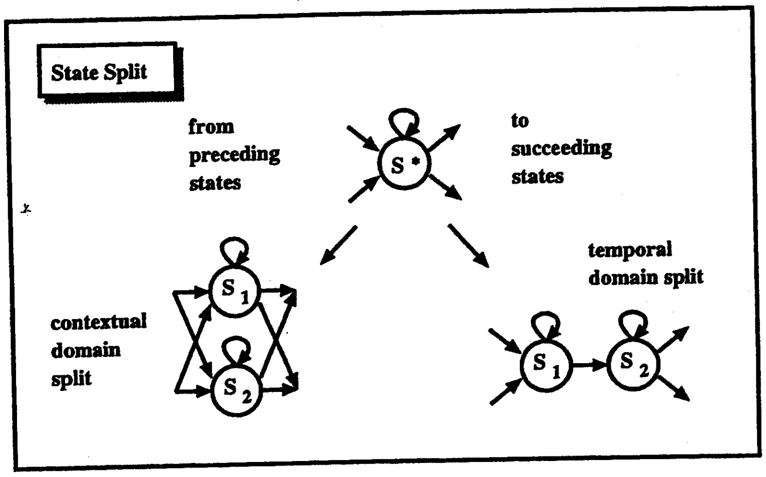

Figure 5. Example of split of state s^* into states s_1 and s_2 for contextual and temporal domains.

Each observation is described by a set of factors, i.e., its preceding, center and succeeding phonemes. A contextual split may only be performed on a factor at a time. Additionally let us group the observation data y_t by phonemes j with $y_j \in y_{j(s,f)}$, where $j(s,f)$ represent a set of phonemes in a certain factor f and state s.

As in TM-SSS the purpose is to represent the HMnet using tied-mixtures. A state s will be represented by its output distribution vector $b(s)$ with L components $b(s) = (b_1(s), \ldots, b_l(s), \ldots, b_L(s))$, where l represents one of the codebook indices of the VQ (Vector Quantization). The goal is then to estimate two new vectors of output distribution coefficients $b(s_1), b(s_2)$, so that the maximum likelihood criterion is observed. This is basically a divisive clustering problem. To solve this problem we use Chou's partitioning algorithm [15] as in ML-SSS [11], choosing the loss function

$$\mathcal{L}(y_1^T, b) = -\log \mathrm{P}(y_1^T | b) \tag{24}$$

Chou's partitioning algorithm is applied here, since it assures that the optimal partition in the sense of maximal likelihood will be selected and that it is linear on the number of clustered phonemes $j(s,f)$. We perform clustering of the data taking into account the whole phoneme, i.e. we do not use Viterbi alignment to extract the data for the split state. In fact we are clustering phonemes between two HMM's which differ only on the split state s^c that may take values in s_k where $k = 1, 2$.

Under this objective equation 24, the "centroid" function becomes

$$\begin{aligned} b(s_k) &= \operatorname*{argmin}_{b(s_k)} E[\mathcal{L}(y, b(s_k))|s^c] \\ &= \operatorname*{argmax}_{b(s_k)} E[\log \mathrm{P}(y|b(s_k))|s^c] \\ &= \operatorname*{argmax}_{b(s_k)} \sum_t \sum_l \gamma_t(s^c, l) \log b_l(s_k) + K \end{aligned} \tag{25}$$

K turns out to be constant for state s^c due to the unaffected terms depending on constrained fix states, transition probabilities and Gaussian probabilities. The new transition terms and Gaussian terms sum their counts to the same value before the hypothetical split.

The maximum in this function is given by the re-estimation formulas of $b(s)$ evaluated from the clustered data. The divergence function, not considering the constant terms for s_1, becomes

$$\begin{aligned} & d(y, b(s_1)) = E[\mathcal{L}(y, b(s_1))|s^c] - i(s) \\ & = - \sum_{j \in j(s_1, f)} \sum_{t: y_t \in y_j} \sum_l \gamma_t(s^c, l) \log \mathrm{P}(y_t|b_l(s_1)) \\ & + \sum_{j \in j(s^c, f)} \sum_{t: y_t \in y_j} \sum_l \gamma_t(s(j), l) \log \mathrm{P}(y_t|b_l(s(j))) \end{aligned} \tag{26}$$

The phonemes $j \in j(s, f)$ are clustered matching them to its *nearest neighbor* $b(s_k)$. $i(s)$ is the minimum impurity as defined in Chou's theorem [15].

For a single phoneme $j \in j(s, f)$ the divergence becomes

$$\begin{aligned} & d(y_j, b(s_1)) = E[\mathcal{L}(y_j, b(s_1))|s] - i_j(s) \\ & = - \sum_{t: y_t \in y_j} \sum_l \gamma_t(s^c, l) \log \mathrm{P}(y_t|b_l(s_1)) \end{aligned}$$

$$+ \sum_{t:y_t \in y_j} \sum_{l} \gamma_t(s(j), l) \log P(y_t | b_l(s(j))) \tag{27}$$

The inequality used to split the phoneme j from space A to the spaces A_k, according to Chou's theorem is defined as

$$\begin{array}{lll} j \in A_1 & \text{if} & d(y_j, b(s_1)) < d(y_j, b(s_2)) \\ j \in A_2 & \text{otherwise} & \end{array} \tag{28}$$

Substituting the distance equation 27 in the inequality 28 we get, for the ith clustering iteration, the split equation

$$\sum_{l} \gamma_j(s^c, l) \log b_l^{(i-1)}(s_1) \geq \sum_{l} \gamma_j(s^c, l) \log b_l^{(i-1)}(s_2) \tag{29}$$

where

$$\gamma_j(s, l) = \sum_{t:x_t = x_{j(s,f)}} \gamma_t(s, l)$$

In this equation we are not considering the minor effects that fixed mean μ_l and variance Σ_l may have on the divergence as a distance function.

The state split is performed, from the split state s^c, initially by creating two states s_1 and s_2, i.e. two new set of mixture weights.

In the contextual domain the initial weight vector for state s_1 is obtained by copy (to assure that the likelihood will not decrease) and for state s_2 by disturbing the original weights $b_l(s^c)$ by $(1 + (-1)^l \epsilon')$. The new weights obtained are then renormalized, so that they still sum up to 1. ϵ' is arbitrarily chosen, typically $1.0e^{-3}$.

From the clustered data new tied-mixtures weights for the two new hypothetical states are re-estimated, constituting the two new centroids for the next iteration. The clustering is iterated until the change in the state split gain becomes smaller than an arbitrarily chosen ϵ, typically $1.0e^{-5}$. If there is no positive split gain, the value of ϵ is reduced by a factor of 10 as many times as necessary until a gain is achieved.

The split design algorithm may be summarized as follows.

Split Design Algorithm for Factor f

(All the terms below depend on f, but f is fixed here so it will be omitted)

- **Initialization** (iteration $i = 0$) Initialize the distribution parameter centroids for the two new hypothetical states:

$$\begin{aligned} b^{(0)}(s_1) &= \left[b_1^{(0)}(s_1), ...b_l^{(0)}(s_1), ...b_L^{(0)}(s_1)\right] \\ &= [b_1(s^c), ...b_l(s^c), ...b_L(s^c)] = b(s^c) \\ b^{(0)}(s_2) &= \left[b_1^{(0)}(s_2), ...b_l^{(0)}(s_2), ...b_L^{(0)}(s_2)\right] \\ b_l^{(0)}(s_2) &= (1 + (-1)^l \epsilon) b_l(s^c) \forall l \end{aligned} \tag{30}$$

- **Iterate** for $i = 1, 2, \ldots$

  1. Find new binary partition $\{A_1^{(i)}, A_2^{(i)}\}$:
     For each $j : j \in j(s, f)$, assign all the $y_t : y_t \in y_j$ to $A_1^{(i)}$ if

$$\begin{aligned} &\sum_l \gamma_j(s^c, l) \log b_l^{(i-1)}(s_1) \\ &\geq \sum_l \gamma_j(s^c, l) \log b_l^{(i-1)}(s_2) \end{aligned} \tag{31}$$

     otherwise, assign all the y_t to $A_2^{(i)}$.

  2. Find new centroids $\{b^{(i)}(s_k) : k = 1, 2\}$.

$$b_l^{(i)}(s_k) = \frac{\sum_{j \in j(s_k, f)^{(i)}} \gamma_j(s, l)}{\sum_{j \in j(s_k, f)^{(i)}} \sum_l \gamma_j(s, l)} \tag{32}$$

     where $\gamma_j(s, l)$ have been stored during the previous BW re-estimation.

3. Test for convergence: stop if the partition does not change or if

$$\frac{G(s^c)^{(i)} - G(s^c)^{(i-1)}}{G(s^c)^{(i-1)}} < \eta$$

where $G(s^c)$ is the contextual gain (Equation 23) and η is an heuristically chosen convergence threshold. Note that $G(s^c)^{(i)} \geq G(s^c)^{(i-1)}$.

3.3 Experimental Data

Isolated word recognition experiments for one male Japanese speaker (speaker MHT of ATR database [4]) were performed to show the effectiveness of the TM-SSS algorithm. The database consists of one utterance each of the most frequent 5240 Japanese words labelled with 55 phoneme labels.

We use the ML-SSS algorithm as a baseline for comparison with the TM-SSS algorithm in the experiments that follow.

A codebook size common to both TM-SSS and ML-SSS models was set to 1024 Gaussians. In order to keep the codebook size fixed among all models, ML-SSS HMnets were retrained with multiple mixands per state. For example, the 256 state ML-SSS HMnet was retrained with 4 mixands per state, resulting in a comparable number of 1024 Gaussians per HMnet.

Table 1. Feature extraction.

parameter	Value
sampling rate	16000 Hz
frame shift	10 ms
frame length	25 ms
pre-emphasis coef.	0.97
parameters	12 MFCC, 12 Δ MFCC, 1 Δ log power

Table 1 describes the feature extraction.

3.3.1 Speaker-dependent HMnet Experiments

The experiments were conducted under the following conditions:

- The initial HMnet consists of 3 states aligned left to right, representing all the phonemes in all the contexts.
- Only contextual split was performed.
- TM-SSS is performed with codebook of 1024 Gaussians.
- HMnets are split up to 256 and 512 states.
- ML-SSS HMnets are retrained with 4 and 2 Gaussians per state, respectively.
- Embedded re-estimation, i.e. not using time information of the phoneme labels, is performed to compensate for labeling errors.

Up to here we have trained two sets of models, one a set of TM (Tied-Mixture) models, obtained using the TM-SSS algorithm and another a set of GM (Continuous Density Gaussian Models) models, constructed using the ML-SSS algorithm.

For a fairer comparison between the TM-SSS and ML-SSS algorithms we also estimated a TM model from a GM model and vice-versa. In this case, neither a difference in the number of parameters, nor a difference in the HMM structure exists at all. These converted models were created as follows:

- TM-SSS models are converted to GM models, by constraining the number of mixands per state to 4 and 2, for HMnets with 256 and 512 states per model, respectively.
- Similarly, ML-SSS models are converted to TM models, by initially taking 10% of the mixand weights and distributing them uniformely as mixands weights for the mixands of the remaining states.

All converted models are finally retrained using embedded re-estimation with the training data, in order to optimize the model parameters to the new model structure.

3.3.2 Word Recognition Results

Recognition results for TM-SSS and ML-SSS algorithms and their respective converted models are summarized in Table 2 and Table 3. The training data are the 2620 even-numbered words and the test data are the 2620 odd-numbered words.

Table 2. Word recognition error (%) for ML-SSS and TM-SSS with 1024 Gaussians and 256 states each.

training	final topology	
algorithm	TM	GM
TM-SSS	2.79	3.74
ML-SSS	5.27	5.27

Table 3. Word recognition error (%) for ML-SSS and TM-SSS with 1024 Gaussians and 512 states each.

training	retrain topology	
algorithm	TM	GM
TM-SSS	2.21	3.63
ML-SSS	4.08	4.08

In Table 2 and Table 3 training algorithm refers to the algorithm used to get the first HMM topology, in the case of TM-SSS resulting in a tied-mixture topology (TM) and in the case of ML-SSS in a continuous density Gaussian mixture topology (GM). The results of TM-SSS algorithm with a GM topology and of ML-SSS with a TM topology refer to the converted models.

Table 2 and Table 3 show that an HMM model constructed using the TM-SSS algorithm nearly halves the recognition error rate compared to an HMM model created using the ML-SSS algorithm. This holds true for splitting up to 256 states and 512 states.

For a meaningful analysis, the two approaches have to be compared within the same topology type, i.e. TM and GM, respectively, so that the total number of free parameters is the same even at the level of the number of mixand weights.

For the GM topology, when the TM topology obtained using TM-SSS was converted to a GM topology, the TM-SSS algorithm still did better (3.74% vs 5.27% for 256 states, 3.63% vs 4.08% for 512 states).[1] This proves that TM-SSS algorithm is effective on its choice of final shared state triphones for the HMnet.

Taking the TM topology as a point for comparison, we also did better with the TM-SSS algorithm, because the GM model obtained using the ML-SSS algorithm did not show a decrease in the error rate, because of the well-known sensitivity of HMM's to initial values. In other words, although more free parameters are available in the TM topology, these parameters are not used efficiently and the error rate remained at 5.27% for 256 states and 4.08% for 512 states. In both cases the corresponding TM-SSS trained models outperform the ML-SSS trained models (2.79% vs 5.27% for 256 states, 2.21% vs 4.08% for 512 states).

3.3.3 Center Phoneme

Another way to measure the effectiveness of the new approach is to analyze how the phonemes have been split. In an HMnet, every state path defines a model representing a set of triphones (or generalized allophones). In SSS these sets of triphones are convex spaces which may be represented by sets of phonemes, each one representing one context (preceding, center and succeeding phoneme set).

For an HMnet with 256 states, almost all the center phonemes should have already been split. Table 4 and Table 5 shows the triphones whose center phonemes remain unsplit in the final HMnet, as well as their occurrence for 256 and 512 states respectively.

We can see from Table 4 and Table 5 that using the TM-SSS algorithm almost all the center phonemes have been split. With ML-SSS an important amount of unsplit center phonemes, equivalent to 27.2% of all triphones contexts, remains. We think that the improvement with TM-SSS is due to the more accurate split gain calculation.

[1]The increase in error rate from 2.79% to 3.74% for 256 states, 2.21% to 3.63% was somehow expected as the total number of free parameters is reduced by the conversion.

Table 4. Unsplit center phonemes for HMnet with 256 states (speaker MHT).

ML-SSS %	TM-SSS %
(ao, o, o-) (o, o-) (a, a-, aa) (ou, oo) (e, ei, ee) (ei ee) (m, g) (gy, ny) (t, p) (k, t, p) (ch, hy)	(e, ee)
27.18%	2.96%

Table 5. Unsplit center phonemes for HMnet with 512 states (speaker MHT).

ML-SSS %	TM-SSS %
(ao, o, o-) (o, o-) (a, a-, aa) (ou, oo) (ei, ee) (m, g) (gy, ny) (t, p) (k, t, p) (ch, hy) (a, aa) (ao, o)	
21.02%	0.0%

3.4 Summary

In this section we proposed a new algorithm named TM-SSS (Tied-mixture Successive State Splitting). This new algorithm extends the use of tied-mixture representation not only to the final phase of an acoustic model generation algorithm, but also to its topology training phase.

For the same number of Gaussians the results using TM-SSS are about 31% (relative decrease in error rate) better than using the ML-SSS algorithm.

The superiority of the TM-SSS algorithm is shown not only by its recognition rate but also by the visible quality of the final phoneme split, as attested by its performance on splitting the center phonemes.

Despite its good performance it has a major draw back that is its computational cost for large tasks. That is exactly the problem we tackle in the next section.

4. Fast Tied-Mixture Successive State Splitting (FTM-SSS)

TM-SSS (Tied-Mixture Successive State Splitting) is an algorithm that automatically generates context dependent HMM's with shared states, transition probabilities and a Gaussian codebook. This algorithm proved to achieve higher recognition rates than ML-SSS (Maximum Likelihood Successive State Splitting) for a single speaker, but the algorithm was too slow for speaker independent tasks. The most demanding computational task was to update the codebook every time a state is split. Previous attempts to reduce the number of splits resulted in poorer recognition rates.

However we found that once the codebook provides an unbiased representation of the acoustic space we can keep it constant during the state split process, updating it just once the HMM's set is defined and thus saving a lot of computation, without significantly affecting the recognition rate. Besides this reduction, further improvements were possible due to this approach, which allowed us to train speaker independent male and female acoustic models in a reasonable time.

In this section, we thus propose an extension to the TM-SSS algorithm, the so-called Fast Tied-Mixture Successive State Splitting algorithm (FTM-SSS). The recognition results for similar number of free parameters between monophone and FTM-SSS triphone (FTM-SSS based) acoustic models showed about the same recognition rate for male and an increase of about 3.6% for female, requiring only 1.5 times the computation amount of a monophone acoustic model.

As a result the FTM-SSS has been applied to the ASJ (Acoustic Society of Japan) and JNAS (Japanese Newspaper Article Sentences) database (more than 60 hours). Recognition results for the 100 sentences taken from the IPA (Information-technology Promotion Agency) CD-ROM, for both male and female acoustic models are reported. The performance is also compared to the models present in the distribution that had been generated with the HTK toolkit. The FTM-SSS acoustic models represent an increase in recognition accuracy of 2% when compared to monophones, with a similar number of free parameters.

4.1 Speeding up TM-SSS

In this section we will describe the proposed FTM-SSS algorithm and point out the differences in respect to the original TM-SSS algorithm, during the description.

Of particular interest in this thesis is the behavior of the Gaussian codebook during the state split process in the case of the TM-SSS [1] algorithm. In Fig. 6a we see a set of allophones being split a couple of times, according to the TM-SSS algorithm. Trying to reduce the number of times we update the codebook has an adverse effect as shown in Fig., 6b. The idea is then to keep the error caused by the codebook constant, so that the state split order is minimally affected by the codebook, Fig. 6c shows how the proposed method behaves.

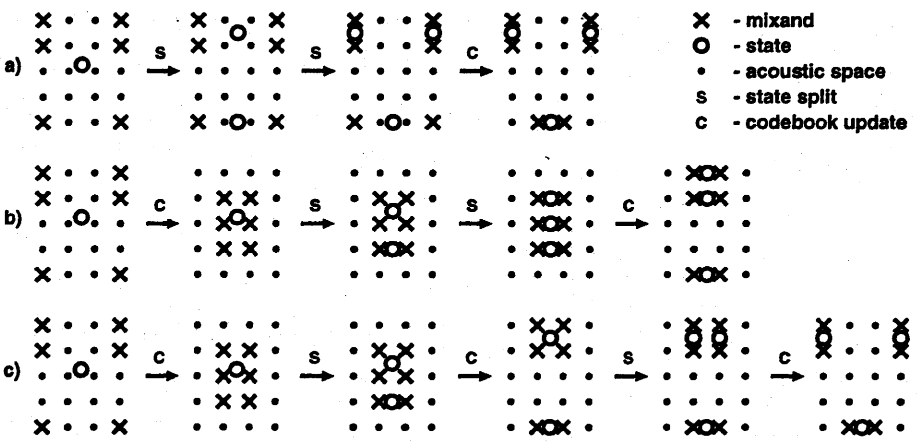

Figure 6. State split and means of codebook moves with Baum Welch (BW) re-estimation. a) just splits ending with BW, b) BW alternated with more than one state split, c) alternated BW and state split. a and c behave similarly, but b leads to non optimal acoustic model structures.

With the original TM-SSS we needed to train all the data together, i.e. we couldn't separate data in its central phoneme and train it in separate processes, because the codebook was shared among all the data. Now with a fixed codebook this is possible, so we can take advantage of multiprocessor computers to speed up the acoustic model generation.

4.1.1 FTM-SSS Algorithm

The original TM-SSS algorithm starts with a simple HMM constructed to represent all the phonemes in the database. In FTM-SSS we start with all the center phonemes split so we can process different phonemes in parallel. In TM-SSS it is in fact a disadvantage to split in advance the center phonemes, because the codebook must be updated during the state split process, so we can not benefit from splitting in advance the center phonemes for parallel processing purposes.

Both algorithms are usually initialized with a three state left-to-right HMM for representing its initial allophones. In TM-SSS the Baum-Welch algorithm is applied to the initial HMM to get the state occurrence counts and to update the HMM parameters. In FTM-SSS the codebook is just the result of the VQ (Vector Quantization) of the database.

From the statistics of the re-estimation, a split gain is evaluated simulating a state split. For each candidate state split s^c, a gain $G(s^c)$ is obtained that is dependent on the pdf that this state represents. The state that gives the maximum expected likelihood gain $s^* = \text{argmax}_s^c\, G(s^c)$ is split and all the affected states are re-estimated with the Baum-Welch algorithm. Affected states are all states in HMM's which contain the split state. The process is repeated, i.e. the HMnet is grown until it reaches a pre-defined state number, the increase in gain is negligible or there are no more states to split (the gain function is below 1.0e-5).

The difference between TM-SSS and FTM-SSS is that in the first the Baum-Welch procedure over the affected states updates all the parameters of the acoustic model, including the codebook, while in the second only the transition probabilities and mixand weights are updated.

Finally, triphone contexts that are lost during the split, the so-called unseen triphones, are placed in the resulting HMnet structure. This is performed using a tree of non overlapping phoneme space environments [19]) that is produced as a result of the TM-SSS training. The unseen triphones (gaps and holes of the phoneme space environment) are placed in the leaves of this tree, based on the co-occurrence counts of phoneme labels collected from the split history.

At the end of the acoustic model generation in case of FTM-SSS a couple of Baum-Welch interactions are carried out to also accommodate the codebook to the final acoustic model.

The original TM-SSS and the proposed FTM-SSS algorithms are described below in a compact form:

```
                    TM-SSS Algorithm

1. create a codebook
    (a) flat start (estimate means and variances)
    (b) vector quantization to get the codebook
    (c) Baum-Welch (BW) of initial HMM
2. iterate over the candidate states:
    (a) get state split information for each domain and factor
    (b) find best split for each domain and factor
    (c) split the state with the highest expected likelihood gain
    (d) if update codebook [2]
            i. run BW, train means, variances, weights and transitions for all
               states
           ii. use only the most significant $L$ mixands
        else
            i. run BW over affected states (states in HMM's which contain the
               split state), only train weights and transitions
3. recover unseen triphones contexts based on split history
```

[2]As stated at the beginning of the Sec. 4 it is beneficial, from the point of view of recognition accuracy, to always update the codebook (and that is the approach used in TM-SSS when comparing with FTM-SSS).

```
FTM-SSS Algorithm

1. create a codebook
   (a) flat start(estimate means and variances)
   (b) vector quantization to get the codebook
2. iterate over the candidate states:
   (a) get state split information for each domain and factor
   (b) find best split for each domain and factor
   (c) split the state with the highest expected likelihood gain
   (d) run BW over affected states, only train weights and transitions
3. iterate over all data
   (a) run BW, train means, variances, weights and transitions for all states
4. recover unseen triphones contexts based on split history
```

Both TM-SSS and FTM-SSS algorithm share the same procedure to get the state split information as follows:

```
Get State Split Information

1. iterate over the domains and factors
   (preceding, center or succeeding phonemes):
   (a) from original state create two new hypothetical states
   (b) iterate until convergence is met
       i. for all data
          A. assign data to the more likely state
       ii. estimate new states and split gain
```

For a given data sample, there is a unique path through the HMnet. It means that when we split a state we are assigning the data that is passing that state to either of the two new states. As the number of possible combinations of splits may be too large, a recursive algorithm [15, 22] is used to find the optimal split: the initial state is copied to provide two new states to start the recursion, one which is a copy of the original state and a second that is a perturbed copy of it (mixand weights multiplied by 1.0e-3).

All the data that passed through the original split state is assigned to the most likely hypothetical state. From the two sets of data the pdf parameters for the new states are re-estimated and the process is repeated until a convergence criterion is met.

What simplifies the use of a tied-mixture system is that we constrain the codebook during the state split to be immutable. Hence just adjusting the mixture weights is enough to evaluate the state split.

The training algorithm follows the classical tied-mixture BW algorithm [7].

In order to derive the gain function we define the observed data y_1^T as $y_1^T = \{y_1, y_2, \ldots, y_T\}$ and the related hidden states s_1^T as $s_1^T = \{s_1, s_2, \ldots, s_T\}$.

Requiring that means and variances of the codebook are constant during the split, as stated above, the gain function is expressed as a difference of the expected log likelihood before and after the split. For each state split iteration k the expected log likelihood becomes:

$$
\begin{aligned}
&Q(\theta^{(k+1)}|\theta^{(k)}) = E[\log \mathrm{P}(y_1^T, s_1^T|\theta^{(k+1)})|y_1^T, \theta^{(k)}] \\
&= \sum_{s_1^T} \mathrm{P}(s_1^T|y_1^T, \theta^{(k)}) \log \mathrm{P}(y_1^T, s_1^T|\theta^{(k+1)}) \\
&= \sum_{s:s=s_t, s'=s_{t-1}}^{S} \sum_t \xi_t(s, s') \log a(s, s') \\
&+ \sum_{s:s=s_t}^{S} \sum_t \sum_l \gamma_t(s, l) \log b_l(s) \\
&+ \sum_{s:s=s_t}^{S} \sum_t \sum_l \gamma_t(s, l) \log N(y_t|\mu_l, \Sigma_l)
\end{aligned}
\tag{33}
$$

where

$$\begin{aligned}
\gamma_t(s,l) &= \mathrm{P}(s_t = s, \nu_l | y_1^T, \theta^{(k)}) \\
\xi_t(s,s') &= \mathrm{P}(s_t = s, s_{t-1} = s' | y_1^T, \theta^{(k)}) \\
a(s,s') &= \mathrm{P}(s_t | s_{t-1}, \theta^{(k+1)}) \\
b_l(s) &= \mathrm{P}(\nu_l | s_t, \theta^{(k+1)}) \\
N(y_t|\mu_l, \Sigma_l) &= \mathrm{P}(y_t | \nu_l, \theta^{(k+1)})
\end{aligned}$$

and $a(s,s')$ is the transition probability from the state $s = s_{t-1}$ to state $s' = s_t$, $b_l(s)$ is the weight of the codebook vector ν_l and $N(y_t|\mu_l,\Sigma_l)$ is the Gaussian mixture of the observation y_t with respect to ν_l. θ are the HMnet parameters.

Note that this equation allows us, assuming that the transition probabilities are fixed as in ML-SSS[5], to consider only the contribution to $Q(\theta^{(k+1)}|\theta^{(k)})$ that comes from the candidate split state s^c. Hence, our gain $G(s^c)$ resumes to:

$$\begin{aligned}
G(s^c) = \; & Q(\theta^{(k+1)}|\theta^{(k)})(s_1,s_2) - Q(\theta^{(k+1)}|\theta^{(k)})(s^c) \\
= \; & \textstyle\sum_{s,s'} M_1(s,s') \log a(s,s') - M_1(s^c,s^c) \log a(s^c,s^c) \\
+ \; & \textstyle\sum_l [\sum_s M_2(s,f,l) \log b_l(s) - M_2(s^c,f,l) \log b_l(s^c)] \\
+ \; & \textstyle\sum_{j(s^c,f)} [M_3(s_1,j) + M_3(s_2,j) - M_3(s^c,j)]
\end{aligned} \tag{34}$$

where

$$\begin{aligned}
M_1(s,s') &= \sum_{j(s,f)} \xi_{j(s,f)}(s,s') \\
M_2(s,f,l) &= \sum_{j(s,f)} \sum_{t:x_t = x_{j(s,f)}} \gamma_t(s,l) \\
M_3(s,j) &= \sum_{t:x_t = x_{j(s,f)}} \sum_l \gamma_t(s,l) \log N(y_t|\mu_l,\Sigma_l)
\end{aligned}$$

and j represents a phoneme, with $j \in j(s,f)$, where $j(s,f)$ represents all the generalized allophones [19] in a certain factor $f \in \{preceding, center, succeeding\}$ phonemes for state s. With $(s,s') \in \{(s_1,s_1)\}$ for the contextual domain split and $(s,s') \in \{(s_1,s_1),(s_1,s_2),(s_2,s_2)\}$ for the temporal domain split (see Fig. 7).

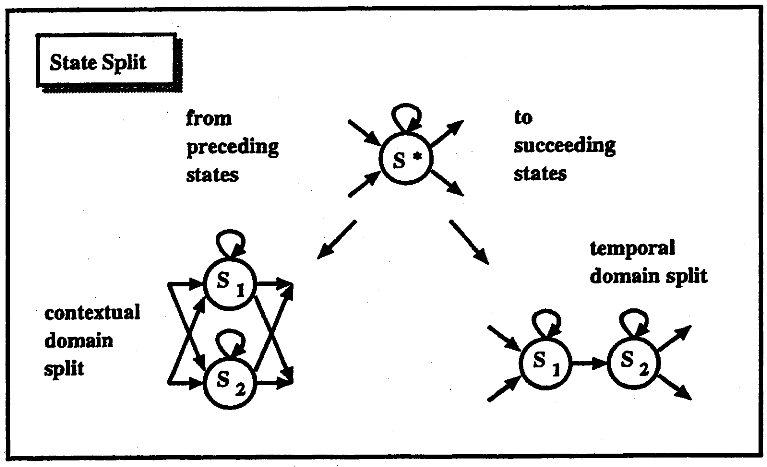

Figure 7. Example of split of state s^* into states s_1 and s_2 for contextual and temporal domains.

The first term $M_1()$, in equation 34 is independent of the chosen factor f and constant due to the constraint of fixed transition probabilities during the split. Additionally, due to the constraint of constant codebook during the state split, the last term $M_3()$ is constant too. These constraints reduce the calculation of the gain to the evaluation of the term $M_2()$.

4.2 Experimental Data

Speaker independent continuos speech recognition experiments for male and female speakers (ASJ [17] and JNAS [10] database) were carried out with the same training data that had been used to construct the acoustic models of the IPA [20] distribution. The training database consists of 20414 sentences uttered by male speakers and 20958 sentences uttered by female speakers. Separate 100 male sentences and 100 female sentences were used for testing from speakers not present in the training database. The sentences were labeled with 43 phoneme labels.

As baseline, we used monophone HMM's, with 16 mixtures per state for comparison with the FTM-SSS algorithm in the experiments that follow. All the models have around 2000 Gaussians whether it is a tied-mixture or a Gaussian mixture structure.

Table 6 describes the feature extraction. The experiments were conducted

Table 6. Feature extraction.

parameter	Value
sampling rate	16000 Hz
frame shift	10 ms
frame length	25 ms
pre-emphasis coef.	0.97
parameters	12 MFCC, 12 Δ MFCC, 1 Δ log power

under the following conditions:

- Monophone models are trained with 3 state aligned left to right per phoneme.
- FTM-SSS models are trained with a codebook of about 2000 mixands
- Embedded re-estimation, i.e. not using time information of the phoneme labels, is performed to compensate for labeling errors, for all acoustic models.

4.2.1 Speaker-independent HMnet Experiments

For the following experiments, we have been using version 2.2b of Julius, that, among other things, recognizes tied-mixture acoustic models in HTK format and takes left and right intra-word contexts, and right inter-word contexts into account. From now on whenever we mention Julius we are referring to version 2.2b. A new feature in this version is pruning the number of mixands that will be evaluated in order to speed up recognition.

For our experiments we used the ASJ and JNAS male and female training sentences described in the IPA distribution CD and the test data with 100 sentences for each gender, used for IPA acoustic models evaluation.

Acoustic models for monophones, FTM-SSS and PDT (Phonetic Decision Trees) were generated. The PDT model was trained using the HTK toolkit[14] and was then converted to a tied-mixture format by merging all the Gaussians and applying a standard flooring of 1.0e-5 to all mixand weights, properly rescaling the mixand weights to make them sum up to 1.0. All models are finally retrained with at least 3 Baum-Welch iterations to deal with differences that may arise from structure conversion.

Word recognition results for female acoustic models for FTM-SSS and PDT algorithms are summarized in Table 7 for different pruning levels to verify the effect of pruning in both algorithms.

Table 7. Word recognition correct/accurate (%) for FTM-SSS and PDT for different levels of pruning (tmix represents the number of most significant mixands that must be evaluated)

tmix	algorithm	
	FTM-SSS	PDT
32	91.00/89.92	82.48/78.68
64	90.55/89.22	83.54/79.94
128	90.93/89.73	84.18/81.90
256	90.93/89.66	85.33/83.24
512	90.93/89.66	86.82/84.35

As we can see from Table 7 the FTM-SSS algorithm produces an acoustic model that is more resistant to pruning, because it builds its structure based on a tied-mixture codebook, while in case of the PDT generated acoustic model, it is artificially generated by merging the mixtures.

From now on we will use the monophone acoustic models as baseline. Results for male and female models are summarized in Table 8.

Table 8 shows that an HMM model constructed using the FTM-SSS algorithm leads to about 2% increase in the recognition rate while only increasing the decoding time by 50%. Table 8 also shows that the HMM model constructed using the FTM-SSS algorithm achieves a higher recognition rate than using the acoustic model generated using PDT algorithm for this task. For $tmix = 32$ obtain increase of more than 10% in both genders for FTM-SSS when compared with PDT.

Table 8. Word recognition correct/accurate (%) for monophone, FTM-SSS and PDT, pruning up to the first highest 32 and 512 mixtures present in the codebook

training algorithm	tmix	gender		relative time
		male	female	
monophone	-	83.34/82.46	87.20/86.25	1.0
FTM-SSS	32	84.56/82.53	91.00/89.92	1.5
PDT	32	79.68/74.11	82.48/78.68	2.2
FTM-SSS	512	83.65/82.31	90.93/89.66	4.3
PDT	512	84.84/79.76	86.82/84.35	5.7

4.3 Summary

In this second part of the thesis we proposed a new algorithm named FTM-SSS (Fast Tied-Mixture Successive State Splitting). FTM-SSS works also for speaker-independent tasks, and thus allows to apply the benefits of TM-SSS (Tied-Mixture Successive State Splitting) without sacrificing recognition performance. It is one step forward for using context-independent, speaker independent acoustic models in real-time applications.

5. Conclusion

This thesis has to cope with a problem found in speech recognition systems that has both a theoretical and practical aspect. The essence of this problem is the representation of HMM states by single Gaussians during the HMM topology generation. Theoretically this is insufficient as a representation, and resulting errors are propagated into the system. The pure application of the theoretical solution, namely the Gaussian Mixture approach is not practically feasible at the state of HMM topology design.

It is known that in the Gaussian Mixture case a state split would have a strong impact in the Gaussians of the split state. However in the Tied-Mixture case that effect is exactly the opposite. A state split in the TM case produces just a slight change in the Gaussian codebook. It was observed that most of the computation cost is due to updating the codebook while performing the state split. Both Tied-Mixture and Gaussian Mixture pdf representations are equivalent and richer than a single Gaussian representation. This lead to the hypothesis that it is possible to use a Tied-Mixture representation keeping the codebook unmutable during a state split.

In this thesis we examined this hypothesis and proposed an automatic algorithm named FTM-SSS (Fast Tied Mixture Successive State Split) algorithm for HMM topology construction that solves the problems mentioned earlier. We have been able to draw the following conclusions about this approach:

- FTM-SSS solves a theoretical problem of previous automatic acoustic model algorithms, namely the representation of the pdf of a state by a single Gaussian, during the construction of the HMM topology. Moreover it is practical in the sense that it does so without being as computationally expensive as the approach of representing states by Gaussian Mixtures.

- FTM-SSS reduces the error rate by about 31% compared to the original ML-SSS algorithm and by about 10% compared to the Phonetic Decision Trees (PDT) based algorithms. This comparison was done with models of complexity similar to a baseline monophone HMM set and fast pruning applied to both FTM-SSS and PDT (in order to be competitive in terms of speed).

- FTM-SSS makes distinctions in the early stages of the HMM topology construction that are more consistent with phonetic knowledge. These distinctions were observed in FTM-SSS but not in previous algorithms such as Maximum Likelihood Successive State Split (ML-SSS).

Both quantitative and qualitative results show that a better HMM topology was achieved by using FTM-SSS. This is important to the performance a speech recognition system, since any error in the HMM topology is propagated and it is difficult if not impossible to compensate for this at later stages.

The results of the second part of the thesis also show that:

- the algorithm is fast enough to be applied to large tasks such as the JNAS and IPA databases, which contain together more than 60 hours of training data.
- training data speech should be as equally distributed in the acoustic space as possible, such that it does not affect the decisions taken by the HMM state clustering algorithm when producing the HMM topology.

The contribution of this thesis is then to show that it is possible to grow a cluster tree with a very rich representation of the probability density function (pdf) of a candidate state split (with a Tied-Mixture representation) and at the same time be computationally feasible, given the assumption that the codebook change during the split can be neglected.

It has been demonstrated that the proposed approach is effective for large context-independent, speaker independent databases such as the JNAS and the IPA databases.

6. Future Work

Suggestion of directions that can be taken follow the work presented in this thesis are:

- acoustic duration models. Should them also be included in the cluster tree? Should we also consider its dependency?
- dimensioning of the HMM topology tree. In particular to better understand when we should to stop splitting a state, in order to avoid undertraining problems later. It is specially important when looking for highest possible accuracy that can be achieved. Should it be in terms of occurrence counts, mass occurrence, or what?
- design a recognition engine that chooses the degree of dependency of the acoustic models based on its duration (at run time). For example, investigating whether we should use full/left/right context dependent models (a cluster tree leave) or context independent models, based on the state occurrence during training/recognition.

Acknowledgements

A lot of research has been placed in ASR systems and the result of this work is only possible due to the contribution of a large number of researches all around the world in the many areas involved in ASR systems, such as: data recording, data transcription (phonetics), feature extraction (echo cancellation, noise reduction, channel normalization), acoustic modeling (clustering, training algorithms), language modeling, confidence measures, rejection, search algorithms, pruning strategies. This is my acknowledgement to the work of those pioneers, now anonymous, that made our research now possible.

Now, more directly involved in this thesis, I will start by specially thanking my academic advisor Professor Kiyohiro Shikano for his friendship, advices and many academic and research opportunities and to Dr. Satoshi Nakamura (former associate professor of NAIST and now head of the first research lab. , ATR spoken language communication research laboratories) and Shiro Ise for the very same reasons. Also, it is important to mention friends like Norimi Youdou (for the language models) and Harald Singer (for the introduction to ML-SS and proof reading of my papers).

I would like also to thanks to Professor Yuji Matsumoto for his valuable comments as a thesis commitee and his group specially for helping with language modeling issues and their friendship.

Also important was the input from Professor Nick Campbell and Professor Hiroshi Saruwatari for their valuable comments as a thesis commitee.

I am indebted also to the members of the IPA group, namely Professor T. Kawahara, Prof. K. Takeda and Dr. A. Lee for their help with the database and with the recognition engine Julius. We also acknowledge the financial support of the IPA.

I would like to thanks my wife Adelia Marli Contri Girardi, my kids Marcelo and Alice Girardi, parents and relatives for here support and comprehension when I needed to stay away or spend nights working to make this thesis.

I also would like to say I will miss all my friends from the Shikano Lab. and those from the dormitory of NAIST (a nice environment that really helps towards research). Just to name some friends (now Dr.'s): Eli Yamamoto, Tetsuya Takiguchi, Takeshi Yamada, Marc Hogenhout, Tang Yuan Sheng, Mai Komatsu and many others.

Also I would like to thanks to Ikoma Network for the lessons of Japanese and above all their friendship. A consistent and warm support that we find in Japan, but that is difficult to find anywhere else.

List of Publications

Journal Publications

- Alexandre Girardi, Harald Singer, Kiyohiro Shikano, and Satoshi Nakamura, “Maximum Likelihood Successive State Splitting Algorithm for Tied-Mixture HMnet”, IEICE transactions on information and systems, Vol.83 No.10, pp.1890-1897, 2000-10.
- Alexandre Girardi, Kiyohiro Shikano and Satoshi Nakamura, “Fast Tied-Mixture Successive State Splitting Algorithm”, paper submited to the Proc. Speech Communication.

International Conferences and Workshops

- Alexandre Girardi, Kiyohiro Shikano, Satoshi Nakamura, ”Creating speaker independent HMM models for restricted database using STRAIGHT-TEMPO morphing”, Proc. Internat. Conf. on Spoken Language Process. (ICSLP), December 1998.
- Alexander Girardi, Harald Singer, Kiyohiro Shikano, Satoshi Nakamura, ”Maximum Likelihood Successive State Splitting Algorithm for Tied-Mixture HMNET”, Eurospeech, pages 119-122, September 1997.

Presentations at Meetings in Japan

- ジラルジ、シンガー、鹿野、中村、” Fast Tied-Mixture Successive State Splitting Algorithm” 、日本音響学会研究発表講演論文集、2-6-12、77-78、March 1998.
- A.Girardi, H.Singer, K.Shikano, S.Nakamura,”Maximum likelihood successive state splitting algorithm for tied mixutre HMnet”, 日本音響学会研究発表講演論文集、3-6-1、89-90、March 1997.

References

[1] A. Girardi, H. Singer, K. Shikano, S. Nakamura, "Maximum Likelihood Successive State Splitting Algorithm for Tied-Mixture HMNET," *Proc. EUROSPEECH*, volume 1, pp.119-122, September 1997.

[2] A. Lee, T. Kawahara, S. Doshita "An Efficient Two-Pass Search Algorithm Using Word Trellis Index," *Proc. ICSLP98*, pp., December 1998.

[3] F. Jelinek, "Self-Organized Language Modeling for Speech Recognition," *In Readings in Speech Recognition [Waibel and Lee, 1990]*, pp.450-506. Morgan Kaufmann.

[4] H. Kuwabara, Y. Sagisaka, K. Takeda, and M. Abe, "Construction of ATR Japanese speech database as a research tool," Technical Report TR-I-0086, ATR, 1989. (in Japanese).

[5] H. Singer and M. Ostendorf, "Maximum likelihood successive state splitting," *Proc. ICASSP*, pp.601-604, Atlanta, 1996.

[6] I. H. Witten, T. C. Bell, "The zero frequency problem: Estimating the probabilities of novel events in adaptive text compression," *IEEE Trans. Information Theory*, vol. 37, No. 4, pp. 1085-1094, 1991.

[7] J. R. Bellegarda and D. Nahamoo, "Tied mixture continuous parameter modeling for speech recognition," *IEEE Trans. Signal Process.*, vol.38, no.12, pp.2033-2045, 1990.

[8] J. Takami, S. Sagayama, "A successive State spliting algorithm for efficient allophone modeling," *Proc. ICASSP*, vol.1, pp.573-576, 1992.

[9] K. F.Lee et al., "Allophone Clustering for Continuous Speech Recognition," *ICASSP*, vol.2, pp.749-752, 1990.

[10] K. Itou, M. Yamamoto, K. Takeda, T. Takezawa, T. Matsuoka, T. Kobayashi, K. Shikano and S. Itahashi, "JNAS: Japanese Speech Corpus

for Large Vocabulary Continuous Speech Recognition Research," *The Journal of Acoustic Society of Japan (E)*, volume 20, number 3, pp.199-206, May 1999.

[11] M. Ostendorf and H. Singer, "HMM Topology Design using Maximum Likelihood Successive State Splitting," *CSL*, volume 11, number 1, pp.17-41, 1997.

[12] N. Yodo, K. Shikano, S. Nakamura, "Compression Algorithm of Trigram Language Models based on Maximum Likelihood Estimation," *Proc. ICSLP98*, pp., December 1998.

[13] P.C. Chang, B.H. Juang, "Discriminative Training of Dynamic Programming Based Speech Recognizers," *IEEE Trans. Speech and Audio Process.*, pp.135-143, volume 1, number 2, April 1993.

[14] P.C. Woodland, C.J. Leggetter, J.J. Odell, V. Valtchev and S.J. Young, "The 1994 HTK large vocabulary speech recognition system," *Proc. ICASSP*, pp.73-76, Detroit, May 1995.

[15] P. Chou, "Optimal partitioning for classification and regression trees," *IEEE Trans. PAMI*, vol.13, no.4, pp.340-354, April 1991.

[16] R. Schwartz and Y. Chow and O. Kimball and S. Roucos and M. Krasner and J. Makhoul, "Context-Dependent Modeling for Acoustic-Phonetic Recognition of Continuous Speech," *Proc. ICASSP*, volume 3, pp.1205-1208, Florida, March 1985.

[17] S. Hayamizu, S. Itahashi, T. Kobayashi, and T. Kakezawa, "Design and Creation of speech and text corpora of dialogue," *IEICE Trans. Inf. Syst. E76-D(1)*, pp.17-22, 1993.

[18] S. J. Young, J. J. Odell and P. C. Woodland, "Tree-based state tying for high accuracy acoustic modeling," *Proc. ARPA Workshop on Human Language Technology*, pp.307-312, 1994.

[19] S. Sagayama, "Phoneme environment clustering for speech recognition," *Proc. ICASSP*, volume 1, pp.397-400, Glasgow, May 1989.

[20] T. Kawahara, T. Kobayashi, K. Takeda, N. Minematsu, K. Itou, M. Yamamoto, A. Yamada, T. Utsuro and K. Shikano, "Sharable Software Repository for Japanese Large Vocabulary Continuous Speech Recognition," *Proc. ICSLP*, pp.3257-3260, Dec 1998.

[21] T. Kawahara, A. Lee, T. Kobayashi, K. Takeda, N. Minematsu, K. Itou, M. Yamamoto, A. Yamada, T. Utsuro and K. Shikano, "Common Platform of Japanese Large Vocabulary Continuous Speech Recognizer Assessment," *Proc. First International Workshop on East-Asian Language Resource and Evaluation (EALREW)*, pp.117-122, May 1998.

[22] Y. Linde, A. Buzo and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp.84-95, Jan. 1980.