TR-IT-0117

Modification of SSS for Speaker-Independent HM-Net Design

オステンドーフ マリ
Mari Ostendorf

シンガー ハラルド
Harald Singer

1995.06

This report describes a new approach to successive state splitting that uses the maximum likelihood criterion directly in choosing specific splits, rather than taking a two step process of choosing the state with the most variability and then choosing the best split for that state. By testing splits directly, the algorithm is better suited to speaker-independent HM-Net training, which we believe will improve recognition performance on spontaneous speech. Preliminary experiments are described for assessing recognition performance gains and training cost reduction associated with the algorithm.

Modification of SSS for Speaker-Independent HM-Net Design

M. Ostendorf H. Singer

June 16, 1995

Abstract

This report describes a new approach to successive state splitting that uses the maximum likelihood criterion directly in choosing specific splits, rather than first choosing the state with the most variability and then choosing the best split for that state. By testing splits directly, the algorithm is better suited to speaker-independent HM-Net training, which we believe will improve recognition performance on spontaneous speech. Preliminary experiments are described for assessing recognition performance gains and training cost reduction associated with the algorithm.

1 Introduction

Successive state splitting (SSS) is a powerful technique for HMM design that provides a mechanism for automatically learning the most appropriate HMM topology [1]. The basic idea behind SSS is that a network of HMM states (referred to as an HM-Net) can be increased in size by choosing to split the state with the most variability, and then picking the best splitting domain (either temporal or contextual) for that state. The iterative application of this splitting results in an HM-Net that efficiently represents context and temporal variability of specified subword units (e.g. phones or moras). SSS has been used successfully by ATR and shown to outperform other HMM design techniques in several studies [2, 3, 4].

A disadvantage of SSS, as it is currently implemented, is that it only works well for training topologies on speaker-dependent data. In speaker-independent training, the state with the most variability, which would be chosen by SSS, is likely to reflect speaker variability rather than coarticulation or temporal effects. In order to use SSS for building a speaker-independent model, one first designs a speaker-dependent topology and then retrains this model on speaker-independent data, as in [5]. Although this solution works well in experiments with carefully read speech by professional speakers [6], it is likely that it will be a limiting factor in recognition under less controlled conditions. For spontaneous speech, in particular, the optimal topology for one speaker may not be a good choice for another speaker with a different accent, speaking rate or style. Therefore, the goal of this work is to reformulate the SSS approach to allow for speaker-independent HM-Net training.

A technique similar to successive state splitting used in many HMM systems is divisive distribution clustering, sometimes referred to as decision tree context modeling. Divisive clustering of

distributions using decision tree design techniques was first proposed in [7, 8] for whole phone model clustering, and later extended to state-level clustering for tied mixtures [9] and single Gaussians [10, 11, 12]. All approaches used either Viterbi or forward-backward alignment to associate training observations with states given some pre-specified HMM topology, followed by decision tree growing of contextual splits under some objective function related to maximum likelihood of the training data. Unlike SSS, decision tree context modeling has been successfully used in speaker-independent HMM training.

An important difference between the decision tree and SSS approaches is that the choice of which distribution to split in decision tree modeling is based on a specific contextual split rather than on the generic measure of state distribution variance used in SSS. This difference suggests a solution to the problem of speaker-independent training in SSS: simply choose the state to split based on the most likely splitting domain for that state, rather than choosing the splitting domain after choosing the state. At a high level, this difference is simply a reordering of the main steps in the SSS algorithm [1], as summarized in Section 2. However, the tests for splits change form, since the most likely case is now found directly from the data rather than from pre-specified mixture distributions. In addition, since these tests are called frequently, it is important that they be implemented efficiently, which impacts the splitting objective function as discussed in Section 3. Efficient algorithms for generating comparable "scores" for a split in the contextual and temporal domains are described in Sections 4 and 5, respectively. In our reformulated version of SSS, HM-net design is very similar to the distribution clustering algorithms used in other recognition systems, except that SSS allows unrestricted distribution sharing as opposed to sharing within specific phone-dependent regions [10, 11]. Together the SSS changes result in an algorithm that gives slightly better performance on speaker-dependent data in preliminary experiments, as shown in Section 6. This is the most difficult test for the new algorithm since it is on data that SSS was tuned on, and we therefore expect significant gains for speaker-independent training. General implications of the results and recommendations for future research are outlined in Section 7.

2 Reformulation of SSS

The original version of the SSS algorithm, as described in [1], is an iterative algorithm that progressively grows the HM-Net. First, the state is selected to be split according to which has the largest divergence between its two mixtures, and then splits in the contextual and temporal domains are tested. Figure 1 illustrates the topology changes, showing how a selected state would be split into two states in the contextual domain and the temporal domain. The algorithm proceeds as follows.

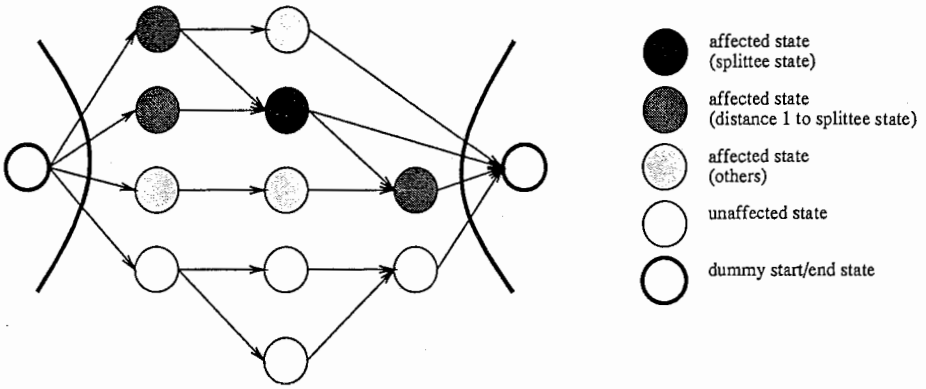
Original SSS Algorithm

- **Initialization:**
  1. Run Baum-Welch (2 Gaussian mixture)
- **Iterate:**
  1. Find best state to split
  2. Find the best domain and factor to split the state
     (a contextual split test and two temporal split tests for the chosen state)
  3. Determine affected states and get new initial mixture distribution parameters for each
     (K context clustering calls for K affected states)
  4. Run Baum-Welch on affected states
     (2 Gaussian mixture distribution)

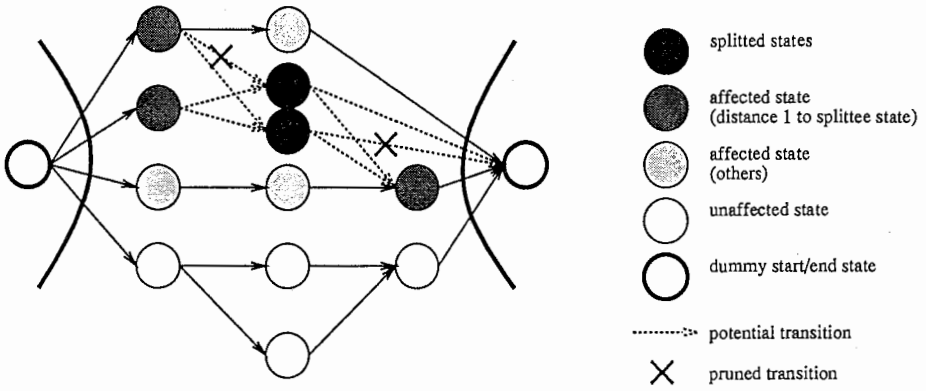
The term "affected states" includes all states that might have parameters change because of this split, given that phone boundaries are fixed. (Phone boundaries might be known given by hand-marked labels or by a Viterbi alignment.) More specifically, the affected states are all states in the subnet of states connected to the current state to be split after the network has been cut at the dummy beginning and end nodes, as illustrated in Figure 1. By this definition, nearly all states are affected by every split, until more specific phone-dependent subnets start to evolve. Note that, for a contextual split, some new paths between states may be impossible because of a mismatch in contextual dependence, and these paths are pruned, as illustrated in Figure 1 (b) with the "X" to indicate path pruning.

The key problem with the original SSS algorithm is that the best state to split is chosen before the actual split is chosen. The output distribution for each state is a mixture of two Gaussians, and the "best" state is that which has the largest separation between the two mixture components. However, these mixture components are generic and do not necessarily correspond to an allowable split, and so the best state to split by this criterion may not in fact be the best choice given the constraints on allowable splits. In speaker-independent training, for example, the mixture components might be well separated because of speaker differences, but this variability cannot be modeled by adding a new state if the allowable splits are in terms of phonetic context or temporal structure. By choosing the state to split separately from the split itself, we also lose the guarantee of non-decreasing likelihood, although in practice it would be rare to see a decrease in likelihood.

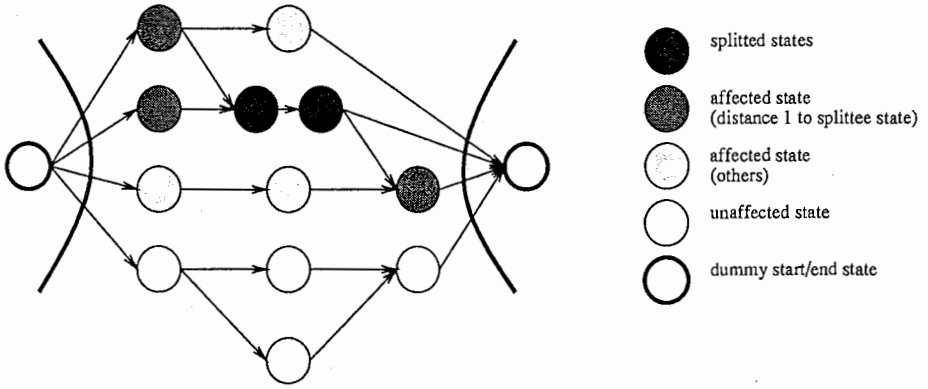
One attempt to address this problem involved adding a splitting dimension in the speaker domain to capture variability such as speaker and gender differences. Contextual and temporal splits are tied across speakers so that repeated copies of the HM-Net topology are estimated. Two variations of this approach were proposed, referred to as 3D-SSS and SP-SSS, differing in when splits in the speaker dimension are allowed [6]. If splits are allowed in the speaker-domain at all



a) Original network.



b) Contextual split.



c) Temporal split.

Figure 1: Illustration of topology changes from the original network (a) to a contextual split (b) or to a temporal split (c). The states which are "affected" by the split, as defined in the SSS implementation, are indicated by shading.

points in SSS topology growing, then the algorithm becomes very expensive. Unfortunately, this approach did not lead to performance improvements, perhaps because of the large amount of state redundancy.

Here, we propose a different solution to the speaker-independent HM-Net topology training problem, which is simply to reorder the steps of finding the best split for a state and picking the best state to split. The new algorithm, which we will refer to here as SI-SSS to distinguish it from SSS, proceeds as follows:

Restructured SI-SSS Algorithm

- **Initialization:**
  1. Run Baum-Welch (single Gaussian)
  2. Get split info for all states
- **Iterate:**
  1. Find best state to split and split
  2. Determine affected states and run Baum-Welch on affected states (single Gaussian distribution)
  3. Find the best domain and factor for future split of this and, optionally, affected states ($1-K$ contextual split tests and one temporal split test for K affected states, depending on update options)

Step 3 in the SSS algorithm as it is implemented, that of finding the initial mixture parameters for the new states, is very similar to finding the best split of phonetic context. Initialization involves a VQ design procedure operating on sample means for different contexts, which is similar to the partitioning algorithm proposed in Section 4. By modifying this step slightly and saving the gain from the best split for later testing, we can effectively eliminate Step 2 and at the same time have a more accurate search. A further advantage of the restructured algorithm is that Baum-Welch training is on single Gaussian distributions, which runs much faster than training on Gaussian mixtures.

Although the Baum-Welch training step will be much faster in the SI-SSS algorithm than in the SSS algorithm, we expect the computational cost of the two approaches will be the same order of magnitude. If all affected states are updated, then the number of contextual split tests in both algorithms is essentially the same ($K+1$ for SSS vs. K for SI-SSS, assuming K affected states). The SI-SSS contextual split test will be somewhat more expensive than the mixture initialization step in SSS, since a maximum likelihood clustering criterion is used rather than minimum distance, but the difference should be less than a factor of two and this step is a relatively small part of the overall SSS computation. The SI-SSS temporal split will also be more costly, as described further in Section 5, requiring Baum-Welch training of the two distributions resulting from the state to be

split rather than the single forward algorithm pass used in the SSS temporal split. In addition, there will be potentially K SI-SSS temporal split tests rather than two SSS temporal split tests. However, the cost of the temporal splits is a very small part of the overall algorithm, since the amount of data to process with the forward algorithm (or forward-backward in SI-SSS) is small, only that which maps to a single state, and since temporal splits are eventually disallowed by maximum state length constraints. Therefore, the additional cost of the SI-SSS temporal splits should not be problematic. In fact, in Section 6, SI-SSS is shown to be faster than SSS for speaker-dependent training on 2620 isolated words.

Even if computation time of SI-SSS is similar to or slightly faster than SSS, it may still be of interest to reduce the HM-Net design cost. For both SSS and SI-SSS, the cost could be reduced by only re-initializing parameters (for SSS) or only re-evaluating the best split (for SI-SSS) for a subset of the affected states. For example, one might designate three levels of affected states: (1) the two new states generated by the splits, (2) all states immediately adjacent to these two states, and (3) all other affected states. In SSS, it may not be necessary to re-initialize the mixture parameters for the states in set (3). In SI-SSS, it may be reasonable to re-estimate fewer parameters of the splits for the group (3) states, assuming that the split will change minimally. The possible SI-SSS options, in order of increasing computation, include:

- Keep the same split and update only the means and variances of the split to find the new gain.
- Keep the split domain (e.g. left context), but re-evaluate the best partitioning of contexts in that domain, initializing the partitioning algorithm with the previous context for faster convergence.
- Re-evaluate the state entirely.

Note that the two new states must be evaluated for all possible splits, and that it is only the other affected states that one might not want to re-evaluate entirely. If the affected states are completely re-evaluated, then the restructured SI-SSS algorithm is guaranteed to give larger increases in the likelihood of the training data at any single step than the SSS algorithm starting from the same model. In practice, however, good results may be achieved without complete re-evaluation and with a significantly lower cost.

3 State Splitting and Constrained ML Estimation

There are three possible general objective functions related to maximum likelihood that one might use in split design. The simplest approach, as followed in several other studies [9, 10, 12], is to align the training data to states in some pre-specified topology and then cluster the resulting state distributions to maximize the joint likelihood of the data and the given state sequence. This approach is essentially Viterbi-style training, which has been used successfully in many applications but is known to be sub-optimal relative to Baum-Welch training.

A second option is to maximize likelihood of the observations directly, but computing likelihood requires running the forward algorithm between fixed points, such as utterance boundaries. Thus a direct likelihood criterion is only practical if intermediate fixed points are used, such as the phone boundaries in SSS. The likelihood of a split is computed using the forward algorithm over all data samples and states within the fixed boundary times that the splittee state falls into. The resulting split goodness measure has the advantage that it is a true likelihood, and that it incorporates the effect of a split on neighboring states. A disadvantage is that phone boundaries are required, and typically SSS has been used with hand-marked boundaries. Viterbi-aligned phone boundaries are likely to work almost as well, but this has not been verified in experiment. However, the real disadvantage of the likelihood splitting criterion is that it is simply too expensive to be used in SI-SSS, where it would be called much more often.

Our solution to this problem is to maximize the expected log likelihood, rather than likelihood, taking advantage of the same Expectation-Maximization (EM) algorithm concepts [13] that are behind the standard Baum-Welch training algorithm. The fundamental result behind the EM algorithm is that increasing the expected log likelihood of the observed data y_1^T and the hidden or unobserved components s_1^T (e.g. HMM states)

$$Q(\theta|\theta^{(p)}) = E_{\theta^{(p)}}[\log \mathrm{p}(y_1^T, s_1^T|y_1^T, \theta)] \tag{1}$$

at worst gives no change to the likelihood of the observed data $L(\theta) = \log \mathrm{p}(y_1^T|\theta)$

$$Q(\theta|\theta^{(p)}) \geq Q(\theta^{(p)}|\theta^{(p)}) \quad \Longrightarrow \quad L(\theta) \geq L(\theta^{(p)}). \tag{2}$$

Because of the conditional independence assumptions in the HMM, the expected log likelihood can be written as

$$\begin{aligned} Q(\theta|\theta^{(p)}) &= E[\log \mathrm{p}(y_1^T, s_1^T|\theta)|y_1^T, \theta^{(p)}] = \sum_{s_1^T} \mathrm{p}(s_1^T|y_1^T, \theta^{(p)}) \log \mathrm{p}(y_1^T, s_1^T|\theta) \\ &= \sum_{s_1^T} \mathrm{p}(s_1^T|y_1^T, \theta^{(p)}) \sum_t \left[\log \mathrm{p}(y_t|s_t, \theta_{A(s)}) + \log \mathrm{p}(s_t|s_{t-1}, \theta_{B(s)})\right] \\ &= \sum_s \sum_t \gamma_t(s) \log \mathrm{p}(y_t|s, \theta_{A(s)}) + \sum_{s,s'} \sum_t \xi_t(s,s') \log \mathrm{p}(s_t = s|s_{t-1} = s', \theta_{B(s)}), \end{aligned} \tag{3}$$

where

$$\gamma_t(s) = \mathrm{p}(s_t = s|y_1^T, \theta^{(p)}) \tag{4}$$

$$\xi_t(s,s') = \mathrm{p}(s_t = s, s_{t-1} = s'|y_1^T, \theta^{(p)}). \tag{5}$$

The form of equation 3 allows for separate maximization of the distribution parameters $\theta_{A(s)}$ and transition probabilities $\theta_{B(s)}$ for each state s. This allows us to estimate the parameters for a single state (or two states after splitting) so that the expected likelihood is increased, thereby guaranteeing that there is no decrease in likelihood of the observed data.

More specifically, in designing a split for state s^*, we maximize $Q(\theta|\theta^{(p)})$ subject to the constraint the $\gamma_t(s)$ and $\xi_t(s,s')$ are fixed for all $s \neq s^*$. If the initial split is chosen appropriately, as will be

discussed in Sections 4 and 5, the constrained function $Q(\theta|\theta^{(p)})$ is guaranteed to be non-decreasing since the terms depending on $s \neq s^*$ do not change and the likelihood due to other terms cannot decrease. Therefore $L(\theta)$ is guaranteed to be non-decreasing. The gain in expected log likelihood for split S from s^* to s_0 and s_1

$$\begin{aligned} G(S) &= \sum_{s=s_0,s_1} \sum_t \gamma_t(s) \log \mathrm{p}(y_t|s, \theta_{A(s)}) - \sum_t \gamma_t(s^*) \log \mathrm{p}(y_t|s^*, \theta_{A(s^*)}) \\ &\quad + \sum_{s=s_0,s_1} \sum_{s'=s_0,s_1} \sum_t \xi_t(s,s') \log a_{ss'} - \sum_t \xi_t(s^*,s^*) \log a_{s^*s^*} \qquad (6) \\ &= \sum_{s_0,s_1} N_1(s) \log \mathrm{p}(y_t|s, \theta_{A(s)}) - N_1(s^*) \log \mathrm{p}(y_t|s^*, \theta_{A(s^*)}) \\ &\quad + \sum_{s=s_0,s_1} \sum_{s'=s_0,s_1} N_2(s,s') \log a_{ss'} - N_2(s^*,s^*) \log a_{s^*s^*} \qquad (7) \end{aligned}$$

where $a_{ss'} = \mathrm{p}(s_t = s|s_{t-1} = s', \theta_{B(s)})$ and

$$N_1(s) = \sum_t \gamma_t(s) \qquad (8)$$

$$N_2(s,s') = \sum_t \xi_t(s,s'). \qquad (9)$$

The gain due to the observation distribution parameters alone can be expressed as

$$\begin{aligned} G_C(S) &= \sum_{s=s_1,s_2} N_1(s) \log \mathrm{p}(y_t|s, \theta_{A(s)}) - N_1(s^*) \log \mathrm{p}(y_t|s^*, \theta_{A(s)}) \\ &= 0.5 \left[\sum_{m=1}^{M} \left[N_1(s^*) \log \sigma_m^2(s^*) - N_1(s_0) \log \sigma_m^2(s_0) - N_1(s_1) \log \sigma_m^2(s_1) \right] \right. \\ &\quad \left. - N_1(s^*) \log \left[1 + \frac{N_1(s_0) N_1(s_1)}{N_1(s^*)^2} \sum_{m=1}^{M} \frac{(\mu_m(s_0) - \mu_m(s_1))^2}{\sigma_m^2(s^*)} \right] \right] \qquad (10) \end{aligned}$$

assuming that the distribution is described by a diagonal covariance and the subscript m indicates an element of the M-dimensional vector. This particular form of the gain uses the combined mean and covariance likelihood criterion described in [10] (based on a result from [14], Chapter 10.3). For contextual splits, where state transition probabilities are held constant, Equation 10 gives the total expected gain. For temporal splits, on the other hand, the total expected gain is

$$\begin{aligned} G(S) &= G_C(S) - N_2(s^*,s^*) \log a_{s^*s^*} + N_2(s_0,s_0) \log a_{s_0 s_0} \\ &\quad + N_2(s_0,s_1) \log a_{s_1 s_0} + N_2(s_1,s_1) \log a_{s_1 s_1} \\ &= G_C(S) - N_2(s^*,s^*) \log a_{s^*s^*} + N_2(s_0,s_0) \log a_{s_0 s_0} \\ &\quad + (N_1(s_0) - N_2(s_0,s_0)) \log(1 - a_{s_0 s_0}) + N_2(s_1,s_1) \log a_{s_1 s_1} \qquad (11) \end{aligned}$$

Equations 10 and 11 give a criterion by which we can compare different candidate splits within and across domains and across states, and choose the split which most increases the expected likelihood of the entire training set. Note that equations 10 and 11 do not give the increase in

likelihood *per se*, but rather the increase in expected likelihood, and so maximizing $G(S)$ over S only guarantees that likelihood is non-decreasing, not necessarily that we have chosen the split that maximally increases likelihood.

Since Equations 10 and 11 measure increase in the expected joint likelihood of the observations and states, they take a different form than the test used in SSS for choosing a splitting domain, which is based on observation likelihood. In addition, Equations 10 and 11 take a different form from the criterion used in SSS for determining the best node to split (Equation 1 in [1]), but in this case the SI-SSS criterion is preferable. The SSS criterion is a measure of the distance between the two generic mixture components and not the gain in likelihood relative to having a single state, and it cannot be related to an increase in the likelihood of the training data in any way.

A cost of using the state likelihoods $\gamma_t(s)$ and $\xi_t(s, s')$ in split design is an increase in memory requirements. To reduce memory requirements, we take advantage of a technique used in SSS, which is to use phone boundaries (hand-marked or Viterbi aligned) to restrict the set of states that have non-zero probabilities at each time, i.e. to reduce the size of $\{\gamma_t(s)\}$.

4 Efficient Search of Contextual Splits

Since SSS is basically a divisive clustering algorithm, it can benefit from advances in addressing an analogous problem: decision tree design [15]. In decision tree design, the problem is to design a function $\hat{Y} = f(\mathbf{X})$ to predict Y from $\mathbf{X}$. If Y takes on values $y \in R^M$ then the function is usually called a regression tree, and if $y \in \{1, \ldots, M\}$ it is called a classification tree. Rather than predicting Y directly, the decision tree function f can also be used instead to estimate a probability distribution $\hat{p}(y|\mathbf{X}) = p(y|f(\mathbf{X}))$, as in the tree language model used in speech recognition [16]. The distribution estimate interpretation corresponds to the use of divisive distribution clustering in speech recognition, e.g. [10, 11], and so decision tree design methodology applies here.

In decision tree design, or divisive clustering in general, the typical approach is a greedy growing algorithm, that successively grows the tree taking the split which most improves the objective function at each step. This algorithm requires testing all possible current leaves of the tree, all possible variables X (an element of $\mathbf{X}$), and all possible ways to split on variable X. Since choosing the best split for variable X is the most frequently called routine, it is important that it be relatively fast. For the case where discrete variable X has J values, there are roughly 2^{J-1} possible binary splits to test, which is prohibitively expensive. Breiman *et al.* [15] give a fast solution for the case where $M = 2$. Later, in [17], Chou provides an algorithm for fast split design for the more general case where $J \geq 2$ and $M \geq 2$. Although Chou's algorithm is only locally optimal for many tree design objective functions, it is linear in M and J and therefore much more efficient than the previously proposed CART algorithm [15] that is exponential in one or the other of those parameters when $M > 2$. In the HM-Net design problem, using phone models for example, the $\mathbf{X}$ is comprised of categorical variables that are the possible splitting domains (e.g. temporal, or left, right or center phonetic context). For any one of the context domains, the values that X takes on are the phone labels ($N = 26$ phones in Japanese). Thus, the HM-Net problem of state splitting is analogous to decision tree categorical question design and can benefit from an algorithm for efficient

search of possible splits.

We begin this section by reviewing the Chou partitioning algorithm [17], and then show how the algorithm is applied for the maximum Gaussian log likelihood objective function. We will depart from standard decision tree terminology (and notation), using the term "state" rather than "node" and "HM-Net" rather than "tree", in order to make the application to HMM design clear. One difference from standard decision tree design, is that observations may not be assigned to a single node or state, but rather there is a probability distribution describing the likelihood of observations being in different states. To simplify the initial discussion, we will assume that observations are associated with a unique state, which can be obtained with Viterbi alignment. Then, we will show how the result can be extended for use in Baum-Welch-style training. In the appendix, we show that the mixture initialization technique used in SSS is very similar to the Chou algorithm (using minimum error rather than maximum likelihood) and so there is little additional cost to implementing the maximum likelihood search into SSS.

4.1 General Categorical Split Design Algorithm

In this section we summarize the partitioning algorithm of Chou [17] for splitting a state s using variable X. Say that the values x_j that would lead to state s form the set A_s. We begin by defining $\mathcal{L}(y, \hat{y})$ as a loss function that is to be minimized in HM-net (or decision tree) design. The variable $\hat{y}$ is a representation of y, which may take on values in the same space as Y (as in quantization, regression or direct classification) or it can be a probability distribution that represents Y (as in the tree language model and distribution clustering examples above).

The "impurity" of a state s in the HM-net is the minimum possible expected loss in a state given by

$$i(s) = E[\mathcal{L}(Y, \theta(s))|s], \tag{12}$$

where $E[f(Y)|s]$ is a conditional expectation given $S = s$ and $\theta(s)$ is the "centroid" of s

$$\theta(s) = \underset{\hat{y}}{\operatorname{argmin}}\, E[\mathcal{L}(Y, \hat{y})|s]. \tag{13}$$

The divergence $d(s, \hat{y})$ is the difference in expected loss from using $\hat{y}$ instead of centroid $\theta(s)$ as the representation for state s:

$$d(s, \hat{y}) = E[\mathcal{L}(Y, \hat{y})|s] - i(s). \tag{14}$$

In designing a split for state s, we start with $i(s)$ fixed and

$$i_J(s) = \sum_j \mathrm{P}(x_j|s) i(x_j)$$

as the minimum possible impurity (also fixed) which is achieved with an J-ary split, where x_j are the possible values that the contextual factor X can take on. The impurity of a binary split into s_0 and s_1 is

$$i_2(s) = \sum_{k=0,1} \mathrm{P}(s_k|s) i(s_k).$$

Let

$$i(s) - i_J(s) = [i(s) - i_2(s)] + [i_2(s) - i_J(s)] = \Delta_1 + \Delta_2.$$

Since $i(s)$ and $i_J(s)$ are fixed, then their difference is fixed, and maximizing Δ_1, as is traditional in split design with greedy growing, is equivalent to minimizing Δ_2. Chou [17] shows that

$$\Delta_2 = \sum_j \mathrm{p}(x_j|s) d(x_j, \theta(\alpha(x_j))),$$

which means that minimizing Δ_2 can be interpreted as a quantizer design problem, where the goal is to design the "encoder" $\alpha(x_j)$ and the "decoder" or centroids $\theta(s_k)$ to minimize the expected divergence. (Note that the encoder can be described in terms of the partition $A_k = \{x_j : \alpha(x_j) = s_k\}$ for $k = 0, 1$.) A locally optimal solution to this problem can be found using an iterative algorithm analogous to the K-means algorithm, or the Linde-Buzo-Gray algorithm for vector quantization [18]: iteratively re-estimate α and θ until convergence (or the relative change in average loss is smaller than some threshold). More explicitly, the two steps are:

1. For each x_j, find new encoder

$$\alpha(x_j)^{(p+1)} = \operatorname*{argmin}_i d(x_j, \theta(s_i)^{(p)})$$

   which gives for $k = 0, 1$

$$A_k^{(p+1)} = \{x_j : \alpha(x_j)^{(p+1)} = k\}$$

2. For $k = 0, 1$, find new decoder

$$\theta(s_k)^{(p+1)} = \operatorname*{argmin}_\theta E[\mathcal{L}(Y, \theta)|s_k] = \operatorname*{argmin}_\theta \sum_{x_j \in A_k^{(p+1)}} \mathrm{p}(x_j|s_k) d(x_j, \theta)$$

For the special case of categorical predictions, Chou's iterative partitioning algorithm is similar to the iterative algorithm proposed by Nádas *et al.* [19], with differences in the mechanics of the two steps because of the $\max \Delta_1$ vs. $\min \Delta_2$ interpretations.

Chou [17] shows that this algorithm can be used with a variety of loss functions including, for example, the weighted squared error for regression ($y \in R^M$) and the weighted Gini index and log likelihood for classification ($y^T = [00 \cdots 010 \cdots 0]$ an M-valued class indicator with a 1 in the mth row to indicate class m). Here we outline the algorithm specifically for the maximum log likelihood objective function, assuming that distributions are characterized by Gaussians.

4.2 Implementation for Maximum Gaussian Log Likelihood

For the problem of clustering Gaussians, $y \in R^M$ corresponds to a cepstral vector in our speech recognition application. Each element of $\mathbf{X}$ is a possible split domain (e.g. the left context phone label), and X is an element of $\mathbf{X}$ that takes on a discrete set of J values (e.g. the 26 possible Japanese phones). We assume a parametric Gaussian model, $\mathrm{p}(y|s)$ with mean $\mu(s)$ and covariance matrix $\Sigma(s)$. A state will then be represented by $\theta(s) = (\mu(s), \Sigma(s))$. Recall that the space of

possible values of X corresponding to state s is characterized by A_s. The goal is to find the best split of s into s_0 and s_1 where $A_s = A_0 \cup A_1$.

Referring back to the general algorithm, we need to determine $d(s,\theta)$ and the formula for finding the optimal decoder, based on some specified $\mathcal{L}(y,\theta)$. If the objective is maximum likelihood, then $\mathcal{L}(y,\theta) = -\log \mathrm{p}(y|\theta)$. Under this objective function, equation 13 becomes

$$\begin{aligned} \theta(s) &= \operatorname*{argmin}_{\theta} E[\mathcal{L}(Y,\theta)|s] = \operatorname*{argmax}_{\theta} E[\log \mathrm{p}(Y|\theta)|s] \\ &= \operatorname*{argmax}_{\theta} \sum_{t:x_t \in A_s} \log \mathrm{p}(y_t|\theta) \end{aligned} \tag{15}$$

using the empirical distribution since we are training from data and the true $\mathrm{p}(y|s)$ is unknown. Note that this is the standard maximum likelihood parameter estimate, which gives mean $\mu(s)$ and covariance $\Sigma(s)$. Then the divergence (equation 14) becomes

$$\begin{aligned} d(s,\theta) &= E[\mathcal{L}(Y,\theta)|s] - i(s) \\ &= -\sum_{t:x_t \in A_s} \log \mathrm{p}(y_t|\theta) + \sum_{t:x_t \in A_s} \log \mathrm{p}(y_t|\theta(s)) \\ &= \frac{1}{2}\Bigg[N_s \log|\Sigma| + \sum_{t:x_t \in A_s} (y_t-\mu)^t \Sigma^{-1} (y_t-\mu) \\ &\qquad -N_s \log|\Sigma(s)| - \sum_{t:x_t \in A_s} (y_t-\mu(s))^t \Sigma(s)^{-1} (y_t-\mu(s)) \Bigg] \end{aligned} \tag{16}$$

where N_s is the number of observations that map to state s and $\theta = (\mu, \Sigma)$. The superscript t indicates vector transpose, and $|A|$ represents the determinant of a matrix A.

Binary split design at state s for a single J-valued variable proceeds as follows.

Maximum Likelihood Split Design Algorithm

- **Initialization** ($p = 0$) Assign initial values to the distribution parameters for the two hypothesized states:

$$\theta^{(0)}(s_0) = \theta(s) = (\mu(s), \Sigma(s))$$
$$\theta^{(0)}(s_1) = (\mu(s)(1+\epsilon), \Sigma(s))$$

  This particular choice ensures that likelihood will increase since one of the states has the original state distribution parameters, analogous to the approach used in vector quantizer design.

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

  1. Find new binary partition $\{A_0^{(p)}, A_1^{(p)}\}$:
     For each $x_j, j = 1, \ldots, J$, assign x_j to $A_0^{(p)}$ if

$$\sum_{t:x_t=x_j} \log \mathrm{p}(y_t|\theta^{(p-1)}(s_0)) \geq \sum_{t:x_t=x_j} \log \mathrm{p}(y_t|\theta^{(p-1)}(s_1)) \tag{17}$$

     otherwise, assign x_j to $A_1^{(p)}$.

  2. Find centroids $\{\theta^{(p)}(s_k) = (\mu^{(p)}(s_k), \Sigma^{(p)}(s_k)) : k = 0, 1\}$ using standard maximum likelihood parameter estimation.

$$\mu^{(p)}(s_k) = \frac{1}{N_k} \sum_{t:x_t \in A_k^{(p)}} y_t \tag{18}$$

$$\Sigma^{(p)}(s_k) = \frac{1}{N_k} \sum_{x_j \in A_k^{(p)}} \sum_{t:x_t=x_j} (y_t - \mu^{(p)}(s_k))(y_t - \mu^{(p)}(s_k))^t \tag{19}$$

     where $N_k = \sum_{x_j \in A_k^{(p)}} N_j$, N_j is the number of elements in $\{t : x_t = x_j\}$, and $N_0 + N_1 = N_s$.

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

$$\frac{L^{(p)} - L^{(p-1)}}{L^{(p-1)}} < \eta$$

     where $L^{(p)} = -N_0 \log|\Sigma^{(p)}(s_0)| - N_1 \log|\Sigma^{(p)}(s_1)|$ (see Appendix A for a derivation), and η is a heuristically chosen convergence threshold. Note that $L^{(p)} \geq L^{(p-1)}$.

For both steps in the algorithm above, we can save computation by using sufficient statistics to represent the data rather than accumulating log probabilities for every data point. Specifically, we first compute the cumulative statistics that describe the data y_t associated with state s for each

value x_j that the variable of interest X might take on. Let N_j represent the number of samples (frames) in state s that have $X = x_j$. Define first and second order statistics

$$S_j^1(s) = \sum_{t:x_t=x_j, s_t=s} y_t \tag{20}$$

$$S_j^2(s) = \sum_{t:x_t=x_j, s_t=s} y_t y_t^t \tag{21}$$

These statistics are computed once for state s in the initialization step and stored together with counts N_j. In the paragraphs below, we show how these statistics can be used in the re-partitioning test (Equation 17) and parameter re-estimation. An alternative solution, which may be more efficient for full covariance distributions, is given in Appendix B for a different set of statistics (also sufficient).

We begin by expanding the re-partitioning test (Equation 17)

$$\begin{aligned} &2N_j \log|\Sigma(s_0)| + \sum_{t:x_t=x_j} (y_t - \mu(s_0))^t \Sigma(s_0)^{-1} (y_t - \mu(s_0)) \\ &\quad \leq 2N_j \log|\Sigma(s_1)| + \sum_{t:x_t=x_j} (y_t - \mu(s_1))^t \Sigma(s_1)^{-1} (y_t - \mu(s_1)), \end{aligned} \tag{22}$$

dropping the superscript (p) indicating iteration number to simplify the notation. The summation terms can be simplified to use the statistics given by Equations 20 and 21, as follows,

$$\begin{aligned} &\sum_{t:x_t=x_j} (y_t - \mu(s_0))^t \Sigma(s_0)^{-1} (y_t - \mu(s_0)) \\ &\quad = \sum_{t:x_t=x_j} tr\left[(y_t - \mu(s_0))(y_t - \mu(s_0))^t \Sigma(s_0)^{-1}\right] \\ &\quad = tr\left[\sum_{t:x_t=x_j} (y_t - \mu(s_0))(y_t - \mu(s_0))^t \Sigma(s_0)^{-1}\right] \\ &\quad = tr\left[\sum_{t:x_t=x_j} (y_t y_t^t - y_t \mu(s_0)^t - \mu(s_0) y_t^t + \mu(s_0)\mu(s_0)^t) \Sigma(s_0)^{-1}\right] \\ &\quad = tr\left[(S_j^2 - S_j^1 \mu(s_0)^t - \mu(s_0)(S_j^1)^t + N_j \mu(s_0)\mu(s_0)^t) \Sigma(s_0)^{-1}\right] \end{aligned}$$

where we used the identity $z^t A z = tr(z z^t A)$ and the fact that the trace function $tr(\cdot)$ is a linear operator. Combining these results with Equation 22, we get the following test

$$\begin{aligned} &2N_j \log|\Sigma(s_0)| + tr\left[(S_j^2 - 2S_j^1 \mu(s_0)^t + N_j \mu(s_0)\mu(s_0)^t) \Sigma(s_0)^{-1}\right] \\ &\quad \leq 2N_j \log|\Sigma(s_1)| + tr\left[(S_j^2 - 2S_j^1 \mu(s_1)^t + N_j \mu(s_1)\mu(s_1)^t) \Sigma(s_1)^{-1}\right], \end{aligned} \tag{23}$$

The parameter re-estimation equations using the sufficient statistics are:

$$\mu(s_k) = \frac{1}{N_k} \sum_{x_j \in A_k} \sum_{t:x_t=x_j} y_t = \frac{1}{N_k} \sum_{x_j \in A_k} S_j^1(s) \tag{24}$$

$$\begin{aligned}\Sigma(s_k) &= \frac{1}{N_k}\sum_{x_j\in A_k}\sum_{t:x_t=x_j}(y_t-\mu(s_k))(y_t-\mu(s_k))^t \\ &= \frac{1}{N_k}\sum_{x_j\in A_k}\sum_{t:x_t=x_j}(y_t y_t^t - y_t\mu(s_k)^t - \mu(s_k)y_t^t + \mu(s_k)\mu(s_k)^t) \\ &= \frac{1}{N_k}\sum_{x_j\in A_k}(S_j^2 - S_j^1\mu(s_k)^t - \mu(s_k)(S_j^1)^t + N_j\mu(s_k)\mu(s_k)^t). \end{aligned} \tag{25}$$

Both the likelihood test and the parameter re-estimation equations simplify if we assume diagonal covariances. To simplify the cluster likelihood test, note that

$$\log|\Sigma| = \sum_{m=1}^{M}\log\sigma_m^2 \tag{26}$$

$$tr(\Sigma_A\Sigma_B^{-1}) = \sum_{m=1}^{M}\sigma_{A,m}^2/\sigma_{B,m}^2. \tag{27}$$

Then the new re-partitioning test becomes

$$C_0+\sum_m\frac{S_{j,m}^2(s)-2S_{j,m}^1(s)\mu_m(s_0)+N_j\mu_m(s_0)^2}{\sigma_m^2(s_0)} \le C_1+\sum_m\frac{S_{j,m}^2(s)-2S_{j,m}^1(s)\mu_m(s_1)+N_j\mu_m(s_1)^2}{\sigma_m^2(s_1)} \tag{28}$$

where

$$C_k = N_j\log(\prod_m\sigma_m^2(s_k)).$$

Equation 25 also simplifies if covariances are assumed to be diagonal, to

$$\sigma_m^2(s_k) = \frac{1}{N_k}\sum_{x_j\in A_k}(S_{j,m}^2 - 2S_{j,m}^1\mu_m(s_k)+N_j\mu_m(s_k)^2) \tag{29}$$

$$= \mu_m(s_k)^2 + \frac{1}{N_k}\sum_{x_j\in A_k}(S_{j,m}^2 - 2S_{j,m}^1\mu_m(s_k)) \tag{30}$$

for $m=1,\ldots,M$.

To extend this algorithm to the case where observations are associated probabilistically with states, via the Baum-Welch algorithm rather than via the Viterbi algorithm, one simply weights each term inside the sum of equations 20 and 21 by the likelihood that it is in the state that is being updated. Specifically, let $\gamma_t(s)$ correspond to the probability that the state at time t is s. Then, the new sufficient statistics are:

$$\begin{aligned} S_j^1(s) &= \sum_{t:x_t=x_j}\gamma_t(s)y_t \\ S_j^2(s) &= \sum_{t:x_t=x_j}\gamma_t(s)y_t y_t^t \\ N_j(s) &= \sum_{t:x_t=x_j}\gamma_t(s) \end{aligned}$$

The $\gamma_t(s)$ terms must be computed using both a forward and backward pass. This information is in principle available from the Baum-Welch iteration in SSS and SI-SSS, but there is no data structure in SSS that saves all of this information and it needs to be added for SI-SSS. Note that the $\gamma_t(s)$ terms are not needed in SSS to find either the best node to split or the best split given a node, since the best node to split is chosen from the generic mixture distributions and the best split uses the forward algorithm.

5 Constrained Design of Temporal Splits

In HM-net design, using the greedy search strategy outlined in Section 2, our goal is to maximally increase the likelihood of the training data at each step. In Section 3, we argued for a constrained EM approach, where the increase in expected likelihood to the HM-net as a whole is simply the difference in expected likelihood for the splittee state vs. the two new states. Constraining the forward-backward counts in designing the contextual split is straightforward, since the likelihood of two states in parallel sum to give the likelihood of the original state. However, the likelihood of two states in sequence is not given by a simple sum.

In SSS, design of a temporal split involves using the HMM forward algorithm and changing the state likelihoods ($\gamma_t(s)$, the likelihood that the state is s at time t) of states other than the splittee state, in which case a larger portion of the network must be evaluated to establish the change in likelihood for the HM-net as a whole. In addition to the extra cost of evaluating a possibly large subnet, a problem with allowing the likelihoods of other states to change for the temporal split and not the contextual split is that there will be a bias towards choosing temporal splits.

The constrained EM criterion addresses both of these problems in the design of temporal splits, with the constraints being that the likelihoods of states other than the splittee state do not change in the parameter estimation stage of split design. (Recall that likelihoods of all affected states will later be updated in the Baum-Welch re-estimation phase of step 2 of the SI-SSS algorithm, after the split is chosen.) To be more explicit, let s^* be the splittee state and let q_0 and q_1 be the two states resulting from a temporal split, as illustrated in Figure 2. (We use the notation q for the hypothetical new states and s^* for the candidate state to be split, to make the relationship between these more clear.) The parameters that must be estimated to describe the new state are $\theta = \{\mu(q_0), \sigma(q_0), \nu(q_0), \mu(q_1), \sigma(q_1), \nu(q_1)\}$, where $\mu(q)$ is the mean vector for state q, $\sigma(q)$ is the vector of variances, and $\nu(q)$ is the probability of returning to state q from state q, i.e. the self loop transition probability. In order to insure that only these parameters in the HM-net change and no others do, we require the following constraints:

$$\begin{aligned} \gamma_t(s^*) &= \gamma_t(q_0) + \gamma_t(q_1) \\ \xi_t(s^*, s^*) &= \xi_t(q_0, q_0) + \xi_t(q_1, q_0) + \xi_t(q_1, q_1) \end{aligned}$$

where

$$\begin{aligned} \gamma_t(i) &= p(s_t = i|\mathcal{Y}) \\ \xi_t(i,j) &= p(s_t = i, s_{t-1} = j|\mathcal{Y}) \end{aligned}$$

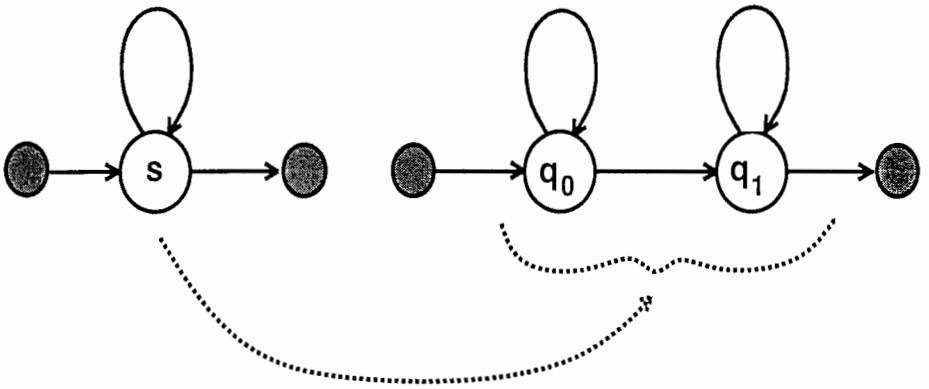


Figure 2: Temporal split of s into q_0 and q_1.

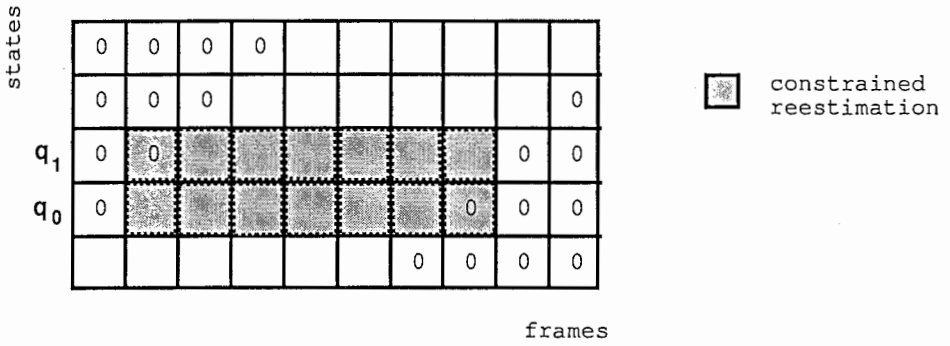


Figure 3: Illustration of data and states used in computing $\tilde{\gamma}_t(q)\ \tilde{\xi}_t(q,q')$ for a temporal split. The zeroes indicate impossible state-observation pairs.

are the standard terms needed for HMM re-estimation and $\mathcal{Y}$ represents the full training set.

These constraints can be easily satisfied by defining

$$\begin{aligned}\tilde{\gamma}_t(q) &= p(q_t = q|s_t = s^*, \mathcal{Y}) \\ \tilde{\xi}_t(q,q') &= p(q_t = q, q_{t-1} = q'|s_t = s^*, s_{t-1} = s^*, \mathcal{Y})\end{aligned}$$

and using the definition of conditional probability and the redundancy of $s_t = s^*$ to get

$$\begin{aligned}\gamma_t(q) &= p(q_t = q|\mathcal{Y}) = p(q_t = q, s_t = s^*|\mathcal{Y}) = \tilde{\gamma}_t(q)\gamma_t(s^*) \\ \xi_t(q,q') &= p(q_t = q, q_{t-1} = q'|\mathcal{Y}) = p(q_t = q, q_{t-1} = q', s_t = s^*, s_{t-1} = s^*|\mathcal{Y}) = \tilde{\xi}_t(q,q')\xi_t(s^*,s^*).\end{aligned}$$

The terms $\tilde{\gamma}_t(q)$ and $\tilde{\xi}_t(q,q')$ can be computed using the standard forward-backward algorithm using only data where $\gamma_t(s^*) > 0$ and having non-zero state likelihood only for states q_0 and q_1 so that $\tilde{\gamma}_t(q_0) + \tilde{\gamma}_t(q_1) = 1$. To thus constrain the forward-backward algorithm is simply a matter of initializing the forward and backward passes appropriately, or passing a subset of the full data structure as illustrated by the shaded region in Figure 3.

Once the terms $\tilde{\gamma}_t(q)$ and $\tilde{\xi}_t(q,q')$ are computed, the parameters θ are estimated according to

$$\mu_m(q) = \frac{\sum_t \tilde{\gamma}_t(q)\gamma_t(s^*)y_{t,m}}{\sum_t \tilde{\gamma}_t(q)\gamma_t(s^*)} \tag{31}$$

$$\sigma_m(q) = \frac{\sum_t \tilde{\gamma}_t(q)\gamma_t(s^*)y_{t,m}^2}{\sum_t \tilde{\gamma}_t(q)\gamma_t(s^*)} - \mu_m(q)^2 \quad (32)$$

$$\nu(q) = \frac{\sum_t \tilde{\xi}_t(q,q)\xi_t(s^*,s^*)}{\sum_{q'}\sum_t \tilde{\xi}_t(q',q)\xi_t(s^*,s^*)} \quad (33)$$

Note that the forward-backward algorithm that is used to compute $\tilde{\gamma}_t(q)$ and $\tilde{\xi}_t(q,q')$ cannot be used to find the likelihood of the observations that map to the two new states, so relative change in likelihood cannot be used as a stopping criterion for the temporal split retraining. Since the split will later be retrained with the Baum-Welch algorithm, it is reasonable to simply run a fixed number of retraining iterations, and four iterations were used here.

One problem with designing a temporal split is that, unlike the contextual split, there is no guarantee of non-decreasing likelihood. Although the SI-SSS temporal split re-estimation procedure guarantees non-decreasing likelihood because it is a constrained version of the EM algorithm, the split from one to two states cannot be initialized in such a way as to guarantee no decrease in likelihood. (A reasonable initial estimate, which is used in this work, is to use the observation distribution of the original state and choose the transition probabilities such that the expected duration of the two hypothesized states together is the same as the expected duration of the original state.) In practice, decreases in likelihood, though rare, do sometimes occur, in which case the temporal split for that state would never be chosen. The SSS temporal splitting algorithm suffers from a similar problem, since it chooses the best temporal split for a fixed set of Gaussians which may not be well-matched to a temporal domain split since they were not designed specifically for that type of split. However, the SSS algorithm may not avoid a bad temporal split, since nodes are split based on distance between mixture components and irrespective of the actual consequence of the split. Of course, the SSS algorithm could potentially achieve a bigger immediate gain than the SI-SSS temporal split by allowing state re-alignment in split design, but this difference is probably small because SI-SSS allows state re-alignment in the immediately following Baum-Welch re-estimation step. Thus, on balance, we feel that the SI-SSS temporal split test represents an improvement over the SSS temporal split test.

6 Experiments

In this section we describe two series of experiments that represent preliminary work in establishing the effectiveness of SI-SSS. After describing the corpora used in the work, we present results for speaker-dependent speech recognition on read speech. This experiment represents the most difficult test for SI-SSS, because it is the paradigm that SSS was developed under and because the very controlled nature of the corpus does not present the type of problems that SI-SSS is aimed at solving. Next, we describe experiments on a multi-speaker task, as a preliminary step demonstrating gains in a task closer to our goal of speaker-independent recognition. Finally, we discuss results using SSS on spontaneous speech under the standard ATR training paradigm, aimed at obtaining the best possible SSS baseline which will be the target for SI-SSS to improve on in later experiments.

6.1 P digm

The goal of this work is to develop better speaker-independent models for spontaneous speech, to obtain the best possible performance when operating in speaker-independent recognition mode as well as to provide a better starting point for speaker adaptation. Since the amount of spontaneous speech data available at the time of this study was not sufficient for training with the Baum-Welch algorithm, we focused on training an initial model on read speech and adapting that model using vector field smoothing on spontaneous speech.

Several corpora were used in these experiments. A Japanese corpus of isolated read words consisting of the most frequent 5240 words (A-set) is used for initial topology training in the speaker-independent experiments (1-2 speakers) and for the speaker-dependent experiments (6 speakers). All sentences in the read speech corpora are hand-transcribed with a phoneme label sequence, and start and end points for each speech segment, which facilitates SSS training. The speech was recorded from professional speakers under low noise conditions. In the speaker-independent experiments, the prototype models designed on the A-set were then retrained on a subset of the speaker-independent read speech database (C-set), consisting of 15 speakers each uttering 50 phoneme-balanced sentences three times at different speaking rates. The C-set data was recorded with the same type of microphone as the A-set data. Pause units were hand-marked, but not phone boundaries. The A-set and C-set corpora are described in [20]. Finally, a corpus of spontaneous speech was used for evaluating recognition performance [21]. This corpus was divided into training and test sets, as specified in [22]. All spontaneous speech training data is from a set of bi-lingual conversations involving 1 Japanese, 1 English speaker and 2 translators (i.e. the onseigengo part). Three different companies collected the data and therefore the quality, e.g. SNR, is quite different. Only the non-translator speech is used, and a small number of pause units of greater than six seconds in length were omitted from the training set due to memory constraints.

For the development test data used in speaker-independent experiments, we used both the onseigengo and the onsei part of the spontaneous speech corpus, i.e. mono-lingual conversations between two Japanese speakers. This data appears more "spontaneous", in the sense that there are more filled pauses in these utterances. The test set contains four female speakers (15 conversational turns, 9711 phonemes) and three male speakers (16 conversational turns, 11231 phonemes).

The analysis parameters consisted of sampling rate of 12000 Hz, frame shift of 5 ms, frame length of 20 ms, pre-emphasis of 0.98, LPC analysis with order 16 and calculation of 16 cepstral, 16 delta cepstral, power and delta power values. The length of the triangular regression window for the delta cepstral calculation was 9 frames on either side, i.e. a two-sided window of 90 ms. Recognition experiments were performed using a one-pass Viterbi algorithm with the phonotactic constraints of Japanese language expressed as phoneme-pair grammar [23].

6.2 Speaker-Dependent HM-net Experiments

In order to verify that the SI-SSS algorithm always performs at least as well as the SSS algorithm, we conducted initial experiments in speaker-dependent mode. 200 and 400 state single Gaussian models and a 3-mixture 400 state model were trained on the even-number words of the A-set for

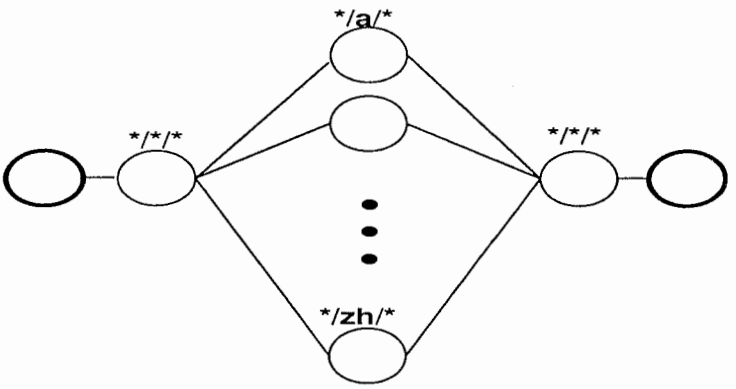


Figure 4: Initial HM-net topology for the speaker-dependent experiments.

each speaker (2620 words). The initial topology used twenty-six states, one center state for each of the 24 phones and a single left and right state shared by all phones, as shown in Figure 4, to reduce the initial HMnet training time and guarantee that each phone is identifiable. The models were tested on 1310 of the odd-numbered words. After designing the HM-net topology, a maximum of 21 iterations of Baum-Welch re-iteration were run to estimate single Gaussian state observation distributions. (For the single Gaussian models, usually fewer than 10 iterations were required, using a threshold test on the relative likelihood gain.) The results are summarized in Table 1, which shows that on average and in almost all cases, results for SI-SSS are slightly better than for SSS. The one exception is speaker MHT, which was used in much of the SSS development work. Differences are not significant, as expected since the main variability in state distributions for this speaker-dependent data is contextual, particularly because the speakers are professional and the recordings are high quality.

In addition, we noticed that SI-SSS distributes multiple allophones across more phones than does SSS, particularly for the 200-state topology. SI-SSS results in more allophones for the consonants than SSS, as well as a somewhat more uniform distribution of allophones over the vowels. The distribution differences are particularly striking for /a/ vs. /u/, where SSS many more allophones for /a/ than for /u/ and SI-SSS has similar numbers for the 400 state model and more allophones for /u/ in the 200 state model.

Computation time for SSS and SI-SSS was measured for each successive split. Computation time for SI-SSS is significant less than for SSS, particularly after all possible temporal splits have been chosen since temporal splits are more costly for SI-SSS. (The number of temporal splits is limited here, since the maximum number of states in sequence is limited to four which effectively establishes a minimum phone duration constraint of 20 ms.) Figure 5 illustrates the difference in computation costs for speaker FTK. On the other hand, SI-SSS requires more memory than SSS because the Baum-Welch state likelihoods must be stored for use in split design. For the 2620 word training set size, the difference in cost is roughly 80MB vs. 50MB. We estimate that speaker-independent training with 10 speakers and 1000 words per speaker could be run using 100MB main memory and swapping the parameter file to disk.

Table 1: *Speaker-dependent phoneme recognition accuracy for SSS vs. SI-SSS topology design.*

	% Accuracy					
	200 states 1 mix		400 states 1 mix		400 states 3 mix	
Speaker	SSS	SI-SSS	SSS	SI-SSS	SSS	SI-SSS
MHT	93.9	92.8	95.4	94.5	96.1	96.0
MAU	93.6	93.2	95.2	95.2	96.4	96.7
MXM	91.7	91.9	93.6	93.9	95.3	95.1
FTK	91.5	91.1	92.9	94.0	94.7	95.0
FMS	89.7	91.3	91.9	93.2	94.2	94.6
FYM	90.7	92.4	92.9	93.6	95.1	95.5
avg	91.9	92.1	93.7	94.1	95.3	95.5

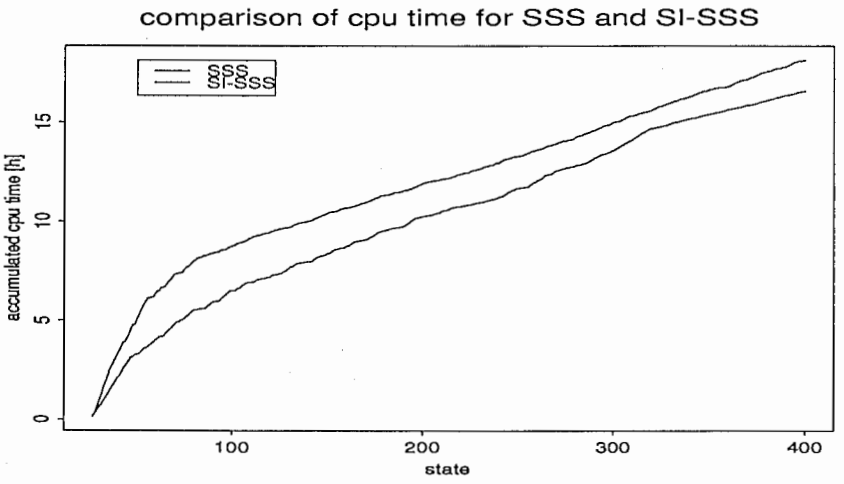


Figure 5: Difference in CPU time required to split each state in sequence for SSS vs. SI-SSS, designing up to 400 states on speaker FTK.

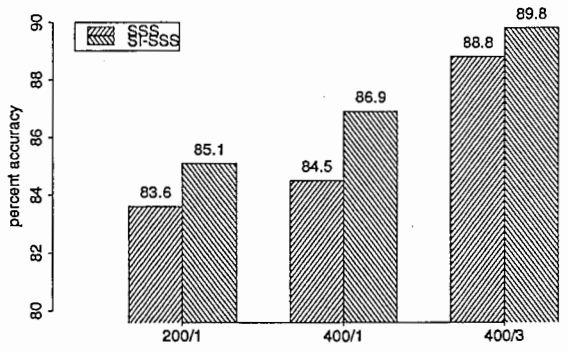


Figure 6: Phoneme recognition accuracy for SSS vs. SI-SSS for a multi-speaker recognition task.

6.3 Multi-Speaker HM-net Experiments

Next we conducted a multi-speaker recognition experiment comparing SSS and SI-SSS. From each of the 6 speakers MAU, MHT, MXM, FYM, FMS, FTK we selected randomly 500 words from the even numbered words of the 5240 word database [20]. This gives a total of 3000 words, which is about the same number as has been used for each of the speaker-dependent experiments. Clearly, more training data is needed for a multi-speaker or speaker-dependent model, but the experiment was conceived mainly for debugging purposes. The same HM-net design procedures were used as in the speaker-dependent experiment, i.e. models were retrained with a single Gaussian (1 mixture) for 200 states, and 1 and 3 mixtures for 400 states. For recognition we tested in multi-speaker mode, using 100 randomly selected words from each of the above 6 speakers.

The results are shown in Fig. 6. SI-SSS consistently performs better than SSS, the difference being greatest for the higher context resolution model (400 states) using only 1 mixture. Using 3 mixtures the difference is smaller, which is not surprising since the lack of allophones can be compensated for by the introduction of mixtures.

6.4 Baseline SSS Experiments on Spontaneous Speech

A series of experiments were conducted with the SSS algorithm, following the paradigm used in [24], in order to establish a good SSS baseline for spontaneous speech. The model building strategy, which is outlined in Figure 7, progressively adds complexity and data to the training process in order to obtain a robust model and avoid problems of local optima. The first step is to train the HM-Net on speaker-dependent A-set data (the 2620 even-numbered words); speaker MHT is used as the prototype for the male and gender-independent models, and speaker FTK is used for the female models. Since the resulting model from SSS has two mixtures per state, the models are then retrained on the same data to have a single Gaussian per state, using 20 iterations of Baum-Welch re-estimation (or improvement of less than 10^{-5}). Next, a set of 15 speaker-dependent models are trained using vector field smoothing (VFS) [25], using representative speakers from the C-set data chosen in previous experiments using tree-structured speaker clustering [26]. The VFS step

involved first running Baum-Welch re-estimation for both means and variances (variances are only allowed to increase) and then smoothing the means with the original parameters using a smoothing rate of five and a neighborhood of six. The 15 speaker-adapted models, each with 1 mixture per state, were then merged into a 5 mixture model with mixture weights proportional to the number of speakers in each of the 5 clusters, as in [27]. The particular choice of speakers differed for the speaker-independent and gender-dependent models, but all were at the highest nodes of the tree possible depending on the gender constraint. The 5-mixture model is then retrained using 5 iterations of the Baum-Welch algorithm on the pooled data from the same 15 speakers using 150 sentences from each. A three-state, ten-mixture silence model was then concatenated with the 5 mixture HM-net. (The three-state silence model is used here because, in preliminary experiments with a different test set of spontaneous speech, we found that the phoneme recognition accuracy rate was consistently higher for models using a three-state silence than those using a one-state silence model.) Finally, this speaker-independent model is optionally adapted using spontaneous speech data with VFS, using the same smoothing procedure as described for speaker adaptation above.

We conducted a series of experiments where we varied:

- the number of states
- whether or not power was used as a feature
- gender-dependent vs. gender-independent
- whether or not adaptation was used.

The biggest effects were obtained by doing adaptation and using gender-dependent models, with the best results being an average of 76.2% accuracy on the two test subsets combined using a gender-dependent model with 400 states, power and adaptation to spontaneous speech. The detailed results are summarized below.

Since the recording conditions of the spontaneous speech corpus was so different from the recording conditions for the read speech training corpus, we felt that including power as a feature might be more harmful than helpful. Figure 8 shows that the difference in performance is not significant for $\pm$power, particularly when the model is not adapted to spontaneous speech. Since there seemed to be a slight overall improvement with the use of power, it was retained in further experiments. There is a significant gain in performance due to using adaptation (25% reduction in error rate), which is consistent with the results in [24] using the same paradigm but a different test set. Note that there is only a small gain in going from 200 to 400 states, which is unexpected based on previous experiments on read speech but might be explained by the fact that the HM-net topology was designed on read speech from a single speaker.

Next, we looked at the effect of using gender-dependent models, and found a significant gain in performance as illustrated in Figure 9 for models that used power as a feature. Error reduction relative to the gender-independent read-speech model is 17-20% for gender-dependent models and 24-25% for adaptation to spontaneous speech, so adaptation gives a bigger gain of the two factors

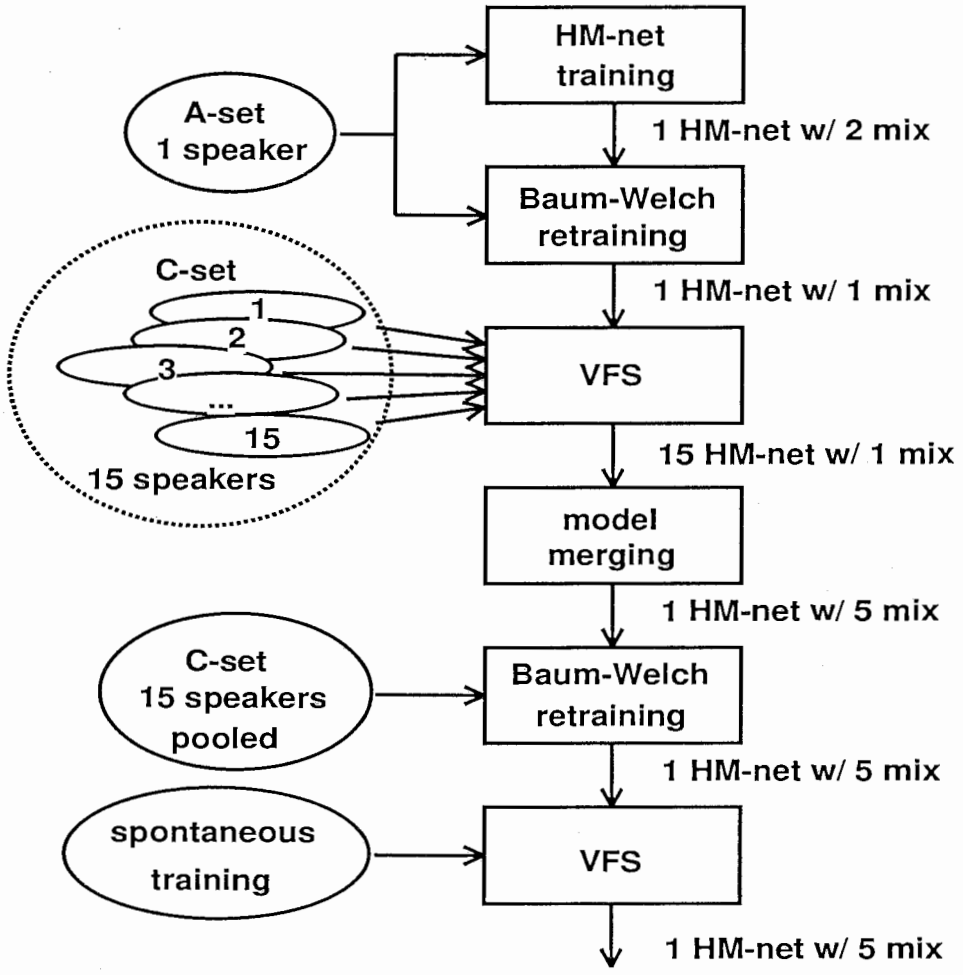


Figure 7: Block diagram of steps used to train a speaker-independent model for spontaneous speech.

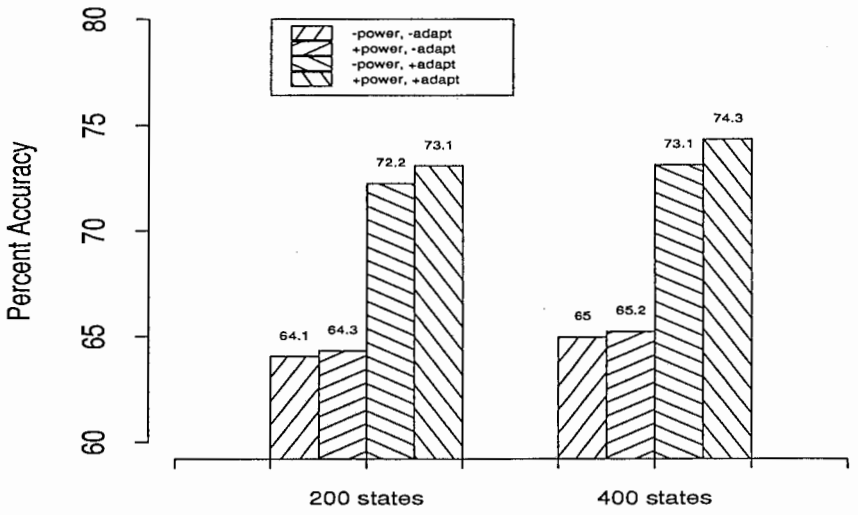


Figure 8: Average recognition results on spontaneous speech using gender-independent models for 200 vs. 400 states, ± power as a feature, and ± adaptation.

alone. The combination of the two techniques gives a small additional improvement, 11% error reduction for 200 states, but only 5% for 400 states. Again, there is very little gain from increasing the number of states from 200 to 400.

As in most speech recognition experiments, the gains in average recognition performance are small relative to the variability across speakers. Figure 10 illustrates this variability for the same conditions as shown in Figure 9, with the 200 and 400 state cases connected by lines. Note that the variability in performance is much higher for the 400 state model than the 200 state model trained only on read speech data, though the average performance of the 400 state model is higher. This variance decreases somewhat with gender-dependent modeling and further with adaptation.

7 Discussion

In summary, in this paper we have proposed a new algorithm for HMM topology design, that is both an extension of successive state clustering and a generalization of decision tree distribution clustering. SSS is limited in that it cannot handle speaker-independent training data, primarily because states are chosen to be split based on variance and not based on the gain that would be achieved by a specific split. By choosing the node and the candidate split at the same time, we can avoid splitting states that simply reflect speaker variability and therefore achieve better performance in a speaker-independent task, hence the name SI-SSS. Using a maximum expected likelihood

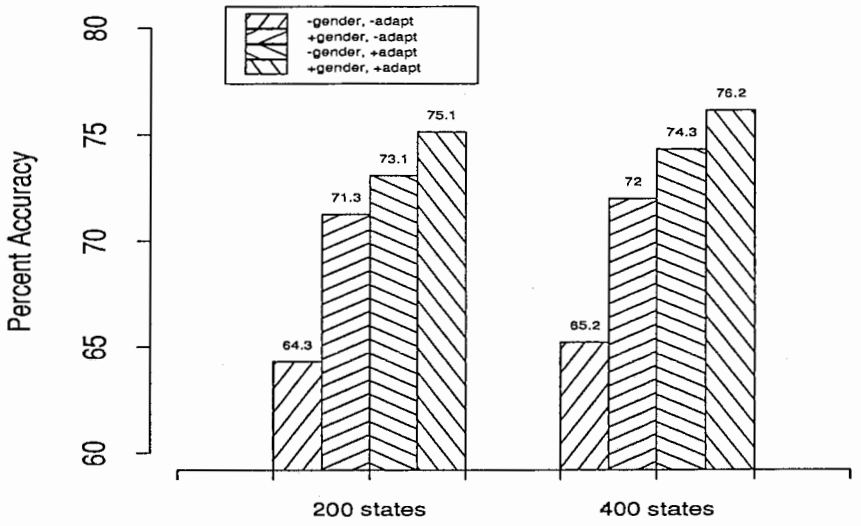


Figure 9: Average recognition results on spontaneous speech using power as a feature, comparing gender-independent vs. gender-dependent models, 200 vs. 400 states, and ± adaptation.

criterion, we can also ensure that node splitting always improves the likelihood of the training data. By allowing both contextual and temporal splits, SI-SSS also represents a generalization of decision tree distribution clustering techniques.

Experimental results on a speaker-dependent task – the most difficult test – demonstrate that SI-SSS performs as well as SSS with lower computational costs. We anticipate that in training on speaker-independent data, the advantages of SI-SSS will play a significant role and enable the design of robust HMM topologies, particularly for recognition of spontaneous speech. Preliminary experiments on a multi-speaker task support this hypothesis, but to fully establish that SI-SSS outperforms SSS and achieves the goal of robust speaker-independent training, several experiments remain. We hope to continue the work by first running speaker-independent read speech training experiments, testing on both read and spontaneous speech, and then finally moving to speaker-independent spontaneous speech training.

Being a generalization of decision tree clustering, SI-SSS inherits some of the advantages and disadvantages of decision tree clustering. On the positive side, trees provide a robust back-off mechanism for unseen contexts, and easily allow for incorporating more general contextual features beyond simply triphone context. Thus, it may be interesting to experiment using SI-SSS with wider context windows, stress labels, syllable/mora structure and/or other possible conditioning factors. On the negative side, the recursive partitioning of training data that is part of the tree design process is not an efficient use of data. Nodes that are split on the same question in different

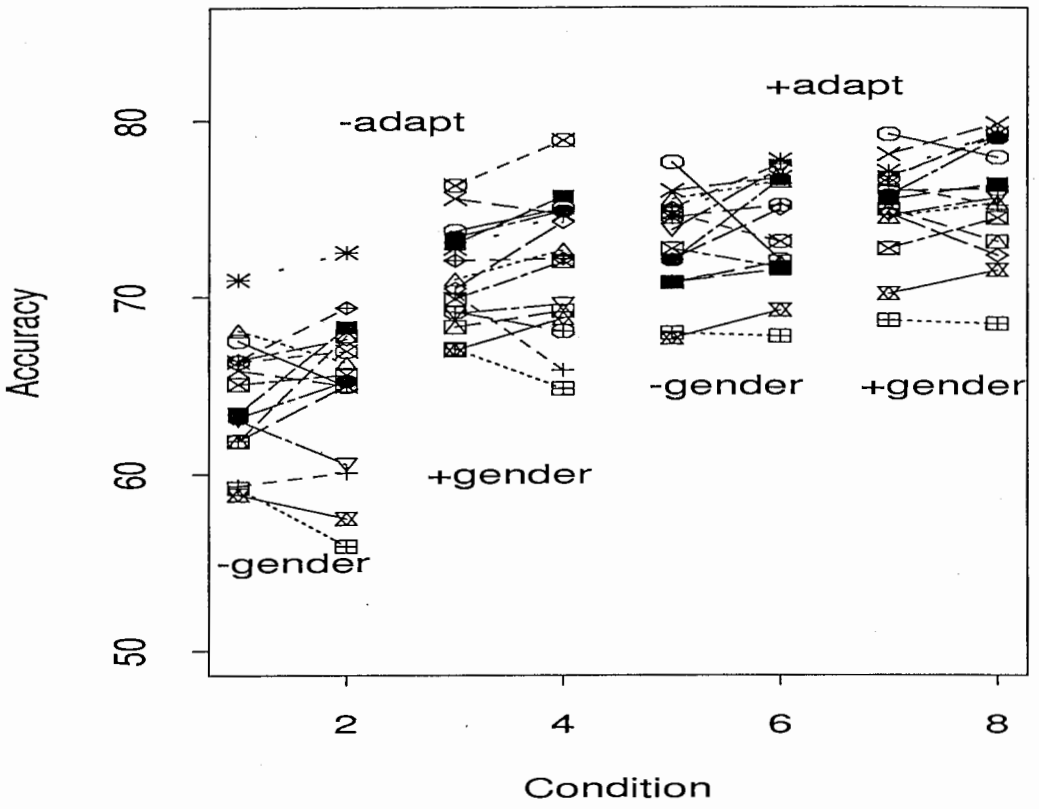


Figure 10: Recognition results on spontaneous speech using power as a feature, comparing gender-independent vs. gender-dependent models, 200 vs. 400 states, and ± adaptation, separated by speaker and dialog. The line segments connect the 200 and 400 state cases.

parts of the tree cannot benefit from the large amount of training data associated with the two nodes together. One means of addressing this problem is merging of nodes, as explored in SSS [29], general decision tree design [17], and distribution clustering [12, 28]. Allowing node merging at all steps of tree growing is expensive; more practical alternatives include node merging only after a maximum size has been reached or after a full tree has been grown.

Although SI-SSS offers both performance and speed improvements over SSS, it does involve the extra cost of higher memory requirements to store the forward-backward counts needed in the constrained EM estimation procedure. In this work, which has trained on relatively small amounts of data, we have addressed the problem of memory cost by using the hand-marked phone boundaries (used in SSS) to constrain the set of non-zero forward-backward counts. To further reduce the memory requirements, one might disallow state sharing across phones and train phones separately, as in [11]. Another possible approach would be to restrict split gain re-estimation to the nodes immediately touching the splittee state, as proposed in Section 2.

Clearly, there are many avenues for future work that build on the SI-SSS algorithm and the experimental result reported here. In particular, we anticipate that SI-SSS will represent an important tool for advancing the state of the art in spontaneous speech recognition.

A Maximum Likelihood of Gaussian Data

This appendix gives a proof of a standard result (e.g. [14]) that the maximum likelihood of a sample set of independently and identically distributed Gaussian vectors $\{y_t; t = 1, \ldots, N\}$ is proportional to the determinant of the ML covariance estimate:

$$\max_{\mu,\Sigma} \prod_t \mathrm{p}(y_t|\mu,\Sigma) = \frac{e^{-NM/2}}{(2\pi)^{NM/2}|\hat{\Sigma}|^{N/2}} \tag{34}$$

where

$$\hat{\Sigma}_{ML} = \frac{1}{N}\sum_t (y_t - \hat{\mu})(y_t - \hat{\mu})^t \tag{35}$$

This result is needed to get the simple convergence test used in the maximum likelihood split design algorithm in Section 4.2.

Proof: First, expand out the probability terms:

$$\begin{aligned} \max_{\mu,\Sigma} \sum_t \log \mathrm{p}(y_t|\mu,\Sigma) &= \sum_t \log \mathrm{p}(y_t|\hat{\mu},\hat{\Sigma}) \\ &= \frac{-1}{2}\left[NM\log(2\pi) + N\log|\hat{\Sigma}| + \sum_t (y_t - \hat{\mu})^t \hat{\Sigma}^{-1}(y_t - \hat{\mu})\right] \end{aligned} \tag{36}$$

Then, using the fact that $x^t A x = tr(x x^t A)$ to reduce the last term:

$$\sum_t (y_t - \hat{\mu})^t \hat{\Sigma}^{-1}(y_t - \hat{\mu}) = \sum_t tr\left[(y_t - \hat{\mu})(y_t - \hat{\mu})^t \hat{\Sigma}^{-1}\right] \tag{37}$$

$$= tr\left[(\sum_t (y_t - \hat{\mu})(y_t - \hat{\mu})^t)\hat{\Sigma}^{-1}\right] \quad (38)$$

$$= tr\left[N\hat{\Sigma}\hat{\Sigma}^{-1}\right] = Ntr(I) = NM \quad (39)$$

for y_t of dimension M. Combining this result with Equation 36 gives

$$\max_{\mu,\Sigma} \sum_t \log \mathrm{p}(y_t|\mu,\Sigma) = \frac{-1}{2}\left[NM\log(2\pi) + N\log|\hat{\Sigma}| + NM\right]$$

or equivalently the expression in equation 34.

Combining the likelihood of two clusters, as in Section 4.2, gives

$$\begin{aligned}
&\max_{\mu_0,\Sigma_0} \sum_{t=1}^{N_0} \log \mathrm{p}(y_t|\mu_0,\Sigma_0) + \max_{\mu_1,\Sigma_1} \sum_{t=1}^{N_1} \log \mathrm{p}(y_t|\mu_1,\Sigma_1) \\
&\quad = -\frac{1}{2}\left[N_0 M\log(2\pi) + N_0\log|\hat{\Sigma}_0| + N_0 M + N_1 M\log(2\pi) + N_1\log|\hat{\Sigma}_1| + N_1 M\right] \\
&\quad = -\frac{1}{2}\left[NM\log(2\pi) + NM + N_0\log|\hat{\Sigma}_0| + N_1\log|\hat{\Sigma}_1|\right]
\end{aligned}$$

The likelihood test in Section 4.2 uses

$$L^{(p)} = -N_0\log|\Sigma^{(p)}(s_0)| - N_1\log|\Sigma^{(p)}(s_1)|$$

since the other terms are constants that depend only on s and do not change with the different partitionings of the observations in s.

B Alternative Statistics in Split Design

For the case where the observation distributions are assumed to be full covariance Gaussians, it may be more convenient to define the first and second order statistics

$$\mu_j(s) = \frac{1}{N_j} \sum_{t:x_t=x_j, s_t=s} y_t \quad (40)$$

$$\Sigma_j(s) = \frac{1}{N_j} \sum_{t:x_t=x_j, s_t=s} (y_t - \mu_j(s))(y_t - \mu_j(s))^t \quad (41)$$

Again, these statistics are computed once for state s in the initialization step and stored together with counts N_j. In this case, we simplify Equation 22 as follows:

$$\begin{aligned}
&\sum_{t:x_t=x_j} (y_t - \mu(s_0))^t\Sigma(s_0)^{-1}(y_t - \mu(s_0)) \\
&\quad = \sum_{t:x_t=x_j} (y_t - \mu_j(s) + \mu_j(s) - \mu(s_0))^t\Sigma(s_0)^{-1}(y_t - \mu_j(s) + \mu_j(s) - \mu(s_0)) \\
&\quad = \sum_{t:x_t=x_j} (y_t - \mu_j(s))^t\Sigma(s_0)^{-1}(y_t - \mu_j(s)) + N_j(\mu_j(s) - \mu(s_0))^t\Sigma(s_0)^{-1}(\mu_j(s) - \mu(s_0)) \\
&\quad = tr(N_j\Sigma_j\Sigma(s_0)^{-1}) + N_j(\mu_j(s) - \mu(s_0))^t\Sigma(s_0)^{-1}(\mu_j(s) - \mu(s_0))
\end{aligned}$$

where in the second step we used the fact that the cross terms sum to zero, and in the third step we used the identity $z^tAz = tr(zz^tA)$ and the fact that the trace function $tr(\cdot)$ is a linear operator, so

$$\sum_{t:x_t=x_j} (y_t-\mu_j(s))^t\Sigma(s_0)^{-1}(y_t-\mu_j(s)) = tr\left[\sum_{t:x_t=x_j} (y_t-\mu_j(s))(y_t-\mu_j(s))^t\Sigma(s_0)^{-1}\right] = tr\left[N_j\Sigma_j\Sigma(s_0)^{-1}\right].$$

Combining these results with Equation 22, we get the following test

$$\begin{aligned}\log|\Sigma(s_0)| + tr\left[\Sigma_j\Sigma(s_0)^{-1}\right] + (\mu_j(s)-\mu(s_0))^t\Sigma(s_0)^{-1}(\mu_j(s)-\mu(s_0)) \\ \leq \log|\Sigma(s_1)| + tr\left[\Sigma_j\Sigma(s_1)^{-1}\right] + (\mu_j(s)-\mu(s_1))^t\Sigma(s_1)^{-1}(\mu_j(s)-\mu(s_1)). \quad (42)\end{aligned}$$

The parameter re-estimation equations using the sufficient statistics are:

$$\mu(s_k) = \frac{1}{N_k}\sum_{x_j\in A_k}\sum_{t:x_t=x_j} y_t = \frac{1}{N_k}\sum_{x_j\in A_k} N_j\mu_j(s) \quad (43)$$

$$\begin{aligned}\Sigma(s_k) &= \frac{1}{N_k}\sum_{x_j\in A_k}\sum_{t:x_t=x_j}(y_t-\mu(s_k))(y_t-\mu(s_k))^t \\ &= \frac{1}{N_k}\sum_{x_j\in A_k} N_j\left[\Sigma_j(s)+(\mu_j(s)-\mu(s_k))(\mu_j(s)-\mu(s_k))^t\right] \quad (44)\end{aligned}$$

using the same sort of trick as in simplifying the re-partitioning test to get the covariance estimate.

Both the re-partitioning test and the parameter re-estimation equations again simplify if we assume diagonal covariances. For the mean and variance statistics, the re-partitioning test becomes

$$C_0 + \sum_m \frac{\sigma^2_{j,m}(s)+(\mu_{j,m}(s)-\mu_m(s_0))^2}{\sigma^2_m(s_0)} \leq C_1 + \sum_m \frac{\sigma^2_{j,m}(s)+(\mu_{j,m}(s)-\mu_m(s_1))^2}{\sigma^2_m(s_1)} \quad (45)$$

where

$$C_k = \log(\prod_m \sigma^2_m(s_k))$$

as before. The covariance estimate becomes

$$\sigma^2_m(s_k) = \frac{1}{N_k}\sum_{x_j\in A_k} N_j\left[\sigma^2_{j,m}(s)+(\mu_{j,m}(s)-\mu_m(s_k))^2\right] \quad (46)$$

for $m = 1,\ldots,M$.

C Code Changes

Although the new SSS algorithm is in theory a relatively minor change from the baseline algorithm, the actual implementation requires modification of several routines (indicated in italics), as summarized below.

State data structure changes. In the baseline system, two Gaussians are stored for each state. In the SI system, we need to store a single Gaussian for each state, plus some representation of the candidate split. At the very least, we must store a description of the split (variable indicating factor and list indicating elements if the split is in the context domain). In addition, for choosing the best state to split, we must either store the gain associated with that split (a scalar), or the means and variances of the resulting states from that split as well as the current mean and variance since all are require to calculate the gain. (The gain associated with a split is given by Equation 11.) If the means and variances are stored, then they do not need to be recalculated before Baum-Welch re-estimation, but this calculation is probably a relatively small cost. The gain from a particular split can be calculated in *find_splittee_state*, or can be calculated in *split_stateSI* (see below) and stored to avoid some (not much) redundant computation in *find_splittee_state*.

Other data structure changes. As mentioned above, a new data structure must be added in order to save $\gamma_t(s)$ and $\xi_t(s, s')$ for all observation times t, possible states s for that time, and the two possible preceding states s'. This results in additional memory requirements proportional to the amount of training data.

train_HMnet. The baseline routine has the following structure (ignoring memory allocation steps and "DOMAIN_TC" which is 3D-SSS):

1. Initialization using *training* and optionally *viterbi* (Step 1 [1])
2. Find the best state to split *(find_splittee_state)* (Step 2 [1])
3. Find the best split for this state, testing different possible domains (left, right or mid context, temporal split AB, and temporal split BA). Within the context split set-up, test different factors (phone sets) with *split_factor*. (Step 3 [1])
4. Do the split *(load_condition)* and log file print outs
5. Determine states/data affected for limiting Baum-Welch
6. Initialize mixture parameters for new states with *init_param0* (Step 4 [1])
7. Retrain 2-mixture state network using *training* (Step 5 [1]), and optionally run *viterbi*.

Note that *viterbi* is used only to get missing phone boundaries (where labelers left out a boundary because they could not decide) and not to associate data samples with specific states.

This routine needs to change primarily in the order of these steps, which is facilitated by introducing some new and revised subroutines (indicated by "SI" ending):

1. Initialization using
   (a) *training* and *viterbi*
   (b) Find best split factor for each state *(split_stateSI)*.

2. Find the best state to split *(find_splittee_stateSI)*

3. Do the split *(load_condition)* and log file print outs. The new state inherits splittable domains from the previous state.

4. Determine states/data affected for limiting Baum-Welch

5. Retrain single Gaussian state network using *training*, and optionally run *viterbi*.

6. Find best split factor for new states and optionally refine split factors for affected states *(split_stateSI)*.

A new subroutine *(split_stateSI)* is needed which is a combination of step 3 and the subroutine *(init_param0)* in the old version, and of similar complexity. If optional split factor refining is used, then there will be fewer calls to *(split_stateSI)* in SI-SSS than to *init_param0* in SSS, decreasing the computational cost.

find_splittee_state. The baseline version of *find_splittee_state* first computes the divergence between the mixtures for each state (equation (1) in [1], which is equivalent to the likelihood ratio test in [10]), and then loops to find the state with the largest divergence that also has a valid domain to split. Since all "mixtures" for states in the SI version will be associated with valid splits, then this second loop is not needed in *find_splittee_stateSI*. (The domain checking code may be useful in *split_stateSI* though.) The specific implementation of the first loop depends on data structure choices, i.e. whether the divergence is pre-stored along with the mixture components, but the test should be that given in equation 11.

split_factor. This routine is called in *train_HMnet* to choose the best factor (left, middle or center) for splitting in the context domain. The forward algorithm is first called to compute the forward (alpha) probability of being in each of the two new states at each possible time. (Note that this work is redone for the best context split in *cal_total_prob.*) For each factor, these probabilities are accumulated to determine which state an element (phone label) is assigned to by the *split_element* routine, and the total probability for the factor is updated accordingly. Then there are various fixes in case not enough data is assigned to one or the other of the two states. Parts of this subroutine may be useful in the new *split_stateSI* subroutine, depending on whether Baum-Welch or Viterbi clustering is used. The small data "fixes" in this routine will not be needed in SI-SSS since such splits will not be allowable in the restructured code.

split_state2. This routine is called in *train_HMnet* to set up pointers/lists etc. for later evaluating a split in the temporal domain.

init_param0. This routine is called in *train_HMnet* before Baum-Welch training to initialize the distribution parameters (means, variances and mixture coefficients) for the new states. The high level approach is to design a two codeword vector quantizer, training on the means of the different

possible phonetic contexts, and then to use the means that are farthest apart. This approach is very similar to the Chou partitioning algorithm [17], except that uses Euclidean distance rather than a maximum likelihood objective function, and the VQ procedure does not take into account the number of observations associated with the different possible contexts. The specific steps of *init_param0* include: 1) accumulate first and second order statistics for each possible contextual splitting factor (left, right, center phone) and each possible element of that factor (e.g. phone in this context associated with the state to be split), 2) run a 2 codeword VQ on the means of each element for this factor as if it was data to be quantized, 3a) if there is no factor with enough data to split it, then run VQ on the data directly, 3b) else choose the factor that has the largest difference between the resulting VQ means, 4) estimate means, variances and mixture weights based on the clustering for the best case based on the sufficient statistics computed earlier, and 5) initialize transition probabilities to be uniform. This new SI-SSS contextual split routine will be very similar to *init_param0*, with some changes to the VQ routines to implement the maximum likelihood split design algorithm described in Section 4.2.

vq. This routine is called in *init_param0* as described above. It iteratively (1) splits all codewords and (2) runs the LBG algorithm with the minimum distortion criterion for this number of codewords, until the desired number of codewords is reached. In this case, there is only one step since the desired number is two. (For larger numbers of codewords this algorithm is costly – one should do all the splitting first and then run the LBG algorithm.) To implement maximum likelihood clustering, the subroutines used in *vq* need to be changed as follows:

- *distance-ML:* rather than including a vector of means as an argument, include two vectors of the sufficient statistics S_j^1 and S_j^2, which are then used to compute the ML distance:

$$d(x, \mu, \sigma^2) = 2\log(\prod_m \sigma_m^2) + \sum_m \frac{S_{j,m}^2(s) - 2S_{j,m}^1(s)\mu_m + \mu_m^2}{\sigma_m^2} \tag{47}$$

  To save computation, you might pre-compute and pass $\log \prod_m \sigma_m$ as an additional argument.

- *caLcent-ML:* add as an argument an array with the count for each "data" point, N_j, which is needed to compute

$$N_k = \sum_{j:\alpha(x_j)=k} N_j.$$

  Recall that j is the index to the different possible elements (e.g. phone labels). N_k is used both in parameter re-estimation and in computing the overall likelihood for the convergence test.

split_stateSI. The new step 3 should have a routine called for each affected state s that implements the following steps:

1. Check that there is more than some minimum number of observations associated with state s. If not, assign the splitting gain to be zero (unsplittable) and exit.

2. *If this is a new state,* check that there is a valid domain to split state s. (Might use code from *find_splittee_state.*) If not, assign the splitting gain to be zero (unsplittable) and exit. *Else, if this is an old, affected state,* set allowable domains and factors to test according to options outlined in Section 2.

3. Loop over all allowable domains to test

   (a) Check that this state is splittable in this domain

   (b) *If the domain is context* then run *init_param0SI*, which should have the following steps:

      i. Accumulate statistics for this state.

      ii. Run the 2-cluster *vq-ML* routine. If there is a valid split for this domain, it returns means and covariances since these are computed in the clustering.

   (c) *If the domain is temporal* then run new temporal split routine *split_tempSI.* Initialize the state transition probabilities to have self-loop probability at half of that of the original state (to have the same average duration for the two split states together as for the original).

   (d) Test for the domain that gives the highest likelihood gain, using $L^{(p)}$ as defined in the maximum likelihood split design algorithm for both the context domains and the temporal domain.

   (e) Compute $G(S)$ for comparing the temporal and contextual domains, and save for use later in determining the best node to split.

split_tempSI. In the original version of SSS, a temporal split was evaluated by testing the two orderings of the mixture components and measuring their likelihood by running the forward algorithm on the data associated with the original state in the *cal_total_prob* routine. In the new version, we no longer have pre-defined mixture components to test, which means that we need to estimate the distributions but also that we only need to run one test. The new routine to compute a temporal split should first copy the original state so that there are two identical states in a row (with self-loop probabilities chosen so that the expected duration of the two states together is the same as that for the original state), and then run a few iterations of the constrained Baum-Welch algorithm outlined in Section 5, to estimate the new distribution parameters. Running four Baum-Welch iterations is at most four times the cost of the two SSS temporal splits together and should be sufficient to assess the goodness of this domain. Since the cost of the current temporal split evaluation is a relatively small part of the overall SSS algorithm, this increase should not be a problem. Only the distributions of the two new states should be re-estimated, since the overall domain test only considers the gain in going from one to two states.

D Equation Labels in Source Code

Below we provide a list of equation numbers and names of these equations used in source code documentation.

- Equation 10: Contextual split expected likelihood gain
- Equation 11: Temporal split expected likelihood gain
- Equation 28: Re-partitioning test
- Equation 30: Contextual split variance estimate
- Equation 31: Temporal split mean estimate
- Equation 32: Temporal split variance estimate
- Equation 33: Temporal split self loop probability estimate

References

[1] J. Takami and S. Sagayama, "A successive state splitting algorithm for efficient allophone modeling," *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.,* vol. I, 1992, pp. 573-576.

[2] A. Nagai and J. Takami and S. Sagayama, "The SSS-LR Continuous Speech Recognition System: Integrating SSS-Derived Allophone Models and a Phoneme-Context-Dependent LR Parser," *Proc. Int'l. Conf. on Spoken Language Proc.,* pp. 1511–1514, 1992.

[3] A. Nagai and K. Yamaguchi and A. Kurematsu, "ATREUS: A Comparative Study of Continuous Speech Recognition Systems at ATR," *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.,* vol. II, pp. 139–142, 1993.

[4] S. Sagayama and J. Takami and A. Nagai and H. Singer and K, Yamaguchi and K. Ohkura and K. Kita and A. Kurematsu, "ATREUS: a Speech Recognition Front-end for a Speech Translation System, *Proc. European Conf. on Speech Commun. and Technology,* pp. 1287–1290, 1993.

[5] T. Kosaka, S. Matsunaga and S. Sagayama, "Tree-Structured Speaker Clustering for Speaker-Independent Continuous Speech Recognition," *Proc. Int'l. Conf. on Spoken Language Proc.,* pp. 1375-1378, 1994.

[6] J. Takami and T. Kosaka and S. Sagayama, "Automatic Generation of Speaker-Common Hidden Markov Network by Adding the Speaker Splitting Domain to the Successive State Splitting Algorithm," *Proc. Acoust. Soc. Jap.,* pp. 155-156, 1992. (In Japanese.)

[7] L. R. Bahl, P. V. de Souza, P. S. Gopalakrishnan, D. Nahamoo and M. A. Picheny, "Decision trees for phonological rules in continuous speech," in *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.,* 1991, pp. 185-188.

[8] K.-F. Lee, S. Hayamizu, H.-W. Hon, C. Huang, J. Swartz and R. Weide, "Allophone Clustering for Continuous Speech Recognition," *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.,* pp. 749-752, April 1990.

[9] X. Huang, F. Alleva, M.-Y. Hwang and R. Rosenfeld, "An overview of the SPHINX-II speech recognition system," *Proc. ARPA Workshop on Human Language Technology,* 1993, pp. 81-86.

[10] A. Kannan, M. Ostendorf and J. R. Rohlicek, "Maximum Likelihood Clustering of Gaussians for Speech Recognition," *IEEE Trans. on Speech and Audio Proc.,* vol. 2, no. 3, 1994, pp. 453-455.

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

[12] L. Bahl, P. de Souza, P. Gopalakrishnan, D. Nahamoo and M. Picheny, "Context-dependent vector quantization for continuous speech recognition," *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.,* vol. II, pp. 632-635, 1993.

[13] A. P. Dempster, N. M. Laird and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society,* Vol. 37, No. 1, 1977, pp. 1–38.

[14] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis,* J. Wiley & Sons, New York, 1984.

[15] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification and Regression Trees,* Wadsworth International Group, 1984.

[16] L. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer, "A tree-based statistical language model for natural language speech recognition," *IEEE Trans. on Acoust., Speech, and Signal Proc.,* Vol. 37, No. 7, pp. 1001-1008, 1989.

[17] P. A. Chou, "Optimal partitioning for classification and regression trees," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* Vol. 13, No. 4, pp. 340-354, April 1991.

[18] 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.

[19] A. Nádas, D. Nahamoo, M. Picheny and J. Powell, "An iterative "flip-flop" approximation of the most informative split in the construction of decision trees," *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.,* pp. 565-568, 1991.

[20] A. Kurematsu, K. Takeda, Y. Sagisaka, S. Katagiri, H. Kuwabara, and K. Shikano, "ATR Japanese speech database as a tool of speech recognition and synthesis," *Speech Communication,* 9:357–363, 1990.

[21] N. Uratani, T. Takezawa, H. Matsuo and C. Morita, "ATR Integrated Speech and Language Database," ATR Technical Report TR-IT-0056, 1994. (in Japanese)

[22] H. Singer, T. Shimizu, R. Isotani, and T. Takezawa, "Development testsets and administrative tools for ATR's non-read speech databases (SLDB and SDB)," Technical Report TR-IT-0118, ATR, 1995.

[23] H. Singer and J. Takami, "Speech recognition without grammar or vocabulary constraints," *Proc. Int'l. Conf. on Spoken Language Proc.*, pp. 2207-2210, 1994.

[24] S. Matsunaga, T. Kosaka and T. Shimizu, "Speaking-style and speaker adaptation for the recognition of spontaneous dialogue speech," *Proc. European Conf. on Speech Commun. and Technology*, 1995. (to appear)

[25] K. Ohkura, M. Sugiyama and S. Sagayama, "Speaker adaptation based on transfer vector field smoothing with continuous mixture density HMMs," *Proc. Int'l. Conf. on Spoken Language Proc.*, pp. 369-372, 1992.

[26] T. Kosaka and S. Sagayama, "Tree-Structured Speaker Clustering For Fast Speaker Adaptation," *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.*, pp. 570-573, 1994.

[27] T. Kosaka and S. Matsunaga and M. Kuraoka, "Speaker-Independent Phone Modeling Based on Speaker-Dependent HMMs' Composition and Clustering," *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.*, pp. 441-444, 1995.

[28] L. Bahl, P. de Souza, P. Gopalakrishnan, D. Nahamoo and M. Picheny, "Robust methods for using context-dependent features and models in a continuous speech recognizer," *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.*, vol. I, pp. 533-536, 1994.

[29] J. Takami, "Improvement in representing efficiency of hidden Markov networks by a state splitting and merging algorithm," *Proc. Acoust. Soc. Jap.*, pp. 7-8, Fall 1995. (in Japanese)

[30] R. Kuhn, A. Lazarides, Y. Normandin and J. Brousseau, "Improved decision trees for phonetic modeling," *Proc. of the Int. Conf. on Acoust., Speech and Signal Proc.*, pp. 552-555, 1995.