Discriminative Training for Large-Vocabulary Speech Recognition Using Minimum Classification Error

Erik McDermott, *Member, IEEE*, Timothy J. Hazen, *Member, IEEE*, Jonathan Le Roux, Atsushi Nakamura, *Member, IEEE*, and Shigeru Katagiri, *Fellow, IEEE*

***Abstract*—The minimum classification error (MCE) framework for discriminative training is a simple and general formalism for directly optimizing recognition accuracy in pattern recognition problems. The framework applies directly to the optimization of hidden Markov models (HMMs) used for speech recognition problems. However, few if any studies have reported results for the application of MCE training to large-vocabulary, continuous-speech recognition tasks. This article reports significant gains in recognition performance and model compactness as a result of discriminative training based on MCE training applied to HMMs, in the context of three challenging large-vocabulary (up to 100 k word) speech recognition tasks: the Corpus of Spontaneous Japanese lecture speech transcription task, a telephone-based name recognition task, and the MIT JUPITER telephone-based conversational weather information task. On these tasks, starting from maximum likelihood (ML) baselines, MCE training yielded relative reductions in word error ranging from 7% to 20%. Furthermore, this paper evaluates the use of different methods for optimizing the MCE criterion function, as well as the use of precomputed recognition lattices to speed up training. An overview of the MCE framework is given, with an emphasis on practical implementation issues.**

***Index Terms*—Discriminative training, pattern recognition, speech recognition.**

I. INTRODUCTION

MANY techniques exist for improving recognizer performance in the face of difficult conditions present in applications of speech recognition technology, including better feature extraction, pronunciation modeling, acoustic modeling, noise handling, and language modeling [1], [2].

Discriminative training has been used for speech recognition for many years already [3]–[10]. In recent years, the few groups that have had the resources to implement discriminative training for large scale speech recognition tasks have primarily used the maximum mutual information (MMI) framework [11]–[13]. Expanding studies first presented in [14]–[16], we here focus instead on the minimum classification error (MCE) framework for discriminative training [7], [17]–[20]. MCE is directly geared at minimizing word string recognition error, in contrast with MMI, which is targeted at optimizing mutual information between an utterance and the correct string.

The study described in this article generalizes the MCE discriminant function so as to model *sets* of word strings rather than single strings. Weighted finite state transducers (WFSTs) [21] are used to implement this approach. Furthermore, heavy use is made of batch-oriented optimization methods that are suitable to parallelization over many computers [22], [23]—a crucial point given the computation-intensive nature of discriminative training. However, in its essentials, the framework is the same as that adopted in, e.g., [19]. The central point of this article is that this framework is fully capable of yielding significant improvements in recognition accuracy for difficult, large-scale recognition tasks. There are important implementational issues, but these are no more problematic than for the MMI framework. The strong gains in performance described here should lay to rest concerns that MCE could only be used for small-vocabulary or noise-free tasks.

Among many studies, MCE has been evaluated on isolated words, Resource Management, TIMIT, TI digits and SieTill data sets [22], [24]–[27], all rather small or limited by today's standards. The preliminary study presented in [28] evaluated a novel MCE-derived training method on the DARPA Communicator travel reservation task with a training set of 46 h and a recognition vocabulary of 2600 words. In contrast, the recognition tasks examined here have training sets ranging from 39 to 230 h of audio, and vocabularies ranging from 2 k to 100 k words. The tasks examined are: 1) the large-vocabulary Corpus of Spontaneous Japanese (CSJ) lecture speech transcription task [29] that is well known in Japan; 2) a telephone-based name recognition task; and 3) the MIT JUPITER weather information continuous speech recognition task. The TIMIT phone classification task [30] is also used for more detailed comparison of optimization methods than is possible on the larger tasks.

The primary evaluation presented in this article, for each task, is the standard comparison of MCE-trained models (of different size) versus maximum likelihood (ML) baselines. The improved recognition accuracy that results from discriminative training follows the classic MCE versus ML picture [22], but for much more challenging tasks than used in most MCE studies to date. This article also addresses several important issues.

1) Comparison of Optimization Methods: We describe and evaluate different gradient-based optimization methods for minimizing the MCE criterion function. In particular, the performance of straightforward gradient descent is compared against methods known as "Quickprop" and "Rprop." Quickprop uses information about the second derivative, while Rprop

Manuscript received November 8, 2004; revised December 6, 2005. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Li Deng.

E. McDermott, A. Nakamura, and S. Katagiri are with the NTT Communication Science Laboratories, Kyoto 619-02, Japan (e-mail: mcd@cslab.kecl.ntt.co.jp).

T. J. Hazen is with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

J. Le Roux is with the Graduate School of Computer Science, Telecommunications, and Electronics, University of Paris VI, Paris 75005, France, and also with the Graduate School of Computer Science and Technology, Department of Information, Physics, and Computing, University of Tokyo, Tokyo 113–0033, Japan.

Digital Object Identifier 10.1109/TASL.2006.876778

performs dynamic scaling of the update step for each parameter. Even in the context of discriminative training, these methods are not as widespread in the speech recognition community as Baum–Welch-style optimization. This article aims to clarify their use and effectiveness on several tasks.

2) Use of Lattices During MCE Training: We evaluate the use during MCE training of result lattices generated from a previous recognition pass using the full recognition grammar. The lattices correspond to a set of likely candidates for each utterance, for a given initial acoustic model. They can then be loaded in for each utterance during MCE training and used instead of the full recognition grammar, resulting in a substantial speed up. This has been used with the MMI framework for years now [11], [12], [31]. Results for lattice-based MCE training have not yet been reported—though discussion of this can be found in [27]. (For related work, see also [32]). The results presented here show that this method is effective for MCE as well. The task used for this evaluation is a 22 k word telephone-based name recognition task.

3) Strict versus Flexible References (Use of String Sets to Define Correct and Incorrect Recognition): This article explicitly formalizes the notion that the desired correct output for a given utterance is not just a single string of words, but a *set* of strings, all deemed to be correct. The MCE discriminant functions are easily generalized to reflect a set-oriented definition. This approach is evaluated in the context of MCE training on the telephone-based name recognition task, where "flexible" references reflecting the possible insertion of different types of hesitations and sentence completions were compared against the more standard "strict" references, which adhere to the precise contents of the utterance transcription. This use of sets of strings is one way of telling the discriminative training procedure about "don't care" symbols in the utterance transcription. In our implementation, WFSTs are used to model sets of strings in a flexible manner that is well suited to discriminative training.

4) Importance of "Rival" Training Over "Corrective" Training: The experiments described here for the MIT JUPITER telephone-based continuous-speech recognition task include a comparison related to a simple but fundamental point at the heart of effective discriminative training. Two different types of MCE loss function were compared that result in, respectively, "rival training" and "corrective training." The former endeavors, directly or indirectly, to increase separation between correct and top incorrect strings; the latter only does so for *incorrectly* recognized utterances. Note that corrective training occurs with MMI if only the top recognition candidate is used as the denominator lattice, or if no measures (such as "acoustic scaling") are taken to prevent the correct string from dominating the denominator N-best list or lattice [27]. In the experiments described here, a clear benefit for rival training was found. One can view this comparison as one between MCE and a coarse implementation of MMI (i.e., that only used a small N-best list and/or no acoustic scaling).

II. OVERVIEW OF THE MCE FRAMEWORK

MCE first arose out of a broader family of approaches to pattern classifier design known as generalized probabilistic descent (GPD) [7], [17], [20]. The MCE loss function, defined in terms of discriminant functions for each category (in the following, the categories are strings or string sets), is a smoothed approximation of the recognition error rate. It can then be used as the criterion function for optimization [18], [22]. Recent theoretical work has analyzed the link between the smooth MCE loss function and the true classification risk [33], [34]. Through minimization of this criterion function, MCE is aimed directly at minimizing classification error rather than at learning the true data probability distributions, the target of ML estimation via Baum–Welch or Viterbi training. An important point is that the smoothing parameters in the MCE formalism operate not just to enable gradient-based optimization, but also to estimate performance on unseen data [33].

Though the focus of this paper is the application of MCE to the design of acoustic models for speech recognition, many other applications exist. Still within speech recognition, there has been significant work applying MCE to feature extraction [35], [36]. See [37] for related discussion. MCE has also been applied to handwriting recognition [38], image processing [39], document classification [40], and machine translation [41].

A. Contrast With Related Methods

For alternative approaches to discriminative training in the context of speech recognition, the reader is referred to [4]–[6], [8], [9], and in particular to the maximum mutual information (MMI) method [3], [11], [42], which derives from information theory rather than decision theory.

A precise theoretical characterization of the asymptotic properties of rival approaches to discriminative training is beyond the scope of this article, but some loose observations can be made. The inability of ML estimation, under incorrect modeling assumptions, to find a separating solution even when it exists has been illustrated for very simple classification scenarios [22]. It has been shown that in some cases (again, where the modeling assumptions are incorrect) MMI can outperform ML [3]. However, cases have also been found showing that the parameter set optimizing MMI does not correspond to the optimal classifier [43]. In contrast, by definition, MCE approximates classification performance regardless of the modeling assumptions. Via control of the smoothing parameters, the approximation of classification performance can be made arbitrarily precise. Therefore, the parameter set optimizing the MCE criterion can in principle correspond to optimal classification performance.

The recently proposed minimum phone error (MPE) approach [45], [46] shares features with both MCE and MMI. MPE too has a smoothing parameter which can make the criterion correspond strictly to classification accuracy, but using a different formulation than MCE. MPE has the advantage of explicitly modeling phone or word accuracy, whereas MCE typically only models string accuracy.[1] In practice, it may be more accurate to view MPE as a *model-based estimate* of recognition accuracy: model-based posterior probabilities are explicitly used to weight phoneme or word accuracy. In this sense it may be more limited by modeling assumptions than MCE, which does not require accurate posterior probabilities

[1]Proposals for incorporating word accuracy into the MCE framework can be found in [22].

to estimate recognition error. However, the analysis of the links between MCE and MPE is still in its early stages.

B. HMM-Based String-Level Discriminant Function

The string-level MCE formalism [19] defines *discriminant functions* for *string categories*. These are taken to refer to specific sequences of phones or words corresponding to the transcription of a speech utterance. Representing a speech utterance belonging to string S_j as a sequence of feature vectors $\mathbf{x}_1^T = (\mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$, the functional form of the joint log-probability of $\mathbf{x}_1^T$, and the best Viterbi state segmentation $\Theta^j = (\theta_1^j, \ldots, \theta_t^j, \ldots, \theta_T^j)$ are used to define the HMM discriminant function for S_j [19]

$$\begin{aligned} g_j\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right) &= \log P(S_j) + \log P_{\mathbf{\Lambda}}\left(\mathbf{x}_1^T, \Theta^j\right) \\ &= \log P(S_j) + \sum_{t=1}^{T} \log a_{\theta_{t-1}^j \theta_t^j} \\ &\quad + \sum_{t=1}^{T} \log b_{\theta_t^j}\left(\mathbf{x}_t\right). \end{aligned} \tag{1}$$

Here, a_{uv} denotes a state transition probability and $b_s(\cdot)$ denotes a Gaussian mixture used to model the observation probability of a feature vector in state s. $\mathbf{\Lambda}$ denotes the entire set of system parameters; here it consists of the transition probabilities and the means, covariances and mixing weights used to define $b_s(\cdot)$. $P(S_j)$ denotes the prior probability of string S_j, modeled using a language model that is left open for now.

C. Weighted Finite-State Transducer (WFST)-Based Generalization of MCE Discriminant Function to String Sets

In this paper, the definition of MCE discriminant functions is generalized from strings to *string sets*. The main benefit of using string sets rather than just strings is that they offer a model of transcription uncertainty. Such uncertainty arises in many practical situations, for example, where the presence or absence of a word (e.g., interword silence) is in doubt, or where a word has several possible phonetic realizations. Including all possible variations into a string set model of the utterance is one way to represent this uncertainty. The task of finding the best realization for a given acoustic input is then given to the decoding process.

In the context of discriminative training, string sets can also be designed to model "don't care" variations around target words. This idea is evaluated in one of the experiments described in this article.

A natural choice for representing and using such string sets is to use WFSTs [21]. Informally, a WFST represents a mapping from input to output symbol sequences. It consists of a set of nodes connected by arcs; each arc is associated with input and output symbols, and possibly scores that are accumulated as the WFST is traversed from start to end nodes. In the context of speech recognition, the input symbols are typically HMM-based phone models (or HMM states themselves), the output symbols are typically words, and the output scores are typically log probabilities derived from a language model. The reader is referred to [21] for detailed discussion of WFSTs and their application to speech recognition. Time-synchronous beam search [46] can be used to find the best path through a WFST for a given acoustic input, generating the acoustic and language model scores, as well as the state segmentation information, necessary for the discriminant function. The use of WFSTs in the overall training scheme is illustrated in Fig. 1. The following sections detail this approach.

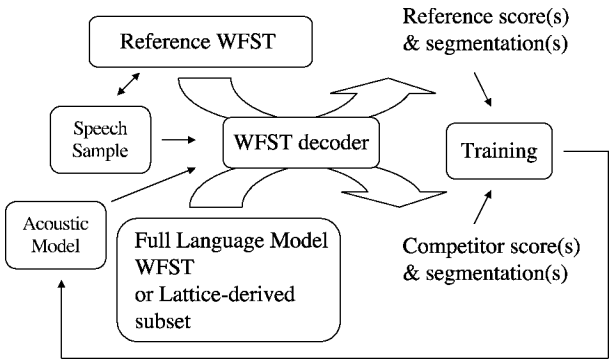


Fig. 1. MCE training using weighted finite-state transducers.

D. Discriminant Function for Correct String Set

In the following, each training utterance $\mathbf{x}_1^T$ is taken to belong to a string set, $\mathcal{K}$, represented using a *reference WFST* modeling the utterance content. The MCE discriminant function (1) for the correct string S_k is then replaced by a discriminant function for the correct string set $\mathcal{K}$

$$g_{\mathcal{K}}\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right) = \max_{k|S_k \in \mathcal{K}} \log P_{\mathbf{\Lambda}}\left(\mathbf{x}_1^T, \Theta^k\right) \tag{2}$$

with the search for the best string S_k within $\mathcal{K}$ being performed by the WFST decoder.

Effectively, each reference WFST is a miniature recognition grammar, complete with language model scores and word pronunciations, modeling the possible set of strings taken to be correct for the corresponding utterance. An example of a reference WFST is shown in abbreviated form in Fig. 2.

E. Discriminant Function for Incorrect String Set

As long as the same cost of 1 is assigned to recognizing strings not in the correct set, there is no need to distinguish incorrect strings from each other. They can all be lumped into a single incorrect string set $\mathcal{J}$. This set is simply the complement of the correct string set $\mathcal{K}$, within the overall set $\mathcal{G}$ allowed by the grammar for the given task, i.e., $\mathcal{J} = \mathcal{G} - \mathcal{K}$. The corresponding discriminant function can be defined as

$$g_{\mathcal{J}}\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right) = \max_{j|S_j \notin \mathcal{K}} g_j\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right) \tag{3}$$

which can be seen as the limiting case of a more general expression

$$g_{\mathcal{J}}\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right) = \log\left[\frac{1}{|\mathcal{J}|} \sum_{j|S_j \notin \mathcal{K}}^{|\mathcal{G}|} e^{g_j\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right)\psi}\right]^{\frac{1}{\psi}} \tag{4}$$

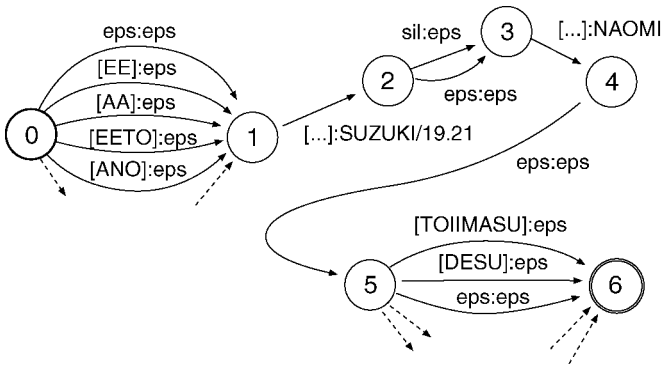


Fig. 2. Reference WFST modeling the name "Suzuki Naomi" as well as possible hesitations ("ee," "aa," "ano," and "eeto"), silence insertion, and utterance completions ("toiimasu" and "desu"). Arc symbols to the left of the colon are input symbols, those to the right are output symbols; language model scores, if any, are shown after a '/'; "eps" refers to epsilon, the null symbol. Symbols in brackets refer to the phonetic realization of a given word.

where $|\mathcal{G}|$ denotes the total number of strings permitted by the recognition grammar and $|\mathcal{J}|$ the total number of incorrect strings.[2] Here the parameter ψ controls the extent to which the top incorrect strings dominate the bracketed part of the expression. Small values of ψ can be used to "unweight" the top incorrect strings (in a manner similar to "acoustic scaling" used in MMI studies [11]), which may help generalization. Note that a similar "soft-max" definition could be used instead of (2) for the correct string. In practice, depending on the value of ψ, an "N-best" approximation of (4) can be used. A more sophisticated approximation is to use lattices [27], [47].

As for the correct string set, the incorrect string set can be represented by a WFST, or more practically, by the WFST representing the overall task grammar $\mathcal{G}$ followed by a filtering step after decoding.

F. Misclassification Measure and Loss Function

The MCE *misclassification measure* compares the discriminant function value for the correct string set $\mathcal{K}$ with that for the incorrect string set $\mathcal{J}$

$$d_{\mathcal{K}}\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right) = -g_{\mathcal{K}}\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right) + g_{\mathcal{J}}\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right). \tag{5}$$

This can be based on either (4) or (3), which only requires the top incorrect string set. Note that the sign of the misclassification measure reflects the correctness or incorrectness of the classification decision for training token $\mathbf{x}_1^T$. The correspondence is exact when using (3).

G. MCE Loss Function

A typical choice of MCE loss function is the sigmoid

$$\ell\left(d_{\mathcal{K}}\left(\mathbf{x}_1^T, \mathbf{\Lambda}\right)\right) = \ell(d_{\mathcal{K}}) = \frac{1}{1 + e^{-\alpha d_{\mathcal{K}}}} \tag{6}$$

[2]As $|\mathcal{J}|$ disappears when taking the derivative of (4) with respect to $\mathbf{\Lambda}$, it does not have to be calculated.

where the abbreviation $d_{\mathcal{K}} = d_{\mathcal{K}}(\mathbf{x}_1^T, \mathbf{\Lambda})$ is used for clarity. Clearly, when the misclassification measure is positive, the loss function will be close to 1; when it is negative, it will be close to 0. This behavior depends on the steepness of the loss function, here controlled by the positive scalar α.

Another choice is the "chopped" sigmoid

$$\ell(d_{\mathcal{K}}) = \begin{cases} 0, & d_{\mathcal{K}} < -|Q| \\ 1/(1 + e^{-\alpha d_{\mathcal{K}}}), & \text{otherwise.} \end{cases} \tag{7}$$

Here, Q (a scalar) acts as a margin, beyond which a correctly recognized token incurs no loss. For some tasks, this has been found to make the optimization process easier. However, use of this loss function introduces another parameter to tune, Q.

Other choices for the loss function are possible, including piecewise linear functions [24], [33]. Regardless of the loss function used, the steepness parameter(s) must be chosen for each task. In particular, the range of the misclassification measure (5) is highly task-dependent. Normalizing (5) by the number of frames in each utterance can help the tuning process. Simple heuristics can be used to automate the choice of steepness parameters. For instance, one can choose α such that $\ell(\sigma_d) = 0.9$, where σ_d is the standard deviation of $d_{\mathcal{K}}$ on the training set. For (7), one can set $Q = \sigma_d/2$.

A convenient choice that does not require tuning steepness parameters is the simple linear loss function, $\ell(d_{\mathcal{K}}) = d_{\mathcal{K}}$. This was used in some of the experiments described in the following. For this choice, of course, the 0–1 approximation of classification performance is lost. Discriminative training based on such a loss function is then directed at increasing overall separation between correct and top incorrect categories. In the context of the JUPITER experiments, a comparison between the linear loss and the sigmoid loss is presented.

Appendix A details the gradient of the MCE loss function with respect to the modifiable system parameters, typically the Gaussian means, covariances, and mixing weights.

III. Optimization Methods

Given the loss gradient (see Appendix A) the empirical MCE loss (i.e., the loss function summed over the training set) can be minimized using several different approaches to optimization [17], [22], [48]–[50]. In this section several gradient-based optimization methods are described.

A. Online and Batch Probabilistic Descent

Probabilistic descent (PD) [51] is a very simple and remarkably effective online optimization method. It consists in computing the gradient of the loss function for each training token $\mathbf{X}_n$ and updating parameters in the opposite direction, by a proportion determined by a learning rate ϵ_n that decreases as the token presentation index n increases

$$\mathbf{\Lambda}^{(n+1)} = \mathbf{\Lambda}^{(n)} - \epsilon_n \nabla_{\mathbf{\Lambda}} \ell\left(d_{\mathcal{K}}\left(\mathbf{X}_n, \mathbf{\Lambda}^{(n)}\right)\right). \tag{8}$$

Here, $\mathbf{X}_n$ denotes a generalized pattern token, that could be either a single feature vector or a sequence of such feature vectors, and $\mathbf{\Lambda}^{(n)}$ denotes the set of model parameters at iteration n.

The power of such online algorithms is that they exploit redundancies in the data, allowing learning to proceed very quickly [52]. However, online algorithms cannot be parallelized using the straight-forward data-parallelism approach that applies to batch algorithms, where different processors can accumulate gradient information for the same model over subsets of the training data before each model update. This means that although PD converges quickly in terms of required number of iterations, in practice it suffers compared to parallelizable batch algorithms, which may require more iterations, but for which each iteration proceeds much more quickly. On the other hand, the purely batch version of PD typically converges slowly.

One approach is to use a compromise between strictly batch and strictly online modes, and update the model every n tokens. Parallelization then becomes possible. In the following, this approach is referred to as "semibatch." One can also consider a hybrid of semibatch and batch, i.e., an evolving update period which starts small but grows eventually to include the entire training set.

The proper setting of the initial learning rate ϵ_0 is important. A simple heuristic is to use the largest initial value that does not lead to unstable learning. In the experiments described here, ϵ_n decreases linearly from ϵ_0 to 0 over the course of N iterations.

B. Quickprop

Quickprop [53] is a simple batch-oriented second-order optimization method loosely based on the classic Newton's method. Quickprop was first applied to MCE-based optimization in [22] and [23]; for an application to MMI, see [54]. It can be run in batch mode and is therefore suitable for parallelization over multiple computers.

Quickprop can be seen as a rough approximation to Newton's method. The central idea in Newton's method is to build a quadratic approximation $M(\cdot)$ of the function of interest $F(\cdot)$, using the first three terms of the Taylor series expansion of the function, around the current point $\mathbf{\Lambda}$, for a given step $\mathbf{s}$, and then to solve for the step $\mathbf{s}^N$ that leads to a point where the gradient of the model is zero, i.e., $\nabla M(\mathbf{\Lambda}+\mathbf{s}^N)=0$. The solution $\mathbf{s}^N$ is the Newton step

$$\mathbf{s}^N = -\left(\nabla^2 F(\mathbf{\Lambda})\right)^{-1}\nabla F(\mathbf{\Lambda}). \tag{9}$$

The optimization procedure is to update the parameter vector $\mathbf{\Lambda}$ at iteration p according to $\mathbf{\Lambda}^{(p)} = \mathbf{\Lambda}^{(p-1)} + \mathbf{s}^N$. In general, if the Hessian matrix is positive definite, and the initial value of $\mathbf{\Lambda}$ is sufficiently close to the optimum, Newton's method converges rapidly to a local minimum of the criterion function [55]. However, there is usually no guarantee that the Hessian is positive definite and properly conditioned. Furthermore, the size of the Hessian—the square of the length of the parameter vector—means that the true Hessian cannot be represented in practice.

The latter problem is addressed in Quickprop by the use of a diagonal approximation of the Hessian

$$\frac{\partial^2 F\left(\mathbf{\Lambda}^{(p)}\right)}{\partial \lambda_i^2} \approx \frac{\frac{\partial F\left(\mathbf{\Lambda}^{(p)}\right)}{\partial \lambda_i} - \frac{\partial F\left(\mathbf{\Lambda}^{(p-1)}\right)}{\partial \lambda_i}}{\Delta\lambda_i^{(p-1)}} \tag{10}$$

where $\mathbf{\Lambda}^{(p)}$ denotes the parameter vector $\mathbf{\Lambda}$ at iteration p, λ_i the ith component of $\mathbf{\Lambda}$ and $\Delta\lambda_i^{(p-1)}$ the ith component of the update step $\mathbf{s}$ at iteration $p-1$. Clearly, this is a rough approximation, which is strictly accurate only for a small update step, or a quadratic $F(\mathbf{\Lambda})$. However, the point is not to estimate the Hessian accurately so much as to obtain some information about the Hessian and use it to move closer to the function optimum.

Quickprop addresses the positive definiteness of the approximated Hessian in the following manner. The sign of the gradient with respect to each parameter is examined for successive iterations. If the sign is the same, the Hessian is considered insufficiently positive, and the simple gradient, multiplied by a learning rate ϵ is added to the Newton step when computing the parameter update

$$\mathbf{s} = -\left[\left(\nabla^2 F(\mathbf{\Lambda})\right)^{-1} + \epsilon\right]\nabla F(\mathbf{\Lambda}). \tag{11}$$

If the sign is different, the Hessian is considered sufficiently positive—a minimum is considered likely to exist between the preceding and current parameter values—and the approximated Hessian is used by itself in computing the update step [i.e., according to (9)]. The method is therefore a compromise between Newton's method and simple gradient descent.

There are yet additional controls on the update step. If the update step for a particular parameter is larger than a task-dependent limit, or greater than μ times the previous update step, it is scaled back to μ times the previous update step. Finally, if the update step has the same sign as the current gradient (i.e., it is pointing uphill), or if it is close to zero in magnitude, the step reverts to that suggested by the simple gradient. A pseudocode description of the core of the algorithm in our implementation is shown in Fig. 3. The proper setting of ϵ is important, and will be discussed in relation to some of the experiments described in the following. In general, we have not found it difficult to find effective values. In all the experiments, the task-dependent limit was set at 10.0 and μ at 1.75.

Many variations around this scheme are possible. Furthermore, several related "modified Newton's methods"[3] exist that address the size, positiveness, and proper estimation of the Hessian, as well as the proper scaling of the update step, with much more sophisticated techniques [55], [56]. However, Quickprop is easy to implement and quite effective in practice, as the results in the following demonstrate.

C. Rprop

Rprop [57], which stands for "Resilient back-propagation," is a batch optimization algorithm well-known in the field of artificial neural networks. Its basic principle is to eliminate the possibly harmful influence of the size of the partial derivative

[3]For example, BFGS methods.

```
## Quickprop - inner loop

FOR i = 1..L,      # Loop over parameter vector

  dlambda = dlambda_vec[i]                  # get first derivative, time t
  dlambda_old = dlambda_old_vec[i]          # get first derivative, t-1
  lambda_update = lambda_update_vec[i]      # get size of last update step

  # Calculate (approximate) diagonal second derivative
  d2lambda = (dlambda - dlambda_old) / lambda_udpate

  # Calculate modified Newton step
  gradient1_part = -epsilon * dlambda
  IF (d2lambda > 0):
    gradient2_part = -dlambda/d2lambda
    IF (dlambda_old * dlambda > 0):  # do gradients point the same way?
      delta = gradient1_part + gradient2_part
    ELSE:
      delta = gradient2_part
  ELSE:
    delta = gradient1_part;

  # Limit absolute update step size
  IF ((ABS(delta) > ABS(mu * lambda_update))
      OR
      (ABS(delta) > TASK_LIMIT)):
    delta = SIGN(delta) * ABS(mu * lambda_update)

  # If going uphill, or update step is near zero, use simple gradient
  IF (((delta * dlambda) > 0.0)
      OR
      (ABS(delta) < TINY)):
    delta = gradient1_part

  lambda_update_vec[i] = delta  # new update step for i-th component
```

Fig. 3. Inner loop of the Quickprop algorithm. First and second derivatives are assumed to have been previously calculated. The algorithm balances parameter updates suggested by an approximated (second-order) Newton's method with those suggested by simple (first-order) gradient descent.

on the update step. As a remedy, only the sign of the derivative is used, determining the direction, but not the magnitude, of the update $\Delta\lambda_i^{(p)}$ for the pth iteration

$$\Delta\lambda_i^{(p)} = \begin{cases} -\Delta_i^{(p)}, & \text{if } \frac{\partial F\left(\mathbf{\Lambda}^{(p)}\right)}{\partial \lambda_i} > 0 \\ +\Delta_i^{(p)}, & \text{if } \frac{\partial F\left(\mathbf{\Lambda}^{(p)}\right)}{\partial \lambda_i} < 0 \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

The magnitude $\Delta_i^{(p)}$ of the update is different for each parameter (though initialized uniformly at a user-determined $\Delta^{(0)}$), and evolves according to a very simple adaptation rule

$$\Delta_i^{(p)} = \begin{cases} \eta^+ \cdot \Delta_i^{(p-1)}, & \frac{\partial F\left(\mathbf{\Lambda}^{(p-1)}\right)}{\partial \lambda_i} \cdot \frac{\partial F\left(\mathbf{\Lambda}^{(p)}\right)}{\partial \lambda_i} > 0 \\ \eta^- \cdot \Delta_i^{(p-1)}, & \frac{\partial F\left(\mathbf{\Lambda}^{(p-1)}\right)}{\partial \lambda_i} \cdot \frac{\partial F\left(\mathbf{\Lambda}^{(p)}\right)}{\partial \lambda_i} < 0 \\ \Delta_i^{(p-1)}, & \text{otherwise} \end{cases} \tag{13}$$

where $0 < \eta^- < 1 < \eta^+$.

The motivation for the procedure is as follows. Each time the derivative with respect to a parameter changes sign, it is regarded as an indication that the last update was too large and that it jumped over a minimum. The update value is thus decreased by a factor η^-. If the sign of the derivative does not change, this is interpreted as an indication that the parameter is in a shallow region of the error surface, and the update value is increased in order to speed up convergence. A common choice for the update parameters is $\eta^- = 0.5$ and $\eta^+ = 1.2$.

It is possible for the standard version of the algorithm to skip over local minima. We thus implemented two versions of the Rprop algorithm. The first one ("Standard Rprop") is the original version described by (12) and (13). The second one is a modified version of "Rprop with weight-backtracking" described in [57]. It consists in not allowing update step adaptation in the iteration following an update step decrease. The "weight-backtracking" version turned out to be the most effective, and is the one used in the experiments described in the following.

In addition to using Rprop in batch mode, we can also consider semibatch versions, as well as hybrids of semibatch and batch mentioned earlier (i.e., the use of an evolving update period which starts small but grows eventually to include the entire training set).

D. Extended Baum–Welch Algorithm

Recent work has used reestimation style optimization [58] similar to that used for MMI to minimize the MCE criterion function [27], [47]. In particular, [47] reports results for a comparison between MCE, MPE, and MMI evaluated on the Wall Street Journal task. This work also uses a lattice update rather than the N-best approach adopted in this article, emphasizing the point that the issue of whether to use the Extended Baum–Welch algorithm, or a lattice update, is independent from the choice of criterion function.

IV. TIMIT Phone Classification

Though far from being a large-scale speech recognition task, the TIMIT phone classification task [30] is a useful benchmark for preliminary experiments. In particular, we used TIMIT to examine the convergence of different MCE optimization algorithms, and analyze the sensitivity of Quickprop to the learning rate ϵ.

A. Database, Feature Extraction, and Baseline HMM

The standard TIMIT training set (3696 utterances) and "core" test set (192 utterances) were used. The feature vectors used consist of 39 Mel Frequency Cepstral Coefficients (MFCCs), deltas, and delta–deltas using a 25-ms window and a 10-ms shift rate. Though 48 phones are modeled, the common procedure of mapping these to 39 phones during testing was followed [59]. Each phone was modeled using a three-state HMM with eight Gaussians per state.

B. Comparison of Optimization Methods

The optimization methods described in Section III were used to minimize a phone-level MCE loss function (7), defined for labeled speech segments. After some tuning of learning rate parameters, each optimization method was run for 44 iterations, where one iteration corresponds to a full presentation of the training set. This is considerably more iterations than is necessary for TIMIT, but helps clarify the convergence of the respective algorithms. All versions of PD—that is, online, semibatch, and batch—were run with a learning rate that decreased linearly to zero over the course of the 44 iterations. The initial value of the learning rate was tuned independently for all optimization methods. Semibatch PD was run with a model update period of

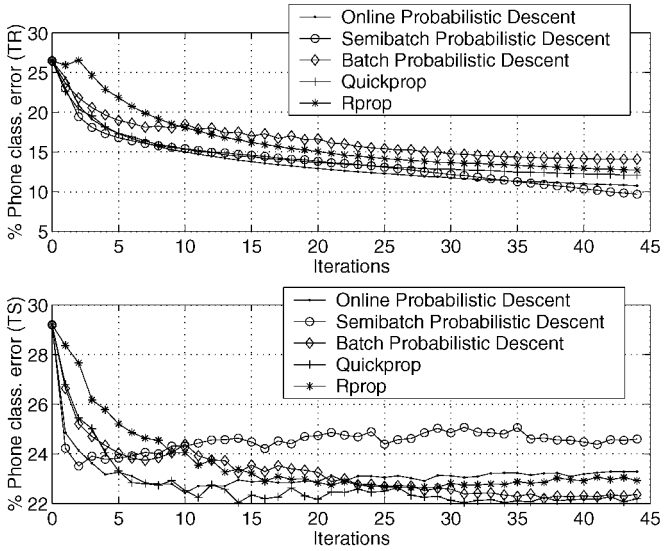


Fig. 4. Course of training for different optimization methods: evolution of phone classification performance measured on TIMIT training set (top) and TIMIT core test set (bottom).

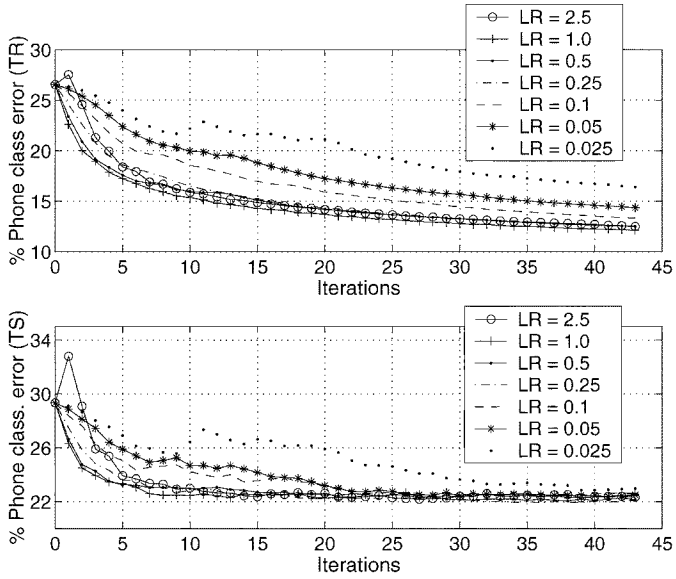


Fig. 5. Course of training for Quickprop run with different values of the learning rate, ϵ: evolution of phone classification performance measured on TIMIT training set (top) and TIMIT core test set (bottom).

TABLE I
PHONE CLASSIFICATION ERROR RATES ON THE TIMIT CORE TEST SET FOR DIFFERENT OPTIMIZATION METHODS

ML	PD	SB-PD	B-PD	Quickprop	Rprop
29.2%	22.4%	23.5%	22.2%	22.0%	22.7%

15 utterances. As described in Section III, Rprop dynamically adjusts its step size for each parameter, given a starting learning rate, while Quickprop uses a fixed learning rate.[4]

Fig. 4 shows training and test set phone classification error rates after each iteration for each method. Error rates for iteration 0 correspond to the maximum likelihood/viterbi training baseline. Note that plotting the MCE loss function itself, rather than phone classification error, would yield very similar graphs, since for this experiment the 0–1 MCE loss function used was defined at the phone segment level. The best test set classification error rates for each method over all iterations are shown in Table I ("ML": maximum likelihood baseline; "SB-PD": semibatch PD; "B-PD": batch PD). These results correspond to the best we could do if we were to use a development set to select the model to use on the core test set. The best result obtained, 22.0% classification error for Quickprop, is a significant improvement over the maximum likelihood baseline model's performance on the core test set, 29.2%, corresponding to a relative error rate reduction of 25% for this model size.

C. Investigation of Sensitivity to Learning Rate

An additional set of experiments on TIMIT examined the effect of running Quickprop with different learning rates. Quickprop was run for 44 iterations for a range of learning rates, from 20.0 to 0.0025. Fig. 5 illustrates the results on both training and testing sets for the range 2.5 to 0.025. With $\epsilon = 2.5$, performance degrades at first but eventually recovers. With $\epsilon = 0.05$, learning is slow, but eventually attains performance similar to that for the other choices of ϵ. Setting $\epsilon \geq 5.0$ led to performance degradation with no subsequent recovery, while $\epsilon \leq 0.025$ led to overly slow and increasingly unstable learning. The latter observation is consistent with the discussion in Section III-B: a small ϵ means the algorithm will rely more on the estimated Hessian, which can be dangerous.

Though not detailed here, the other optimization methods are similar to Quickprop in their behavior with respect to the learning rate: a learning rate that is too small typically leads to very slow and ineffective learning, a learning rate that is too large leads to instability. However, it is not hard to find a range of effective values.

D. Summary and Discussion

Several optimization methods were applied to HMM design using the MCE criterion function on the TIMIT phone classification task. The main result is that the MCE criterion function can be optimized quite effectively using several different methods. Furthermore, these methods are not particularly sensitive to the choice of learning rate. This was examined in detail for the Quickprop method, where a broad range of learning rates yielded reasonable convergence.

Examining the evolution of performance on the training set, we see that online and semibatch PD converge much more rapidly than batch PD. The differences between the other methods are small. Performance differences on the test set are also small, with the exception of semibatch PD, which does not perform as well as the other methods. In light of the effectiveness of semibatch PD on the training set, this may be an instance of over-training.

A slight edge in test set performance was observed for Quickprop. The classification error rate of 22.0% obtained for Quickprop is the best result known to us for a standard HMM on this well-known task [23], [60]–[62]. Note, however, that this result was selected with direct reference to the test set; strictly

[4]Experiments giving Quickprop a decreasing learning rate did not suggest any benefit to this.

speaking, it should be described as an upper-bound on performance in the scenarios investigated. Only one model size was investigated; better performance may result from MCE training applied to smaller or larger models. However, the point of these experiments was not to get the best possible performance on TIMIT, but rather to investigate the performance of the different optimization methods.

V. Corpus of Spontaneous Japanese Lecture Transcription

The Corpus of Spontaneous Japanese (CSJ) lecture speech transcription task has recently been used as a standard benchmark by several groups in Japan [29]. The data consists of audio and corresponding text data collected from lectures at science and engineering conferences in Japan. The speaking style observed in the lectures ranges from a formal reading style to casual and spontaneous styles of speech. The test set and test reference transcriptions are standard for this task. Here, we report recognition results using two trigram language models (with back-off bigram and unigram models), using vocabularies of 30 k and 100 k words, respectively.

The primary aim of the experiments described here was to evaluate the effectiveness of MCE training in improving the performance of baseline ML models in the context of a large vocabulary continuous speech recognition task. The first set of experiments used a single baseline ML model to compare different optimization methods when applied to minimizing the MCE loss function, along the lines of the comparison presented in the previous section for the TIMIT phone classification task. The second set of experiments evaluated the use of several different baseline model sizes and topologies, with just one optimization method, Quickprop.

A. Target Language Models

The first trigram language model WFST used in our CSJ recognition tests is based on a vocabulary of 30 000 words, uses Witten–Bell back-off smoothing, and has a perplexity of 70.6. The out-of-vocabulary rate for the test set given this language model is 2.28%. The second trigram language model is based on a vocabulary of 100 000 words, uses Kneser–Ney back-off smoothing, and has a perplexity of 75.3. This language model leads to a test set out-of-vocabulary rate of 1.36%.

B. Database

1) Training Set: The CSJ A set (male and female), consisting of about 186 k utterances was used for MCE training. This amounts to approximately 230 h of audio, corresponding to 953 lectures, each from a different lecturer.

2) Test Set: The data set used for testing is the standard CSJ test set, consisting of 10 lecture speeches, each from a different speaker, comprising 130 min of audio in total.

C. Setting Up Reference WFSTs

As described in Section II, the MCE discriminant function (2) for the correct string set is based on a WFST model of the utterance. The starting point for generating such WFSTs is simply the word sequence of the utterance transcription. Variations around that (optional insertion of silence between words, inclusion of "don't care" words, etc.) can then be considered. Since the discriminant function includes language model scores, these must be incorporated into the reference WFST too.[5]

On the CSJ task, setting up such WFSTs was much harder than it might appear. The primary reason for this is that there is no standard definition of the word unit in Japanese. As a result, there was large mismatch in character segmentations found in the utterance transcriptions and those found in the text data used to train the language models. The CSJ transcriptions originally available to us used a coarse grain of decomposition, segmenting the transcription kanji and kana content into units closer to phrases than to words. In contrast, the standard trigram language models were trained on text data using a much finer, word-like grain of segmentation. For effective discriminative training, it is important that what is given as the correct, desired output of the recognizer be consistent with the language model used. Our approach was to resegment the utterance transcriptions using the same rules as used in the creation of the trigram language models. A WFST was then created for each resegmented transcription.

These WFSTs start out very simple, representing just the transcription word sequence, with the optional insertion of silence between words. Standard WFST techniques were then applied. The reference WFSTs were composed with a dictionary WFST representing word pronunciations, and then with a triphone loop WFST mapping context-independent phone sequences into context-dependent ones. These steps are all performed offline; for every utterance, the MCE training module then must read in a reference WFST from disk.

D. Unigram Language Models for MCE Training

An important choice is that of the language model to use during MCE training. This will be used to compute the discriminant function for the competing incorrect string set [(3) or (4)].

Though the target language models are 30 k and 100 k word trigrams, there are reasons to consider using different language models during training. First, many words in the training set are not covered by the 30 k word trigram. The system has no chance of recognizing these words, and asking the discriminative training process to try to correct the ensuing errors by modifying the acoustic model is not appropriate. Second, training with a full trigram may simply be too slow to permit more than a few experiments. Third, discriminative training using a less constrained grammar can in some cases generalize better to unseen data [11].

In the first set of CSJ experiments (focusing on optimization methods) described in this study, the approach was to create a 48 000 word unigram WFST covering all the words in the speech training set, as well as all the words in the 30 k word trigram used for testing. The many homonyms found in the CSJ database were merged, as word writings and part-of-speech tags

[5]If one uses a linear loss function instead of the sigmoid, the language model scores for the transcription do not affect the MCE gradient and so are not necessary.

do not benefit recognition using a unigram. Using common techniques to improve WFST compactness, the final unigram WFST only contains around 80 000 arcs. In these experiments, only the 30 k trigram was used for testing.

In the second set of CSJ experiments (focusing on model size), the approach was to use a unigram based on all words in the CSJ language model text training set, again merging all homonyms into single word entries. This resulted in a 68 000 word unigram WFST, containing around 110 000 arcs. This unigram too covers all words in the speech training set.[6] In these experiments, both the 30 k and 100 k trigram language models were used for testing.

For both unigram WFSTs, time-synchronous beam search through the WFST using the SOLON decoder [46] is fast enough that MCE training could be carried out without resorting to lattices to speed up computation.

E. Feature Extraction and Baseline HMMs

The feature vectors consist of 38 or 39 components: MFCCs (after cepstral mean subtraction), deltas, and delta–deltas, computed every 10 ms for 25-ms windows over the speech signal. In the first set of CSJ experiments, described in Section V-H, the static energy component was removed after computation of the corresponding delta and delta–delta components, resulting in a 38 component feature vector. Triphones are modeled using phonetic decision tree-based clustering [63]. The triphone set is based on 43 base phones that include utterance initial/final silence and within-utterance silence. Each phone is modeled using three state positions, in a linear arrangement. After tree construction, the number of Gaussians was increased iteratively by performing Viterbi training on the training set and splitting the Gaussian with the largest mixing weight [64].

The following acoustic model topologies were used:

- 2000 states, 16 and 32 Gaussians per state (for totals of 32 k and 64 k Gaussian pdfs, respectively);
- 3000 states, eight and 16 Gaussians per state (totals of 24 k and 48 k Gaussian pdfs, respectively);
- 4000 states, 16 Gaussians per state (a total of 64 k Gaussian pdfs).

F. Language Model Weight During Testing

The language model weight should be tuned appropriately when evaluating acoustic models. The approach adopted here was to try different weights, in steps of 1.0, on the entire test set.[7] The optimal weight after discriminative training ranged from 8.0 to 12.0, compared to an optimal weight of 13.0 to 15.0 for the baseline ML models.

G. Beam Size During Testing

For the results presented here, the beam threshold was adjusted to result in similar amounts of CPU time for each model tested, maintaining an overall CPU time of about six times real time for each test. Previous results showed that few search errors occur for beams of this size or larger.

[6]The speech training set corresponds to transcriptions that comprise a subset of the overall CSJ language model text training data.

[7]There is as of yet no standard development set for the CSJ task.

TABLE II
WORD ERROR RATES (%) FOR DIFFERENT MCE OPTIMIZATION METHODS ON THE CORPUS OF SPONTANEOUS JAPANESE AND RELATIVE ERROR RATE REDUCTIONS (%) COMPARED TO THE ML PERFORMANCE. TRAINING LM: 48 k WORD UNIGRAM; TESTING LM: 30 k WORD TRIGRAM. ALL RESULTS ARE FOR AN HMM TOPOLOGY OF 3000 STATES AND EIGHT GAUSSIANS PER STATE

	ML	SB-PD	Quickprop	SB-Rprop	Rprop
Train	47.7	41.1 (13.3)	34.0 (28.3)	31.0 (34.6)	30.2 (36.3)
Test	23.8	22.2 (6.7)	22.0 (7.6)	22.8 (4.2)	22.4 (5.9)

TABLE III
ERROR RATES FOR MCE VERSUS ML BASELINES ON THE CORPUS OF SPONTANEOUS JAPANESE, AND RELATIVE ERROR RATE REDUCTIONS (%) COMPARED TO ML PERFORMANCE. TRAINING LM: 68 k WORD UNIGRAM; TESTING LM: 30 k WORDS

# States	# Gaussians/state	# Gaussians	ML	MCE/Quickprop
2000	16	32k	23.4	22.3 (4.7)
2000	32	64k	22.4	21.0 (6.3)
3000	8	24k	24.1	22.5 (6.6)
3000	16	48k	23.1	20.8 (10.0)
4000	16	64k	22.8	20.8 (8.8)

H. MCE Training: Optimization Methods

For these experiments, a single acoustic model was used, with 3000 Gaussian mixture states and eight Gaussians per state (24 k Gaussian pdfs in all).

Starting with the baseline ML model, MCE optimization runs based on semibatch PD ("SB-PD," with an update period of 20 000 utterances), Quickprop, semibatch Rprop ("SB-Rprop," with an update period starting at 20 000 utterances, but doubling after every full presentation of the training set), and batch Rprop, were carried out for six iterations (i.e., six full presentations of the training set). The model at the end of training was used for testing. Learning rate parameters for each method were coarsely tuned on the training set in preliminary runs using two to three iterations. Rprop was found to be significantly easier to tune than the other methods. In all the CSJ experiments, a linear loss function was used, and only the top incorrect string used. In recent work, use of up to 99 incorrect strings with $\psi = 0.01$ in (4), was evaluated, but no significant improvements were obtained.

The word error rates for each method on the training set using the unigram WFST, and on the test set using the standard 30 k word trigram, are shown in Table II.

I. MCE Training: Model Topology and Size

The aim of these experiments was to investigate the impact of MCE training on ML baselines with different numbers of states and Gaussians per state. These experiments are based on the latest "CSJ 2004" release of the database, and use feature vectors with 39 rather than 38 components; the results are therefore not strictly comparable to results described in Section V-H.[8]

Here the 68 k word unigram described in Section V-D was used for MCE training. Seven iterations of MCE training using Quickprop were carried out for each model topology, and the final models tested using both the 30 k and 100 k word trigrams. The results are shown in Tables III and IV.

[8]The differing results for 3000 states and eight Gaussians/state in Tables II and III are thus based on different experimental conditions.

TABLE IV
ERROR RATES FOR MCE VERSUS ML BASELINES ON THE CORPUS OF SPONTANEOUS JAPANESE, AND RELATIVE ERROR RATE REDUCTIONS (%) COMPARED TO ML PERFORMANCE. TRAINING LM: 68 k WORD UNIGRAM; TESTING LM: 100 k WORDS

# States	# Gaussians/state	# Gaussians	ML	MCE/Quickprop
2000	16	32k	23.0	21.1 (8.3)
2000	32	64k	21.7	20.5 (5.5)
3000	8	24k	23.7	21.1 (11.0)
3000	16	48k	22.4	20.5 (8.5)
4000	16	64k	22.1	20.1 (9.1)

J. Results & Discussion

Optimization: The results on the CSJ task show that for optimization on the training set, Rprop is very effective. Given the simplicity of the method and the ease of parameter tuning, Rprop seems like an attractive choice. However, Quickprop yielded the best test performance. It is interesting to note that here semibatch PD is the least effective method on the training set, but its test set performance is the second best. The opposite pattern was observed for TIMIT phone classification. More tests with these methods on the CSJ task are needed to determine whether the behavior reported here is representative.

Altogether, the results obtained on both TIMIT and CSJ show significant improvements over the maximum likelihood baseline for all optimization methods evaluated, indicating that the MCE criterion can be optimized effectively using rather different methods.

Model Size/Topology: Using 30 k and 100 k word trigram language models for test set evaluation, MCE yielded relative error rate reductions from 4.7% to 11% for several ML baselines with different topologies. The best MCE-obtained results for the 30 k and 100 k word trigrams, respectively, are word error rates of 20.8% and 20.1%. The best ML results are word error rates of 22.4% and 21.7%, respectively. Compared to these ML results, MCE afforded relative error rate reductions of 7.1% and 7.4%, respectively. Though these relative improvements are lower than those for the other tasks described in this article, the very large number of homonyms in Japanese may limit the effectiveness of discriminative training of acoustic models on this task.

Ongoing and Future Work: Ongoing work using the 100 k trigram language model for MCE training has so far yielded results that are very similar to those for the unigram-based training approach detailed here. Thus, on this task, training with a unigram language model does not seem to help generalization compared to training with the target trigram language model. However, training with the 100 k trigram is rather time-consuming, so training with a unigram still has merit. Future work on this task should evaluate the use of an MCE update based on a lattice of incorrect candidates [14]. Discriminative speaker adaptive training [16] could also be effective for this task.

More discussion of the results on the CSJ task can be found in Section VIII.

VI. LARGE-VOCABULARY TELEPHONE-BASED NAME RECOGNITION

The task examined in this section is that of recognizing Japanese names spoken over the telephone, in the real-world setting of people telephoning a call center to request that information (catalogs, pamphlets, etc.) be mailed to them. The goal was to evaluate the effectiveness of MCE for an offline speech recognition system used to transcribe the contents of each call, in particular the caller's name. The specific issues investigated for MCE training include the effects of model size, lattice-based and phone bigram approximations to the full language model, and a strict versus flexible approach to reference modeling. This study extends previous work [14] and was recently summarized in [16].

A. Database and Data Characteristics

A database of more than 40 h of utterances in this real-world setting was collected and transcribed. Most utterances are from different callers. Some broad aspects of the database can be described qualitatively. Every utterance contains a family name and a given name. There is great variation in speaking style, and a large proportion of the calls were made from cellular phones and noisy acoustic environments. Another feature of the data is the presence of false starts, hesitations, and various fillers ("eeto. . .," "ano. . .," etc. at the beginning of the utterance; "desu," "to mooshimasu," etc., at the end) that can bracket the caller's name.

Training, Test, and Development Sets: From the overall data set, a training set of 35 500 utterances (about 39 h of audio) and a test set of 6428 utterances were selected. A separate development set of 948 utterances was also prepared for the purpose of tuning language model scaling factors. In addition to the transcriptions assembled for training and test sets, phoneme segment start/end times for 1200 training utterances were hand-labeled; these were used as seed data to initialize the recognizer.

B. Target Language Model

From a corpus of 130 000 name listings, 16 602 unique family names and 5755 unique given names were found. A WFST was designed to model 16 576 family names and 5744 given names, covering about 99.8% of the family and given names found in the corpus. Unigram probabilities were estimated from the corpus and incorporated into the WFST. The set of names used covers all utterances in the training set described above—out-of-vocabulary (OOV) utterances were either removed from the training set, or the corresponding names added to the lexicon. In addition to the set of names, some simple types of hesitations and pauses are modeled, along the lines of the patterns described in Section VI-A. The test set contains a small number ($<1\%$) of utterances not modeled by this language model. Fig. 6 illustrates the WFST used.

After weight-pushing and network optimization, the network contained 489 756 nodes and 1 349 430 arcs. (Further compression is possible via composition with the WFST—often referred to as "H"—mapping HMM states to triphones, but this was not performed in this study). The size of the vocabulary modeled (22 320 names in all), and the fact that both cross-word and within-word triphones were modeled, are significant improvements compared to the initial study presented in [14].

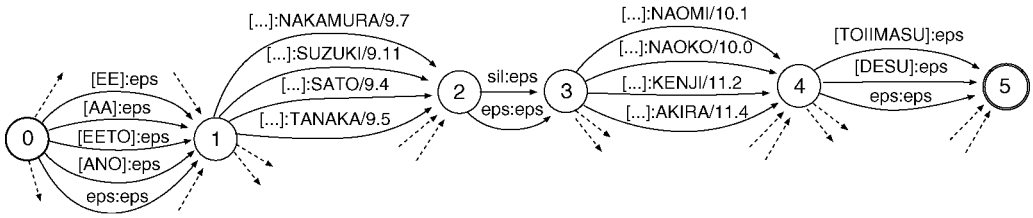


Fig. 6. Outline of the WFST-based target language model for the telephone-based name recognition task. The WFST represents 22 320 family and given names, as well as possible utterance-initial hesitations, silence insertion, and utterance-final completions. Arc symbols to the left of the colon are input symbols, those to the right are output symbols; language model scores, if any, are shown after a '/'; "eps" refers to epsilon, the null symbol. Symbols in brackets refer to the phonetic realization of a given word.

C. Strict Reference WFSTs

Since the task is primarily a name recognition task, word segmentation did not pose the problem it did for the CSJ task. The first set of reference WFSTs created was based on a straightforward representation of the given transcription for each utterance, including fillers, family name and given name, and associated language model scores. The triphone[9] and lexicon[10] WFSTs are then composed with the WFSTs from the previous step,[11] so that each resulting WFST is a miniature recognition network, whose input symbols are triphones, and whose output symbols are word IDs with language model scores. These reference WFSTs can then be used with the decoder to calculate the MCE discriminant function for the correct string set (2), as well as the corresponding state segmentations necessary to compute the MCE derivatives, as described in Section II. Since they adhere to the exact word content of the transcriptions, and are not characterized by much if any branching,[12] they are referred to in the following as "strict reference WFSTs."

D. Flexible Reference WFSTs

As described in Section II-C, there is often uncertainty in utterance transcriptions. For the name recognition task, it was observed that though the target names were transcribed accurately, fillers were often dropped or transcribed incorrectly. For this task, however, only accurate recognition of the names is of interest. Hence, a "flexible reference," representing, in addition to the family name and given name, all possible fillers around and in between the two names (along the lines described in Section VI-A), provides a better definition of the desired output. This concords with the notion, formalized in Section II, that the desired output is any string out of a set of strings, all considered to be correct.

The use of corresponding "flexible reference WFSTs" during MCE training to model the utterance content (2) was evaluated for this task. One such WFST is illustrated in Fig. 2, which models possible hesitations at the start of the utterance, possible silence insertion between the family and the given name, and possible completions at the end of the utterance.

When using flexible reference WFSTs, the MCE loss function corresponds closely to string-level name recognition, and is not affected by success or failure in recognizing fillers. The discriminative training task is easier than when using strict reference WFSTs. It can be hoped that optimization will consequently be more efficient.

E. Modeling References With a Phone Bigram

This study evaluated a third approach to modeling the correct string set used in (2): a simple phonetic transcription, coupled with language model scores from a phone bigram. This approach is typically, but not necessarily, coupled with the use of a looping phone bigram recognition network for generating the competitor categories.

F. Language Model for MCE Training: Target Language Model

The target recognition WFST, described in Section VI-B, can be used for MCE training as is. In tandem with the SOLON decoder, it can be used to compute the discriminant function for the competing incorrect string set [(3) or (4)], as well as the corresponding state segmentations necessary to compute the MCE derivatives.

Since the recognition WFST was designed to cover all utterances in the training set, there is no out-of-vocabulary issue. As it is essentially a unigram language model, decoding can be performed in a reasonable time. Furthermore, this choice matches the set of competitors considered during training with the set of competitors that applies during testing.

G. Language Model for MCE Training: Lattices

The use of lattices for discriminative training has been reported for the maximum mutual information (MMI) approach already for some time now [10], [31]. Large-scale speech recognition systems have been successfully designed using this approach [11]. In contrast, only recently have studies reported results for latticed-based MCE [47].

Lattices can be seen to have two roles for discriminative training. The first is that they can be used to "push away" a very large number of competing strings. In the context of MMI, this serves to maximize mutual information, defined as the ratio of the likelihood of the correct string to the posterior probabilities of all possible strings (including the correct string). The lattice primarily models the denominator of this ratio. In the context of MCE, a lattice can be used to model the general form of the discriminant function for the set of incorrect strings (4), as discussed in [27] and [47].

[9]*Input*: triphones; *output*: phones.

[10]*Input*: phones; *output*: word IDs.

[11]*Input*: word IDs; *output*: word IDs and language model scores.

[12]What branching there is in the strict reference WFSTs comes from pronunciation variants for the same name.

However, lattices can play another important, practical role: that of speeding up the learning procedure. This is the role evaluated here. Assuming that the model does not change too much over the course of learning, lattices can be generated for all utterances for the initial model, and then repeatedly reused over the course of training. This is a common practice in MMI implementations. To our knowledge, the study described in this article is the first to evaluate such a use for MCE.[13]

The approach evaluated here consists in first generating lattices for each utterance, and then removing the segmentation time information. Saved in WFST format, the result can be used as the recognition grammar during MCE training, still only using the top competing incorrect string, as expressed in (3). Using lattice-derived WFSTs as the language model for each utterance is much faster than using the full language model.

Each lattice WFST is made during beam search through the target language model WFST. The lattice receives a new arc for every output-emitting arc activated during search. This usually happens somewhere in the word once a phone sequence becomes unique. In this study, when generating the lattices, the beam width was set to be about three times larger than that used during training with the full language model. Across the configurations used, the resulting set of lattices contained on average 800 arcs per lattice, in contrast with 1 349 430 arcs for the target language model.

H. Language Model for MCE Training: Phone Bigram

Another possibility is to use a phone bigram to calculate the discriminant function for the top incorrect string set, (3) or (4). In this study, a bigram-constrained connected triphone loop was used.

In tandem with a phone bigram to model the correct string set (Section VI-E), using a bigram-constrained connected phone loop as the language model during MCE training is a practical choice. With this approach, discriminative training will attempt to optimize phoneme string recognition accuracy in a task-independent manner. The hope is that the enhanced phoneme recognition accuracy will translate into better word recognition when given the target language model. This approach can be a simple way of obtaining improvements over the ML/Viterbi training baseline.

I. Feature Extraction and Baseline HMMs

All utterances were converted into a sequence of feature vectors based on RASTA-processed mel-scale filter-bank cepstral coefficients (MFCC) [66]. A 21-component feature vector was extracted for every 10 ms of the waveform, consisting of ten static MFCC-RASTA components, ten deltas, and a delta energy component.

Triphone models of several sizes were clustered using phonetic decision trees [63]. The triphone set is based on 28 base phones that include utterance initial/final silence, within-utterance silence, and a model for the click of a telephone being hung up.

[13]The results of this study were first reported in [16].

TABLE V
CONFIGURATIONS OF TRIPHONE HMMS USED

# States	# Gaussians/State	Total # Gaussians
187	4	748
187	12	2244
187	20	3740
547	12	6564
547	20	10940
547	36	19692
547	50	27350

TABLE VI
NAME ERROR RATE FOR DIFFERENT MODEL CONFIGURATIONS: ML BASELINE AND VARIOUS MCE DESIGN METHODS

# Gaussians	ML	Phone	FullLM	Lattice	Lattice+FlexRef
748	43.1%	35.4%	31.4%	32.8%	31.0%
2244	37.7%	31.7%	27.2%	28.1%	27.9%
3740	35.4%	31.0%	-	-	26.8%
6564	32.9%	29.7%	26.8%	28.2%	26.4%/25.5%
10940	31.4%	27.6%	-	-	26.3%/25.0%
19692	30.1%	26.8%	-	-	25.6%/25.1%
27350	29.6%	-	-	-	-

Two different log-likelihood increase thresholds were used during tree construction, resulting in two sets of trees with different tree depths and sizes. The shallower tree set, containing 187 triphone states, was used to make three models with, respectively, 4, 12, and 20 Gaussians per leaf node mixture; the deeper tree set, containing 547 triphone states, was used to make four models with, respectively 12, 20, 36, and 50 Gaussians per leaf node mixture. Deeper trees than those reported here were also evaluated but showed worse performance. The different HMM configurations are summarized in Table V.

Given these models, name recognition using the SOLON decoder with the recognition WFST described in Section VI-B was evaluated. The recognition results for each configuration are shown in Table VI. The error rates reported here are equal to 100 minus name accuracy; name accuracy is based on both family name and given name recognition accuracy. We do not report string accuracy, i.e., recognition accuracy for family and given name considered as a unit.

J. MCE Training

MCE training based on Quickprop was carried out using different choices of reference and language models. For all settings evaluated, the sigmoid loss function (6) was used. Furthermore, up to 40 iterations of MCE training were carried out. This is a large number of iterations, but acts as insurance against a poor choice of ϵ, which can affect the speed of convergence of Quickprop.

1) Phone Bigram: The first approach evaluated was the use of a phone bigram model for the utterance reference, described in Section VI-E, together with a bigram-constrained triphone loop model as the language model, described in Section VI-H. MCE training for this setting was carried out for six of the ML-trained acoustic model configurations. The name recognition results for the resulting MCE-trained model are shown in Table VI in the column for *Phone*.

2) Target Language Model and Strict References: The second set of experiments evaluated the use of the target

language model to define the set of competing strings (see Sections VI-B and VI-F), together with the use of the strict reference WFST model for the references (see Section VI-C). Three of the ML-trained model configurations were used as initial models for MCE training. Even parallelized over multiple machines, the large number of iterations and the heavier decoding demands of using the full language model led to a training time of up to two days. The test set results are shown in Table VI in the *FullLM* column.

3) Lattices and Strict References: Recognition lattices were generated for all training utterances for three of the ML-trained model configurations. MCE training was carried out using as language model for each utterance the corresponding lattice WFST (Section VI-G). The reference model was again based on strict reference WFSTs (Section VI-C). Training is now much faster than training with the target language model, by a factor of about three. The results for lattice based MCE training are shown in Table VI in the *Lattice* column. The results are very close to those for MCE training with the full language model.

4) Lattices and Flexible References: Recognition lattices were generated for all training utterances for an additional three ML-trained model configurations. A total of six ML-trained models were then used as the initial models for MCE training, again using lattice WFSTs as the language model for each utterance (Section VI-G), but this time using flexible reference WFSTs (Section VI-D). Training proceeded more quickly than when using the strict WFSTs, significantly raising the training set recognition rate after each iteration. For some of the configurations, lattices had to be recreated half-way through the training procedure. The test results are shown to the left of the "/" character in Table VI in the *Lattice* + *FlexRef* column.

5) Lattices, Flexible References, and L_p-Normed N-Best Incorrect Candidates: The experiments described so far all assumed the use of (3), i.e., the use of a single best incorrect string candidate. Using instead the general form, (4), and a choice of ψ that prevents the top few incorrect strings from dominating the expression, may help generalization to test data. This is related to "acoustic scaling" used in MMI studies [11], [44]. This was investigated for three of the ML-trained baseline models. Using $\psi = 0.1$ and the top 30 incorrect strings during lattice-based MCE training with flexible reference WFSTs resulted in performance gains. These results are shown to the right of the "/" character in the *Lattice* + *FlexRef* column of Table VI.

K. Results

All results for the name recognition task are shown in Tables VI. VII shows the relative recognition error reduction for different model sizes as a result of MCE training, using the best MCE result for each configuration, and in the last row, the relative error reduction for the best overall MCE model compared to the best overall ML model.

L. Discussion

The experiments on the name recognition task primarily investigated the effect of MCE training applied to ML baseline models of different sizes. The study also evaluated lattice-based and phone bigram approximations to the full language model, and strict versus flexible approaches to reference modeling. Some specific points of discussion follow.

TABLE VII
RELATIVE NAME ERROR REDUCTION AS A RESULT OF MCE TRAINING FOR GIVEN MODEL SIZES, AND RELATIVE ERROR REDUCTION FOR BEST MCE VERSUS BEST ML MODELS ACROSS ALL MODEL SIZES

# Gaussians	Error Reduction
748	27.9%
2244	26.1%
3740	24.4%
6564	22.4%
10940	20.4%
19692	16.4%
10940 (MCE)/ 27350 (ML)	15.5%

1) Model Size: As shown in Table VI, the smallest MCE-trained model described here, with 748 Gaussians, outperforms the baseline models with up to 10 940 Gaussians. The best MCE-trained model, with 10 940 Gaussians, yielded a relative performance improvement of 20.4% over the baseline model of that size, and 15.5% relative improvement over the best overall baseline model. As shown in Table VII, the benefit of MCE training, though largest for the small configurations, applies to all the configurations examined. These are very large gains in performance.

2) Phone Bigram: MCE training with the bigram-constrained triphone loop as language model yielded significant improvements over the ML baseline. This approach is easy to implement, and may not be too time-consuming, depending on the depth of the decision trees. However, training with the full language model yields significantly better performance than training with the triphone loop.

3) Lattices: The use of pregenerated lattices to generate per-utterance WFST language models is a reasonable approximation to the full language model, and speeds up decoding by a factor of about three. The results obtained in this study show that MCE training can be performed successfully using this approximation. Saving the lattices, stripped of segmentation time information, in WFST format, and then using them with the same central decoder, makes implementation straightforward. Further gains in computation time could result if one were to fix the segmentation times.

4) Flexible Reference Model: This article proposed the use of string *set* level discriminant functions for use with MCE-based discriminative training. The approach provides a coherent way of making the MCE loss function reflect the true recognition target in the context of keyword-oriented tasks, as exemplified in the "flexible reference" WFST model tested on the name recognition task. Compared with the conventional "strict reference" approach, this approach yielded gains in performance for all configurations examined.

5) N-Best Rather Than 1-Best: A benefit was found to smoothing the misclassification measure (5) via the exponent ψ, using top 30 incorrect strings. This is still a very small number of candidates considering the 22 k word language model. It can be surmised that using many more candidates, ideally via a lattice-based model of (4) [27], could yield additional gains in performance.

VII. JUPITER WEATHER INFORMATION

JUPITER is a telephone-based conversational system that can understand and respond to queries related to weather forecasts for roughly 500 cities around the world [67]. The system is accessible to anyone via a toll-free number in the USA. It uses a mixed-initiative dialogue strategy which allows callers the freedom to express their queries in whatever form they wish. Because of this approach, JUPITER's recognizer is forced to handle highly spontaneous continuous speech. Difficult artifacts such as background and foreground noises, out-of-vocabulary words, false starts, partial words, filled pauses, and strong accents are common, making the recognition task challenging. The speech recognition module used in JUPITER is the SUMMIT recognizer [68]. The acoustic model in SUMMIT consists of landmark and segment models defined for variable-duration utterance segments.

This section describes results obtained from applying MCE training to the SUMMIT acoustic models. The MCE implementation and experimental setup used in this study are independent from the SOLON-based setup described so far. Furthermore, it should be noted that the variable-duration segment modeling approach adopted in SUMMIT is quite different from the HMM-based approach implemented in SOLON. Evaluating the effect of MCE training in the JUPITER/SUMMIT context thus provides insight into the generality of the method's ability to improve the recognition performance of acoustic models that are not limited to standard HMMs.

As with the evaluations for the other tasks described in this article, the main target of investigation is the effect of MCE training on ML baseline models. In addition, specific points investigated on the JUPITER task include a comparison of the typical MCE sigmoid loss with a linear loss function, as well as with a piecewise linear loss function that only results in parameter modification when recognition errors occur (a type of optimization often referred to as "corrective training").

MCE results for JUPITER were first presented in [16]. Here the experimental setup for evaluation is expanded to include a 33 k word language model in addition to the standard JUPITER 1924 word language model. This evaluation sheds light on the ability of MCE training to improve performance not just for the criterion used during training, but for more general criteria as well.

A. Database

The data sets used in these experiments were primarily derived from calls made to the JUPITER weather information system.

1) Training Set: The baseline acoustic models were trained using 140 769 utterances, amounting to 120 hours of audio. A subset of 101 965 utterances was used for MCE training. These utterances only contain words covered by the standard JUPITER 1924 word vocabulary.

2) Development Set: A development set of 4894 held-out JUPITER utterances was available to tune the MCE training parameters (sigmoid slope α and Quickprop learning rate ϵ), and to examine the convergence of the training procedure.

3) Test Set: A held-out set of 2905 JUPITER utterances was used for evaluation. Within this set, 80% of the utterances are spoken entirely with words contained in the standard 1924 word vocabulary used by the JUPITER recognizer. The remaining 20% of the data contains at least one out-of-vocabulary (OOV) word or incomplete partial word in each utterance. When using the expanded language model (described in Section VII-B), the OOV rate decreases slightly, from 20% to 17%.

Because errors caused specifically by OOV words cannot be recovered by improvements to the recognizer itself, the JUPITER conversational system relies on a combination of OOV word detection, recognition confidence scoring, and error-recovery dialogue techniques to recover from recognition errors introduced by the presence of out-of-vocabulary words [69], [70]. The MCE evaluations described here focus on the in-vocabulary portion of the evaluation set (i.e., 80% of the full evaluation set). However, results on the full evaluation set (including the OOV utterances) are reported as well.

B. Target Language Models

The primary language model used in the evaluations here is the standard language model for JUPITER, based on a 1924 word trigram, with back-off bigram and unigram models. The language model uses a basic vocabulary of around 1000 words to model different ways of asking questions related to weather information, allowing for hesitations, false starts, and other types of disfluencies. The remaining vocabulary items are city names, such as "New York City," "Santa Monica," etc.

A trigram with an expanded vocabulary of 33 031 words was used as well. The additional vocabulary items (not found in the standard 1924 word language model) are primarily city names.

C. Reference WFSTs

SUMMIT, like SOLON, is a WFST-based recognition architecture. The JUPITER training utterances come with word-level transcriptions, represented in WFST format; language model scores were added to these using WFST composition with the target language model. The transcriptions are flexible in that they allow for variation in many contractions and reductions, such as "what's" for "what is," but strict with regard to filled pauses and false starts, which are assumed to be transcribed accurately.

D. Language Model for MCE Training

The language model used for MCE training is the standard 1924 word trigram language model used for JUPITER. This is the case even when using the expanded 33 k word language model for testing.

When filtering the correct string out of the recognition output for MCE training, words that are not used in calculating the word error rate, such as "uh" and "um," are ignored. Furthermore, contractions and reductions are considered equivalent to their canonical forms.

E. SUMMIT Baseline Acoustic Model

SUMMIT's standard real-time recognition configuration performs acoustic modelling using a set of 1388 context-dependent landmark models [68]. Each hypothesized landmark is modeled with a set of MFCC averages collected from regions, up to 75 ms away, surrounding the landmark. The acoustic information surrounding each landmark is represented by a 112-dimension feature vector which is reduced to 50 dimensions using a principal

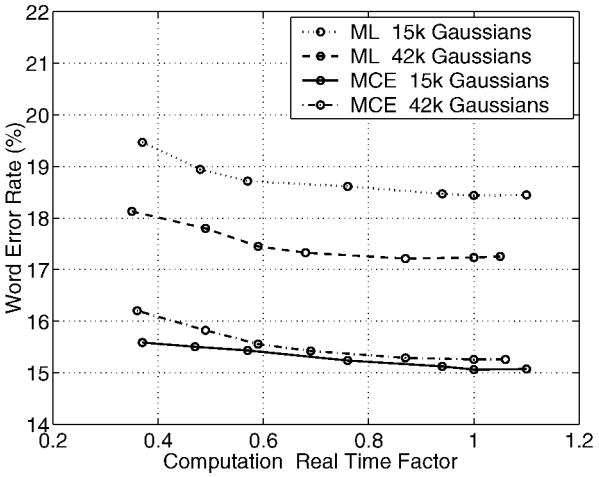


Fig. 7. Comparison of ML and MCE training for two different model set sizes (in-vocabulary evaluation set, 33 k word language model).

TABLE VIII
WORD ERROR RATE COMPARISON OF ML TRAINED MODELS AND MCE TRAINED MODELS, EVALUATED AT A REAL-TIME COMPUTATION SETTING (STANDARD 2 k WORD JUPITER LANGUAGE MODEL)

	Word Error Rate		Relative Error
Test Set	ML-42k-gauss	MCE-15k-gauss	Rate Reduction
In Vocab.	16.3%	13.1%	19.7%
Out of Vocab.	54.6%	56.7%	-1.7%
All Data	23.9%	21.8%	8.9%

TABLE IX
WORD ERROR RATE COMPARISON OF ML TRAINED MODELS AND MCE TRAINED MODELS, EVALUATED AT A REAL-TIME COMPUTATION SETTING (EXPANDED 33 k WORD JUPITER LANGUAGE MODEL)

	Word Error Rate		Relative Error
Test Set	ML-42k-gauss	MCE-42k-gauss	Rate Reduction
In Vocab.	17.2%	15.3%	11.5%
Out of Vocab.	63.0%	67.5%	-7.2%
All Data	23.8%	22.8%	4.4%

component analysis (PCA) rotation matrix. These 50-dimension vectors form the input to the recognizer's acoustic model classifier, which assigns context-dependent acoustic scores for each landmark using mixture Gaussian models. The landmark scores are combined with segment duration scores and class trigram language model scores during the search to produce the full score.

The baseline set of landmark models was trained using standard ML Viterbi-style EM training. The number of mixtures per model was heuristically set to be 1 mixture component for every 50 training samples with a maximum of 75 mixture components per model. Previous experiments showed that using more than a maximum of 75 components per model does not improve recognition performance of ML trained models. This produced a model set with 41 677 total Gaussian components, for an average of 30 Gaussians per model over the full set of 1388 models. In the following, this baseline set of models is referred to as the "ML-42k-gauss" model set.

In order to evaluate the ability of MCE training to produce accurate models using fewer parameters, a second set of ML-trained models was also produced. These models were trained with at most 15 Gaussian components per model, producing a model set using 15 325 total components, for an average of 11 Gaussians per model. In the following, this smaller set of ML-trained models is referred to as the "ML-15k-gauss" model set. The ML-15k-gauss model set thus has 63% fewer parameters than the ML-42k-gauss model set.

Fig. 7 shows the in-vocabulary performance of the two sets of ML baseline models as a function of computation time, for the expanded JUPITER language model. Computation time is controlled via the pruning threshold used during decoding. Tables VIII and IX show the ML performance at the point of real time computation, for the standard and expanded language models, respectively.

F. MCE Loss Function

Three loss functions were evaluated:

1) Standard Sigmoid: Equation (6) with steepness α set so that the function tapers off for a substantial number of training set utterances that are very strongly correct or incorrect (see Section II-G which discusses the tuning of α).

2) Linear Loss Function: $\ell(d_{\mathcal{K}}) = d_{\mathcal{K}}$. Use of this loss means the optimization procedure is directed at maximizing overall *separation* between correct and best incorrect hypotheses, with no consideration of the binary cost of misclassification. Increasing separation might help generalization to unseen data. This setting corresponds to an MMI-derived approach, "rival training," in which only correct and best incorrect strings are used [71].

3) "Corrective" Piecewise Linear Loss Function: $\ell(d_{\mathcal{K}}) = d_{\mathcal{K}}$ for $d_k > 0$, and 0 otherwise. This function corresponds to a *corrective training* strategy, which only performs parameter modifications for incorrectly recognized utterances. This choice of loss function corresponds to the MMI-derived corrective training approach [4], [27], [71]—though the optimization method used here is Quickprop, and not the Extended Baum–Welch algorithm.

G. MCE Training

The mean vectors, variances, and Gaussian mixing weights of SUMMIT's landmark-based acoustic models were modified via MCE-based training. The MCE derivatives described in Appendix A were applied to SUMMIT landmark models with little modification.

MCE training was carried out using the two ML baselines as initial model configurations, and using different choices of loss function (Section VII-F). Preliminary experiments comparing Quickprop with the batch probabilistic descent (BPD) method (discussed in Section III-A) showed an advantage for Quickprop. The following experiments all used Quickprop as the optimization method. For each of the conditions examined, 12 iterations of MCE-based optimization were used. For all experiments, only the top incorrect string candidate was used to model the set of competitors, corresponding to the use of (3).

1) MCE versus ML Baseline, Sigmoid Loss Function: MCE optimization with the standard sigmoid loss function was used to produce a "MCE-42k-gauss" model from the ML-42k-gauss models, and a "MCE-15k-gauss" model set from the ML-15k-gauss models. The models performing best on the development

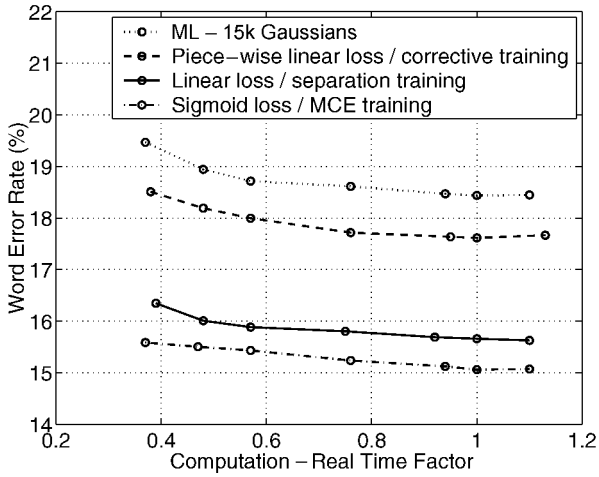


Fig. 8. Comparison of three different loss functions used to train the ML-15k-gauss model (in-vocabulary evaluation set, 33 k word language model).

set (over all 12 MCE iterations) were then chosen for evaluation. Fig. 7 shows the performance of the MCE trained models as a function of computation time allotted to the decoding process, using the in-vocabulary evaluation set and the expanded 33 k word language model. Computation time is controlled via the search pruning threshold.

2) Sigmoid, Linear, and "Corrective" Piecewise Linear Loss Functions: Fig. 8 shows the performance of models derived from training with the three different MCE loss functions, using the ML-15k-gauss model set as the starting point, again on the in-vocabulary evaluation set and the expanded 33 k word language model. Again, the development set was used to choose the best MCE models over all 12 iterations.

H. Results, Standard 2 k Word JUPITER *Language Model*

Table VIII summarizes the results for the best MCE trained model set on the entire evaluation set when using the standard 1924 word language model. Overall performance of the MCE-15k-gauss model set (trained with the sigmoid loss) versus the baseline ML-42k-gauss model set is shown, as well as performance on in-vocabulary (IV) and OOV portions of the evaluation data. These results were obtained using a pruning threshold that achieves real time performance for each model set. MCE training produced a relative word error rate reduction of 8.9% over the full evaluation set, and a relative word error reduction of 19.7% on the IV portion of the data.

I. Results, Expanded 33 k Word Language Model

When tested on the full evaluation set (IV and OOV) using the 33 k word language model, the MCE-42k-gauss model set performs slightly better than the MCE-15k-gauss model set. Table IX summarizes the test results when using the expanded 33 k word language model, for both the MCE-42k-gauss model set and the baseline ML-42k-gauss model set. Performance is divided into IV and OOV components, for a pruning threshold that achieves real time performance for each respective model set. Overall, MCE training produced a relative word error rate reduction of 4.4% over the full evaluation set, and a relative word error reduction of 11.5% on the IV portion of the data.

J. Summary and Discussion

The experiments on the JUPITER weather information task primarily investigated the effect of MCE training applied to two ML baseline models. The study also evaluated the effect of different MCE loss functions.

For both the standard 1924 word JUPITER language model and an expanded 33 k word language model, significant improvements over the ML baseline were obtained as a result of MCE training, showing the effectiveness of MCE on a real-world continuous speech recognition task. For the standard language model, discriminative training yielded a 8.9% relative performance improvement overall, and a 63% reduction in the number of parameters compared to the baseline model. For the IV portion of the test set, MCE training yielded a 19.7% relative reduction in word error compared to the ML baseline. With the expanded language model, the overall performance improvement is 4.4%, and 11.5% for the IV portion of the evaluation data. Additional points are discussed in the following.

1) Model Size: As shown in Fig. 7, the smaller ML-15k-gauss model set performs significantly worse than the larger ML-42k-gauss model set, indicating that the additional parameters present in ML-42k-gauss are required to model accurately the probability density functions of the individual classes. However, when using MCE training, there is little difference in performance between the MCE-15k-gauss model set and the MCE-42k-gauss model set. This demonstrates the significant reduction in parameters that MCE training enables.

2) Sigmoid versus Linear Loss Function: As shown in Fig. 8, the models trained with the standard sigmoid loss function show slightly better performance than those trained with the linear loss function. Even though the 0–1 sigmoid loss function is oriented towards optimal *string* recognition accuracy, and the focus here is on *word* recognition accuracy, there appears to be an advantage to using a nonlinear function of separation rather than separation itself. The results show that an equal weighting of the training tokens is not as effective as a loss function which assigns greater weight to the training tokens closest to the (string) classification decision boundary.

3) Rival versus Corrective Training: As shown in Fig. 8, both sigmoid and linear loss functions resulted in significantly better performance than the piecewise linear loss function corresponding to pure corrective training. This clearly shows the benefit of increasing the separation between correct and best incorrect strings even for correctly recognized utterances.

4) OOV Utterances: As shown in Table VIII, MCE training did not improve the performance of the recognizer on the out-of-vocabulary portion of the data. In fact, relative degradations of 1.7% and 7.2% occurs on the OOV data for the standard and expanded language models, respectively. This degradation in performance on utterances containing OOV words is not unexpected, as the MCE training process only used in-vocabulary utterances. (Recall that while the ML baselines were trained on 140 769 utterances, MCE training was limited to a 101 965 utterance in-vocabulary subset). A straightforward remedy to the problem may be to expand the MCE training language model to cover all 140 k training utterances. This was the approach adopted for the CSJ and name recognition tasks described earlier in the article.

5) Expanded 33 k Word Language Model: The improvements obtained for the expanded 33 k word task, 4.4% overall and 11.5% for in-vocabulary utterances, are more modest than for the standard language model. Compared to evaluation with the standard language model, there is an even greater degradation on OOV utterances. This is particularly so for the MCE-trained models, which show a 7.2% relative increase in word error rate compare to the ML baseline. However, it should be kept in mind that MCE training used the standard 1924 word vocabulary, not the expanded 33 k word vocabulary. Many confusions that can arise with the expanded vocabulary were therefore not observed during model design. In spite of this mismatch, MCE yielded substantial benefits to recognition with the expanded language model.

K. Possibilities for Further Improvements

Clearly, it would be desirable to use the entire training set, rather than just a subset, for MCE training. Using the full set might yield additional improvements. Furthermore, training with the expanded 33 k word language model should improve performance when using that language model.

All results presented here for the JUPITER task are based on a 1-best implementation of the MCE discriminant function for the incorrect string set, corresponding to the use of (3) rather than (4) with a small ψ. Using many more incorrect candidates, via either a large N-best list or a lattice [27], may yield significant additional improvements.

VIII. Overall Summary and Discussion

This article reported evaluation results for MCE-based discriminative training performed on several speech recognition tasks, ranging from phone classification to 100 k word continuous speech recognition. Aside from the primary comparison of MCE versus baseline ML performance, common to all the studies described here, each study examined a slightly different set of issues. Table X summarizes (for the large-scale tasks) the task characteristics, issues investigated, and relative error rate reductions (RERRs) obtained from MCE training. "Lattice approximation" in this table refers to the use of lattices to speed up the training process. "Flexible reference" refers to the string *set* oriented model of the desired output outlined in this article. The RERR figures listed are those described in the Results section for the different tasks. The two figures listed for the CSJ column refer to the 30 k and 100 k word language models, respectively. The two figures listed for the JUPITER column refer to the standard 1924 word language model and expanded 33 k word language model, respectively. Both sets of JUPITER figures refer to the in-vocabulary evaluation.

TABLE X
SUMMARY OF TASK CHARACTERISTICS, ISSUES INVESTIGATED, AND MCE-OBTAINED RELATIVE ERROR RATE REDUCTION (RERR) FOR EACH TASK

	CSJ	Names	JUPITER
Vocabulary	30k/100k	22k	2k/33k
Language model	3-gram	1-gram	3-gram
Continuous speech	✓	-	✓
Optimization	✓	-	-
Model size	✓	✓	✓
Lattice approximation	-	✓	-
Flexible reference	-	✓	-
Loss function type	-	-	✓
RERR	7.1%/7.4%	15.5%	19.7%/11.5%

Gradient-Based Optimization: A number of gradient-based optimization techniques were evaluated in this article. The ability to optimize the MCE loss function successfully across different tasks and system architectures is a crucial point for MCE. One reason for the widespread use of MMI is the availability of a corresponding re-estimation algorithm, the extended Baum–Welch algorithm [42]. Similar reestimation algorithms have been suggested for MCE [49], [50]. These approaches require an approximation of the MCE loss function. In contrast, the approach described in this article was to keep the MCE loss function as it is, and use gradient-based optimization. Though this means a learning rate must be tuned, this is not hard in practice, and furthermore is not very different from the need to set the reestimation constant when using the extended Baum–Welch algorithm [72]. Given the practical utility of gradient-based descent, demonstrated here, combined with a wealth of theoretical analysis [73], [74], there seems to be no reason to shy away from the use of gradient descent in one of its many practical instantiations.

Of course, better optimization methods for MCE, including reestimation style approaches, should continue to be an ongoing topic of investigation. Recent work has evaluated the application of the extended Baum–Welch algorithm to MCE optimization [47].

Effect of Model Size: For the name recognition task and the JUPITER task, the effect of MCE-based discriminative training was particularly pronounced for small models, as shown in Table VI and Fig. 7. On the Corpus of Spontaneous Japanese, the relative reduction in error rate resulting from MCE training was not as strongly linked to model size, with some of the largest error reductions coming for the largest models, as shown in Tables III and IV. Clearly, the interest of discriminative training is not merely to improve the performance of small models, but also to generate top state-of-the-art performance, using whatever size is necessary. This goal was achieved for all the tasks considered: the MCE trained models significantly outperformed the best ML models.

Benefits of High-Performing Small Models: At the same time, the merits of improved performance with small models should not be discounted. Small models often result in lighter computation requirements for the same performance, and by definition lead to lighter storage requirements. The latter factor can be crucial for speech recognition implementations on small computational devices, such as PDAs and cell phones.

Towards a Lattice-Based MCE Update: All but one set of MCE results presented here were generated using (3), which is to say, a 1-best implementation of (4). [The remaining set of results used just 30 incorrect strings to implement (4)]. The gains in performance reported in this article appear to us to be especially significant when one considers the simplicity of the 1-best/N-best approach used. Nonetheless, a much richer implementation of (4) may yield significantly better results. In particular, on the large vocabulary and highly unconstrained CSJ task, where the difference between the top few N recognition outputs is typically limited to small, localized variations, a much

larger set of competitors may be desirable for discriminative training. Note that the CSJ RERRs reported here, around 7%, are low compared to the RERRs for the other tasks. They are also low compared to MMI results reported for tasks such as Switchboard [11]. This result may show the limitations of the 1-best or N-best paradigm, calling for a more sophisticated lattice-based MCE update, along the lines first described in [27] and recently evaluated in [47]. There are other explanations for the CSJ results, though. The baseline performance for this task may be very "language model heavy," in the sense that the language model may be doing a larger share of the work than the acoustic model, compared to the other tasks examined in this article. As a result, discriminative training of the acoustic model may not have as large an impact. Consistent with this point is the possibility that the very large number of homonyms in Japanese may limit the effect of discriminative training of acoustic models. Furthermore, speaker adaptive training could have a large impact on this task.

MCE, Corrective Training, and MMI: The scope of the present study was limited to showing what can be achieved with MCE, and not to comparing MCE with the more established approach to large scale discriminative training, MMI (or its word accuracy oriented successor, MPE). Nonetheless, a clear benefit to MCE-style separation-oriented training compared to the MMI-related corrective training approach was described here for the JUPITER task. Corrective training can be seen as a rough approximation to MMI, using the single top recognition output to represent the denominator of the MMI criterion function. However, it is not hard to implement a substantially better version of MMI by using more recognition outputs (with either an N-best list or a lattice) in tandem with acoustic scaling. The reader is referred to [47] for a comparison of MCE, MMI, and MPE, using lattice-based updates for all three methods.

IX. CONCLUSION

This article reviewed the MCE framework for discriminative training of HMM-based speech recognition systems and reported MCE results on challenging large vocabulary speech recognition tasks. Several batch-oriented optimization methods suitable for training on large databases using multiple computers were described and evaluated. It was shown that recognition lattices, represented as WFSTs, can be used successfully to speed up the MCE training procedure. The use of a *string set* level discriminant function, implemented using WFSTs, was proposed as a means to make the discriminative training process focus on correct recognition of the desired keyword sequence, while ignoring irrelevant filler words.

Overall, compared to traditional design based on ML/Viterbi Training, MCE-based discriminative training on these tasks yielded significant gains in recognition accuracy and system compactness. On the 22 k name recognition task, an acoustic model with 748 Gaussians, trained with MCE, outperformed baseline ML models with up to 10 940 Gaussians. The best MCE-trained model on this task outperforms the best ML model by a relative error rate difference of 15.5%, and has nearly three times fewer parameters. A similar pattern was observed on the 2 k and 33 k word JUPITER weather information tasks examined. On the Corpus of Spontaneous Japanese lecture speech transcription task, relative improvements of 7.1% and 7.4% were obtained over the ML baseline when evaluating with 30 k and 100 k word trigrams, respectively. These results were all obtained using a rather simple 1-best or N-best MCE update. Significant additional improvements may come from the use of methods such as lattice-based MCE updating and speaker adaptive training, as well as from the extension of the MCE formalism to reflect word accuracy rather than string accuracy, in a manner related to similar extensions to MMI [44], [45].

APPENDIX

A. MCE Gradient

The derivatives for the MCE loss function have been described in several sources [22], [75] but the reader may find the following summary useful.

Here only the gradient for a single token is described; if using a batch-oriented optimization algorithm, the gradient should be summed over all training tokens. Assuming that the speech token $\mathbf{x}_1^T$ belongs to string set $\mathcal{K}$, the derivative of the loss $\ell(d_{\mathcal{K}}(\mathbf{x}_1^T, \mathbf{\Lambda}))$ w.r.t. a component $\boldsymbol{\phi}_s$ of an observation probability $b_s(\mathbf{x}_t)$ on Viterbi state sequence $\Theta^j = (\theta_1^j, \theta_2^j, \ldots, \theta_t^j, \ldots, \theta_T^j)$ for a string S_j is

$$\frac{\partial \ell_{\mathcal{K}}}{\partial \boldsymbol{\phi}_s} = \frac{\partial \ell_{\mathcal{K}}}{\partial d_{\mathcal{K}}} \frac{\partial d_{\mathcal{K}}}{\partial g_j} \sum_{t|\theta_t^j = s} \frac{1}{b_s(\mathbf{x}_t)} \frac{\partial b_s(\mathbf{x}_t)}{\partial \boldsymbol{\phi}_s} \tag{14}$$

where the abbreviations $\ell_{\mathcal{K}} = \ell(d_{\mathcal{K}})$ and $g_j = g_j(\mathbf{x}_1^T, \mathbf{\Lambda})$ were used. Furthermore, from (6)

$$\frac{\partial \ell_{\mathcal{K}}}{\partial d_{\mathcal{K}}} = \alpha \ell_{\mathcal{K}} (1 - \ell_{\mathcal{K}}) \tag{15}$$

and from (5) used with the general form, (4)

$$\frac{\partial d_{\mathcal{K}}}{\partial g_j} = \begin{cases} -1, & S_j \in \mathcal{K} \\ \frac{e^{g_j \psi}}{\sum_{i|S_i \notin \mathcal{K}}^{M} e^{g_i \psi}}, & S_j \notin \mathcal{K}. \end{cases} \tag{16}$$

Assuming a large value of ψ (corresponding to the use of (3) rather than (4)), the latter expression reduces to

$$\frac{\partial d_{\mathcal{K}}}{\partial g_j} = \begin{cases} -1, & S_j \in \mathcal{K} \\ 1, & j = \arg\max_{i|S_i \notin \mathcal{K}} g_i \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

In this case, the derivative only exists for the correct and best incorrect strings.

In practice, $\partial \ell_{\mathcal{K}} / \partial \boldsymbol{\phi}_s$ is accumulated along Θ^j for each string S_j, adding to the partial derivative of the loss function with respect to each component $\boldsymbol{\phi}_s$, which potentially ranges over all mixing weight, mean vector, and covariance components. When the number of possible strings is prohibitively large, only the top

incorrect strings need be used to form an approximation of (4). Accumulating in this manner over all training tokens allows one to form the overall gradient, that can then be plugged into the optimization procedure.

We now specify the rest of the partial derivatives. The Gaussian mixture density $b_s(\mathbf{x}_t)$ used in the definition of $g_j(\mathbf{x}_1^T, \mathbf{\Lambda})$ in (1) is defined as

$$b_s(\mathbf{x}_t) = \sum_{i=1}^{I_s} c_{s,i} N(\mathbf{x}_t, \boldsymbol{\mu}_{s,i}, \mathbf{\Sigma}_{s,i}) \tag{18}$$

where $\boldsymbol{\mu}_{s,i}$ is a vector of D components $\mu_{s,i,d}$, and $\mathbf{\Sigma}_{s,i}$ is a diagonal matrix of D components $\sigma^2_{s,i,d}$. Using the abbreviation $N_{s,i}(\mathbf{x}_t) = N(\mathbf{x}_t, \boldsymbol{\mu}_{s,i}, \mathbf{\Sigma}_{s,i})$, the partial derivatives of $b_s(\mathbf{x}_t)$ with respect to the transformed mixing weight $\bar{c}_{s,i} = \log c_{s,i}$, mean vector component $\mu_{s,i,d}$, and transformed inverse covariance component $\bar{\sigma}_{s,i,d} = \log \sigma^{-1}_{s,i,d}$, are, respectively,

$$\begin{aligned}
\frac{\partial b_s(\mathbf{x}_t)}{\partial \bar{c}_{s,i}} &= c_{s,i} N_{s,i}(\mathbf{x}_t) \\
\frac{\partial b_s(\mathbf{x}_t)}{\partial \mu_{s,i,d}} &= c_{s,i} N_{s,i}(\mathbf{x}_t)(x_{t,d} - \mu_{s,i,d}) \\
\frac{\partial b_s(\mathbf{x}_t)}{\partial \bar{\sigma}_{s,i,d}} &= c_{s,i} N_{s,i}(\mathbf{x}_t) \left(1 - e^{2\bar{\sigma}_{s,i,d}} (x_{t,d} - \mu_{s,i,d})^2\right).
\end{aligned} \tag{19}$$

These terms are then used to expand $\partial b_s(\mathbf{x}_t)/\partial \boldsymbol{\phi}_s$ in (14). Adaptation of $\bar{c}_{s,i}$, followed by the back transformation $c_{s,i} = \exp(\bar{c}_{s,i})$ during parameter updating, enforces the constraint that the mixing weights must stay positive. The additional constraint that the mixing weights must sum to one can be maintained simply by renormalizing the weights after each iteration of MCE (see [19] for a transformation-based approach). Adaptation of the transformed inverse covariance term, $\bar{\sigma}_{s,i,d}$, results in greater numerical accuracy than adaptation of $\sigma_{s,i,d}$ itself. Finally, in the interest of numerical stability, a division by $\sigma^2_{s,i,d}$ term has been dropped from the true derivative for the mean [19].

ACKNOWLEDGMENT

The authors would like to thank the MIT Spoken Language Systems Group for providing many of the WFST tools used in this work. They wouls also like to thank T. Hori, T. Oba, and K. Shimomura with help on the CSJ references and language models.

REFERENCES

[1] B. M. D. Azzopardi, S. Semnani, and R. Wiseman, "Improving accuracy of telephony-based, speaker-independent speech recognition," in *Int. Conf. Spoken Language Processing*, 1998, vol. 2, pp. 301–304.

[2] J. Glass and T. Hazen, "Telephone-based conversational speech recognition in the JUPITER domain," in *Int. Conf. Spoken Language Processing*, 1998, vol. 4, pp. 1327–1330.

[3] P. Brown, "The acoustic-modeling problem in automatic speech recognition," Ph.D. dissertation, Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, 1987.

[4] T. Applebaum and B. B. Hanson, "Enhancing the discrimination of speaker independent hidden Markov models with corrective training," in *Proc. IEEE ICASSP*, 1989, vol. 1, pp. 302–305.

[5] H. Franco and A. Serralheiro, "Training HMMs using a minimum error approach," in *Proc. IEEE ICASSP*, 1990, vol. 1, pp. 357–360.

[6] A. Ljolje, Y. Ephraim, and L. Rabiner, "Estimation of hidden Markov model parameters by minimizing empirical error rate," in *Proc. IEEE ICASSP*, 1990, vol. 2, pp. 709–712.

[7] S. Katagiri, C.-H. Lee, and B.-H. Juang, "New discriminative training algorithms based on the generalized descent method," in *Proc. IEEE Worshop Neural Networks for Signal Processing*, 1991, pp. 299–308.

[8] H. Gish, "A minimum classification error, maximum likelihood, neural network," in *Proc. IEEE ICASSP*, 1992, vol. 2, pp. 289–292.

[9] J. Hampshire, "A differential theory of learning for efficient statistical pattern recognition," Ph.D. dissertation, Dept. Elect. Comput. Eng., Carnegie Mellon Univ., Pittsburgh, PA, 1991.

[10] V. Valtchev, J. J. Odell, P. C. Woodland, and S. J. Young, "Lattice-based discriminative training for large vocabulary speech recognition," in *Proc. Int. Conf. Spoken Language Processing*, 1996, vol. 2, pp. 605–609.

[11] P. Woodland and D. Povey, "Large scale discriminative training of hidden Markov models for speech recognition," *Comput. Speech Lang.*, vol. 16, pp. 25–47, 2002.

[12] L. Nguyen and B. Xiang, "Light supervision in acoustic model training," in *Proc. IEEE ICASSP*, 2004, vol. 1, pp. 185–188.

[13] G. Evermann, H. Chan, M. Gales, B. Jia, D. Mrva, P. Woodland, and K. Yu, "Training LVCSR systems on thousands of hours of data," in *Proc. IEEE ICASSP*, 2005, vol. 1, pp. 209–212.

[14] E. McDermott, A. Biem, S. Tenpaku, and S. Katagiri, "Discriminative training for large vocabulary telephone-based name recognition," in *Proc. IEEE ICASSP*, 2000, vol. 6, pp. 3739–3742.

[15] E. McDermott and T. J. Hazen, "Minimum classification error training of landmark models for real-time continuous speech recognition," in *Proc. IEEE ICASSP*, 2004, vol. 1, pp. 937–940.

[16] E. McDermott and S. Katagiri, "Minimum classification error for large scale speech recognition tasks using weighted finite state transducers," in *Proc. IEEE ICASSP*, 2005, vol. 1, pp. 113–116.

[17] S. Katagiri, C.-H. Lee, and B.-H. Juang, "A generalized probabilistic descent method," in *Proc. Acoustical Society of Japan, Ser. Fall Meeting* ser. Fall Meeting, Sep. 1990, pp. 141–142.

[18] B.-H. Juang and S. Katagiri, "Discriminative Learning for minimum error classification," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 40, no. 12, pp. 3043–3054, Dec. 1992.

[19] W. Chou, B.-H. Juang, and C.-H. Lee, "Segmental GPD training of HMM based speech recognizer," in *Proc. IEEE ICASSP*, Mar. 1992, vol. 1, pp. 473–476.

[20] S. Katagiri, B.-H. Juang, and C.-H. Lee, "Pattern recognition using a family of design algorithms based upon the generalized probabilistic descent method," *Proc. IEEE*, vol. 86, no. 11, pp. 2345–2373, Nov. 1998.

[21] M. Mohri, F. Pereira, and M. Riley, "Weighted finite state transducers in speech recognition," in *Proc. Automatic Speech Recognition Workshop*, 2000, pp. 97–106.

[22] E. McDermott, "Discriminative training for speech recognition," Ph.D. dissertation, School Eng., Waseda Univ., Tokyo, Japan, Mar. 1997.

[23] E. McDermott and S. Katagiri, "String-level MCE for continuous phoneme recognition," in *Proc. Eurospeech*, Rhodes, Greece, Sep. 1997, pp. 123–126.

[24] ——, "Prototype based discriminative training for various speech units," *Comput. Speech Lang.*, vol. 8, pp. 351–368, 1994.

[25] W. Chou, C.-H. Lee, and B.-H. Juang, "Minimum error rate training based on N-best string models," in *Proc. IEEE ICASSP*, 1993, vol. 2, pp. 652–655.

[26] W. Chou, C.-H. Lee, B.-H. Juang, and F.-K. Soong, "A minimum speech error rate pattern recognition approach to speech recognition," *Int. J. Pattern Recognition Artificial Intelligence, Special Issue on Speech Recognition for Different Languages*, vol. 8, no. 1, pp. 5–31, 1994.

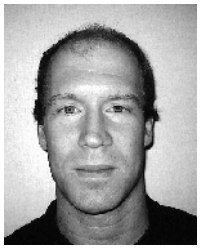
[27] R. Schlueter, W. Macherey, B. Muller, and H. Ney, "Comparison of discriminative training criteria and optimization methods for speech recognition," *Speech Communication*, vol. 34, no. 3, pp. 287–310, 2001.

[28] H. Jiang, O. Siohan, F.-K. Soong, and C.-H. Lee, "A dynamic in-search discriminative training approach for large vocabulary speech recognition," in *Proc. ICASSP*, 2002, vol. 1, pp. 113–116.

[29] T. Kawahara, H. Nanjo, T. Shinozaki, and S. Furui, "Benchmark test for speech recognition using the corpus of spontaneous japanese," in *Proc. Spontaneous Speech Processing Recognition Workshop*, Tokyo, Japan, 2003, pp. 135–138.

[30] A. Robinson, "Application of recurrent nets to phone probability estimation," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 298–305, Mar. 1994.

[31] Y. Normandin, R. Lacouture, and R. Cardin, "MMIE training for large vocabulary continuous speech recognition," in *Proc. Int. Conf. Spoken Language Processing*, 1994, vol. 3, pp. 1367–1370.
[32] B. Liu, H. Jiang, J.-L. Zhou, and R.-H. Wang, "Discriminative training based on the criterion of least phone competing tokens for large vocabulary speech recognition," in *Proc. IEEE ICASSP*, 2005, vol. 1, pp. 117–120.
[33] E. McDermott and S. Katagiri, "A derivation of minimum classification error from the theoretical classification risk using Parzen estimation," *Comput. Speech Lang.*, vol. 18, pp. 107–122, 2004.
[34] R. Schlueter and H. Ney, "Model-based MCE bound to the true Bayes' error," *IEEE Signal Process. Lett.*, vol. 8, no. 5, pp. 131–133, May 2001.
[35] A. Biem, S. Katagiri, E. McDermott, and B.-H. Juang, "An application of discriminative feature extraction to filter-bank based speech recognition," *IEEE Trans. Speech Audio Process.*, vol. 9, no. 2, pp. 96–110, Mar. 2001.
[36] R. Chengalvarayan and L. Deng, "HMM-based speech recognition using state-dependent, discriminatively derived transforms on Mel-warped DFT features," *IEEE Trans. Speech Audio Process.*, vol. 5, no. 3, pp. 243–256, May 1997.
[37] L. Deng, K. Wang, and W. Chou, "Guest editorial," *IEEE Signal Process. Mag.*, vol. 22, no. 5, pp. 12–14, Sep. 2005.
[38] A. Biem, "Minimum classification error training of hidden Markov models for handwriting recognition," in *Proc. IEEE ICASSP*, 2001, vol. 3, pp. 1529–1532.
[39] C. Yen, S.-S. Kuo, and C.-H. Lee, "Minimum error rate training for PHMM-based text recognition," *IEEE Trans. Image Process.*, vol. 8, no. 8, pp. 1120–1124, Aug. 1999.
[40] S. Gao, W. Wu, C.-H. Lee, and T.-S. Chua, "A maximal figure-of-merit learning approach to text categorization," in *Proc. ACM SIGIR Conf. Research and Development in Information Retrieval*, 2003, pp. 174–181.
[41] F. J. Och, "Minimum error rate training in statistical machine translation," in *ACL: Proc. Assoc. Comput. Ling.*, 2003, pp. 160–167.
[42] Y. Normandin, "Hidden Markov models, maximum mutual information estimation, and the speech recognition problem," Ph.D. dissertation, Dept. Elect. Eng., McGill Univ., Montreal, QC, Canada, 1991.
[43] P. S. Gopalakrishnan, D. Kanevsky, A. Nadas, D. Nahamoo, and M. Picheny, "Decoder selection based on cross-entropies," in *Proc. IEEE ICASSP*, 1988, vol. 1, pp. 20–23.
[44] D. Povey and P. Woodland, "Minimum phone error and I-smoothing for improved discriminative training," in *Proc. IEEE ICASSP*, 2002, vol. 1, pp. 105–108.
[45] D. Povey, "Discriminative training for large vocabulary speech recognition," Ph.D. dissertation, Univ. Cambridge, Cambridge, U.K., 2004.
[46] T. Hori, C. Hori, and Y. Minami, "Fast on-the-fly composition for weighted finite-state transducers in 1.8 million-word vocabulary continuous speech recognition," in *Proc. Int. Conf. Spoken Language Processing*, 2004, vol. 1, pp. 289–292.
[47] W. Macherey, L. Haferkamp, R. Schlueter, and H. Ney, "Investigations on error minimizing training criteria for discriminative training in automatic speech recognition," in *Proc. Eurospeech*, 2005, pp. 2133–2136.
[48] R. Schlueter, W. Macherey, S. Kanthak, H. Ney, and L. Welling, "Comparison of optimization methods for discriminative training criteria," in *Proc. Eurospeech*, 1997, vol. 1, pp. 15–18.
[49] Q. Li and B.-H. Juang, "A new algorithm for fast discriminative training," in *Proc. IEEE ICASSP*, 2002, vol. 1, pp. 97–100.
[50] X. He and W. Chou, "Minimum classification error linear regression for acoustic model adaptation of continuous density HMMs," in *Proc. IEEE ICASSP*, 2003, vol. 1, pp. 556–559.
[51] S. Amari, "A theory of adaptive pattern classifiers," *IEEE Trans. Electron. Comput.*, vol. EC-16, no. 3, pp. 299–307, Mar. 1967.
[52] L. Bottou, "Une approche théorique de l'apprentissage connectionniste: Application à la reconnaissane de la parole," Ph.D. dissertation, Univ. de Paris Sud, Paris, France, 1991.
[53] S. E. Fahlman, An empirical study of learning speed in back-propagation networks Canergie Mellon Univ., Pittsburgh, PA, 1988, Tech. Rep..
[54] S. Kapadia, V. Valtchev, and S. J. Young, "MMI training for continuous phoneme recognition on the TIMIT database," in *Proc. ICASSP*, 1993, vol. 2, pp. 491–494.
[55] R. Battiti, "First- and second- order methods for learning: Between steepest descent and newton's method," *Neural Comput.*, vol. 4, pp. 141–166, 1992.
[56] J. Leroux and E. McDermott, "Optimization methods for discriminative training," in *Proc. Eurospeech*, 2005, pp. 3341–3344.
[57] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. IEEE Intl. Conf. Neural Networks*, San Francisco, CA, 1993, pp. 586–591 [Online]. Available: http://www.citeseer.ist.psu.edu/riedmiller93direct.html
[58] D. Kanevsky, "A generalization of the Baum algorithm to functions on nonlinear manifolds," in *Proc. IEEE ICASSP*, 1995, pp. 473–476.
[59] K.-F. Lee and H.-W. Hon, "Speaker-independent phone recognition using hidden Markov models," *IEEE Trans. Acoustics, Speech, Signal Process.*, vol. 37, no. 11, pp. 1641–1648, Nov. 1989.
[60] P. Clarkson and P. Moreno, "On the use of support vector machines for phonetic classification," in *Proc. IEEE ICASSP*, 1999, vol. 2, pp. 585–588.
[61] A. Halberstadt and J. Glass, "Heterogeneous measurements and multiple classifiers for speech recognition," in *Proc. ICSLP*, Sydney, Australia, Dec. 1998, pp. 995–998.
[62] A. Gunawardana, M. Mahajan, A. Acero, and J. Platt, "Hidden conditional random fields for phone classification," in *Proc. Eurospeech*, 2005, pp. 1117–1120.
[63] J. Odell, "The use of context in large vocabulary speech recognition," Ph.D. dissertation, Univ. Cambridge, Cambridge, U.K., 1995.
[64] S. Young and P. Woodland, "State clustering in hidden Markov model-based continuous speech recognition," *Comput. Speech Lang.*, vol. 8, pp. 369–383, 1994.
[65] L. Wang and P. Woodland, "Discriminative adaptive training using the MPE criterion," in *Proc. ASRU*, 2003, pp. 279–284.
[66] H. Hermansky and N. Morgan, "RASTA processing of speech," *IEEE Trans. Speech Audio Process.*, vol. 2, no. 4, pp. 578–589, Mar. 1994.
[67] V. Zue *et al.*, "JUPITER: A telephone-based conversational interface for weather information," *IEEE Trans. Speech Audio Process.*, vol. 8, no. 1, pp. 85–96, Jan. 2000.
[68] J. Glass, "A probabilistic framework for segment-based speech recognition," *Comput. Speech Lang.*, vol. 17, no. 2–3, pp. 137–152, Apr.–Jul. 2003.
[69] T. Hazen, S. Seneff, and J. Polifroni, "Recognition confidence scoring for use in speech understanding systems," *Comput. Speech Lang.*, vol. 16, no. 1, pp. 49–67, Jan. 2002.
[70] I. Bazzi and J. Glass, "A multi-class approach for modelling out-of-vocabulary words," in *Proc. ICSLP*, Denver, CO, Sep. 2002, pp. 1613–1616.
[71] C. Meyer and G. Rose, "Improved noise robustness by corrective and rival training," in *Proc. ICASSP*, 2001, vol. 1, pp. 293–296.
[72] D. Willett, "Error-weighted discriminative training for HMM parameter estimation," in *Proc. Int. Conf. Spoken Language Processing*, 2004, pp. 1661–1664.
[73] J. E. Dennis and R. B. Schnabel, "Numerical methods for unconstrained optimization and nonlinear equations," in *Classics in Applied Mathematics*. Philadelphia, PA: SIAM, 1996.
[74] L. E. Scales, *Introduction to Non-Linear Optimization*. New York: Macmillan, 1985.
[75] D. Rainton and S. Sagayama, "Minimum error classification training of HMMs—Implementation details and experimental results," *J. Acoust. Soc. Jpn.*, vol. 13, no. 6, pp. 379–387, 1992.



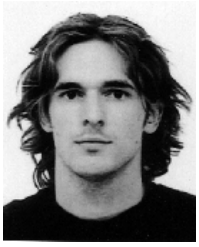
Erik McDermott (M'03) received the B.S. degree from Stanford University, (Symbolic Systems Program), Stanford, CA, in 1987 and the Ph.D. degree from the School of Engineering, Waseda University, Tokyo, Japan, in 1997.

He has been a Research Specialist at Nippon Telegraph and Telephone Corporation (NTT), Communication Science Laboratories, Kyoto, Japan, since December 1999. From 1987 to 1991, he worked at ATR Auditory and Visual Perception Research Laboratories, Kyoto, and at ATR Human Information Processing Laboratories from 1991 to 1999. He spent six months in 2000 in the SRI Speech Technology Laboratory, and one month in 2003 in the Massachusetts Institute of Technology Spoken Language Systems Group. His research has primarily been in speech recognition, with a focus on acoustic modeling paradigms for hidden Markov model (HMM)-based speech recognition. His recent work has focused on improving both theoretical and practical aspects of discriminative training applied to speech recognition system design.

Timothy J. Hazen (M'04) received the B.S., M.S., and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, in 1991, 1993, and 1998, respectively.

He is a Research Scientist at the MIT Computer Science and Artificial Intelligence Laboratory, where he works in the areas of automatic speech recognition, automatic person identification, multimodal speech processing, and conversational speech systems.

Dr. Hazen is an Associate Editor for the IEEE TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING.



Jonathan Le Roux received the degree in mathematics from the Ecole Normale Supérieure, Paris, France, the M.Sc. degree in partial differential equations from the University of Paris XI, Paris, in 2001, and the M.Sc. degree in stochastic processes from the University of Paris VI, Paris, in 2003 and is currently pursuing the Ph.D. degree at the Graduate School of Computer Science, Telecommunications and Electronics of Paris, University of Paris VI, and in the Graduate School of Information Science and Technology, Department of Information Physics and Computing, University of Tokyo, Tokyo, Japan.

Atsushi Nakamura (M'03) received the B.E., M.E., and Dr. Eng. degrees from Kyushu University, Fukuoka, Japan, in 1985, 1987, and 2001, respectively.

In 1987, he joined Nippon Telegraph and Telephone Corporation (NTT), Musashino Electrical Communication Laboratories, Tokyo, Japan, where he was engaged in the research and development of network service platforms, including studies on application of speech processing technologies into network services. From 1994 to 2000, he was with Advanced Telecommunications Research (ATR) Institute, Kyoto, Japan, as a Senior Researcher, where he was engaged in research on spontaneous speech recognition, the construction of spoken language databases, and the development of speech translation systems. Since April, 2000, he has been with NTT Communication Science Laboratories, Kyoto. His research interests include acoustic modeling of speech, speech recognition and synthesis, spoken language processing systems, speech production and perception, computational phonetics and phonology, and the application of learning theories to signal analysis and modeling.

Dr. Nakamura is a member of the Institute of Electronics, Information and Communication Engineering (IEICE) and the Acoustical Society of Japan (ASJ). He received the Best Paper Award from the IEICE in 2004.

Shigeru Katagiri (M'88–SM'97–F'01) received the B.E. degree in electrical engineering and the M.E. and Dr.Eng. degrees in information engineering from Tohoku University, Sendai, Japan, in 1977, 1979, and 1982, respectively.

From 1982 to 1986, he was with the Electrical Communication Laboratories, Nippon Telegraph and Telephone Public Corporation (currently NTT), Tokyo, Japan. From 1986 to 1998, he was with the Advanced Telecommunications Research Institute, Int. (ATR), Kyoto, Japan. Since 1999, he has been with NTT Communication Science Laboratories (CS Labs), Kyoto, Japan, where he currently serves as Director of NTT CS Labs. In addition to the above, from 1989 to 1990, he was a Visiting Researcher in the Speech Research Department, AT&T Bell Laboratories, Murray Hill, NJ. From 1998 to 2004, he also served as an Adjunct Professor at the Graduate School of Kyoto University. His research interests include studies on pattern recognition and multimodal telecommunications.

Dr. Katagiri is an NTT R&D Fellow.