Discriminative n-gram language modeling [1]

Brian Roark [*,2]

Center for Spoken Language Understanding, OGI School of Science & Engineering at Oregon Health & Science University, 20000 NW Walker Rd., Beaverton, OR 97006, USA

Murat Saraclar [2]

Boğaziçi University
34342 Bebek, Istanbul, Turkey

Michael Collins

MIT CSAIL/EECS
Stata Center, Bldg 32-G484
Cambridge, MA 02139

Abstract

This paper describes discriminative language modeling for a large vocabulary speech recognition task. We contrast two parameter estimation methods: the perceptron algorithm, and a method based on maximizing the regularized conditional log-likelihood. The models are encoded as deterministic weighted finite state automata, and are applied by intersecting the automata with word-lattices that are the output from a baseline recognizer. The perceptron algorithm has the benefit of automatically selecting a relatively small feature set in just a couple of passes over the training data. We describe a method based on regularized likelihood that makes use of the feature set given by the perceptron algorithm, and initialization with the perceptron's weights; this method gives an addition 0.5% reduction in word error rate (WER) over training with the perceptron alone. The final system achieves a 1.8% absolute reduction in WER for a baseline first-pass recognition system (from 39.2% to 37.4%), and a 0.9% absolute reduction in WER for a multi-pass recognition system (from 28.9% to 28.0%).

Key words: Speech Recognition, Language Modeling, Discriminative Training

1 Introduction

A crucial component of any speech recognizer is the *language model* (LM), which assigns scores or probabilities to candidate output strings in a speech recognizer. The language model is used in combination with an acoustic model, to give an overall score to candidate word sequences that ranks them in order of probability or plausibility.

A dominant approach in speech recognition has been to use a "source-channel", or "noisy-channel" model. In this approach, language modeling is effectively framed as probability estimation: the language model's task is to define a distribution over the source – i.e., the possible strings in the language. Markov (n-gram) models are often used for this task, whose parameters are optimized to maximize the likelihood of a large amount of training text. Recognition performance is a direct measure of the effectiveness of a language model; an indirect measure which is frequently proposed within these approaches is the perplexity of the LM (i.e., the negative log probability it assigns to some held-out data set).

This paper explores alternative methods for language modeling, which complement the source-channel approach through discriminatively trained models. The language models we describe do not attempt to estimate a generative model $P(w)$ over strings. Instead, they are trained on acoustic sequences with their transcriptions, in an attempt to directly optimize error-rate. In particular, we use the perceptron algorithm to build a discriminative global linear model, and we also explore global conditional log-linear models (GCLMs) as a parameter estimation method. [3] We describe how these models can be trained over lattices that are the output from a baseline recognizer. We also give a number of experiments comparing the two approaches. The perceptron method gave a 1.3% absolute improvement in first-pass recognition error on the Switchboard domain, and a 0.5% multi-pass improvement; the GCLM methods we describe provide absolute gains of 1.8% for first-pass recognition and 0.9% for multi-pass.

A central issue that we discuss is feature selection. The number of distinct n-grams

* Corresponding Author
Email addresses: roark@cslu.ogi.edu (Brian Roark), murat.saraclar@boun.edu.tr (Murat Saraclar), mcollins@csail.mit.edu (Michael Collins).

[1] Parts of this study were presented in conferences (Roark et al., 2004a,b).

[2] The authors were with AT&T Labs-Research when the work presented here was performed.

[3] In a previous paper (Roark et al., 2004b), we described these models as an instance of Conditional random fields (CRFs) (Lafferty et al., 2001). While certainly similar to CRFs, our models are technically not CRFs (see section 2.2 for discussion), so we will not use this term in the current paper for our models.

in our training data is close to 45 million, and we show that GCLM training converges very slowly even when trained with a subset (of size 12 million) of these features. Because of this, we have explored methods for picking a small subset of the available features. [4] The perceptron algorithm can be used as one method for feature selection, selecting around 1.5 million features in total. The GCLM trained with this feature set, and initialized with parameters from perceptron training, converges much more quickly than other approaches, and also gives the best performance on the held-out set. We explored other approaches to feature selection, but found that the perceptron-based approach gave the best results in our experiments.

While we focus on n-gram models, we stress that our methods are applicable to more general language modeling features – for example, syntactic features, as explored in, e.g., Khudanpur and Wu (2000). We intend to explore methods with new features in the future. Experimental results with n-gram models on 1000-best lists show a very small drop in accuracy compared to the use of lattices. This is encouraging, in that it suggests that models with more flexible features than n-gram models, which therefore cannot be efficiently used with lattices, may not be unduly harmed by their restriction to n-best lists.

1.1 Related Work

Large vocabulary ASR has benefited from discriminative estimation of hidden Markov model (HMM) parameters in the form of maximum mutual information estimation (MMIE) or conditional maximum likelihood estimation (CMLE). Woodland and Povey (2000) have shown the effectiveness of lattice-based MMIE/CMLE in large–scale ASR tasks such as Switchboard. Povey and Woodland (2002) have introduced minimum phone error (MPE) and minimum word error (MWE) criteria for the discriminative training of HMM systems. In fact, state-of-the-art acoustic modeling, as seen, for example, at annual Switchboard evaluations, invariably includes some kind of discriminative training.

Discriminative estimation of language models has also been proposed in recent years. Jelinek (1996) suggested an acoustic–sensitive language model whose parameters are estimated by minimizing $H(W|A)$, the expected uncertainty of the spoken text W given A, the acoustic sequence. Stolcke and Weintraub (1998) experimented with various discriminative approaches, including MMIE, with mixed results. This work was followed up with some success by Stolcke et al. (2000) where an "anti-LM", estimated from weighted N-best hypotheses of a baseline ASR system, was used with a negative weight in combination with the baseline LM. Chen et al. (2000) presented a method based on changing the trigram counts discriminatively, together with changing the lexicon to add new words. Kuo et al.

[4] Note also that in addition to concerns about training time, a language model with fewer features is likely to be considerably more efficient when decoding new utterances.

(2002) used the generalized probabilistic descent (GPD) algorithm to train relatively small language models which attempt to minimize string error rate on the DARPA Communicator task. This is an instance of the widely known minimum classification error (MCE) training. Banerjee et al. (2003) used a language model modification algorithm in the context of a reading tutor that listens. Their algorithm first uses a classifier to predict what effect each parameter has on the error rate, and then modifies the parameters to reduce the error rate based on this prediction.

Of the above approaches, only Stolcke and Weintraub (1998) experimented with an objective function of the sort used to estimate the GCLMs we describe in this paper: a straightforward maximum conditional likelihood objective, conditioned on the entire input sequence. [5] This was one of two objectives they investigated for estimation of a unigram model, and it yielded no gain over their baseline unigram model. Our GCLM approach is also similar to the work of Rosenfeld et al. (2001), with the difference that GCLMs are estimated using discriminative methods.

Various other approaches have been proposed that attempt to estimate parameters in a way that minimizes word error rate (Bahl et al., 1993; Goel and Byrne, 2000; Mangu et al., 2000; Mangu and Padmanabhan, 2001; Ringger and Allen, 1996), ranging from discriminative parameter adjustment algorithms (Bahl et al., 1993) to post-processing on recognizer output (Mangu and Padmanabhan, 2001; Ringger and Allen, 1996) and confusion network construction (Mangu et al., 2000; Mangu and Padmanabhan, 2001). One of the earliest papers on the topic (Bahl et al., 1993) motivated its approach by reference to the perceptron algorithm, and, proposed a technique for corrective training of discrete output HMM parameters for acoustic modeling.

2 Global Linear Models

This section describes a general framework, global linear models, and two parameter estimation methods within the framework, the perceptron algorithm and a method based on maximizing the regularized conditional log-likelihood. The linear models we describe are general enough to be applicable to a diverse range of NLP and speech tasks – this section gives a general description of the approach. In section 3 of the paper we describe how global linear models can be applied to speech recognition. In particular, we focus on how the decoding and parameter estimation problems can be implemented over lattices using finite-state techniques.

We follow the framework outlined in Collins (2002, 2004). The task is to learn a mapping from inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. We assume the following compo-

[5] Note, however, that the objective in Stolcke and Weintraub (1998) was unregularized, which, based on our results, is one likely reason that it failed to yield improvements.

Inputs: Training examples (x_i, y_i)
Initialization: Set $\bar{\alpha} = 0$
Algorithm:
For $t = 1 \ldots T$
For $i = 1 \ldots N$
Calculate $z_i = \text{argmax}_{z \in \mathbf{GEN}(x_i)} \Phi(x_i, z) \cdot \bar{\alpha}$
If($z_i \neq y_i$) then $\bar{\alpha} = \bar{\alpha} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$
Output: Parameters $\bar{\alpha}$

Fig. 1. A variant of the perceptron algorithm. The value for T, the number of passes over the training set, is chosen by validation on a held out set. In practice we used the *averaged* parameter values after T iterations, see the text for details.

nents: (1) Training examples (x_i, y_i) for $i = 1 \ldots N$. (2) A function $\mathbf{GEN}$ which enumerates a finite set of candidates $\mathbf{GEN}(x) \subseteq \mathcal{Y}$ for each possible input x. (3) A **representation** Φ mapping each $(x, y) \in \mathcal{X} \times \mathcal{Y}$ to a feature vector $\Phi(x, y) \in \mathbb{R}^d$. (4) A **parameter vector** $\bar{\alpha} \in \mathbb{R}^d$.

The components $\mathbf{GEN}, \Phi$ and $\bar{\alpha}$ define a mapping from an input x to an output $F(x)$ through

$$F(x) = \underset{y \in \mathbf{GEN}(x)}{\text{argmax}} \; \Phi(x, y) \cdot \bar{\alpha} \tag{1}$$

where $\Phi(x, y) \cdot \bar{\alpha}$ is the inner product $\sum_s \alpha_s \Phi_s(x, y)$. The learning task is to set the parameter values $\bar{\alpha}$ using the training examples as evidence. The *decoding algorithm* is a method for searching for the y that maximizes Eq. 1.

2.1 The Perceptron algorithm

We now turn to methods for training the parameters $\bar{\alpha}$ of the model, given a set of training examples $(x_1, y_1) \ldots (x_N, y_N)$. This section describes the perceptron algorithm and the next section describes an alternative method, based on maximizing the regularized conditional log-likelihood.

The perceptron algorithm is shown in figure 1. At each training example (x_i, y_i), the current best-scoring hypothesis z_i is found, and if it differs from the reference y_i, then the weight of each feature is increased by the count of that feature in y_i and decreased by the count of that feature in z_i. The weights in the model are updated, and the algorithm moves to the next utterance.

We will now give a first theorem regarding the convergence of this algorithm. First, we need the following definition:

Definition 1 *Let* $\overline{\mathbf{GEN}}(x_i) = \mathbf{GEN}(x_i) - \{y_i\}$. *In other words* $\overline{\mathbf{GEN}}(x_i)$ *is the set of* incorrect *candidates for an example* x_i. *We will say that a training sequence* (x_i, y_i) *for* $i = 1 \ldots n$ *is* **separable with margin** $\delta > 0$ *if there exists some vector* $\mathbf{U}$ *with* $||\mathbf{U}|| = 1$

such that

$$\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i), \quad \mathbf{U} \cdot \Phi(x_i, y_i) - \mathbf{U} \cdot \Phi(x_i, z) \geq \delta \tag{2}$$

($||\mathbf{U}||$ *is the 2-norm of* $\mathbf{U}$*, i.e.,* $||\mathbf{U}|| = \sqrt{\sum_s \mathbf{U}_s^2}$*.)*

Next, define N_e to be the number of times an error is made by the algorithm in figure 1 – that is, the number of times that the condition $z_i \neq y_i$ is met at any point in the algorithm. We can then state the following theorem (see Collins, 2002, for a proof):

Theorem 1 *For any training sequence* (x_i, y_i) *that is separable with margin* δ*, for any value of* T*, then for the perceptron algorithm in figure 1*

$$N_e \leq \frac{R^2}{\delta^2}$$

where R *is a constant such that* $\forall i, \forall z \in \overline{\mathbf{GEN}}(x_i) \quad ||\Phi(x_i, y_i) - \Phi(x_i, z)|| \leq R$*.*

This theorem implies that if there is a parameter vector $\mathbf{U}$ which makes zero errors on the training set, then after at most $\frac{R^2}{\delta^2}$ passes over the training set the training algorithm will converge to parameter values with zero training errors.[6] A crucial point is that the number of mistakes is independent of the number of candidates for each example (i.e. the size of $\mathbf{GEN}(x_i)$ for each i), depending only on the separation of the training data, where separation is defined above. This is important because in ASR the number of candidates in $\mathbf{GEN}(x)$ is generally exponential in the length of the utterance. All of the convergence and generalization results in (Collins, 2002) depend on notions of separability rather than the size of $\mathbf{GEN}$.[7]

Two questions come to mind. First, are there guarantees for the algorithm if the training data is not separable? Second, how well does the algorithm generalizes to newly drawn test examples (under an assumption that both training and test examples are drawn from the same, unknown distribution $P(x, y)$)? Freund and Schapire (1999) discuss how the theory for classification problems can be extended to deal with both of these questions; Collins (2002) describes how these results apply to NLP problems.

[6] To see this, note that if the algorithm makes a complete pass over the training examples without making any errors, then it must have converged; and furthermore, in the worst case it makes $N_e \leq \frac{R^2}{\delta^2}$ passes over the training set, each with a single error, before converging.

[7] Note, however, that in practice as the size of $\mathbf{GEN}$ becomes larger, the separability of problems may well diminish, although this is not necessarily the case. Even so, the lack of direct dependence on $|\mathbf{GEN}(x)|$ for the perceptron algorithm is somewhat surprising. For example, under the same assumptions for the training set, the tightest known generalization bounds for the support vector machine or large margin solution (which explicitly searches for the parameter vector with the largest separation on training examples) contains a $\log |\mathbf{GEN}(x)|$ factor which is not present in the perceptron convergence or generalization bounds – see (Collins, 2002) for discussion. (Note that Taskar et al. (2003) do describe a tighter bound, but this depends on a modified definition of margins on the training examples.)

Following Collins (2002), we used the *averaged* parameters from the training algorithm in decoding held-out and test examples in our experiments. Say $\bar{\alpha}_i^t$ is the parameter vector after the i'th example is processed on the t'th pass through the data in the algorithm in figure 1. Then the averaged parameters $\bar{\alpha}_{AVG}$ are defined as $\bar{\alpha}_{AVG} = \sum_{i,t} \bar{\alpha}_i^t / NT$. Freund and Schapire (1999) originally proposed the averaged parameter method; it was shown to give substantial improvements over the regular perceptron in accuracy for tagging tasks in Collins (2002). The averaged perceptron can be seen as an approximation of the "voted perceptron", an algorithm also described by Freund and Schapire (1999). The voted perceptron is motivated by statistical bounds on generalization performance given in Freund and Schapire (1999).

2.2 Global Conditional Log-Linear Models

Global conditional log-linear models (GCLMs) use the parameters $\bar{\alpha}$ to define a conditional distribution over the members of $\mathbf{GEN}(x)$ for a given input x:

$$p_{\bar{\alpha}}(y|x) = \frac{1}{Z(x, \bar{\alpha})} \exp\left(\Phi(x, y) \cdot \bar{\alpha}\right) \tag{3}$$

where $Z(x, \bar{\alpha}) = \sum_{y \in \mathbf{GEN}(x)} \exp\left(\Phi(x, y) \cdot \bar{\alpha}\right)$ is a normalization constant that depends on x and $\bar{\alpha}$.

Early work on GCLMs for NLP applied them to parsing (Ratnaparkhi et al., 1994; Johnson et al., 1999). In (Ratnaparkhi et al., 1994), n-best output from an existing probabilistic parser was used to define $\mathbf{GEN}(x)$ for each input sentence x. In (Johnson et al., 1999) all parses from a deterministic parser defined the set $\mathbf{GEN}(x)$. In both of these papers $\Phi(x, y)$ was allowed to include essentially arbitrary features of parse trees y and their yields x.

Conditional random fields (CRFs) (Lafferty et al., 2001; Sha and Pereira, 2003; McCallum and Li, 2003; Pinto et al., 2003) are a sub-class of GCLMs that are particularly relevant to our problem. CRFs define $\mathbf{GEN}(x)$ to be the space of vectors $\{y_1, y_2 \ldots y_{l(x)}\} \in \mathcal{Y}_1 \times \mathcal{Y}_2 \ldots \times \mathcal{Y}_{l(x)}$, where each $\mathcal{Y}_k$ is a finite "label set" for the k'th random variable, and $l(x)$ is a "size" that can vary with the input x. As a simple example, in part-of-speech tagging, for a sentence x of length $l(x)$, $\mathcal{Y}_k$ would be the set of possible part-of-speech tags for the k'th word in the sentence. Crucially, in CRFs, $\Phi(x, y)$ is defined through a graphical structure over the variables $\{y_1, y_2 \ldots y_{l(x)}\}$. The representation $\Phi(x, y)$ is defined as a sum of feature-vectors associated with cliques in the graph. Under these assumptions, assuming that the underlying graph has good properties, [8] CRFs can be trained efficiently in spite

[8] More specifically, the graph must have relatively low "tree-width" for there to be efficient training and decoding algorithms.

of the size of $\mathbf{GEN}(x)$ being exponential in the number of labels $l(x)$. The training algorithm makes use of standard dynamic programming algorithms in Markov random fields to calculate the gradient of the log-likelihood under the model. See (Lafferty et al., 2001) for full details.

The GCLMs that we present for language modeling in the current paper are similar to CRFs, in that $\mathbf{GEN}(x)$ is again exponential in size, and we rely on the "local" nature of our feature vector representation $\Phi(x, y)$ to give us dynamic programming algorithms for training and decoding. As for CRFs, these algorithms are efficient in spite of the large size of $\mathbf{GEN}(x)$. Our approach differs from CRFs, however, in that our problem is not structured in an undirected graphical model, or Markov random field. Instead, we will make use of algorithms and concepts from weighted finite-state automata—as opposed to Markov random fields—in this paper.

Given the definition in Eq. 3, the log-likelihood of the training data under parameters $\bar{\alpha}$ is

$$LL(\bar{\alpha}) = \sum_{i=1}^{N} \log p_{\bar{\alpha}}(y_i|x_i) = \sum_{i=1}^{N} [\Phi(x_i, y_i) \cdot \bar{\alpha} - \log Z(x_i, \bar{\alpha})] \tag{4}$$

Note that this is very similar to the objective functions used in other discriminative training approaches. The objective function for MMIE/CMLE has the same form and the one for MCE could be considered as an extension.

Following Johnson et al. (1999) and Lafferty et al. (2001), we use a zero-mean Gaussian prior on the parameters resulting in the regularized objective function:

$$LL_R(\bar{\alpha}) = \sum_{i=1}^{N} [\Phi(x_i, y_i) \cdot \bar{\alpha} - \log Z(x_i, \bar{\alpha})] - \frac{||\bar{\alpha}||^2}{2\sigma^2} \tag{5}$$

The value σ dictates the relative influence of the log-likelihood term vs. the prior, and is typically estimated using held-out data. (For example, in the experiments in this paper we choose a value of σ that minimizes the word-error-rate on a development data set.) The optimal parameters under this criterion are $\bar{\alpha}^* = \text{argmax}_{\bar{\alpha}} LL_R(\bar{\alpha})$.

We use a *limited memory variable metric* method (Benson and Moré, 2002) to optimize LL_R. There is a general implementation of this method in the Tao/PETSc software libraries (Balay et al., 2002; Benson et al., 2002). This technique has been shown to be very effective in a variety of NLP tasks (Malouf, 2002; Wallach, 2002). The main interface between the optimizer and the training data is a procedure which takes a parameter vector $\bar{\alpha}$ as input, and in turn returns $LL_R(\bar{\alpha})$ as well as the gradient of LL_R at $\bar{\alpha}$. The derivative of the objective function with respect to a parameter α_s at parameter values $\bar{\alpha}$ is

$$\frac{\partial LL_R}{\partial \alpha_s} = \sum_{i=1}^{N} \left[\Phi_s(x_i, y_i) - \sum_{y \in \mathbf{GEN}(x_i)} p_{\bar{\alpha}}(y|x_i) \Phi_s(x_i, y) \right] - \frac{\alpha_s}{\sigma^2} \quad (6)$$

Note that $LL_R(\bar{\alpha})$ is a convex function [9], so that there are no issues with local maxima in the objective function, and the optimization methods we use converge to the globally optimal solution. The use of the Gaussian prior term $||\bar{\alpha}||^2/2\sigma^2$ in the objective function has been found to be useful in several NLP settings. It effectively ensures that there is a large penalty for parameter values in the model becoming too large – as such, it tends to control over-training. The choice of LL_R as an objective function can be justified as maximum a-posteriori (MAP) training within a Bayesian approach. An alternative justification comes through a connection to support vector machines and other large margin approaches. SVM-based approaches use an optimization criterion that is closely related to LL_R – see Collins (2004) for more discussion.

3 Linear models for speech recognition

We now describe how the formalism and algorithms in section 2 can be applied to language modeling for speech recognition.

3.1 The basic approach

As described in the previous section, linear models require definitions of $\mathcal{X}$, $\mathcal{Y}$, x_i, y_i, $\mathbf{GEN}$, Φ and a parameter estimation method. In the language modeling setting we take $\mathcal{X}$ to be the set of all possible acoustic inputs; $\mathcal{Y}$ is the set of all possible strings, Σ^*, for some vocabulary Σ. Each x_i is an utterance (a sequence of acoustic feature-vectors), and $\mathbf{GEN}(x_i)$ is the set of possible transcriptions under a first pass recognizer. ($\mathbf{GEN}(x_i)$ is a huge set, but will be represented compactly using a lattice – we will discuss this in detail shortly). We take y_i to be the member of $\mathbf{GEN}(x_i)$ with lowest error rate with respect to the reference transcription of x_i.

All that remains is to define the feature-vector representation, $\Phi(x, y)$. In the general case, each component $\Phi_s(x, y)$ could be essentially any function of the acoustic input x and the candidate transcription y. The first feature we define is $\Phi_0(x, y)$,

[9] This is a well known property of the regularized likelihood function in Eq. 5. Convexity follows because: 1) $\frac{||\bar{\alpha}||^2}{2\sigma^2}$ is convex in $\bar{\alpha}$; 2) $\log Z(x_i, \bar{\alpha})$ is convex in $\bar{\alpha}$ for any value of x_i (the latter property is central to the theory of exponential families in statistics, see for example section 3 of Wainwright and Jordan (2002)); 3) a function that is a sum of convex functions is also convex.

which is defined as *the log-probability of x, y in the lattice produced by the baseline recognizer*. Thus this feature will include contributions from the acoustic model and the original language model.

Note that the baseline recognizer has a standard language model weight parameterization, which dictates how the language model combines with the acoustic model to produce the baseline score for the word sequence. We did not change this standard parameterization for the work in this paper, i.e. we took the log score output by the baseline recognizer unmodified for use in training and applying the models. The lattice is deterministic, so that any word sequence has at most one path through the lattice. Thus multiple time-alignments for the same word sequence are not represented; the path associated with a word sequence is the path that receives the highest probability among all competing time alignments for that word sequence.

The remaining features are restricted to be functions over the transcription y alone and they track all n-grams up to some length (say $n = 3$), for example:

$$\Phi_1(x, y) = \text{Number of times "}\textit{the the of}\text{" is seen in } y$$

At one level of abstraction, features of this form are introduced for *all* n-grams up to length 3 seen in some training data lattice, i.e., n-grams seen in any word sequence within the lattices. In practice, we consider methods that search for sparse parameter vectors $\bar{\alpha}$, thus assigning many n-grams 0 weight. This will lead to more efficient algorithms, which avoid dealing explicitly with the entire set of n-grams seen in training data.

3.2 Implementation using WFA

We now give a brief sketch of how weighted finite-state automata (WFA) can be used to implement linear models for speech recognition. There are several papers describing the use of weighted automata and transducers for speech in detail, e.g., Mohri et al. (2002), but for clarity and completeness this section gives a brief description of the operations which we use.

For our purpose, a WFA A is a tuple $(\Sigma, Q, q_s, F, E, \rho)$, where Σ is the vocabulary, Q is a (finite) set of states, $q_s \in Q$ is a unique start state, $F \subseteq Q$ is a set of final states, E is a (finite) set of transitions, and $\rho : F \rightarrow \mathbb{R}$ is a function from final states to final weights. Each transition $e \in E$ is a tuple $e = (l[e], p[e], n[e], w[e])$, where $l[e] \in \Sigma$ is a label (in our case, a word), $p[e] \in Q$ is the origin state of e, $n[e] \in Q$ is the destination state of e, and $w[e] \in \mathbb{R}$ is the weight of the transition. A successful path $\pi = e_1 \ldots e_j$ is a sequence of transitions, such that $p[e_1] = q_s$, $n[e_j] \in F$, and for $1 < k \leq j$, $n[e_{k-1}] = p[e_k]$. Let Π_A be the set of successful paths π in a WFA

A. For any $\pi = e_1 \dots e_j$, $l[\pi] = l[e_1] \dots l[e_j]$.

The weights of the WFA in our case are always in the log semiring, which means that the weight of a path $\pi = e_1 \dots e_j \in \Pi_A$ is defined as:

$$w_A[\pi] = \left(\sum_{k=1}^{j} w[e_k] \right) + \rho(e_j) \tag{7}$$

All WFA that we will discuss in this paper are deterministic, i.e. there are no ϵ transitions, and for any two transitions $e, e' \in E$, if $p[e] = p[e']$, then $l[e] \neq l[e']$. Thus, for any string $\mathbf{w} = w_1 \dots w_j$, there is at most one successful path $\pi \in \Pi_A$, such that $\pi = e_1 \dots e_j$ and for $1 \leq k \leq j$, $l[e_k] = w_k$, i.e. $l[\pi] = \mathbf{w}$. The set of strings $\mathbf{w}$ such that there exists a $\pi \in \Pi_A$ with $l[\pi] = \mathbf{w}$ define a regular language $L_A \subseteq \Sigma$.

We can now define some operations that will be used in this paper.

• λA. For a set of transitions E and $\lambda \in \mathbb{R}$, define $\lambda E = \{(l[e],\ p[e],\ n[e],\ \lambda w[e]) : e \in E\}$. Then, for any WFA $A = (\Sigma, Q, q_s, F, E, \rho)$, define λA for $\lambda \in \mathbb{R}$ as follows: $\lambda A = (\Sigma, Q, q_s, F, \lambda E, \lambda \rho)$.

• $A \circ A'$. The intersection of two deterministic WFAs $A \circ A'$ in the log semiring is a deterministic WFA such that $L_{A \circ A'} = L_A \bigcap L_{A'}$. For any $\pi \in \Pi_{A \circ A'}$, $w_{A \circ A'}[\pi] = w_A[\pi_1] + w_{A'}[\pi_2]$, where $l_{A \circ A'}[\pi] = l_A[\pi_1] = l_{A'}[\pi_2]$.

• $\mathbf{BestPath}(A)$. This operation takes a WFA A, and returns the best scoring path $\hat{\pi} = \operatorname{argmax}_{\pi \in \Pi_A} w_A[\pi]$.

• $\mathbf{MinErr}(A, y)$. Given a WFA A, a string y, and an error-function $E(y, \mathbf{w})$, this operation returns $\hat{\pi} = \operatorname{argmin}_{\pi \in \Pi_A} E(y, l[\pi])$. This operation will generally be used with y as the reference transcription for a particular training example, and $E(y, \mathbf{w})$ as some measure of the number of errors in $\mathbf{w}$ when compared to y. In this case, the $\mathbf{MinErr}$ operation returns the path $\pi \in \Pi_A$ such $l[\pi]$ has the smallest number of errors when compared to y.

• $\mathbf{Norm}(A)$. Given a WFA A, this operation yields a WFA A' such that $L_A = L_{A'}$ and for every $\pi \in \Pi_A$ there is a $\pi' \in \Pi_{A'}$ such that $l[\pi] = l[\pi']$ and

$$w_{A'}[\pi'] = w_A[\pi] - \log \left(\sum_{\bar{\pi} \in \Pi_A} \exp(w_A[\bar{\pi}]) \right) \tag{8}$$

Note that

$$\sum_{\pi \in \mathbf{Norm}(A)} \exp(w_{\mathbf{Norm}(A)}[\pi]) = 1 \tag{9}$$

In other words the weights define a probability distribution over the paths.

• $\mathbf{ExpCount}(A, \mathbf{w})$. Given a WFA A and an n-gram $\mathbf{w}$, we define the expected count of $\mathbf{w}$ in A as

$$\mathbf{ExpCount}(A, \mathbf{w}) = \sum_{\pi \in \Pi_A} \exp(w_{\mathbf{Norm}(A)}[\pi]) C(\mathbf{w}, l[\pi])$$

where $C(\mathbf{w}, l[\pi])$ is defined to be the number of times the n-gram $\mathbf{w}$ appears in a string $l[\pi]$.

Given an acoustic input x, let $\mathcal{L}_x$ be a deterministic word-lattice produced by the baseline recognizer. The lattice $\mathcal{L}_x$ is an acyclic WFA, representing a weighted set of possible transcriptions of x under the baseline recognizer. The weights represent the combination of acoustic and language model scores in the original recognizer.

The new, discriminative language model constructed during training consists of a deterministic WFA which we will denote $\mathcal{D}$, together with a single parameter α_0. The parameter α_0 is the weight for the log probability feature Φ_0 given by the baseline recognizer. The WFA $\mathcal{D}$ is constructed so that $L_{\mathcal{D}} = \Sigma^*$ and for all $\pi \in \Pi_{\mathcal{D}}$

$$w_{\mathcal{D}}[\pi] = \sum_{j=1}^{d} \Phi_j(x, l[\pi]) \alpha_j$$

Recall that $\Phi_j(x, \mathbf{w})$ for $j > 0$ is the count of the j'th n-gram in $\mathbf{w}$, and α_j is the parameter associated with that n-gram. Then, by definition, $\alpha_0 \mathcal{L} \circ \mathcal{D}$ accepts the same set of strings as $\mathcal{L}$, but

$$w_{\alpha_0 \mathcal{L} \circ \mathcal{D}}[\pi] = \sum_{j=0}^{d} \Phi_j(x, l[\pi]) \alpha_j$$

and

$$\operatorname*{argmax}_{\pi \in \mathcal{L}} \Phi(x, l[\pi]) \cdot \bar{\alpha} = \mathbf{BestPath}(\alpha_0 \mathcal{L} \circ \mathcal{D}).$$

Thus decoding under our new model involves first producing a lattice $\mathcal{L}$ from the baseline recognizer; second, scaling $\mathcal{L}$ with α_0 and intersecting it with the discriminative language model $\mathcal{D}$; third, finding the best scoring path in the new WFA.

We now turn to training a model, or more explicitly, deriving a discriminative language model $(\mathcal{D}, \alpha_0)$ from a set of training examples. Given a training set (x_i, r_i) for $i = 1 \ldots N$, where x_i is an acoustic sequence, and r_i is a reference transcription, we can construct lattices $\mathcal{L}_i$ for $i = 1 \ldots N$ using the baseline recognizer. We can also derive target transcriptions $y_i = \mathbf{MinErr}(\mathcal{L}_i, r_i)$. The training algorithm is then a mapping from $(\mathcal{L}_i, y_i)$ for $i = 1 \ldots N$ to a pair $(\mathcal{D}, \alpha_0)$. Note that the construction of the language model requires two choices. The first concerns the choice

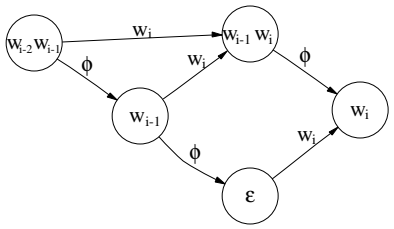

Fig. 2. Representation of a trigram model with failure transitions.

of the set of n-gram *features* Φ_j for $j = 1 \ldots d$ implemented by $\mathcal{D}$. The second concerns the choice of *parameters* α_j for $j = 0 \ldots d$ which assign weights to the n-gram features as well as the baseline feature Φ_0.

Before describing methods for training a discriminative language model, we give a little more detail about the structure of $\mathcal{D}$, focusing on how n-gram language models can be implemented with finite-state techniques.

3.3 Representation of n-gram language models

An n-gram model can be efficiently represented in a deterministic WFA, through the use of failure transitions (Allauzen et al., 2003). Every string accepted by such an automaton has a single path through the automaton, and the weight of the string is the sum of the weights of the transitions in that path. In such a representation, every state in the automaton represents an n-gram history h, e.g. $w_{i-2}w_{i-1}$, and there are transitions leaving the state for every word w_i such that the feature hw_i has a weight. There is also a failure transition leaving the state, labeled with some reserved symbol ϕ, which can only be traversed if the next symbol in the input does not match any transition leaving the state. This failure transition points to the backoff state h', i.e. the n-gram history h minus its initial word. Figure 2 shows how a trigram model can be represented in such an automaton. See Allauzen et al. (2003) for more details.

Note that in such a deterministic representation, the entire weight of all features associated with the word w_i following history h must be assigned to the transition labeled with w_i leaving the state h in the automaton. For example, if $h = w_{i-2}w_{i-1}$, then the trigram $w_{i-2}w_{i-1}w_i$ is a feature, as is the bigram $w_{i-1}w_i$ and the unigram w_i. In this case, the weight on the transition w_i leaving state h must be the sum of the trigram, bigram and unigram feature weights. If only the trigram feature weight were assigned to the transition, neither the unigram nor the bigram feature contribution would be included in the path weight. In order to ensure that the correct weights are assigned to each string, every transition encoding an order k n-gram must carry the sum of the weights for all n-gram features of orders $\leq k$. To ensure that every string in Σ^* receives the correct weight, for any n-gram hw represented explicitly in the automaton, $h'w$ must also be represented explicitly in the automa-

ton, even if its weight is 0. Otherwise, there is no guarantee that a string containing the n-gram $h'w$ will be able to reach the state in the automaton corresponding to this n-gram.

3.4 The perceptron algorithm

The perceptron algorithm is incremental, meaning that the language model $\mathcal{D}$ is built one training example at a time, during several passes over the training set. Initially, we build $\mathcal{D}$ to be the (trivial) automaton that accepts all strings in Σ^* with weight 0. For the perceptron experiments, we chose the parameter α_0 to be a fixed constant, chosen by optimization on the held-out set. The algorithm in figure 1 is implemented as follows:

Inputs: Lattices $\mathcal{L}_i$ and reference transcriptions r_i for $i = 1 \ldots N$. A value for the parameter α_0.

Initialization: Set $\mathcal{D}$ to be a WFA that accepts all strings in Σ^* with weight 0. Set $y_i = \mathbf{MinErr}(\mathcal{L}_i, r_i)$ for $i = 1 \ldots N$.

Algorithm: **For** $t = 1 \ldots T, i = 1 \ldots N$:

- Calculate $z_i = \text{argmax}_{y \in \mathbf{GEN}(x_i)} \Phi(x_i, y) \cdot \bar{\alpha} = \mathbf{BestPath}(\alpha_0 \mathcal{L}_i \circ \mathcal{D})$.
- For all j for $j = 1 \ldots d$ such that $\Phi_j(x_i, y_i) \neq \Phi_j(x_i, z_i)$ apply the update $\alpha_j \leftarrow \alpha_j + \Phi_j(x_i, y_i) - \Phi_j(x_i, z_i)$. Modify $\mathcal{D}$ to incorporate these parameter changes.

In addition, averaged parameters need to be stored (see section 2.1). These parameters will replace the un-averaged parameters in $\mathcal{D}$ once training is completed.

Note that the only n-gram features to be included in $\mathcal{D}$ at the end of the training process are those that occur in either a best scoring path z_i or a minimum error path y_i at some point during training. [10] Thus the perceptron algorithm is in effect doing feature selection as a by-product of training. Given N training examples, and T passes over the training set, $O(NT)$ n-grams will have non-zero weight after training. Experiments suggest that the perceptron reaches optimal performance after a small number of training iterations, for example $T = 1$ or $T = 2$. Thus $O(NT)$ can be very small compared to the full number of n-grams seen in all training lattices. In our experiments, the perceptron method chose around 1.4 million n-grams with non-zero weight. This compares to 43.65 million possible n-grams seen in the training data.

This is a key contrast with conditional log-likelihood maximization, which optimize the parameters of a fixed feature set. Feature selection can be critical in our

[10] In fact, only features that differ in count between the best scoring path and the minimum error path will be updated, resulting in an even smaller set of features with non-zero parameters.

domain, as training and applying a discriminative language model over *all* n-grams seen in the training data (in either correct or incorrect transcriptions) may be computationally very demanding. One training scenario that we will consider will be using the output of the perceptron algorithm (the averaged parameters) to provide the feature set and the initial feature weights for use in the conditional log-likelihood optimization algorithm. This leads to a model which is reasonably sparse, but has the benefit of maximizing the conditional log-likelihood, which as we will see gives gains in performance.

3.5 Global Conditional Log-Linear Models

The GCLM optimization methods that we use assume a fixed definition of the n-gram features Φ_j for $j = 1 \ldots d$ in the model. In the experimental section we will describe a number of ways of defining the feature set. The optimization methods we use begin at some initial setting for $\bar{\alpha}$, and then search for the parameters $\bar{\alpha}^*$ which maximize $LL_R(\bar{\alpha})$ as defined in Eq. 5.

The optimization method requires calculation of $LL_R(\bar{\alpha})$ and the gradient of $LL_R(\bar{\alpha})$ for a series of values for $\bar{\alpha}$. The first step in calculating these quantities is to take the parameter values $\bar{\alpha}$, and to construct an acceptor $\mathcal{D}$ which accepts all strings in Σ^*, such that

$$w_{\mathcal{D}}[\pi] = \sum_{j=1}^{d} \Phi_j(x, l[\pi])\alpha_j$$

For each training lattice $\mathcal{L}_i$, we then construct a new lattice $\mathcal{L}'_i = \mathbf{Norm}(\alpha_0 \mathcal{L}_i \circ \mathcal{D})$. The lattice $\mathcal{L}'_i$ represents (in the log domain) the distribution $p_{\bar{\alpha}}(y|x_i)$ over strings $y \in \mathbf{GEN}(x_i)$. The value of $\log p_{\bar{\alpha}}(y_i|x_i)$ for any i can be computed by simply taking the path weight of π such that $l[\pi] = y_i$ in the new lattice $\mathcal{L}'_i$. Hence computation of $LL_R(\bar{\alpha})$ in Eq. 5 is straightforward.

Calculating the n-gram feature gradients for the GCLM optimization is also relatively simple, once $\mathcal{L}'_i$ has been constructed. From the derivative in Eq. 6, for each $i = 1 \ldots N, j = 1 \ldots d$ the quantity

$$\Phi_j(x_i, y_i) - \sum_{y \in \mathbf{GEN}(x_i)} p_{\bar{\alpha}}(y|x_i)\Phi_j(x_i, y) \tag{10}$$

must be computed. The first term is simply the number of times the j'th n-gram feature is seen in y_i. The second term is the expected number of times that the j'th n-gram is seen in the acceptor $\mathcal{L}'_i$. If the j'th n-gram is $w_1 \ldots w_n$, then this can be computed as $\mathbf{ExpCount}(\mathcal{L}'_i, w_1 \ldots w_n)$. The GRM library, which was presented in Allauzen et al. (2003), has a direct implementation of the function $\mathbf{ExpCount}$, which simultaneously calculates the expected value of all n-grams of order less than or equal to a given n in a lattice $\mathcal{L}$.

The one non-ngram feature weight that is being estimated is the weight α_0 given to the baseline ASR negative log probability. Calculation of the gradient of LL_R with respect to this parameter again requires calculation of the term in Eq. 10 for $j = 0$ and $i = 1 \ldots N$. Computation of $\sum_{y \in \mathbf{GEN}(x_i)} p_{\bar{\alpha}}(y|x_i)\Phi_0(x_i, y)$ turns out to be not as straightforward as calculating n-gram expectations. To do so, we rely upon the fact that $\Phi_0(x_i, y)$, the negative log probability of the path, decomposes to the sum of negative log probabilities of each transition in the path. We index each transition in the lattice $\mathcal{L}_i$, and store its negative log probability under the baseline model. We can then calculate the required gradient from $\mathcal{L}'_i$, by calculating the expected value in $\mathcal{L}'_i$ of each indexed transition in $\mathcal{L}_i$.

We found that an approximation to the gradient of α_0, however, performed nearly identically to this exact gradient, while requiring substantially less computation. Let w_1^n be a string of n words, labeling a path in word-lattice $\mathcal{L}'_i$. For brevity, let $\mathrm{P}_i(w_1^n) = p_{\bar{\alpha}}(w_1^n|x_i)$ be the conditional probability under the current model, and let $\mathrm{Q}_i(w_1^n)$ be the probability of w_1^n in the normalized baseline ASR lattice $\mathbf{Norm}(\mathcal{L}_i)$. Let L_i be the set of strings in the language defined by $\mathcal{L}_i$. Then we wish to compute E_i for $i = 1 \ldots N$, where

$$\begin{aligned} E_i &= \sum_{w_1^n \in L_i} \mathrm{P}_i(w_1^n) \log \mathrm{Q}_i(w_1^n) \\ &= \sum_{w_1^n \in L_i} \sum_{k=1\ldots n} \mathrm{P}_i(w_1^n) \log \mathrm{Q}_i(w_k|w_1^{k-1}) \end{aligned} \tag{11}$$

The approximation is to make the following Markov assumption:

$$\begin{aligned} E_i &\approx \sum_{w_1^n \in L_i} \sum_{k=1\ldots n} \mathrm{P}_i(w_1^n) \log \mathrm{Q}_i(w_k|w_{k-2}^{k-1}) \\ &= \sum_{xyz \in S_i} \mathbf{ExpCount}(\mathcal{L}'_i, xyz) \log \mathrm{Q}_i(z|xy) \end{aligned} \tag{12}$$

where S_i is the set of all trigrams seen in L_i. The term $\log \mathrm{Q}_i(z|xy)$ can be calculated once before training for every lattice in the training set; the $\mathbf{ExpCount}$ term is calculated as before using the GRM library. We have found this approximation to be effective in practice, and it was used for the trials reported below.

When the gradients and conditional likelihoods are collected from all of the utterances in the training set, the contributions from the regularizer are combined to give an overall gradient and objective function value. These values are provided to the parameter estimation routine, which then returns the parameters for use in the next iteration. The accumulation of gradients for the feature set is the most time consuming part of the approach, but this is parallelizable, so that the computation can be divided among many processors.

4 Empirical Results

We present empirical results on the Rich Transcription 2002 evaluation test set (rt02), which we used as our development set, as well as on the Rich Transcription 2003 Spring evaluation CTS test set (rt03). The rt02 set consists of 6081 sentences (63804 words) and has three subsets: Switchboard 1, Switchboard 2, Switchboard Cellular. The rt03 set consists of 9050 sentences (76083 words) and has two subsets: Switchboard and Fisher.

The training set consists of 276726 transcribed utterances (3047805 words), with an additional 20854 utterances (249774 words) as held out data. For each utterance, a weighted word-lattice was produced, representing alternative transcriptions, from the ASR system. From each word-lattice, the oracle best path was extracted, which gives the best word-error rate from among all of the hypotheses in the lattice. The oracle word-error rate for the training set lattices was 12.2%. We also performed trials with 1000-best lists for the same training set, rather than lattices. The oracle score for the 1000-best lists was 16.7%.

To produce the word-lattices, each training utterance was processed by the baseline ASR system. In a naive approach, we would simply train the baseline system (i.e., an acoustic model and language model) on the entire training set, and then decode the training utterances with this system to produce lattices. We would then use these lattices with the perceptron or GCLM training algorithms. Unfortunately, this approach is likely to produce a set of training lattices that are very different from test lattices, in that they will have very low word-error rates, given that the lattice for each utterance was produced by a model that was trained on that utterance. To somewhat control for this, the training set was partitioned into 28 sets, and baseline Katz backoff trigram models were built for each set by including only transcripts from the other 27 sets. Lattices for each utterance were produced with an acoustic model that had been trained on the entire training set, but with a language model that was trained on the 27 data portions that did not include the current utterance. Since language models are generally far more prone to overtrain than standard acoustic models, this goes a long way toward making the training conditions similar to testing conditions.

Our ASR system is based on the AT&T Switchboard system used in the RT03 Evaluations (see Ljolje et al., 2003, for details). The evaluation system was originally designed as a real-time system. For these experiments, we use the system at a slower speed in order to avoid search errors. The system uses a multi-pass strategy to incorporate speaker adaptation as well as more complex language and acoustic models. In the first pass, the system produces an initial hypothesis which is used for speaker normalization and adaptation. In the second pass, lattices are generated using the normalized features and the adapted acoustic models. Both of these passes use a trigram language model, trained on approximately 4 million words of

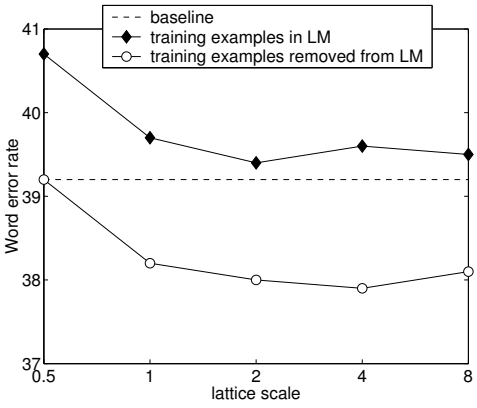

Fig. 3. Leaving all utterances in the training set for the language model that produces the training lattice, versus removing utterances from the training for the language model that produces their word-lattice. Word error rate on Switchboard 2002 eval set at various lattice scale factors.

transcribed telephone conversations. The scores in the lattices are then replaced by scores from a 6-gram language model, trained using MAP estimation (Bacchiani et al., 2006), with the original 4 million words of in-domain data plus an additional 150 million words of out-of-domain text. Finally the lattices are rescored using more complex adapted acoustic models (typically pentaphone models instead of triphone models), and the final hypotheses are obtained.

In order to evaluate the performance of discriminative language modeling as well as its interaction with other components of the ASR system, we used three different configurations:

(1) A simple single-pass (or first pass) system. This configuration was used to measure the performance of discriminative language modeling in isolation.
(2) A system that performs an additional rescoring pass which allows for better silence modeling and replaces the trigram language model score with a 6-gram model. This configuration was used to see the effects of lattice rescoring and more complex language models.
(3) The full multi-pass system, with a modified final pass that generates lattices instead of single hypotheses. This configuration was used to evaluate the interaction of discriminative language modeling with acoustic model adaptation and feature normalization. The effects of using more complex acoustic models which were not used during the training of the discriminative language models was also evaluated.

4.1 Perceptron results

The first trials using the perceptron algorithm look at a simple single-pass recognition system that forms the basis of the AT&T Switchboard system. After each

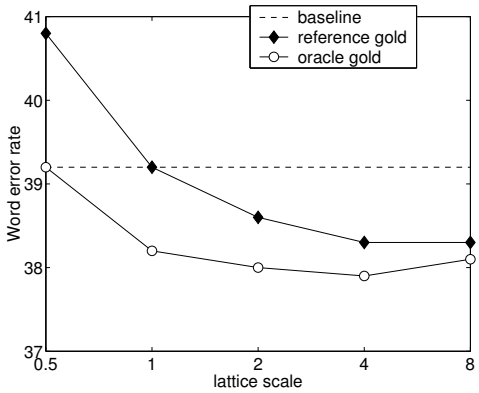

Fig. 4. Using the reference transcription as the gold standard, versus the oracle best path through the lattice. Word error rate on Switchboard 2002 eval set at various lattice scale factors.

iteration over the training set, the averaged perceptron model was evaluated against the held-out training data, and the model with the lowest word-error-rate was chosen for evaluation on the test set. For each training scenario, we built 5 models, corresponding to 5 lattice scaling factors α_0, from 0.5 to 8.0. Each graph shows the baseline performance, which is without a perceptron model; and performance of a perceptron built under our standard training scenario. The standard training scenario is defined as

(1) training lattices produced by removing utterances from their own baseline LM training set
(2) using the oracle best path as the gold standard
(3) with trigram, bigram and unigram features
(4) no n-best extraction from the word lattices

Figure 3 compares the standard scenario just presented with the same scenario, except that the lattices were produced without removing utterances from their own baseline LM training set, i.e. number 1 above is changed. From this plot, we can see several things. First, removing utterances from their own baseline LM training set is necessary to get any improvement over the baseline results at all. This underlines the importance of matching the testing and training conditions for this approach. Our standard approach works best with a lattice scale of 4, which provides a 1.3 percent improvement over the baseline, 37.9 percent WER versus 39.2. All scales α_0 from 1 to 8 are within 0.3% of this best result.

Figure 4 compares the standard training scenario with the same scenario, except the reference transcription is used as the gold standard instead of the oracle best path. At the best scaling factors, the difference is 0.4 percent, but the reference trained model is much more sensitive to the scaling factor.

Figure 5 shows the result of including fewer features in the perceptron model. Including all n-grams of order 3 or less is the best performer, but the gain is very small

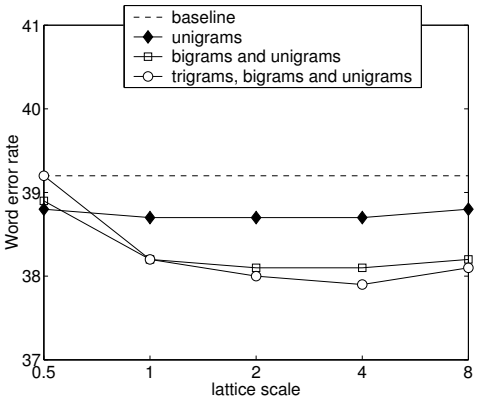

Fig. 5. Using feature sets with n-grams of different orders. Word error rate on Switchboard 2002 eval set at various lattice scale factors.

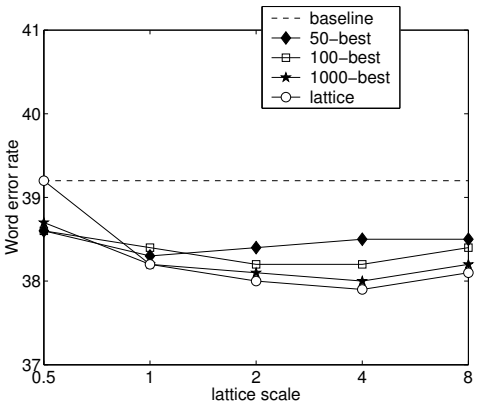

Fig. 6. N-best extraction on training lattices with various values of N, versus using the lattices. Word error rate on Switchboard 2002 eval set at various lattice scale factors.

versus using just bigrams and unigrams. Unigrams and bigrams both contribute a fair amount to performance, but the trigrams add very little over and above those. The lower order models are less sensitive to the lattice scale factor.

Finally, figure 6 shows the result of performing n-best extraction on the training and testing lattices. [11] With n=1000, the performance is essentially the same as with full lattices, and the performance degrades as fewer candidates are included. The n-best extracted models are less sensitive to the lattice scale factor.

The AT&T Switchboard system performs a rescoring pass, which allows for better silence modeling and replaces the trigram language model score with a 6-gram model. Table 1 gives the rt02 and rt03 results for the ASR baselines and perceptron trained on lattices or 1000-best lists for both first-pass recognition and the rescoring-pass. The magnitude of the gain in the rescoring pass is less than for the first pass, but the 0.5 and 0.7 percent improvements over the rescoring-pass base-

[11] The oracle word-error rates for the 50-best, 100-best and 1000-best training sets are 20.8, 19.7, and 16.7 percent, respectively.

Trial	Pass	Iterations	Features	Dev (rt02)	Test (rt03)
ASR Baseline	First	-	-	39.2	38.2
Perceptron, Lattice	First	2	1408572	37.9	36.9
Perceptron, 1000-best	First	2	910322	38.0	37.2
ASR Baseline	Rescore	-	-	37.1	36.4
Perceptron, Lattice	Rescore	2	974159	36.6	35.7
Perceptron, 1000-best	Rescore	3	884186	36.6	35.7

Table 1
Word-error rate results at convergence iteration for various trials, on both Switchboard 2002 test set (rt02), which was used as the dev set, and Switchboard 2003 test set (rt03).

line are also statistically significant ($p < 0.001$), using the Matched Pair Sentence Segment test for WER included with SCTK (NIST, 2000).

4.2 Global conditional log-linear model results

We now describe results for global conditional log-linear models. There are three baselines which we compare against. The first is the ASR baseline, with no reweighting from a discriminatively trained n-gram model. The other two baselines are with perceptron-trained n-gram model re-weighting. The first of these is for a pruned-lattice trained trigram model, which showed a reduction in word error rate (WER) of 1.3%, from 39.2% to 37.9% on rt02. The second is for a 1000-best list trained trigram model, which performed only marginally worse than the lattice-trained perceptron, at 38.0% on rt02.

4.2.1 Perceptron feature set

We use the perceptron-trained models as the starting point for our GCLM training algorithm: the feature set given to the GCLM training algorithm is the feature set selected by the perceptron algorithm; the feature weights are initialized to those of the averaged perceptron. Figure 7 shows the performance of our three baselines versus three trials of the GCLM training algorithm. In the first two trials, the training set consists of the pruned lattices, and the feature set is from the perceptron algorithm trained on pruned lattices. There were 1.4 million features in this feature set. The first trial set the regularizer constant $\sigma = \infty$, so that the algorithm was optimizing raw conditional likelihood. The second trial is with the regularizer constant $\sigma = 0.5$, which we found empirically to be a good parameterization on the held-out set. As can be seen from these results, regularization is critical.

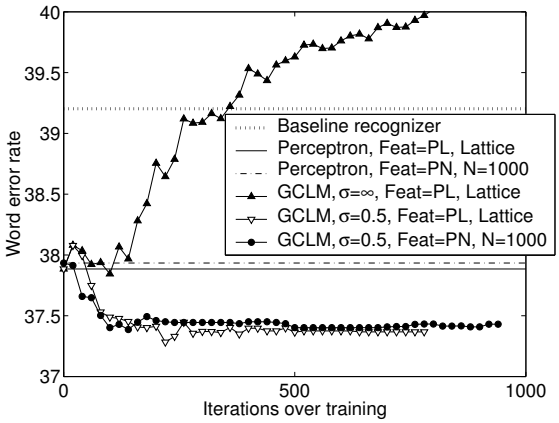

Fig. 7. Word error rate on the rt02 eval set versus training iterations for GCLM trials, contrasted with baseline recognizer performance and perceptron performance. Points are at every 20 iterations. Each point (x,y) is the WER at the iteration with the best objective function value in the interval (x-20,x].

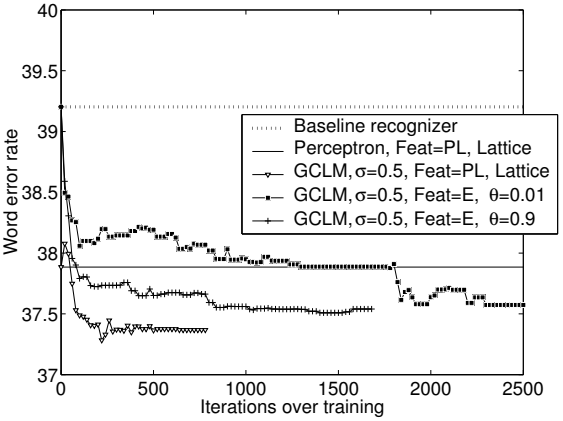

Fig. 8. Word error rate on the rt02 eval set versus training iterations for GCLM trials, contrasted with baseline recognizer performance and perceptron performance. Points are at every 20 iterations. Each point (x,y) is the WER at the iteration with the best objective function value in the interval (x-20,x].

The third trial in this set uses the feature set from the perceptron algorithm trained on 1000-best lists, and uses GCLM optimization on these on these same 1000-best lists. There were 0.9 million features in this feature set. For this trial, we also used $\sigma = 0.5$. As with the perceptron baselines, the n-best trial performs nearly identically with the pruned lattices, here also resulting in 37.4% WER. This may be useful for techniques that would be more expensive to extend to lattices versus n-best lists (e.g. models with unbounded dependencies).

These trials demonstrate that the GCLM training algorithm can do a better job of estimating parameter weights than the perceptron algorithm for the same feature set. As mentioned in the earlier section, feature selection is a by-product of the perceptron algorithm, but the GCLM training algorithm uses a fixed set of features. The next two trials looked at selecting feature sets other than those provided by the

perceptron algorithm.

4.2.2 Other feature sets

In order for the feature weights to be non-zero in this approach, they must be observed in the training set. The number of unigram, bigram and trigram features with non-zero observations in the training set lattices is 43.65 million, or roughly 30 times the size of the perceptron feature set. Many of these features occur only rarely with very low conditional probabilities, and hence cannot meaningfully impact system performance. We pruned this feature set to include all unigrams and bigrams, but only those trigrams with an expected count of greater than 0.01 in the training set. That is, to be included, a trigram must occur in a set of paths, the sum of the conditional probabilities of which must be greater than our threshold $\theta = 0.01$. This threshold resulted in a feature set of roughly 12 million features, nearly 10 times the size of the perceptron feature set. For better comparability with that feature set, we set our thresholds higher, so that trigrams were pruned if their expected count fell below $\theta = 0.9$, and bigrams were pruned if their expected count fell below $\theta = 0.1$. We were concerned that this may leave out some of the features on the oracle paths, so we added back in all bigram and trigram features that occurred on oracle paths, giving a feature set of 1.5 million features, roughly the same size as the perceptron feature set.

Figure 8 shows the results for three GCLM trials versus our ASR baseline and the perceptron algorithm baseline trained on lattices. First, the result using the perceptron feature set provides us with a WER of 37.4%, as previously shown. The WER at convergence for the big feature set (12 million features) is 37.6%; the WER at convergence for the smaller feature set (1.5 million features) is 37.5%. While both of these other feature sets converge to performance close to that using the perceptron features, the number of iterations over the training data that are required to reach that level of performance are many more than for the perceptron-initialized feature set.

Tables 2 and 3 show the word-error rate at the best performing iteration on the development set for the various trials, on both rt02 and rt03, for first-pass and rescoring pass, respectively. All of the first-pass GCLM trials are significantly better than the perceptron performance, using the Matched Pair Sentence Segment test for WER included with SCTK (NIST, 2000). On rt02, the N-best and perceptron initialized GCLM trials were significantly better than the lattice perceptron at $p < 0.001$; the other two GCLM trials were significantly better than the lattice perceptron at $p < 0.01$. On rt03, the N-best GCLM trial was significantly better than the lattice perceptron at $p < 0.002$; the other three GCLM trials were significantly better than the lattice perceptron at $p < 0.001$.

Table 4 presents several trials investigating other methods for selecting features for

Trial	Pass	Features	Iterations	rt02	rt03
ASR Baseline	First	-	-	39.2	38.2
Perceptron, Lattice	First	1408572	2	37.9	36.9
Perceptron, 1000-best	First	910322	2	38.0	37.2
GCLM, Lattice, Percep feats	First	1408572	220	37.3	36.5
GCLM Exact, Lattice, Percep feats	First	1408572	144	37.4	36.7
GCLM, 1000-best, Percep feats	First	910322	140	37.4	36.7
GCLM Exact, 1000-best, Percep feats	First	910322	220	37.4	36.6
GCLM, Lattice, $\theta = 0.01$	First	11816862	2530	37.5	36.6
GCLM, Lattice, $\theta = 0.9$	First	1540260	1432	37.5	36.5

Table 2
First-pass word-error rate results at best-dev-set-performance iteration for various trials, on both Switchboard 2002 test set (rt02), which was used as the dev set, and Switchboard 2003 test set (rt03).

Trial	Pass	Features	Iterations	rt02	rt03
ASR Baseline	Rescore	-	-	37.1	36.4
Perceptron, Lattice	Rescore	974159	2	36.6	35.7
Perceptron, 1000-best	Rescore	884186	3	36.6	35.7
GCLM, Lattice, Percep feats	Rescore	974159	132	36.2	35.5
GCLM Exact, Lattice, Percep feats	Rescore	974159	177	36.2	35.5
GCLM, 1000-best, Percep feats	Rescore	884186	113	36.3	35.4
GCLM Exact, 1000-best, Percep feats	Rescore	884186	100	36.3	35.4

Table 3
Rescoring-pass word-error rate results at best-dev-set-performance iteration for various trials, on both Switchboard 2002 test set (rt02), which was used as the dev set, and Switchboard 2003 test set (rt03).

use in GCLM modeling. The first method is inspired by the fact that the perceptron algorithm only selects features from the best scoring and oracle (minimum error rate) paths. One can do something similar without perceptron training, by restricting features to those occurring in the baseline model 1-best or oracle paths. The first two rows of table 4 demonstrate this to be a viable approach, reaching competitive levels of performance in both first pass and rescoring trials. In the absence of parameter weight starting values provided by the perceptron algorithm, however, it takes many more iterations to reach convergence.

The second two rows of table 4 show the result of restricting the features in the perceptron algorithm to those with an expected count in the corpus greater than a

Trial	Pass	Features	Iterations	rt02	rt03
GCLM, Oracle & Bestpath feats	First	384090	467	37.4	36.6
GCLM, Oracle & Bestpath feats	Rescore	357616	367	36.3	35.5
Perceptron, Lattice, pruned $\theta = 0.2$	Rescore	334156	2	36.6	35.8
GCLM, Lattice, Percep feats pruned $\theta = 0.2$	Rescore	334156	123	36.3	35.6

Table 4
Word-error rate results at best-dev-set-performance iteration for various trials with smaller feature sets, on both Switchboard 2002 test set (rt02), which was used as the dev set, and Switchboard 2003 test set (rt03).

		GCLM	
Features	Percep	approx	exact
Lattice, Percep Feats (1.4M)	7.10	1.69	3.61
N-best, Percep Feats (0.9M)	3.40	0.96	1.40
Lattice, $\theta = 0.01$ (12M)	-	2.24	4.75

Table 5
Time (in hours) for one iteration on a single Intel Xeon 2.4Ghz processor with 4GB RAM.

threshold $\theta = 0.2$. This leads to a feature set of roughly the same size as the trials in the first two rows of the table. Two iterations of the perceptron algorithm provides a starting point to GCLM training with this feature set, which converges to the same performance point as the trial in row 2 of the table, but takes about one third of the iterations to do so.

Finally, we measured the time of a single iteration over the training data on a single machine for the perceptron algorithm, the GCLM training algorithm using the approximation to the gradient of α_0, and the GCLM training algorithm using an exact gradient of α_0. Table 5 shows these times in hours. Because of the frequent update of the weights in the model, the perceptron algorithm is more expensive than the GCLM training algorithm for a single iteration. Further, the GCLM training algorithm is parallelizable, so that most of the work of an iteration can be shared among multiple processors. Our most common training setup for the GCLM training algorithm was parallelized between 20 processors, using the approximation to the gradient. In that setup, using the 1.4M feature set, one iteration of the perceptron algorithm took the same amount of real time as approximately 80 iterations of the GCLM training algorithm.

Trial	Pass	Triphone AM	Pentaphone AM
Baseline	Final	28.9	27.4
Perceptron	Final	28.4	27.1
GCLM, Percep feats	Final	28.0	26.9

Table 6
Multi-pass word-error rate results using the discriminative language model with triphone and pentaphone acoustic models for the final pass on the Switchboard 2003 test set (rt03).

4.3 Results with a multi-pass ASR system

Finally we present results using the multi-pass AT&T Switchboard system used in the RT03 Evaluations (see Ljolje et al., 2003, for details). For simplicity we apply the discriminative language model only at the final pass. Note that results may be improved further by applying the discriminative language model at earlier, adaptation passes in recognition. In fact, application of the perceptron method at each decoding step was shown to improve the performance in Bacchiani et al. (2004).

The word-error rate performance of the system on the rt03 test set is presented in Table 6. Recall that the discriminative language models were estimated using lattices generated by triphone acoustic models. When the final pass uses triphone acoustic models an improvement of 0.9% in WER is observed. This is a 3.1% relative error rate reduction, compared to the 4.6% relative error rate reduction for first-pass recognition. This result demonstrates that the speaker normalization and acoustic model adaptation techniques utilized do not create a significant mismatch. On the other hand, using pentaphone acoustic models we get a 0.5% absolute improvement in WER (1.8% relative reduction), suggesting that the mismatch between the training conditions using triphone acoustic models and the testing conditions using pentaphone acoustic models result in reduced but still significant improvements.

As in Bacchiani et al. (2004), reductions in WER are retained after unsupervised acoustic model adaptation. These results indicate that it is not necessary to perform unsupervised acoustic model adaptation when producing training lattices for this approach, even if such techniques are used at test time.

5 Conclusion

We have contrasted two approaches to discriminative language model estimation on a difficult large vocabulary task, showing that they can indeed scale effectively to handle this size of a problem. Both algorithms have their benefits. The perceptron algorithm selects a relatively small subset of the total feature set, and requires just a

couple of passes over the training data. The GCLM training algorithm does a better job of parameter estimation for the same feature set, and is parallelizable, so that each pass over the training set can require just a fraction of the real time of the perceptron algorithm.

The best scenario from among those that we investigated was a combination of both approaches, with the output of the perceptron algorithm taken as the starting point for GCLM estimation.

We have shown that reducing the mismatch between the training lattices and the test lattices is crucial. A leave-one-partition-out strategy was utilized while estimating the language models used in generating the training lattices. This strategy is hard to employ while estimating the acoustic models and our results suggest that this is not as necessary, at least for the case of acoustic model adaptation.

As a final point, note that the methods we describe do not replace an existing language model, but rather complement it. The existing language model has the benefit that it can be trained on a large amount of text that does not have corresponding speech data, as was done for the 6-gram language model employed in the rescoring pass of the system described here. It has the disadvantage of not being a discriminative model. The new language model is trained on the speech transcriptions, meaning that it has less training data, but that it has the advantage of discriminative training – and in particular, the advantage of being able to learn negative evidence in the form of negative weights on n-grams which are rarely or never seen in natural language text (e.g., "the of"), but are produced too frequently by the recognizer. The methods we describe combine the two language models, making use of their complementary strengths.

References

Allauzen, C., Mohri, M., Roark, B., 2003. Generalized algorithms for constructing language models. In: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics. pp. 40–47.

Bacchiani, M., Riley, M., Roark, B., Sproat, R., 2006. Map adaptation of stochastic grammars. Computer Speech and Language 20 (1), 41–68.

Bacchiani, M., Roark, B., Saraclar, M., 2004. Language model adaptation with MAP estimation and the perceptron algorithm. In: Proceedings of the Human Language Technology Conference and Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL). pp. 21–24.

Bahl, L. R., Brown, P. F., de Souza, P. V., Mercer, R. L., 1993. Estimating hidden markov model parameters so as to maximize speech recognition accuracy. IEEE Transactions on Speech and Audio Processing 1 (1), 77–83.

Balay, S., Gropp, W. D., McInnes, L. C., Smith, B. F., 2002. Petsc users manual, technical Report ANL-95/11-Revision 2.1.2, Argonne National Laboratory.

Banerjee, S., Mostow, J., Beck, J., Tam, W., December 2003. Improving language models by learning from speech recognition errors in a reading tutor that listens. In: Proceedings of the Second International Conference on Applied Artificial Intelligence. Fort Panhala, Kolhapur, India.

Benson, S. J., McInnes, L. C., Moré, J. J., Sarich, J., 2002. Tao users manual, technical Report ANL/MCS-TM-242-Revision 1.4, Argonne National Laboratory.

Benson, S. J., Moré, J. J., 2002. A limited memory variable metric method for bound constrained minimization, preprint ANL/ACSP909-0901, Argonne National Laboratory.

Chen, Z., Lee, K.-F., Li, M. J., October 2000. Discriminative training on language model. In: Proceedings of the Sixth International Conference on Spoken Language Processing (ICSLP). Beijing, China.

Collins, M., 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1–8.

Collins, M., 2004. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In: Bunt, H., Carroll, J., Satta, G. (Eds.), New Developments in Parsing Technology. Kluwer.

Freund, Y., Schapire, R., 1999. Large margin classification using the perceptron algorithm. Machine Learning 3 (37), 277–296.

Goel, V., Byrne, W., 2000. Minimum bayes-risk automatic speech recognition. Computer Speech and Language 14 (2), 115–135.

Jelinek, F., 1996. Acoustic sensitive language modeling. Tech. rep., Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD.

Johnson, M., Geman, S., Canon, S., Chi, Z., Riezler, S., 1999. Estimators for stochastic "unification-based" grammars. In: Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics. pp. 535–541.

Khudanpur, S., Wu, J., 2000. Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling. Computer Speech and Language 14 (4), 355–372.

Kuo, H.-K. J., Fosler-Lussier, E., Jiang, H., Lee, C.-H., May 2002. Discriminative training of language models for speech recognition. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP). Orlando, Florida.

Lafferty, J., McCallum, A., Pereira, F., 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the 18th International Conference on Machine Learning. pp. 282–289.

Ljolje, A., Bocchieri, E., Riley, M., Roark, B., Saraclar, M., Shafran, I., 2003. The AT&T 1xRT CTS system. In: Rich Transcription Workshop.

Malouf, R., 2002. A comparison of algorithms for maximum entropy parameter estimation. In: Proceedings of the Sixth Conference on Natural Language Learning (CoNLL). pp. 49–55.

Mangu, L., Brill, E., Stolcke, A., 2000. Finding consensus in speech recognition: word error minimization and other application of confusion networks. Computer

Speech and Language 14 (4), 373–400.

Mangu, L., Padmanabhan, M., 2001. Error corrective mechanisms for speech recognition. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP).

McCallum, A., Li, W., 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In: Seventh Conference on Natural Language Learning (CoNLL).

Mohri, M., Pereira, F. C. N., Riley, M., 2002. Weighted finite-state transducers in speech recognition. Computer Speech and Language 16 (1), 69–88.

NIST, 2000. Speech recognition scoring toolkit (sctk) version 1.2c. Available at `http://www.nist.gov/speech/tools`.

Pinto, D., McCallum, A., Wei, X., Croft, W. B., 2003. Table extraction using conditional random fields. In: Proceedings of the ACM SIGIR.

Povey, D., Woodland, P., 2002. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP). pp. 105–108.

Ratnaparkhi, A., Roukos, S., Ward, R. T., 1994. A maximum entropy model for parsing. In: Proceedings of the International Conference on Spoken Language Processing (ICSLP). pp. 803–806.

Ringger, E. K., Allen, J. F., 1996. Error corrections via a post-processor for continuous speech recognition. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP).

Roark, B., Saraclar, M., Collins, M., 2004a. Corrective language modeling for large vocabulary ASR with the perceptron algorithm. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP). pp. 749–752.

Roark, B., Saraclar, M., Collins, M., Johnson, M., 2004b. Discriminative language modeling with conditional random fields and the perceptron algorithm. In: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics.

Rosenfeld, R., Chen, S. F., Zhu, X., Jan. 2001. Whole-sentence exponential language models: a vehicle for linguistic-statistical integration. Computer Speech and Language 15 (1), 55–73.

Sha, F., Pereira, F., 2003. Shallow parsing with conditional random fields. In: Proceedings of HLT-NAACL. Edmonton, Canada.

Stolcke, A., Bratt, H., Butzberger, J., Franco, H., Gadde, V. R. R., Plauche, M., Richey, C., Shriberg, E., Sonmez, K., Weng, F., Zheng, J., 2000. The SRI March 2000 Hub-5 conversational speech transcription system. In: Proceedings of the NIST Speech Transcription Workshop.

Stolcke, A., Weintraub, M., 1998. Discriminitive language modeling. In: Proceedings of the 9th Hub-5 Conversational Speech Recog nition Workshop.

Taskar, B., Guestrin, C., Koller, D., 2003. Max margin Markov networks. In: Proceedings of NIPS 2003.

Wainwright, M. J., Jordan, M. I., 2002. Graphical models, exponential families, and variational inference, technical Report 649, UC Berkeley, Dept. of Statistics.

Wallach, H., 2002. Efficient training of conditional random fields. Master's thesis,

University of Edinburgh.
Woodland, P., Povey, D., 2000. Large scale discriminative training for speech recognition. In: Proc. ISCA ITRW ASR2000. pp. 7–16.