Efficient Discriminative Training of Long-span Language Models

Ariya Rastrow, Mark Dredze, Sanjeev Khudanpur

Human Language Technology Center of Excellence
Center for Language and Speech Processing, Johns Hopkins University
Baltimore, MD USA
{ariya,mdredze,khudanpur}@jhu.edu

***Abstract*—Long-span language models, such as those involving syntactic dependencies, produce more coherent text than their n-gram counterparts. However, evaluating the large number of sentence-hypotheses in a packed representation such as an ASR lattice is intractable under such long-span models both during decoding and discriminative training. The accepted compromise is to rescore only the N-best hypotheses in the lattice using the long-span LM. We present *discriminative hill climbing*, an efficient and effective discriminative training procedure for long-span LMs based on a hill climbing rescoring algorithm [1]. We empirically demonstrate significant computational savings as well as error-rate reduction over N-best training methods in a state of the art ASR system for Broadcast News transcription.**

I. Introduction

Automatic Speech Recognition (ASR) relies on a language model (LM) to generate coherent natural language text. The most common language models rely on simple n-gram statistics and a wide range of smoothing and backoff techniques [2][3]. For quite some time, LM research has pushed towards linguistically motivated models, including using semantics [4], word class [5], part of speech tags [6][7], and full sentence parsing [8][9]. These LMs, which include non-local information based on the entire word sequence, deliver significant improvements over the traditional n-gram model. While linguistically appealing, long-span LM integration in full systems still poses a challenge. n-gram LMs can efficiently operate on a lattice representation, but long-span LMs require enumeration of each hypothesis. Since scoring all the hypotheses contained in a lattice is intractable, these models rescore an N-best list from the lattice. List size trades off efficient (smaller N) and accurate (larger N) decoding.

Another trend is towards discriminatively trained models. Generative models estimate the probability of generating the observed sequence, whereas discriminative models need only compare the correct sequence with the highest scoring incorrect outputs, capturing acoustic confusions in the decoded hypotheses [10][11][12]. As has been the experience in the machine learning community, discriminative models, while more complex to train, yield better performance. Since we are often interested in minimizing an objective function (WER), discriminative training based on such criteria often yields better results than minimizing perplexity.

Discriminative models provide the flexibility to include both typical n-gram features and arbitrary features based on the word sequence. Globally normalized (as opposed to locally normalized) discriminative models also dispense with explicit computation of the partition function Z, making it easier to incorporate long-span features. However, unlike generative LMs with long-span dependencies where one has to resort to N-best lists only during decoding, globally normalized discriminative models force the use of N-best lists even for LM *training*. [9], for instance, train a syntactic LM on the 1000-best outputs from a speech lattice. This adds a significant computational cost to training: parsing all 1000 candidate paths. While attractive in terms of output quality, the computational burden prohibits long-span LMs in practice.

There are now alternatives to N-best rescoring with long-span LMs [1][13]. [1] presents a *hill climbing* procedure on ASR lattices that iteratively searches for a higher-scoring hypothesis in a local neighborhood of the current-best hypothesis; the error-reduction from hill climbing matches or surpasses N-best rescoring, with up to two orders of magnitude reduction in the number of evaluated utterances. We seek a similar reduction in the computational complexity of globally normalized discriminative training.

The key idea in this paper may be described as *discriminative hill climbing*. The main idea of a discriminative LM is to learn how to prefer a correct sequence hypothesis over all incorrect alternatives, while the main idea of hill climbing is to move incrementally towards the correct sequence hypothesis via local changes to a current hypothesis. The new LM training procedure we present combines these two ideas: using long-span features to learn how to prefer a "locally better" hypothesis, so that when the trained LM is applied in conjunction with hill climbing, a tendency to move towards the correct hypothesis will result. In effect, we learn how to hill climb.

The remainder of this paper proceeds as follows. We begin by introducing the language model of [9], which provides the parameterization we use in our model. We summarize the hill climbing algorithm of [1] and present discriminative hill climbing. We evaluate in terms of WER and training speedup on the Broadcast News speech recognition task.

II. Discriminatively Training Syntactic LMs

Syntactic language models extend beyond the local n-gram context to incorporate long range syntactic dependencies in the sequence. Recent work has favored discriminative feature based approaches that allow for extracting arbitrary features from the syntactic parse tree. We use the framework (and much of the notation) of Collins et al. [9] and begin with a review of this model.

In speech recognition, an acoustic model P_a induces a conditional distribution over acoustic sequences $\mathbf{a}$ given a sequence of words $\mathbf{w}$. An n-gram language model P_l induces a prior distribution over word sequences. Given these models, recognition is formulated as a search task for finding the most likely word sequence $\mathbf{w}^*$ from the acoustics:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \left(\beta \log P_l(\mathbf{w}) + \log P_a(\mathbf{a}|\mathbf{w})\right) \tag{1}$$

where $\beta > 0$ is the usual language model factor.

Collins et al. extend this model to include generic features; a feature vector $\Phi(\mathbf{a}, \mathbf{w}) \in \mathbb{R}^d$ that encodes arbitrary features of $\mathbf{a}$ and $\mathbf{w}$. A weighted linear combination of features and model parameters $\alpha \in R^d$ is added to the model (Eq. 1):

$$\beta \log P_l(\mathbf{w}) + \log P_a(\mathbf{a}|\mathbf{w}) + \langle \alpha, \Phi(\mathbf{a}, \mathbf{w}) \rangle \tag{2}$$

While $\Phi(\mathbf{a}, \mathbf{w})$ can include features from the input acoustic sequence, we follow Collins et al. and use word sequence dependent features alone. To achieve a compact notation, the first feature is the baseline (first-pass) recognizer score:[1]

$$\Phi_1(\mathbf{a}, \mathbf{w}) = \beta \log P_l(\mathbf{w}) + \log P_a(\mathbf{a}|\mathbf{w}) \tag{3}$$

which yields the new model form:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \langle \alpha, \Phi(\mathbf{a}, \mathbf{w}) \rangle \tag{4}$$

We use the features of Collins et al. (see §3 of Collins et al.), which are based on part of speech tags and a syntactic parse of the word sequence. These features include regular n-gram (in our case 3-gram) features, n-tag features ($[t_{i-2}, t_{i-1}, t_i, w_i t_i]$, where t_i is the POS tag at position i and w_i is the corresponding word, i.e. POS(1) in Collins et al.), and head-to-head (H2H) dependencies.

A. Parameter Estimation

To estimate the *global-linear* model parameters α, Collins et al. rely on the Perceptron learning algorithm, a popular online learning algorithm with wide applications to structured learning problems in NLP [14], and in particular, discriminative language modeling for speech recognition [11]. As an online algorithm, the Perceptron processes training instances one at a time, updating α after each example.

For each training example in each algorithm iteration t, the best scoring hypothesis $\mathbf{w}_m$ according to current model parameters α is selected from the set of possible word sequences in $\mathbf{GEN}(\mathbf{a}_i)$, which is a set of possible word sequence candidates under the first-pass (baseline) recognizer for $\mathbf{a}_i$ (Figure 1). $\mathbf{w}_m$ is compared to $\mathbf{w}_o \in \mathbf{GEN}(\mathbf{a}_i)$, the best available sequence as compared to the reference $\mathbf{s}$ in terms of WER (edit distance $d(\mathbf{w}, \mathbf{s})$). If current α does not yield $\mathbf{w}_o$, an update is made based on $\mathbf{w}_o$ and $\mathbf{w}_m$.

[1]We denote this feature as *baseline feature* throughout the paper.

```
Input:
  Training examples $\{(\mathbf{a}_i, \mathbf{s}_i)\}|_{i=1}^{M}$.
  A feature-vector representation $\Phi(\mathbf{a}, \mathbf{w}) \in \mathbb{R}^d$.
  $\mathbf{GEN}(\mathbf{a}_i)$ : the candidate list for example $\mathbf{a}_i$.
  $\mathcal{T}(\mathbf{w}_i)$: the parse trees for each $\mathbf{w}_i \in \mathbf{GEN}(\mathbf{a}_i)$
Algorithm:
for $t$ : 1..$T$                                                    [Iterations]
  for $i$ : 1..$M$                                                  [Examples]
    $\mathbf{w}_m = \arg\max_{\mathbf{w} \in \mathbf{GEN}(\mathbf{a}_i)} \langle \alpha, \Phi(\mathbf{a}_i, \mathbf{w}) \rangle$
    $\mathbf{w}_o = \arg\min_{\mathbf{w} \in \mathbf{GEN}(\mathbf{a}_i)} d(\mathbf{w}, \mathbf{s}_i)$
    if $\mathbf{w}_m \neq \mathbf{w}_o$                                      [If Mistake]
      for $j$ : 2..$d$                                              [Update]
        $\alpha_j = \alpha_j + \Phi_j(\mathbf{a}_i, \mathbf{w}_o) - \Phi_j(\mathbf{a}_i, \mathbf{w}_m)$
Output: $\alpha$
```

Fig. 1. The Perceptron training algorithm for discriminative language modeling [9]. α_1 is the weight given to the first-pass (baseline) recognizer score and is tuned on development data.

B. N-Best Lists as $\mathbf{GEN}(\mathbf{a})$

The creation of the set $\mathbf{GEN}(\mathbf{a})$, candidate word sequences, is required for training and decoding, when a trained language model is used to rescore the baseline recognizer's output. For n-gram language models, the set of possible outputs $\mathbf{GEN}(\mathbf{a})$ are encoded as a *lattice*: a directed acyclic graph (DAG) with unique start and end nodes, nodes time-stamped with respect to the speech signal $\mathbf{a}$, and edges labeled with words $\mathbf{w}$. Each path in the DAG from start to end corresponds to a candidate time-aligned transcript $\mathbf{w} = w_1 w_2 \ldots w_n$ of $\mathbf{a}$. Finding $\mathbf{w}_m \in \mathbf{GEN}(\mathbf{a})$ can be solved efficiently by composing weighted finite-state automata (WFSA).

However, long-span dependencies (e.g. syntactic features) prevent a LM from being represented efficiently as a WFSA and it is intractable to enumerate all paths. Instead, the conventional solution used by Collins et al. is to approximate the search space with an N-best list, the N highest scoring paths from the baseline recognizer. Still, this can pose significant computational costs as each enumerated path must be tagged and parsed. This imposes a tradeoff in the selection of N; smaller N will suffer from search errors while larger N are computationally expensive. When considering that the complexity of statistical parsers are usually on the order of $O(\ell(\mathbf{w})^2)$ or $O(\ell(\mathbf{w})^3)$, where $\ell(\mathbf{w})$ is the length of the word sequence, and that $N = 1000$ is typical, the LM is often too costly to use in practice.

III. Hill Climbing Rescoring

Given a speech utterance (lattice $\mathcal{L}$) and acoustic sequence $\mathbf{a}$, lattice rescoring under a *global-linear* model (with α) tries to solve the search problem of Eq. 4. Rastrow et al. [1] offer an efficient rescoring method based on the well-known *hill climbing* algorithm, obtaining the same WER reductions with *two orders of magnitude* fewer utterance-length hypothesis evaluations compared to conventional N-best rescoring. Hill climbing is an iterative search that greedily improves the current best path by examining neighborhoods in $\mathcal{L}$. Rastrow et al. define the neighborhood of $\mathbf{w} = w_1 w_2 \ldots w_n$ at position i to be all paths in the lattice whose word sequence may be obtained by editing — deleting, substituting or inserting a word to the left of — w_i; $\mathcal{N}(\mathbf{w}, i)$ denotes this "distance 1

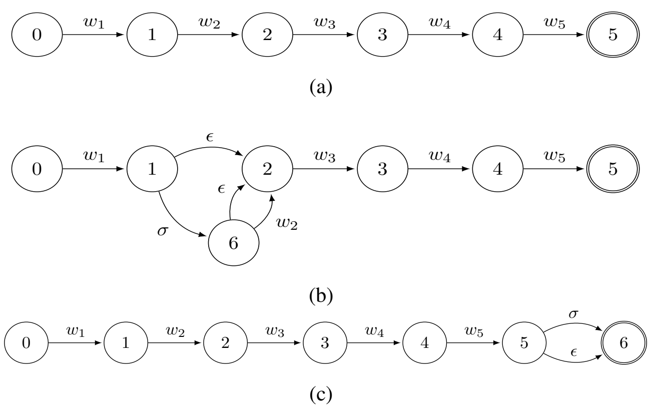

Fig. 2. The FSA representation of the neighborhood set of a given path: (a) Original path. (b) Neighborhood for a specific position. (c) Neighborhood for the last position (required to allow insertions to the right of the last word). σ is a special arc which can be replaced by any word. For example, the path with σ followed by w_2 in (b) corresponds to one insertion to the left of w_2 .

```
Initialization:
$\mathcal{L} \leftarrow$ WFSA of the initial (first-pass) lattice
$\mathbf{v} \leftarrow$ Viterbi path of the initial lattice
$N \leftarrow length(v)$
Algorithm:
repeat
    $i \leftarrow 1$                                                  [Start at position 1]
    while $i \leq N+1$ do
        create FSA $LC(\mathbf{v}, i)$
        $LN(\mathbf{v}, i) \leftarrow LC(\mathbf{v}, i) \circ \mathcal{L}$                       [Neighboring paths]

        $\mathbf{v} \leftarrow \arg\max_{\mathbf{w} \in LN(\mathbf{v},i)} \langle \alpha, \Phi(\mathbf{a}, \mathbf{w}) \rangle$              [Best $\mathbf{w}$]

        $N \leftarrow length(\mathbf{v})$                          [Adjust length and position]
        if DELETION
            $i \leftarrow i$-1
        $i \leftarrow i$+1
    end while
until $\mathbf{v}$ does not change                                   [Stopping criterion]
```

Fig. 3. Hill climbing algorithm using *global-linear* models for a given utterance with acoustic sequence $\mathbf{a}$ and initial (baseline) lattice $\mathcal{L}$.

at position i" neighborhood of $\mathbf{w}$. Given a selected position i in word sequence $\mathbf{w}$, all paths in $\mathcal{L}$ corresponding to word sequences $\mathbf{w}' \in \mathcal{N}(\mathbf{w}, i)$ and the current edge scores are extracted and rescored using Eq. 4; the hypothesis $\hat{\mathbf{w}}'(i)$ with the highest score becomes the new $\mathbf{w}$. The process is repeated with new positions until $\mathbf{w} = \hat{\mathbf{w}}'(1) = \ldots = \hat{\mathbf{w}}'(n)$, i.e. when $\mathbf{w}$ is the highest scoring hypothesis in all 1-neighborhoods.

Rastrow et al. show that $\mathcal{N}(\mathbf{w}, i)$ can be efficiently computed using FSA operations. They define the machine $LC(\mathbf{w}, i)$: all word sequences that can be generated from $\mathbf{w}$ with one edit at position i (Figure 2). A second machine $LN(\mathbf{w}, i) \leftarrow LC(\mathbf{w}, i) \circ \mathcal{L}$ restricts the neighborhood $\mathcal{N}(\mathbf{w}, i)$ to word sequences in $\mathcal{L}$ (the search space) by composing $LC(\mathbf{w}, i)$ with $\mathcal{L}$.[2]

The final rescoring algorithm includes three operations (Figure 3): initialization of the best scoring word sequence in $\mathcal{L}$, generating the neighborhood $LN(\mathbf{w}, i)$ for each position in i, and selecting the new highest scoring sequence $\mathbf{v}$ from $\mathbf{w} \in LN(\mathbf{w}, i)$ based on $\langle \alpha, \Phi(\mathbf{a}, \mathbf{w}) \rangle$. In practice, algorithm run time is dominated by the neighborhood rescoring step. Note that since hill climbing algorithms are greedy (no global maximum guarantee), Rastrow et al. use random-restarts (repeating hill climbing with different initial word sequences $\mathbf{v}$).[3]

IV. Discriminative Hill Climbing Training

We are now ready to present our training algorithm. Just as hill climbing can improve over N-best lists for rescoring, it can be used to improve discriminative LM training. The key insight is that discriminative learning, which learns to select the correct hypothesis over incorrect hypotheses, and hill climbing, which moves towards the best sequence, can be integrated by learning how to take steps towards the correct hypothesis. *Discriminative Hill Climbing* effectively learns how to hill climb.

Observe that during N-best training, $\mathbf{w}_m$ is chosen after scoring N hypotheses, which means we must wait for N steps before updating α. In contrast, hill climbing produces $\mathbf{w}_m$ incrementally, which means that a scoring mistake can be observed before $\mathbf{w}_m$ is fully produced. By updating α as soon as a single hill climbing mistake is made, we can make model corrections faster and learn how to hill climb correctly (in the right direction).

Consider the example utterance in Figure 5(3). The baseline model produces an incorrect word sequence (Figure 5 (baseline)), which mistakes the word "abortion" for "Al Gore's" and "them" for "the". If Figure 5(2) was the best path in an N-best list, which corrects only one mistake, standard N-best list training would update only after all paths have been scored. Instead, the hill climbing algorithm gradually corrects the baseline sequence (Figure 5(1-3)). If "the" is not corrected when the 11th position is visited, the algorithm could immediately update, encouraging hill climbing to move in a better direction.

Discriminative hill climbing proceeds as follows (Figure 4). For each training utterance (given $\mathbf{a}$, $\mathbf{s}$ and $\mathcal{L}$), select the Viterbi path in $\mathcal{L}$ as $\mathbf{v}$. Hill climbing starts improving from $\mathbf{v}$ using current model parameters α. Select a position i in $\mathbf{v}$ (we start from 0 and move left to right) and use the algorithm in §III to generate $LN(\mathbf{v}, i)$. Select the best path $\mathbf{w}_m \in LN(\mathbf{v}, i)$ by parsing, extracting features $\Phi(\mathbf{a}, \mathbf{w})$, scoring with α each path in the neighborhood. Before continuing to the next sequence position, we evaluate if $\mathbf{w}_m$ was the best choice given the reference $\mathbf{s}$. Since, $\mathbf{s}$ may not be in the neighborhood, we instead use the path $\mathbf{w}_o \in LN(\mathbf{v}, i)$ closest to $\mathbf{s}$, $LN(\mathbf{v}, i)$ fixes all paths besides position i, using edit distance $d(\mathbf{w}, \mathbf{s})$. This path can be found by composing the FSA $LN(\mathbf{v}, i)$ (after removing the weights) with the edit distance machine (also known as *levenshtein distance* machine) and compose the result with the finite state transducer (FST) representation of $\mathbf{s}$. The input side of the best path in the resulting composed FST will be $\mathbf{w}_o$.

[2]$LN(\mathbf{w}, i)$ therefore is a WFSA representation of the subset of word sequences $\mathbf{w}'$ in $\mathcal{N}(\mathbf{w}, i)$ that are also present in the initial lattice. The weights associated with the words in $LN(\mathbf{w}, i)$ are the scores from the baseline recognizer, which we will need as the *baseline feature* parameter (α_1) for finding the global model score according to $\langle \alpha, \Phi(\mathbf{a}, \mathbf{w}) \rangle$.

[3]Starting paths (word sequences) are selected by sampling from $\mathcal{L}$ and including the Viterbi path.

```
Initialization:
$\mathcal{L} \leftarrow$ WFSA of the initial (first-pass) lattice
$\mathbf{v} \leftarrow$ Viterbi path of the initial lattice
$\mathbf{s} \leftarrow$ reference word sequence
$N \leftarrow length(v)$
Algorithm:
repeat
  $i \leftarrow 0$
  while $i \leq N+1$ do
    create FSA $LC(\mathbf{v}, i)$
    $LN(\mathbf{v}, i) \leftarrow LC(\mathbf{v}, i) \circ \mathcal{L}$                [Neighboring paths in the lattice]
    $\mathbf{w}_m \leftarrow \arg\max_{\mathbf{w} \in LN(\mathbf{v},i)} \langle \alpha, \Phi(\mathbf{a}, \mathbf{w}) \rangle$ [Best path according to model]
    $\mathbf{w}_o \leftarrow \arg\min_{\mathbf{w} \in LN(\mathbf{v},i)} d(\mathbf{w}, \mathbf{s})$ [Best path according to edit-distance]

    if $d(\mathbf{w}_o, \mathbf{s}) < d(\mathbf{w}_m, \mathbf{s})$
      for $j : 2..d$
        $\alpha_j = \alpha_j + \Phi_j(\mathbf{a}, \mathbf{w}_o) - \Phi_j(\mathbf{a}, \mathbf{w}_m)$
      $\mathbf{v} \leftarrow \mathbf{w}_o$
    else
      $\mathbf{v} \leftarrow \mathbf{w}_m$

    $N \leftarrow length(\mathbf{v})$                [Adjusting length and position]
    if DELETION
      $i \leftarrow i$-1
    $i \leftarrow i$+1
  end while
until $\mathbf{v}$ does not change                [Stopping criterion]
```

Fig. 4. Discriminative hill climbing training for *global-linear* models for a given training utterance with acoustic sequence $\mathbf{a}$, reference $\mathbf{s}$ and lattice $\mathcal{L}$.

To determine if the current hill climbing direction improves the best path with respect to the reference, we compare both $\mathbf{w}_m$ and $\mathbf{w}_o$ to the reference $\mathbf{s}$. If $d(\mathbf{w}_o, \mathbf{s}) < d(\mathbf{w}_m, \mathbf{s})$, then the algorithm is not moving in the optimal direction. In this case, since $\mathbf{w}_m, \mathbf{w}_o \in LC(\mathbf{v}, i)$, this means that $\mathbf{w}_o$ correctly includes, deletes or substitutes a word at position i of $\mathbf{s}$ as compared to $\mathbf{w}_m$. Therefore, model parameters α should be encouraged to select $\mathbf{w}_o$ instead of $\mathbf{w}_m$, which we achieve through a Perceptron update:

$$\alpha_j = \alpha_j + \Phi_j(\mathbf{a}, \mathbf{w}_o) - \Phi_j(\mathbf{a}, \mathbf{w}_m) \ , \tag{5}$$

where α_j is the jth parameter.[4] Notice that while $\Phi(\mathbf{a}, \mathbf{w})$ is based on the entire sequence, only parameters that change based on position i will be updated. Furthermore, If there does not exist a correctable error in the neighborhood because either position i of the neighboring paths is not matched with the reference $\mathbf{s}$ (correction can *not* be made by considering the paths in the neighborhood) or the algorithm already selects the correct path, we do not update the parameters.

After an update the current word sequence becomes the best sequence: $\mathbf{v} \leftarrow \mathbf{w}_o$, which leads to faster convergence during training.[5] If no update occurs, we apply the selected path ($\mathbf{v} \leftarrow \mathbf{w}_m$) and move to the next position. We iterate over positions until hill climbing converges, i.e. no change is available in any position of the sequence. In §VI, we will empirically demonstrate that this procedure requires the evaluation of far fewer word sequences, leading to a dramatic reduction in the number of calls to the parser. Additionally, since we parse many word sequences in a neighborhood, we could obtain further speedups by using lattice parsing [15].

[4]Following [9], we tune α_1 – the baseline model weight – on development set for decoding instead of updating it during training.

[5]Alternatively, we could obtain a new $\mathbf{w}_m$ and repeat the process, or continue to the next position with no change to $\mathbf{v}$.

Remarks: Our approach is reminiscent of learning algorithms that improve sequences through iterative updates. [16] use a Monte Carlo technique for decoding phrase based translation lattices. Bi-directional learning uses a Perceptron to iteratively label each position of a word sequence, updating parameters when a single incorrect label is applied [17].

We believe our approach is well suited to speech lattices. Normal perceptron training updates using the optimal path, but it may be unrealistic for the model to select the optimal sequence from the baseline lattice, which leads to overly aggressive learning. Instead, our method encourages gradual improvement towards the best sequence, which will be selected if available. Otherwise, the model is still trained to improve as many positions of the sequence as possible. This may be more compatible with minimizing WER as we correct as many word positions as possible, as opposed to conventional Perceptron training which favors sentence error rate by selecting the best sequence overall.

A. Additional Efficiencies

We obtain further speedups by parallelization based on the distributed Perceptron of [18] (*iterative parameter mixing* with uniform weights). We divide the training examples $\{(\mathbf{a}_i, \mathbf{s}_i)\}|_{i=1}^{M}$ among S machines and train a single epoch of discriminative hill climbing on each machine in parallel, mixing the model weights α after each iteration. The mixed parameter vector is then used by each machine to start another single epoch of training. The parameter values are uniformly mixed at the end of each iteration: $\alpha_j = \frac{1}{S} \sum_{s=1}^{S} \alpha_j^s$,where t and j indicate the iteration and machine respectively.

For additional efficiency, we reduce the number of parser calls by caching parse results. For each utterance, we store a record of every sequence and its parse. If a parser call requests a previous sequence, we return the cached result. The cached parse tree structures are maintained across iterations. Note that caching cannot be used for N-best list training since every N-best entry is unique and must be parsed before training.

V. Experimental Setup

The ASR system used in this paper is based on the 2007 IBM Speech transcription system for the GALE Distillation Go/No-go Evaluation [19]. The acoustic models are state of the art discriminatively trained models trained on Broadcast News (BN) Hub4 acoustic training data, which contains about $153K$ utterances and transcriptions (400 hours of speech). To train the discriminative language model we use Hub4 utterances with length < 50, giving $115K$ training utterances ($2.6M$ words). The baseline language model is a modified Kneser-Ney backoff 4-gram model trained on 1996 CSR Hub4 Language Model data and EARS BN03 closed captions corpora (about $193M$ words).

To produce word-lattices, each training utterance is decoded/processed by the baseline ASR system. The best path in the lattice in terms of WER with respect to the transcription is used as the reference $\mathbf{s}$ for discriminative hill climbing training. The baseline (4-gram LM) 1-best output had a 10.7%

Baseline	ABORTION		LEGAL	TEAM	IS	ASKING	THE	COURT	TO	ORDER	THEM	RECOUNTING	OF	BALLOTS
(1)	(deletion)	GORE'S	LEGAL	TEAM	IS	ASKING	THE	COURT	TO	ORDER	THEM	RECOUNTING	OF	BALLOTS
(2)	AL	GORE'S	LEGAL	TEAM	IS	ASKING	THE	COURT	TO	ORDER	THEM	RECOUNTING	OF	BALLOTS
(3)	AL	GORE'S	LEGAL	TEAM	IS	ASKING	THE	COURT	TO	ORDER	THE	RECOUNTING	OF	BALLOTS

Fig. 5. An example utterance corrected by our algorithm. The first line (baseline) shows the word sequence provided by the baseline model. Errors are shown in red. After the first pass through the sequence, the algorithm delete's the word ABORTION and inserts the word GORE'S. The second pass adds the word AL and the third pass replace THEM with THE, yielding the correct transcription.

	WER	
Baseline ASR	15.07	
	Training Algorithm	
Hill Climbing Rescoring	**DHC**	**N-best**
1 initial path	14.86	15.01
5 initial paths	**14.74**	14.96

TABLE I
TEST SET RESULTS: RESCORING A BASELINE RECOGNIZER USING A SYNTACTIC LANGUAGE MODEL TRAINED WITH EITHER DISCRIMINATIVE HILL CLIMBING OR CONVENTIONAL $N = 50$-BEST TRAINING.

Parser Complexity	**Complexity-Ratio**
Linear	18.19
Quadratic	15.25
Cubic	13.08

TABLE II
COMPLEXITY-RATIO OF 1000-BEST LIST PERCEPTRON OVER DISCRIMINATIVE HILL CLIMBING AFTER 100 ITERATIONS.

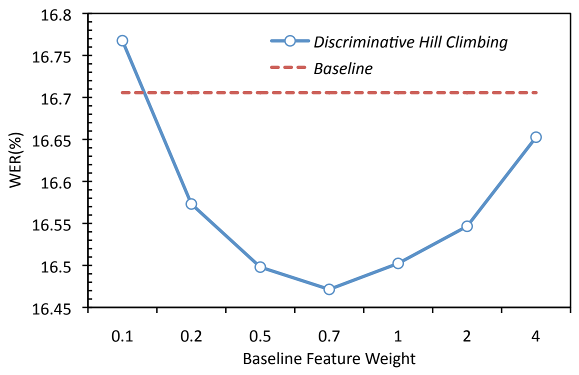

Fig. 6. WER on development data for the baseline recognizer versus discriminative hill climbing for varying values of the baseline feature α_1.

WER w.r.t to these references. The test set is the 2.5 hour rt04 Broadcast News evaluation set ($45K$ words). The dev set for tuning the *baseline feature* weight (α_1) is the dev04f Broadcast News data.

Syntactic features were extracted using the parser of [20] which achieves state of the art performance on BN. It is trained on the Broadcast News treebank from Ontonotes [21] and the WSJ Penn Treebank along with self-training on Hub4 CSR 1996 utterances. The parser has complexity of $O(l(w)^3)$.

VI. RESULTS

We begin with an evaluation of discriminative hill climbing (DHC) in terms of WER improvements on the dev and test sets. We use a model trained for 100 iterations with a total of 3.77M parameters. The baseline ASR system's output is rescored with our discriminative syntactic LM using hill climbing rescoring with either one or five starting paths (§III).

Results are reported for test data in Table I. We observe a 1.4% relative error reduction for one starting initial path and a 2.2% reduction for starting with 5 initial paths as compared to the baseline model, a statistically significant result ($p < 0.01$ using the paired permutation test). We note these improvements are over an already state of the art acoustic model, and our baseline recognizer gives results close to state of the art for the BN task. We next investigate how tuning the baseline feature affects performance on dev data. Figure 6 shows that for a wide range of parameter values, our model improves over the baseline. We selected a value of 0.7 for the test data experiments above, which reduces WER on dev data from 16.71 to 16.47 with one starting path.

Parser computational complexity prohibits training with large N-best lists. We evaluate the training method of Collins et al. with $N = 50$ (Table I); we selected $N = 50$ since it is equivalent in terms of computational complexity to discriminative hill climbing (see below.) Despite having equivalent training complexity, our method improves over N-best list training, even when both models rely on the hill climbing rescoring algorithm.[6] This performance gap can be attributed to the ability of our algorithm to consider options that appear deep in the N-list by exploring paths in the lattice. Our model is thus trained on more informative errors.[7] In fact, the model which was trained using N-best perceptron training has about 1M parameters (only a quarter of our model's size).

A. Analysis

We analyze the behavior of discriminative hill climbing and compare it to the N-best list Perceptron in terms of efficiency. First, we compare the relative training complexity in terms of the parser with hypothesized N-best perceptron training with $N = 1000$, as used by Collins et al. We compute the complexity ratio as:

$$\text{complexity-ratio} = \frac{\sum_{\mathbf{w}} l(\mathbf{w})^i \cdot N(\mathbf{w})}{\sum_{\mathbf{w}} l(\mathbf{w})^i \cdot N_{HC}(\mathbf{w})} , \qquad (6)$$

where $\mathbf{w}$ is the training utterance, $N(\mathbf{w})$ is the size of the N-best list (1000), i.e. the number of parser calls, and $N_{HC}(\mathbf{w})$ is the number parser calls for $\mathbf{w}$ during discriminative hill climbing training (taking into account cached parses). The parser complexity is indicated by i (in our case $i = 3$). A

[6]We obtain a smaller error reduction than reported in Collins et al. because we use a much stronger baseline.

[7]We also explained in §IV that discriminative hill climbing's objective function is directly related to WER whereas N-best discriminative training tries to minimize the sentence error rate.

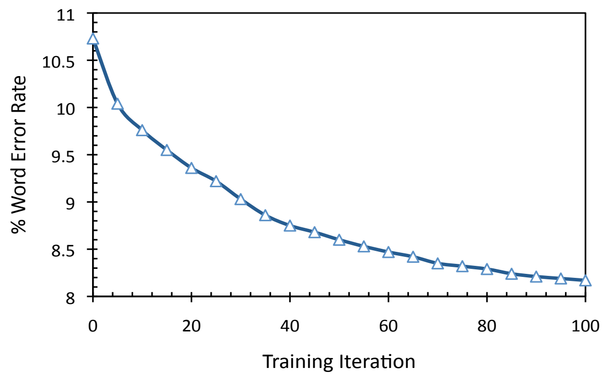

Fig. 7. The WER of training data after each training iteration.

complexity ratio of 10 means that the N-best list training complexity is an order of magnitude larger.

The average number of cumulative parser calls for each utterance ($N_{HC}(\mathbf{w})$) was 42.82. [8] The N-best list has no duplicates, so it must all be parsed (1000 parser calls before the training). Table II shows the complexity-ratio for different parser complexities. For longer utterances, $N(\mathbf{w})$ is fixed whereas $N_{HC}(\mathbf{w})$ must consider more possible paths, which decreases the gap between the two algorithms. Therefore, as i (parser complexity) increases, the term $l(w^i)$ dominates the ratio for longer utterances and the complexity-ratio decreases. For the parser used in this work, we would observe an order of magnitude improvement in complexity. For faster linear parsers, which typically require a reduction in parser accuracy, we would observe more dramatic improvements.

In terms of raw computation, our 100 training iterations took about 6.5 days on 60 machines (390 cpu days). Of this time, about 250 minutes (1.7% of the time) was spent on neighborhood generation and FSA operations for finding oracle paths. For comparison, we can use the complexity-ratio, which is based on parse statistics collected during training, to estimate that it would take more than 10 times as long with the same resources and parser to train an $N = 1000$-best list Perceptron, about 13.8 CPU *years*. Obviously, we cannot run the original Perceptron training algorithm for N=1000 with our state of the art parser.

We can also measure learning effectiveness by considering the WER of the training data after each iteration by discriminative hill climbing. Figure 7 shows the WER after each training iteration. After a single iteration, our algorithm learns to reduce the WER from 10.7% to 10.0%. Also it can be seen from the Figure that after 100 iterations, the training WER stops improving significantly.

VII. Conclusion and Future Work

We have presented discriminative hill climbing, a new training algorithm for long-span language models that normally require the use of N-best lists for training and rescoring. For a syntactic language model that relies on a parser, we demonstrate improvements in WER with *an order of magnitude* reduction in training complexity. We also observe improvements over N-best training with comparable computational resources. Our algorithm can be used to train complex language models on large corpora with state of the art parsers.

There are a number of promising avenues for future work, including considering more aggressive online algorithms, or scaling training to much larger corpora by utilizing more efficient parsers, including lattice parsers. Additionally, improved training time means we can effectively explore new LM features.

[8]Note that we chose to evaluate $N = 50$ in the previous section as it is roughly equivalent to the 42.82 parser calls used by our method. In fact, allowing 50 instead of 43 is being generous to the baseline.

References

[1] A. Rastrow, M. Dreyer, A. Sethy, S. Khudanpur, B. Ramabhadran, and M. Dredze, "Hill climbing on speech lattices : A new rescoring framework," in *ICASSP*, 2011.

[2] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *ICASSP*, 1995.

[3] S. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, 1987.

[4] S. Khudanpur and J. Wu, "Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling," *Computer Speech and Language*, pp. 355–372, 2000.

[5] P. Brown, P. Desouza, R. Mercer, V. Pietra, and J. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[6] T. Niesler, E. Whittaker, and P. Woodland, "Comparison of part-of-speech and automatically derived category-based language models for speech recognition," in *ICASSP*, 1998.

[7] D. Filimonov and M. Harper, "A joint language model with fine-grain syntactic tags," in *EMNLP*, 2009.

[8] C. Chelba and J. Frederick, "Structured language modeling," *Computer Speech and Language*, vol. 14, no. 4, pp. 283–332, 2000.

[9] M. Collins, B. Roark, and M. Saraclar, "Discriminative syntactic language modeling for speech recognition," in *ACL*, 2005.

[10] H.-K. J. Kuo, B. Kingsbury, and G. Zweig, "Discriminative training of decoding graphs for large vocabulary continuous speech recognition," in *Proc. ICASSP*, vol. 4, 2007, pp. 45–48.

[11] B. Roark, M. Saraclar, and M. Collins, "Discriminative n-gram language modeling," *Computer Speech & Language*, vol. 21(2), 2007.

[12] A. Rastrow, A. Sethy, and B. Ramabhadran, "Constrained Discriminative Training of N-gram Language Models," in *ASRU*, 2009.

[13] A. Deoras and F. Jelinek, "Iterative decoding: A novel re-scoring framework for confusion networks," in *ASRU*, 2009.

[14] M. Collins, "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms," in *EMNLP*, 2002.

[15] J. Cheppalier, M. Rajman, R. Aragues, and A. Rozenknop, "Lattice parsing for speech recognition," in *Sixth Conference sur le Traitement Automatique du Langage Naturel (TANL'99)*, 1999.

[16] A. Arun, B. Haddow, P. Koehn, A. Lopez, P. Blunsom, and C. Dyer, "Monte carlo techniques for phrase-based translation," *Machine Translation*, vol. 24, no. 2, August 2010.

[17] L. Shen, G. Satta, and A. Joshi, "Guided learning for bidirectional sequence classification," in *Annual Meeting-Association for Computational Linguistics*, vol. 45, no. 1, 2007, p. 760.

[18] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *NAACL-HLT*, 2010.

[19] S. Chen, B. Kingsbury, L. Mangu, D. Povey, G. Saon, H. Soltau, and G. Zweig, "Advances in speech transcription at IBM under the DARPA EARS program," *IEEE Transactions on Audio, Speech and Language Processing*, pp. 1596–1608, 2006.

[20] Z. Huang and M. Harper, "Self-Training PCFG grammars with latent annotations across languages," in *EMNLP*, 2009.

[21] R. Weischedel, S. Pradhan, L. Ramshaw, M. Palmer, N. Xue, M. Marcus, A. Taylor, C. Greenberg, E. Hovy, R. Belvin, and A. Houston, *OntoNotes Release 2.0*, Linguistic Data Consortium, Philadelphia, 2008.