CORRECTIVE LANGUAGE MODELING FOR LARGE VOCABULARY ASR WITH THE PERCEPTRON ALGORITHM

Brian Roark[†]*, Murat Saraclar*[†]*, and Michael Collins*[‡]

[†]AT&T Labs-Research, 180 Park Ave., Florham Park, NJ 07932, USA
[‡]MIT Artificial Intelligence Laboratory, Room NE43-723
200 (545) Technology Square, MIT Building NE43, Cambridge, MA 02139
[†]{roark,murat}@research.att.com [‡]mcollins@ai.mit.edu

ABSTRACT

This paper investigates error-corrective language modeling using the perceptron algorithm on word lattices. The resulting model is encoded as a weighted finite-state automaton, and is used by intersecting the model with word lattices, making it simple and inexpensive to apply during decoding. We present results for various training scenarios for the Switchboard task, including using n-gram features of different orders, and performing n-best extraction versus using full word lattices. We demonstrate the importance of making the training conditions as close as possible to testing conditions. The best approach yields a 1.3 percent improvement in first pass accuracy, which translates to 0.5 percent improvement after other rescoring passes.

1. INTRODUCTION

Various approaches have been proposed to directly optimize models to minimize error rate [2, 6, 7, 8, 10, 11, 12], ranging from discriminative parameter adjustment algorithms [2, 12] to post-processing on recognizer output [8, 10] and confusion network construction [7, 8]. One of the earliest papers on the topic [2] motivated its approach by reference to the perceptron algorithm, and, proposed a technique for corrective training of discrete output HMM parameters for acoustic modeling. In this paper, we investigate the use of the perceptron algorithm for language modeling, under various training scenarios, using word-lattice output and trigram, bigram and unigram features. We show error-rate reductions of between 0.5 and 1.3 percent on the Switchboard 2002 evaluation set.

The basic idea behind the algorithm is to move the parameter values – in our case n-gram feature costs – in such a way that features associated with the lowest error-rate hypotheses in the training lattices have their costs reduced and features associated with the lowest cost hypotheses have their costs increased.

Our approach has several nice properties. First, the algorithm converges to its best performance within one or two passes over the training data, leading to relatively short training times. Second, because it extracts features from only two strings per utterance for each iteration, rather than from all paths in the word lattice, it is relatively parsimonious in the size of the final feature set. Finally, because it involves a simple linear combination of n-gram feature weights, it can be easily encoded as a weighted finite-state automaton, and simply intersected with word lattices to apply the model.

The paper is organized as follows. First we will present the perceptron training algorithm introduced in [3], in the context of language modeling. Next we will discuss encoding the perceptron model in a deterministic weighted finite state automaton, which allows for rapid intersection with the lattices and counting of features for model update. Finally, we will present empirical trials with different methods for generating training lattices and with different feature sets.

2. THE GENERAL FRAMEWORK

In this section we describe a general framework of linear models that could be applied to a diverse range of tasks, e.g. POS-tagging or ASR hypothesis re-ranking. We then describe a particular method for parameter estimation, which is a generalization of the perceptron algorithm.

2.1. Linear models for language modeling

We follow the framework outlined in [3, 4]. The task is to learn a mapping from inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. For example, $\mathcal{X}$ might be a set of utterances, with $\mathcal{Y}$ being a set of possible transcriptions. We assume:

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

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

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

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.

This framework is general enough to encompass several tasks in natural language modeling, such as part-of-speech tagging and named entity extraction, as detailed in [3]. In this paper we are interested in ASR, where (x_i, y_i), $\mathbf{GEN}$, and Φ can be defined as follows:

- Each training example (x_i, y_i) is a pair where x_i is an utterance, and y_i is the gold-standard transcription for that utterance, either the reference transcription or the hypothesis transcription with the minimum error rate.
- Given an input utterance x, $\mathbf{GEN}(x)$ is a set of hypothesized transcriptions for that sentence. In our case, $\mathbf{GEN}(x)$ will be defined as the paths in a word-lattice output from a baseline recognizer.
- The representation $\Phi(x, y)$ tracks arbitrary features of candidate transcriptions. For example, $\Phi_i(x, y)$ could be defined as the unigram count in the candidate transcription (x, y) of word w_i.

Inputs: Training examples (x_i, y_i)
Initialization: Set $\bar{\alpha} = 0$
Algorithm:
For $t = 1 \ldots T, i = 1 \ldots N$
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.

2.2. The Perceptron Algorithm for Parameter Estimation

We now consider the problem of setting the parameters, $\bar{\alpha}$, given training examples (x_i, y_i). We will briefly review the perceptron algorithm, and its convergence properties – see [3] for a full description. The algorithm and theorems are based on the approach to classification problems described in [5].

Figure 1 shows the algorithm. Note that the most complex step of the method is finding $z_i = \text{argmax}_{z \in \mathbf{GEN}(x_i)} \Phi(x_i, z) \cdot \bar{\alpha}$, which is the decoding problem. We will show in the next section that, in the current case, this can be done with efficient intersection and bestpath extraction algorithms available in the FSM library [9].

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*

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

($||\mathbf{U}||$ *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 $z_i \neq y_i$ for some (t, i) pair. We can then state the following theorem (see [3] 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.[1] 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 [3] depend on notions of separability rather than the size of $\mathbf{GEN}$.[2]

[1]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 $\frac{R^2}{\delta^2}$ passes over the training set, each with a single error, before converging.

[2]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 [3] for discussion.

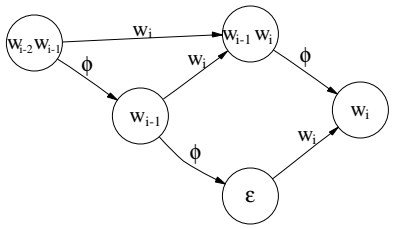

Fig. 2. Representation of a trigram model with failure transitions.

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 $P(x, y)$)? [5] discusses how the theory for classification problems can be extended to deal with both of these questions; [3] describes how these results apply to NLP problems.

As a final note, following [3], we used the *averaged* parameters from the training algorithm in decoding test examples in our experiments, because this provides better generalization to unseen test examples. 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 $\bar{\alpha}_{AVG} = \sum_{i,t} \bar{\alpha}_i^t / NT$. [5] originally proposed the averaged parameter method; it was shown to give substantial improvements in accuracy for tagging tasks in [3].

3. WEIGHTED AUTOMATA ENCODING OF MODEL

This section describes how to encode the perceptron model in a weighted finite state automaton (WFSA), so that the model can be used to re-weight a word-lattice by simply intersecting the lattice with the automaton. Efficient re-weighting of lattices is critical, since it is part of both training and use. By encoding the models as a WFSA, we can take advantage of general algorithms implemented in the AT&T FSM library.

The feature set that we will investigate in the current paper includes n-gram features plus the scaled cost given by the baseline ASR system, i.e. $-\lambda \log \mathrm{P}(A, W)$. In principle, we could learn the scale λ to give the lattice cost in the same manner as the weights to give to n-gram parameters, but for the trials that we will present in the next section, we treat the weight of the lattice cost as a constant scaling factor, and present results with various values[3].

An n-gram model can be efficiently represented in a deterministic weighted finite-state automaton, through the use of failure transitions [1]. 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

[3]Note that lattice weights are interpreted as costs, which changes the sign in the algorithm presented in figure 1.

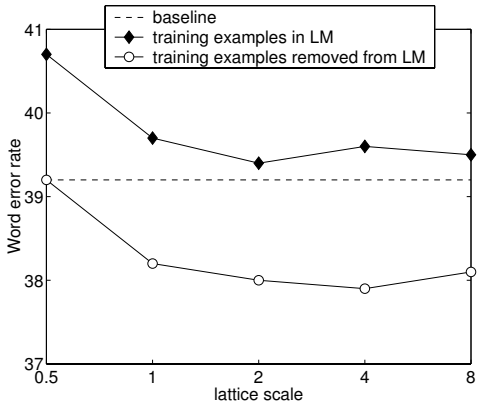

Fig. 3. Leaving all utterances in the training set for the language model that produces the training lattice, versus removing utterances from the training for the language model that produces their word-lattice. Word error rate on Switchboard 2002 eval set at various lattice scale factors.

the input does not label 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 [1] 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 $\leq k$.

The perceptron algorithm is incremental, meaning that the model parameters are updated after every training sentence. Because updating the n-gram parameters involves both the feature summing described in the previous paragraph, and the perceptron averaging presented in the last section, frequently updating the model can be rather expensive. However, since a relatively small subset of features change value after each sentence, one can improve efficiency by summing and averaging only those transitions which include the weight of updated features.

By encoding the perceptron model as an automata in this way, the algorithm in figure 1 reduces to a series of general finite-state operations, which were performed using the FSM library. Given a word lattice $\mathcal{L}_i$, which encodes a weighted set of alternative hypothesis transcriptions for utterance i, and a perceptron automata $\mathcal{P}$, z_i from figure 1 is simply the least cost path through their intersection, $\lambda\mathcal{L}_i \circ \mathcal{P}$, where λ is the scale assigned to the word-lattice costs. The features from this path are counted, as are the features from the gold standard transcription, and the feature values are updated as presented in the algorithm.

4. EMPIRICAL RESULTS

We present empirical results on the Switchboard 2002 eval test set. The test set consists of 6081 sentences (63804 words) and has three subsets: Switchboard 1, Switchboard 2, Switchboard Cellular.

Our training set consisted of 276726 transcribed utterances

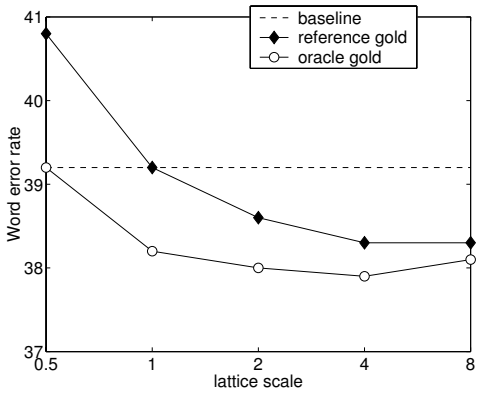

Fig. 4. Using the reference transcription as the gold standard, versus the oracle best path through the lattice. Word error rate on Switchboard 2002 eval set at various lattice scale factors.

(3047805 words), with an additional 20854 utterances (249774 words) as held out data. For each utterance, we produced a weighted word-lattice, representing alternative transcriptions, from the ASR system. From each word lattice, we extracted the oracle best path, 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 percent.

To produce the word-lattices, each training utterance is processed by the baseline ASR system. However, these same utterances are what the acoustic and language models are built from, which leads to better performance on the training utterances than can be expected when the ASR system processes unseen utterances. To somewhat control for this we partitioned the training set into 28 sets, and built baseline Katz backoff trigram models for each set by including only transcripts from the other 27 sets. 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.

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

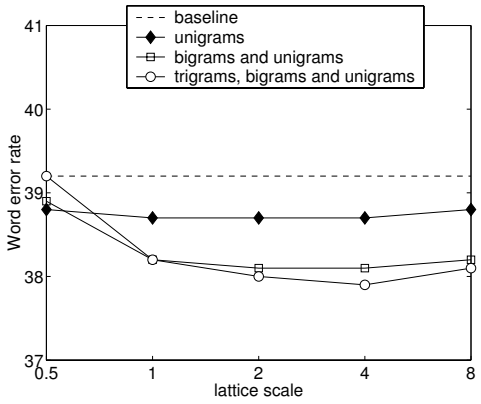

Fig. 5. Using feature sets with n-grams of different orders. Word error rate on Switchboard 2002 eval set at various lattice scale factors.

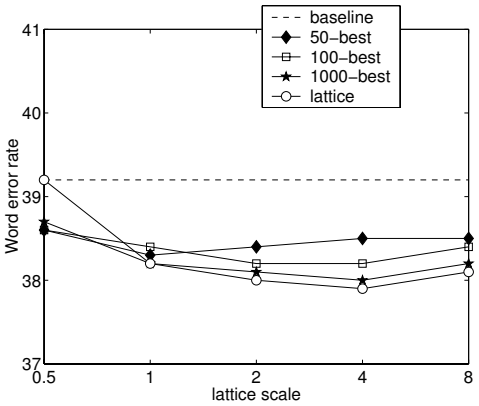

Fig. 6. N-best extraction on training lattices with various values of N, versus using the lattices. Word error rate on Switchboard 2002 eval set at various lattice scale factors.

percent improvement over the baseline, 37.9 percent WER versus 39.2. All scales λ 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 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[4]. 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. The standard scenario outlined above yields a 1.3 percent improvement over the first pass accuracy results. The improvement drops to 0.5 percent after rescoring. This can be explained by the mismatch between the training and test conditions. Perhaps, by making the training lattices more similar to this rescoring condition, further improvement can be obtained. The full system also has speaker normalization and adaptation as well as another rescoring pass with more detailed acoustic models. Although we expect the effect of acoustic model changes to be minor, we may need to better integrate the training setup with the full system for improved results.

5. DISCUSSION

Regarding the training, several observations can be made. In every training scenario, the best perceptron model was obtained after only one or two passes over the training data. The approach is fairly parsimonious in the feature space, since only n-grams from the best scoring path and the oracle path are updated in the model. In the perceptron built in our standard scenario, the total number of features in the model is 1408571, consisting of 30642 unigram features, 438425 bigram features and 939504 trigram features. Furthermore, techniques which reduce the size of the model – e.g. n-best extraction and only using bigrams and unigrams – have little impact on accuracy.

[4]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.

6. REFERENCES

[1] C. Allauzen, M. Mohri, and B. Roark. Generalized algorithms for constructing language models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 40–47, 2003.

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

[3] M. Collins. 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)*, pages 1–8, 2002.

[4] M. Collins. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In *to appear*. 2002.

[5] Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 3(37):277–296, 1999.

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

[7] L. Mangu, E. Brill, and A. Stolcke. Finding consensus in speech recognition: word error minimization and other application of confusion networks. *Computer Speech and Language*, 14(4):373–400, 2000.

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

[9] M. Mohri, F. C. N. Pereira, and M. Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231:17–32, January 2000.

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

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

[12] A. Stolcke and M. Weintraub. Discriminitive language modeling. In *Proceedings of the 9th Hub-5 Conversational Speech Recognition Workshop*, 1998.