DISCRIMINATIVE TRAINING OF DECODING GRAPHS FOR LARGE VOCABULARY CONTINUOUS SPEECH RECOGNITION

Hong-Kwang Jeff Kuo, Brian Kingsbury

IBM T.J. Watson Research Center,
Yorktown Heights, NY 10598
{hkuo,bedk}@us.ibm.com

Geoffrey Zweig

Microsoft Research,
Redmond, WA
gzweig@microsoft.com

ABSTRACT

Finite-state decoding graphs integrate the decision trees, pronunciation model and language model for speech recognition into a unified representation of the search space. We explore discriminative training of the transition weights in the decoding graph in the context of large vocabulary speech recognition. In preliminary experiments on the RT-03 English Broadcast News evaluation set, the word error rate was reduced by about 5.7% relative, from 23.0% to 21.7%. We discuss how this method is particularly applicable to low-latency and low-resource applications such as real-time closed captioning of broadcast news and interactive speech-to-speech translation.

***Index Terms*—** Discriminative training, Finite-state decoding graph, Language model, Pronunciation model, Low-resource speech recognition.

1. INTRODUCTION

In recent years, it has become popular to use an integrated finite-state decoding graph as a pre-compiled search space for efficient decoding for large-vocabulary speech recognition [1]. This decoding graph can be thought of as a finite-state machine that results from the composition of a few weighted finite-state transducers (wFSTs) that incorporate the statistical language model (LM), the pronunciation model, and the decision trees that expand context-independent phones to context-dependent units. With appropriate optimizations, the decoding graph can be made efficient for speech decoding.

Discriminative training of the language model for speech recognition has become an active area of research [2, 3, 4, 5, 6]. The motivation is clear: instead of using maximum likelihood to estimate the LM probabilities, the LM parameters are trained on speech data and corresponding transcripts to minimize the actual speech recognition error rate. In the framework of speech recognizers using an integrated finite-state decoding graph, one can either discriminatively train the LM before constructing the decoding graph or, as recently proposed [7], one can create the decoding graph and then discriminatively train the transition weights in the graph.

Potential advantages of training the decoding graph instead of just the language model include the following. First, the decoding graph combines several models (the language model, pronunciation model, decision trees, silence insertion penalty, etc.) and in some cases it would be better to perform end-to-end optimization of the combined model rather than just one model separately. In addition, it is possible to learn LM context dependent pronunciation probabilities, e.g. "for the *record*" vs. "need to *record*."

In this paper, we extend previous work on discriminative graph training [7] to large vocabulary continuous speech recognition, using context-dependent acoustic models instead of context-independent models. All transition weights in the decoding graph are adjustable except for zero-cost self loops. We also describe methods using FST tools to make it possible to perform discriminative graph training on a large decoding graph and a large amount of training data.

2. DISCRIMINATIVE TRAINING

In this section, we describe how the transition weights of an integrated finite-state decoding graph are adjusted discriminatively to improve the score separation of the correct word sequence from the competing word sequence hypothesis, using the Minimum Classification Error (MCE) criterion [8, 9]. The treatment is similar to [7], with one difference being that we use context-dependent acoustic models, so the sequences are context-dependent state sequences.

Note that the integrated decoding graph (call it G) is essentially a classical Hidden Markov Model (HMM) with two sets of parameters: the acoustic model Λ, consisting of the Gaussian densities in the HMM states, and the transition weights Γ, which specify the costs of transitions between the HMM states. The decoding graph can be constructed according to procedures described in [1, 10], which involve FST composition and optimization of the the language model, pronunciation model, and decision trees of the context-dependent HMM. The language model is a back-off n-gram language model, trained using the conventional maximum likelihood criterion and appropriate smoothing such as modified Kneser-Ney. The pronunciation probabilities may be arbitrarily set to uniform or may be based on estimates from aligning pronunciation variants ("lexemes") to the speech training data.

Given an acoustic observation sequence $X = x_1, x_2, \dots, x_t$ representing the speech signal and a word sequence W, the conditional likelihood of X is approximated as the score of the best path $S = S_1, S_2, \dots, S_t$ through G for input X and output W. We define a discriminant function to be this score, which is a weighted combination of the sum of acoustic log likelihoods $a(X, W, S; \Lambda)$ and transition weights $b(W, S; \Gamma)$:

$$g(X, W, S; \Lambda, \Gamma) = \alpha \cdot a(X, W, S; \Lambda) + b(W, S; \Gamma), \qquad (1)$$

where α is the acoustic model weight. Note that the path S is a sequence of hidden states that actually specify a lexeme (specific pronunciation variant of a word) sequence as well as a leaf (context-dependent HMM state) sequence. The sum of the transition weights of the path includes the sum of the language and pronunciation model log probabilities of the associated lexeme sequence (plus other parameters such as word or silence insertion penalties, etc.).

A common strategy for a speech recognizer is to search for the word sequence W_1 with the largest value for this function:

$$W_1 = \operatorname*{argmax}_{W,S} g(X, W, S; \Lambda, \Gamma). \tag{2}$$

Let W_0 be the known correct word sequence. The misclassification function is defined to be the difference between the discriminant function and the anti-discriminant function, which is normally an $\mathrm{L_p}$ norm weighted combination of the N-best competing hypotheses [3]. For simplicity, we follow [7] and just consider the decoded (single best) hypothesis W_1. Let the misclassification function be

$$d(X; \Lambda, \Gamma) = -g(X, W_0, S_0; \Lambda, \Gamma) + g(X, W_1, S_1; \Lambda, \Gamma). \tag{3}$$

When this misclassification function is strictly positive, a sentence recognition error has been made. To formulate an error function appropriate for gradient descent optimization, a smooth, differentiable function ranging from 0 to 1 such as the sigmoid function is chosen to be the *class loss function* for a specific utterance X_i:

$$l_i(X_i) = l(d(X_i)) = \frac{1}{1 + \exp(-\gamma d(X_i) + \theta)}, \tag{4}$$

where γ and θ are constants which control the slope and the shift of the sigmoid function, respectively. Our objective is to minimize the loss function over all utterances in the training corpus:

$$l(\mathbf{X}) = \sum_i l_i(X_i). \tag{5}$$

The transition-weight parameters can be adjusted iteratively (with step size ϵ) to minimize the objective function using the following update equation:

$$\Gamma_{t+1} = \Gamma_t - \epsilon \nabla l(\mathbf{X}; \Lambda_t, \Gamma_t). \tag{6}$$

The gradient of the loss function is

$$\nabla l(\mathbf{X}; \Lambda_t, \Gamma_t) = \sum_i \frac{\partial l_i}{\partial d_i} \frac{\partial d(X_i; \Lambda, \Gamma)}{\partial \Gamma}, \tag{7}$$

where the first term is the slope associated with the sigmoid class-loss function and is given by:

$$\frac{\partial l_i}{\partial d_i} = \gamma l(d_i)(1 - l(d_i)). \tag{8}$$

If we regard Γ as a vector of transition weights s_j, to compute $\frac{\partial d(X_i;\Lambda,\Gamma)}{\partial \Gamma}$, we can take the partial derivatives with respect to each s_j. Using the definition of d in Equation 3 and after working out the mathematics, we get:

$$\frac{\partial d(X_i; \Lambda, \Gamma)}{\partial s_j} = -I(W_0, s_j) + I(W_1, s_j), \tag{9}$$

where $I(W, s_j)$ denotes the number of times the transition S_j is taken in the best aligned path of X_i to the word sequence W.

For each utterance in the training data, the algorithm is counting the transitions for the correct string and the decoded hypothesis. Transitions for the correct string increase the corresponding transition weights, while those for the decoded string decrease the weights. The amount of increase or decrease is proportional to the step size ϵ, the value of the slope of the sigmoid function and the difference in the number of times the transition appears. The slope of the sigmoid function is close to 0 for very large positive d, so little adjustment is made for a sentence for which the total score of the correct string is much worse than the score of the competing string. This decreases the effect of outliers, for example of utterances whose transcripts are erroneous. Notice that the only dependence on the acoustic scores (more specifically, the difference in the total path scores) in the equations is in the slope $\frac{\partial l_i}{\partial d_i}$, which determines how much influence a particular training sample has in updating the parameters.

With gradient descent optimization, there is a choice of batch mode, which collects the statistics over all the training data before making an update to the model, or online mode, where the model is updated after processing each training sample and typically the sample order is randomized. Although online mode may result in faster convergence, batch mode has the advantage of allowing for parallelism in collecting statistics of the training data. In this paper, we use batch mode training.

3. FST IMPLEMENTATION

In this section, we describe a simple and elegant method for implementing discriminative training for large vocabulary decoding graphs using weighted finite-state transducers. We use an internal IBM FSM toolkit [11], with functionality similar to the publicly available AT&T toolkit [12].

It is easy to instrument a Viterbi decoder to count state transitions (just count on the backtrace). However, it is not obvious how the reference aligns to the decoding graph, especially because of LM backoff arcs. Therefore, we treat both cases in the same way: we produce leaf sequences for the reference and decoded word sequences, and then use a decoder-derived transducer that reads leaves and outputs state transitions to do the counting.

The algorithm consists of the following steps:

1. For each sentence in the training data, find the reference leaf sequence by aligning the reference transcript to the speech data. Encode the leaf sequence as an FSM and attach a dummy arc to store the acoustic score.
2. Decode training speech data using the decoding graph. For each sentence, find the decoded leaf sequence. Construct the FSM and attach a dummy arc to store the acoustic score.
3. Construct a transducer from the decoding graph to transform leaf sequences to state transition sequences, with associated transition weights. (The transition weights will be used later to calculate the path score of the leaf sequence.) Apply the transducer to both reference and decoded leaf sequences.
4. For each sentence, count transitions in the reference and decoded sequences (Equation 9), and weight the count for each transition by the derivative of the sigmoid class loss function (Equation 8) using the difference in total path scores (Equation 3). Accumulate over all utterances (Equation 7) to compute the gradient.
5. Update weights in the decoding graph based on the gradient.
6. Repeat from step 2 until the performance on a held-out set converges.

There are a variety of methods to make the updates based on the gradient [13]. We tried regular gradient descent (Equation 6) and Quickprop [14]. For Quickprop, Equations 7–9 are still the same; the only difference is in the update:

$$\Gamma_{t+1} = \Gamma_t - [(\Delta^2 l(\Gamma))^{-1} + \epsilon] \nabla l(X; \Lambda_t, \Gamma_t), \tag{10}$$

where the Hessian $\Delta^2 l(\Gamma)$ is assumed to be diagonal [13, 14].

4. EXPERIMENTAL SETUP

The experiments are done using a speaker-independent, English broadcast news recognition system. The language model used to build the decoding graph is trained on a 132M word corpus comprising the 1996 English Broadcast News Transcripts (LDC97T22), the 1997 English Broadcast News Transcripts (LDC98T28) and the 1996 CSR Hub4 Language Model data (LDC98T31). It is pruned to a bigram language model with 61K unigrams and 204K bigrams. This very small model size was necessary to turn around multiple experiments. A larger language model was used in lattice rescoring experiments. It was trained on the same 132M word corpus, but was a back-off 4-gram model containing 61K unigrams, 5.7M bigrams, 14M trigrams, and 8.7M 4-grams.

The acoustic model is trained on a 143-hour corpus comprising the 1996 English Broadcast News Speech collection (LDC97S44) and the 1997 English Broadcast News Speech collection (LDC98S71). The recognition features are 40-d vectors computed via an LDA+MLLT projection of 9 spliced frames of 19-d PLP features. The raw PLP features are normalized using utterance-based cepstral mean subtraction. In total, the acoustic model includes 6000 quinphone context dependent states and 120K Gaussian mixture components. The acoustic model is trained using maximum likelihood estimation.

For testing, we use the RT-03 English Broadcast News evaluation set, a collection of six news broadcasts from 2001. The total duration of the test audio is 2.93 hours, and the total number of words in the reference transcripts is 24790. The segmentation of the test set audio was derived from the reference transcripts.

5. RESULTS

The first practical issue that arises when doing discriminative training of decoding graphs is that the reference transcripts of the training data may not match the decoding graph. There may be out-of-vocabulary (OOV) words in the training data due to vocabulary pruning during LM training – we can do nothing about these words. However, some OOVs may be be correctable, e.g. in the reference “um” and “uh” may be separate words while in the decoding graph they are represented as one word with multiple pronunciations, or OOVs may be caused by typographical errors or spelling variations. The first step is to normalize the high frequency OOVs to better match the decoding graph vocabulary.

Because training over all the data takes a long time, we performed some preliminary experiments on a subset of the training data. Since there is a data weighting effect due to the derivative of the sigmoid in Equation 8, an intelligent method is to select the subset based on a range of the misclassification function (Equation 3). By picking $d < 2.0$, we ended up with about 5K training utterances.

Using this training set, we first explored the difference between gradient descent and Quickprop for updating the transition weights. Choosing the proper learning rate (ϵ) is always tricky for gradient descent and to a lesser degree for Quickprop. For gradient descent, the learning rate was chosen by considering the largest absolute value of the gradient, in order to constrain the maximum weight update to be on the order of 0.1.

Figure 1 shows the convergence rate of simple gradient descent and Quickprop. The objective function and word error rate (WER) for the training data, as well as the test data WER, generally decrease with each epoch of training. Quickprop seems to achieve convergence faster then gradient descent, although the test error curve is a bit more bumpy at some points.

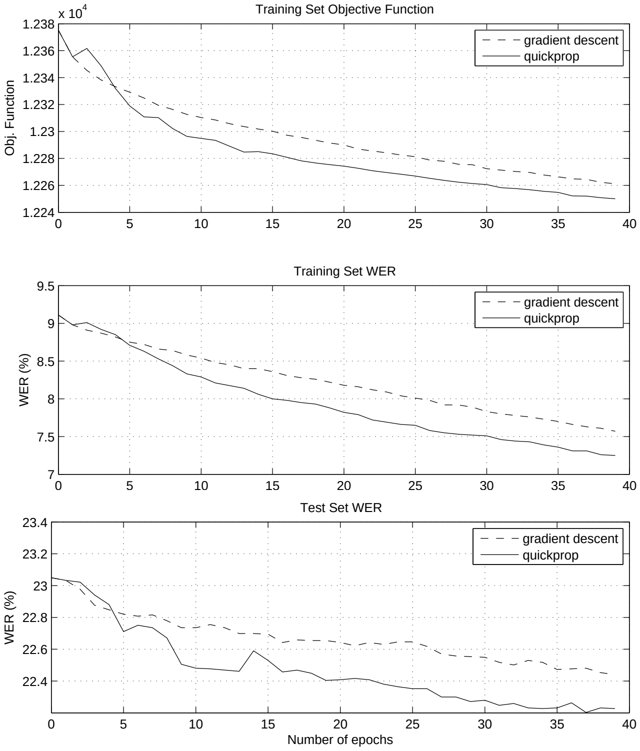

Fig. 1. Convergence rates for gradient descent and Quickprop

	Baseline WER	23.0
d Threshold	No.Training Sentences	WER
2.0	5538	22.3
10.0	24492	21.8
5000.0	56194	21.7

Table 1. Effect of amount of training data on WER

In Table 5, we show the effects of discriminative graph training on the WER, for various amount of training data. Even with 5538 sentences, we are able to get a significant improvement (of 0.7%) because of the way we have chosen them: we chose the sentences that are likely to have the largest contribution to the training. With more (25K sentences), there is further improvement. Based on the best results in the table, starting from a baseline WER of 23.0%, discriminative graph training was able to reduce the WER to 21.7%, representing an absolute improvement of 1.3%, or 5.7% relative.

In many large vocabulary transcription tasks it is common to generate lattices representing a large set of possible output hypotheses and then rescore the lattices with a larger, more complex language model before producing the final output. One objection to the discriminative training of decoding graphs is that any gains achieved through discriminative training will disappear following such language model rescoring. We tested this hypothesis by generating lattices using the baseline decoding graph and the best decoding graph from above (obtained with a d threshold of 5000.0) and rescoring the lattices with a much larger, 4-gram language model. The baseline system’s WER drops to 17.7% after rescoring, while the discriminatively trained system’s WER drops to 17.6%. The absolute

	beam	ML	MCE	Diff	%Diff
one-pass	14	23.0%	21.7%	1.3%	5.7%
	10	23.2%	21.9%	1.3%	5.6%
	9	23.8%	22.3%	1.5%	6.3%
	8	25.7%	23.7%	2.0%	7.8%
	7	33.5%	30.1%	3.4%	10.1%
LM rescored	14	17.7%	17.6%	0.1%	0.6%
	10	18.5%	18.0%	0.5%	2.7%
	9	19.4%	18.9%	0.5%	2.6%
	8	22.2%	21.0%	1.2%	5.4%
	7	31.7%	28.4%	3.3%	10.4%

Table 2. WER as function of decoding beam width

difference in the number of errors between the two systems is 38. Thus, the rescoring objection is upheld by these results. We note, however, that there are a number of applications in which low system latency is vital, such as the real-time closed captioning of news broadcasts, or in which system resources are constrained, such as speech recognition on handheld computers. In such low-latency and resource constrained applications, lattice rescoring may not be possible, so techniques such as discriminative training that improve the decoding graph itself are of practical interest.

Furthermore, discriminatively trained decoding graphs appear to have better pruning behavior. Table 5 shows how WER changes as the beam width for decoding is decreased to reduce computation and memory requirements. The advantage of using an MCE-trained decoding graph is even more apparent when very low beam widths are used. For example, with a beam of 8, the improvement in WER is increased to 2.0% absolute.

6. DISCUSSION AND CONCLUSIONS

We extended discriminative training of decoding graphs to large-vocabulary speech recognition with context-dependent acoustic models. We partially overcame challenges of large graphs and large amounts of training data, using a simple wFST approach and achieving decent improvements on a relatively small decoding graph.

The benefit of using a discriminatively trained decoding graph when a very large language model is available for rescoring is unclear at this time. (Such a language model is so large that it cannot practically be expanded into a decoding graph.) Our current experiments indicate that LM rescoring decreases the benefits of using our discriminatively trained decoding graph. However, future improvements to the training paradigm may show different results. For example, we do not know what will happen if we had trained a much larger decoding graph and then used LM rescoring. Also, our simplified framework for discriminative training does not expose the algorithm to enough realistic acoustic confusions in the training data. Future work could include holding out acoustic training data transcripts from language model training and using lattice or N-best hypotheses. Despite the LM rescoring issue, the discriminatively trained decoding graph is particularly useful in low-latency and low-resource applications such as real-time closed captioning or speech-to-speech translation, where LM rescoring is not possible.

Another line of future work is to more precisely characterize the benefits of integrated training of the decoding graph compared with discriminative LM training. It would also be interesting to determine whether LM context dependent pronunciation modeling, one advantage of integrated decoding graph training, is useful for certain applications.

7. ACKNOWLEDGMENTS

This work was partially supported by the Defense Advanced Research Projects Agency under contract No. HR0011-06-2-0001. We thank Hagen Soltau for help with the IBM ASR package Attila, and Mohamed Afify and Alain Biem for discussions.

8. REFERENCES

[1] Mehryar Mohri, Fernando Pereira, and Michael Riley, “Weighted finite-state transducers in speech recognition,” *Computer Speech and Language*, vol. 16, no. 1, pp. 69–88, 2002.

[2] Andreas Stolcke and Mitch Weintraub, “Discriminative language modeling,” in *Proc. 9th Hub-5 Conversational Speech Recognition Workshop*, 1998.

[3] Hong-Kwang Jeff Kuo, Eric Fosler-Lussier, Hui Jiang, and Chin-Hui Lee, “Discriminative training of language models for speech recognition,” in *Proc. ICASSP 2002*, Orlando, Florida, May 2002.

[4] Vaibhava Goel, “Conditional maximum likelihood estimation for improving annotation performance of N-gram models incorporating stochastic finite state grammars,” in *Proc. ICSLP 2004*, Jeju Island, Korea, Oct. 2004.

[5] Brian Roark, Murat Saraclar, and Michael Collins, “Discriminative n-gram language modeling,” *Computer Speech and Language*, 2006, to appear.

[6] Brian Roark, Murat Saraclar, Michael Collins, and Mark Johnson, “Discriminative language modeling with conditional random fields and the perceptron algorithm,” in *Proc. ACL*, Barcelona, Spain, July 2004.

[7] Shiuan-Sung Lin and François Yvon, “Discriminative training of finite state decoding graphs,” in *Proc. Interspeech 2005*, Lisbon, Portugal, Sept. 2005.

[8] Shigeru Katagiri, Chin-Hui Lee, and Biing-Hwang Juang, “New discriminative algorithm based on the generalized probabilistic descent method,” in *Proc. IEEE Workshop on Neural Network for Signal Processing*, Princeton, Sept. 1991, pp. 299–309.

[9] Biing-Hwang Juang, Wu Chou, and Chin-Hui Lee, “Minimum classification error rate methods for speech recognition,” *IEEE Transactions on Speech and Audio Processing*, vol. 5, no. 3, pp. 257–265, May 1997.

[10] Stanley F. Chen, “Compiling large-context phonetic decision trees into finite-state transducers,” in *Proc. Eurospeech 2003*, Geneva, Switzerland, Sept. 2003.

[11] Stanley F. Chen, “The IBM finite-state machine toolkit,” Tech. Rep., IBM T.J. Watson Research Center, Feb. 2000.

[12] Mehryar Mohri, Fernando C. Pereira, and Michael Riley, “AT&T Finite-State Machine Library,” http://www.research.att.com/~fsmtools/fsm/.

[13] Jonathan Le Roux and Eric McDermott, “Optimization methods for discriminative training,” in *Proc. Interspeech 2005*, Lisbon, Portugal, Sept. 2005.

[14] Scott E. Fahlman, “An empirical study of learning speed in back-propagation networks,” Tech. Rep. CMU-CS-88-162, Carnegie Mellon University, Sept. 1998.