Discriminatively Trained Dependency Language Modeling for Conversational Speech Recognition

Benjamin Lambert, Bhiksha Raj, Rita Singh

Language Technologies Institute, Carnegie Mellon University, USA

benlambert@cmu.edu, bhiksha@cs.cmu.edu, rsingh@cs.cmu.edu

Abstract

We present a discriminatively trained dependency parser-based language model. The model operates on utterances, rather than words, and so can utilize long-distance structural features of each sentence. We train the model discriminatively on *n*-best lists, using the perceptron algorithm to tune the model weights. Our features include standard *n*-gram style features, long-distance co-occurrence features, and syntactic structural features. We evaluate this model by re-ranking *n*-best lists of recognized speech from the Fisher dataset of informal telephone conversations. We compare various combinations of feature types, and methods of training the model.

Index Terms: Automatic speech recognition, natural language processing, language modeling, discriminative training

1. Introduction

Respectable performance at automatic recognition of conversational speech has eluded speech recognition researchers for many years. Recognition of dictated speech, on the other hand, can be quite good.

It seems ironic, that a sequence of very simple clauses (i.e. *I went to the doctor*, *I'm pretty active*, *I run around a lot*, and *I work out*) is more difficult to model and recognize than complex newswire sentences, like *Doctor Salk on the other hand uses everything but the envelope*.

So why is there this discrepancy between recognition of conversational speech and dictated speech? Part of the answer surely lies in acoustic factors, like the cadence and pronunciation, which are less predictable in informal and conversational speech. Another part is due to the insufficiency of the language model (LM) to capture the style of speech. Conventional *n*-gram LMs fall short in modeling conversational speech for a number of reasons. One, for instance, is that a sequence of meaningful words is very often interrupted by filler words and phrases. Thus, we see sentences like this:

> *You know I I went to the an- the doctor you know an- I'm I'm pretty active you know I run around a lot and I work out and try to you know.*

Here, repetitions (*I'm I'm*), false starts (*the an- the*), and fillers (*you know*) severely disrupt the flow of meaningful words.

In this example, even when using a *4*-gram LM, the most useful language model cues are beyond the model's three-word history. Thus, the correct recognition of *went to* can have no effect on the correct recognition of *doctor*. Nor, can the correct recognition of any of *doctor*, *active*, and *work out* have any effect on the recognition of the others.

Our model attempts to capture exactly these sorts of long-distance dependencies. The structured long-distance dependencies, as in *went to ... the doctor*, are identified these using an off-the-shelf dependency parser. The unstructured long-distance dependencies, like those among *doctor*, *active*, and *work*, are simply identified by their co-occurrence.

We utilize existing discriminative techniques to train the model. This paper builds on earlier work, extending it by adding new feature types, new feature combinations, and detailed analysis.

2. Related Research

In order to include long-distance features in our model, it is important that we model entire sentences or utterances rather than individual words (as in a conditional language model). The use of *whole sentence* language models for speech recognition was pioneered by Rosenfeld et al. [1] around 2001. Their model uses shallow parser features, and is trained in the maximum entropy/exponential model framework. Though a very interesting and novel technique, decreases in word error rate were small.

Discriminative training methods have been shown to perform better than maximum likelihood training in many areas, including acoustic modeling [2], *n*-gram language modeling [3], and machine translation [4]. There are various ways one may discriminatively train a model, such as conditional maximum likelihood estimation (CMLE). We use one based on the perceptron algorithm [5] because it is effective, but very simple to implement and use.

Finally, syntactic methods for language modeling and speech recognition have a rich history. In some of these models the parsing is integral to the language model including work by Chelba et al. [6][7] or done as a separate process [8]. We take the latter (and simpler) approach of deriving model features from a parse tree, thus keeping the model and syntactic analysis separate processes. Our overall technique is very similar to Collins et al [5] except that we use a dependency parser rather than a Penn Treebank style syntactic and derive different features from the parse tree.

3. The Model

Our model and training are very similar to that of Collins et al. [5]. The typical method used to score a speech recognition hypotheses, *s*, is the following formula, where *a* represents the acoustic signal.

$$\log(P(s \mid a)) = \log(P_a(a \mid s)) + \alpha \log(P_l(s)) \quad (1)$$

Here, $P_a(a|s)$ is the acoustic model, $P_l(s)$ is the language model, and α is the language weight.

We will assume $P_l(s)$ is some baseline *n*-gram language model. In this paper, as in Collins et al.[5], we identify features of the whole sentence, and adjust the overall score of the sentence, up or down, depending on the feature's weight. The resulting equation is an almost trivial extension of the original equation:

$$\log(P(s \mid a)) = \log(P_a(a \mid s)) + \alpha \log(P_l(s)) + \sum_i^n f_i(s) \cdot w_i \quad (2)$$

The only change we have made is to add the last term, a summation of feature weights to the original score. The final term simply adds the feature weight of each feature in the sentence, s, to the overall score. That is, $f_i(s)$ is an indicator function, returning *one* if the feature is present and *zero* if it is not. The f_i features are whole sentence features that capture syntactic parse information or other long distance dependencies.

3.1. Parameter estimation

There are numerous ways to estimate effective values for the features weights, w_i, in the equation above. The papers of Rosenfeld [1], Roark [3], and Collins [5] share the general form of the function above, but estimate the features weights very differently.

Rosenfeld [1] estimates the weights within a maximum likelihood, maximum entropy framework—and thus set the parameters to maximize the likelihood of a corpus. Roark [3] and Collins [5] on other hand estimate the weights discriminatively (i.e. attempting to minimize word error rate). In Roark [3], they perform a global optimization over the feature weights with the goal of increasing the likelihood of low word error rate (WER) candidates (taken from an *n*-best list). In Collins [5], the feature weights are tuned similarly, but with the perceptron algorithm rather than a global optimization.

We use the perceptron algorithm to train the feature weights in our model. We have come across two different methods for using perceptron to tune the parameter weights, those of Collins [5], and those of Hopkins [9]. The main difference between the two approaches is how pairs of training examples are chosen.

The basic idea is the following. Say we're given two sentences from an n-best list, s_1 and s_2. s_1 is better than s_2 in that it has lower WER (i.e. $wer(s_1) < wer(s_2)$). However, our model prefers s_2 to s_1 (i.e. $P(s_1|a) < P(s_2|a)$). In this case, we'd like to give a small boost to the weights of all the features present in s_1 and a small penalty to the weights of all the features present in s_2. These small adjustments will bring up $P(s_1|a)$ and $P(s_2|a)$ down. This is exactly what we want: we want the language model probabilities to correlate inversely with WER as much as possible.

Inputs:

$nbest_k$	*k n*-best lists
t	iterations
c	pair count
wer(s)	word error rate function
f(s)	returns the features of s
w_i	Initial parameters
r	Learning rate

Algorithm:

loop t **times**:
  k = 0
  loop until k = c:
    let $nbest_k$ be selected randomly
    let s_i be selected randomly from $nbest_k$
    let s_j be selected randomly from $nbest_k$
    if $s_i \neq s_j$ **and** $f(s_i) \neq f(s_j)$ **and**
      $sign(wer(s_i) - wer(s_j)) \neq sign(P(s_i|a) - P(s_j|a))$:
      k++
      for l **in** $f(s_i)$:
        $w_l = w_l + sign(wer(s_i) - wer(s_j)) \times r$
      for l **in** $f(s_j)$:
        $w_l = w_l - sign(wer(s_i) - wer(s_j)) \times r$

Figure 1: *Perceptron-based pairwise ranking optimization*

Collins [5] chooses the pairs, by first selecting the sentence on an *n*-best list that has the lowest WER (referred to in their papers as s_i), and then pairing that with each of the other members of the *n*-best list. Hopkins [9] chooses the pairs by randomly sampling pairs from each *n*-best list (accepting pairs with a probability proportional to the difference in WER). They call this method "pairwise ranking optimization."

We use a method similar to Hopkins [9]; our method is shown in Figure 1. We randomly choose pairs of ASR hypotheses from the *n*-best lists (that have different WER and different features), and use those as the training pairs. We do not sample based on differences in WER. We also decrease the learning rate *r*, by *r/t* after each iteration.

4. N-gram, co-occurrence and dependency features

In this section we describe the features we use in our model: standard *n*-gram features, co-occurrence features, and dependency features. All features are treated as binary, whether they are present in a sentence or not, regardless of how many times they appear.

4.1. *N*-gram features and co-occurrence features

These are pretty self-explanatory. We generate features for each sequence of up to *n* words (*n*=3). We also generate features for each pair of words that co-occurs in a sentence; we call these *x*-grams. *x*-grams are similar to the "trigger" word pairs of Rosenfeld [10].

Examples of these features include:

f_1 = xgram(*I*, *I*)
f_2 = xgram(*I*, *went*)
f_3 = xgram(*I*, *to*) …

f_n = xgram(doctor, active)

4.2. Dependency features

Dependency features are designed to capture structured long-distance dependencies, such as between *I went to* and *the doctor* in the example sentence in the introduction. To identify these dependencies, we use an off-the-shelf dependency parser. The parser assigns typed and directed links, between pairs of words in each sentence. The parser we use, labels the links with Stanford's taxonomy of 53 dependency types [11].

Each dependency link is labeled with one of the 53 grammatical relations. These include relations like that between a verb and its noun subject (*nsubj*), between a verb and its direct object (*dobj*), and so forth. Figure 2 shows a portion of the parse for the example sentence of section 1. This includes a *prep* link between the verb *went* and its preposition *to*, as well as a *pobj* link from the preposition *to* to the object of the preposition *doctor*.

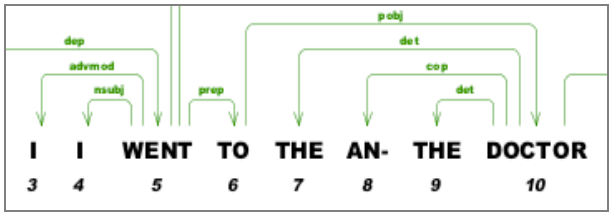

Figure 2: *A portion of a typed dependency parse.*

Each link in the dependency parse tree becomes one feature. Thus, each feature is a triplet of two words and a link type:

f_1 = prep(*went*, *to*)

f_2 = pobj(*to*, *doctor*)

The results in section five also show performance of *untyped* dependencies. By this we mean that we remove the edge labels from the dependencies. Thus, the feature *pobj(to, doctor)* simply becomes *dep(to, doctor)* and *nobj(I, went)* becomes *dep(I, went)*. Here are some example matches for the feature *det(the, school)*:

Taught at ***the same school*** *for seven years…*

I'm in ***the business school*** *actually*

…in **the liberal arts school** you need like three years

5. Feature selection

Many instances of the features types described in the previous section will be relatively useless, or will be redundant with existing *n*-grams. There is no sense inflating the model with these features, so we select only those that are estimated to be most useful. If nothing else, fewer features will mean less time parsing and training. Collins[5] suggests that feature selection is unnecessary when using the perceptron learning algorithm, so we have tried this with and without feature selection, and with varying levels of feature selection.

Identifying those features with the most discriminative power is fairly straightforward. We simply count how often they occur in *correctly* recognized speech and how often they occur in *incorrectly* recognized speech. Features that occur about the same number of times in each have low discriminative power. Those that occur disproportionately more in one or the other have high discriminative power.

To get examples of correctly and incorrectly recognized speech, we took 100,000 utterances from our speech corpus (the Fisher corpus [12]). The corresponding 100,000 reference transcripts became our set of correctly recognized speech (what we call the *positive* set). We then ran the same 100,000 utterances through our speech recognizer. The output of the ASR system—*when it was wrong*—became our set of incorrectly recognized speech (the *negative* set). The size of the set of incorrectly recognized utterances (i.e. WER greater than zero) is 80,271. Table 1 shows some examples of the sentences we used.

Again following Rosenfeld, et al [1][13], we use the following method to select the most useful features, from our positive and negative sets. This is an example of a "two-sample statistical hypothesis test." Equation 3 yields a standard score or *z*-value. If the *z*-value is two (i.e. about two standard deviations) then there's a 96% chance that the feature has *different* distributions in the two sets. The larger the *z*-value, the more discriminative power the feature has. We call this z-score the *utility*.

Positive set (i.e. reference transcripts)	**Negative set (i.e. incorrectly recognized utterances)**
Good how are you doing	Good how are you done
Do you have any idea what's going on	Give any idea what's going on
Because I don't like to cook very much	Can I don't like to cook very much
I tend to eat out	*n/a*

Table 1 *Correctly and incorrectly recognized examples*

In this equation, x is the number of positive sentences that contain the feature, y is the number of negative sentences that contain the feature, n is the number of positive sentences (100,000), and m is the number of negative sentences (80,271). See tables 3 and 4 for examples.

$$\left| \frac{\frac{x}{n} - \frac{y}{m}}{\left(\frac{x+y}{m+n}\right) \cdot \left(1 - \frac{x+y}{m+n}\right) \cdot \left(\frac{n+m}{nm}\right)} \right| \quad (3)$$

6. Experimental results

6.1. Experimental setup

All experiments were on a subset of the Fisher1 dataset [12]. We decoded 110,000 utterances of this dataset with the Sphinx3 ASR system [14]. The LM is a trigram LM trained on Fisher2 [12] and Switchboard transcripts [15]. The acoustic model was also trained on Fisher2. 100,000 of these were used for feature generation and selection. The other 10,000 were used to generate *1,000*-best lists for parameter tuning and evaluation.

For the contrast score weighting of features, we split the 10,000 *n*-best lists into two sets: one for training the model weights with perceptron, and the other for doing the final evaluation.

We parsed the data with the Stanford POS tagger (caseless mode, trained on Switchboard)[16], and the MaltParser [18] (using pre-trained English model 'engmalt.linear-1.7'. The POS tagger is trained on speech, the parser is not.

To train the model weights, we set the parameters described in section 3 as follows. We randomly sampled 10,000,000 hypothesis pairs (c), at each of 20 iterations (t). The learning rate (r) was initially set to 0.00001, and decreased at each iteration. The initial parameters were all set to zero.

Since the algorithm is randomized, we repeated the experiments several times. The WER were always within about 0.01% of one another, and usually within 0.001%. This indicates that sampling 10,000,000 pairs is a sufficiently large number. The numbers were even closer when we sampled 100,000,000 pairs, but the training took much longer per iteration. Each iteration with 10,000,000 pairs took a couple minutes. Since we were looking at 5,000 1,000-best lists, there are at most 5,000,000,000 candidate pairs. Thus, training was performed on about 2% of those pairs at each iteration.

6.2. Features

There were many instances of the feature types described in section 3 in the training corpus. The following table gives a rough idea of how many features types were in the corpus. The first column is the number of distinct feature types of each class of the features that had a utility score of more than 1.96. The second column is number of feature types that occurred in either the positive set of reference transcripts or the negative set of 80,271 incorrectly recognized sentences at least five times. The final column is the total number of feature types in the whole set.

	util > 1.96	freq > 5	all
n-gram	3,112	74,604	787,828
1-gram	403	7,094	20,785
2-gram	1,501	33,180	210,832
3-gram	1,208	34,330	556,211
x-gram	7,022	152,179	962,567
dependency	2,011	41,689	387,632
dep untyped	1,957	40,858	307,258
Total	**14,102**	**309,330**	**2,445,285**

Table 2 *Feature type counts in the training corpus*

6.3. Results

Results are shown as absolute changes in WER. The baseline WER is 26.171%. Each column represents the three methods of feature selection (described in section 6.2).

Overall, we are seeing that any set of these features is able to reduce the WER, albeit a small amount. This indicates that the training method, and features used, are to some degree advantageous to use. However, the decreases are fairly small, one third of one percent at most.

	Features\ feat sel.	util >1.96	freq>5	all
1	1-gram (1g)	-0.12	-0.19	-0.19
2	2-gram (2g)	-0.06	-0.13	-0.13
3	3-gram (3g)	-0.01	-0.05	-0.05
4	x-gram (xg)	-0.16	-0.22	-0.22
5	dependency (dep)	-0.03	-0.07	-0.07
6	dep-untyped (depu)	-0.02	-0.10	-0.10
7	n-gram (ng)	-0.22	-0.20	-0.21
8	ng + xg	**-0.24**	**-0.29**	-0.29
9	ng + dep	-0.20	-0.22	-0.22
10	ng + depu	-0.17	-0.24	-0.24
11	dep + depu	-0.01	-0.07	-0.06
12	xg + dep	-0.17	-0.21	-0.22
13	xg + dep + depu	-0.16	-0.24	-0.26
14	ng + dep + depu	-0.19	-0.25	-0.24
15	ng + xg + dep	-0.24	-0.30	-0.29
16	ng + xg + depu	-0.24	-0.30	**-0.33**
17	ng + xg + dep + depu	-0.24	-0.25	-0.25
18	1g + xg + dep	**-0.27**	-0.26	-0.30
19	1g + 3g + xg + depu		**-0.34**	**-0.34**

Table 3 *Absolute WER reduction using our model with varied features and feature selection techniques*

When we look at the results of each feature type individually (rows 1-6), those with the largest contributions appear to be *x*-grams and *1*-grams. When we start looking at combinations of feature types (row 7-18), then *n*-grams all together begin to dominate as well. Line 17 represents what happens when we include *all* feature types.

When considered individually, the features derived from the dependency parse trees have a much smaller contribution. And, when we include all the *n*-gram features together in combination with the other feature types (lines 8-17 of the table), the feature combinations that have the largest WER reductions are those that include both *n*-grams and *x*-grams, with or without dependency features.

However, if we consider other combinations of features that do not include each of *1*-grams, *2*-grams, and *3*-grams, the dependency features do seem to help achieve the largest reductions in WER. Rows 18 and 19 of the table show the largest WER reduction of *all* feature combinations for each column.

We suspect that these results indicate that the dependency parse features are largely, but not entirely, redundant with the *2*-gram and *x*-gram features.

When comparing the different methods of feature selection, we see that typically the more features we use, the better the results

6.4. Analysis

To better understand these results, we took a closer look at some of the features in the model that had the highest utility scores and that were assigned the largest magnitude weights by the perceptron algorithm. We use the model that had the most feature selection (utility > 1.96), and with *all* feature types (i.e. column 1, row 17).

First, as a sanity check on this technique, the magnitudes of the feature weights appear to be reasonable. Total scores (acoustic + lm, log base 10) of hypotheses in these *n*-best lists tend to be in the range of -1,200 to -200.

Tables 4 and 5 show those features with the largest magnitude learned feature weights (w_i), and largest utility scores, respectively. The most obvious observation is that the n-gram and x-gram features dominate. This makes sense considering the other results.

Feature	x	y	utility	weight
1-gram(probably)	1,224	888	2.309	4.105
2-gram(don't, know)	2,442	1,802	2.743	3.791
x-gram(and, need)	50	68	2.864	3.780
x-gram(hi, are)	335	163	5.304	3.766
1-gram(another)	462	307	2.575	3.621
...				
x-gram(know, any)	89	112	3.194	-3.380
1-gram(floor)	19	33	2.747	-3.718
1-gram(every)	538	523	3.132	-4.011
3-gram(it, would, be)	195	121	2.232	-5.062
1-gram(lotta)	68	78	2.163	-5.417

Table 4 *Features with the largest magnitude learned weights*

Feature	x	y	utility	weight
1-gram(the)	19,339	18,541	19.470	-0.172
1-gram(they)	7,499	7,926	17.916	-0.379
1-gram(it's)	8,454	8,778	17.808	-0.319
1-gram(it)	12,009	11,656	15.695	0.006
1-gram(all)	3,137	3,634	15.428	-1.729

Table 5 Features with the largest 'utility' scores

7. Conclusions

We have shown, as previously has been shown, that perceptron based approaches to whole sentence language model training with syntactic features, can indeed yield decreases in WER. Although dependency parser based features do contribute a small amount to the WER reduction, they appear to be largely redundant with *x*-gram and *2*-gram features. We suspect that eliminating some of this redundancy may yield additional decreases in WER. Additionally, if syntactic and dependency features are to help, we need to take a closer look at them rather than just blindly generating all such features and using the entire (or feature-selected) set.

8. References

[1] R. Rosenfeld, S. Chen, and X. Zhu, "Whole-sentence exponential language models: a vehicle for linguistic-statistical integration," *Computer Speech and Language*, 2001.

[2] Woodland. C. and Povey, D. "Large Scale Discriminative Training For Speech Recognition," *Computer Speech and Language*, 2000.

[3] Brian Roark, Murat Saraclar, and Michael Collins, "Discriminative n-gram language modeling." *Computer Speech and Language*, 2007.

[4] F. Och, "Minimum error rate training in statistical machine translation" *ACL,* 2003.

[5] Michael Collins, Brian Roark, and Murat Saraclar, Discriminative syntactic language modeling for speech recognition. In Proceedings of *ACL*, 2005.

[6] C. Chelba, et al, "Structure and performance of a dependency language model," In Proceedings of *Eurospeech*, 1997.

[7] C. Chelba, and F. Jelinek, "Structured language modeling," *Computer Speech and Language*, 2000.

[8] Wen Wang and Andreas Stolcke Mary P. Harper, The use of a linguistically motivated language model in conversational speech recognition." In Proceedings of *ICASSP*, 2004.

[9] M. Hopkins and J. May, "Tuning as Ranking," In Proceedings of *EMNLP*, 2011.

[10] R. Rosenfeld, "A maximum entropy approach to adaptive statistical language modeling," *Computer speech and language*, 1996.

[11] M-C de Marneffe and C.D. Manning, "Stanford typed dependencies manual," *Stanford University*, Technical report, 2008.

[12] C. Cieri, et al. "Fisher English Training, Speech," *Linguistic Data Consortium*, Philadelphia, 2005.

[13] R.J. Larsen, and M.L. Marx, *An Introduction to Mathematical Statistics and Its Applications*, Prentice Hall, 3rd Edition, 2000.

[14] P. Placeway, S. Chen, M. Eskenazi, U. Jain, V. Parikh, B. Raj, M. Ravishankar, R. Rosenfeld, K. Seymore, M. Siegler, R. Stern, E. Thayer, Hub-4 Sphinx-3 system, 1997.

[15] J. J. Godfrey and E. Holliman, Switchboard-1 Release 2, *Linguistic Data Consortium*, Philadelphia, 1997.

[16] K. Toutanova, D. Klein, C. Manning, and Y. Singer. "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network," In Proceedings of *HLT-NAACL*, 2003.

[17] M. P. Marcus, B. Santorini, M. A. Marcinkiewicz and A. Taylor Treebank-3 *Linguistic Data Consortium*, Philadelphia, 1999.

[18] J. Nivre, and J. Hall, "Maltparser: A language-independent system for data-driven dependency parsing," In Proceedings of *Workshop on Treebanks and Linguistic Theories*, 2005.