Refining Generative Language Models using Discriminative Learning

Ben Sandbank
Blavatnik School of Computer Science
Tel-Aviv University
Tel-Aviv 69978, Israel
sandban@post.tau.ac.il

Abstract

We propose a new approach to language modeling which utilizes discriminative learning methods. Our approach is an iterative one: starting with an initial language model, in each iteration we generate 'false' sentences from the current model, and then train a classifier to discriminate between them and sentences from the training corpus. To the extent that this succeeds, the classifier is incorporated into the model by lowering the probability of sentences classified as false, and the process is repeated. We demonstrate the effectiveness of this approach on a natural language corpus and show it provides an 11.4% improvement in perplexity over a modified kneser-ney smoothed trigram.

1 Introduction

Language modeling is a fundamental task in natural language processing and is routinely employed in a wide range of applications, such as speech recognition, machine translation, etc'. Traditionally, a language model is a probabilistic model which assigns a probability value to a sentence or a sequence of words. We refer to these as *generative language models*. A very popular example of a generative language model is the n-gram, which conditions the probability of the next word on the previous *(n-1)*-words.

Although simple and widely-applicable, it has proven difficult to allow n-grams, and other forms of generative language models as well, to take advantage of non-local and overlapping features.[1] These sorts of features, however, pose no problem for standard discriminative learning methods, e.g. large-margin classifiers. For this reason, a new class of language model, the *discriminative language model*, has been proposed recently to augment generative language models (Gao et al., 2005; Roark et al., 2007). Instead of providing probability values, discriminative language models directly classify sentences as either correct or incorrect, where the definition of correctness depends on the application (e.g. grammatical / ungrammatical, correct translation / incorrect translation, etc').

Discriminative learning methods require negative samples. Given that the corpora used for training language models contain only real sentences, i.e. positive samples, obtaining these can be problematic. In most work on discriminative language modeling this was not a major issue as the work was concerned with specific applications, and these provided a natural definition of negative samples. For instance, (Roark et al., 2007) proposed a discriminative language model for a speech recognition task. Given an acoustic sequence, a baseline recognizer was used to generate a set of possible transcriptions. The correct transcription was taken as a positive sample, while the rest were taken as negative samples. More recently, however, Okanohara and Tsujii (2007) showed that a

[1] Conditional maximum entropy models (Rosenfeld, 1996) provide somewhat of a counter-example, but there, too, many kinds of global and non-local features are difficult to use (Rosenfeld, 1997).

discriminative language model can be trained independently of a specific application by using a generative language model to obtain the negative samples. Using a non-linear large-margin learning algorithm, they successfully trained a classifier to discriminate real sentences from sentences generated by a trigram.

In this paper we extend this line of work to study the extent to which discriminative learning methods can lead to better *generative language models per-se*. The basic intuition is the following: if a classifier can be used to discriminate real sentences from 'false' sentences generated by a language model, then it can also be used to improve that language model by taking probability mass away from sentences classified as false and transferring it to sentences classified as real. If the resulting language model can be efficiently sampled from, then this process can be repeated, until generated sentences can no longer be distinguished from real ones.

The remainder of the paper is structured as follows: In the next section we formally develop this intuition, providing a quick overview of the whole-sentence maximum-entropy model and of self-supervised boosting, two previous works on which we rely. We also present the method we use for sampling from the current model, which for the present work is far more efficient than the classical Gibbs sampling. Our experimental results are presented in section 3, and section 4 concludes with a discussion and a future outlook.

2 Learning Framework

2.1 Whole-sentence maximum-entropy model

The vast majority of statistical language models estimate the probability of a given sentence as a product of conditional probabilities via the chain rule:

$$\mathrm{P}(s) \overset{def}{=} \mathrm{P}(w_1 ... w_n) \overset{def}{=} \prod_{i=1}^{n} \mathrm{P}(w_i \mid h_i) \qquad (1)$$

where $h_i \overset{def}{=} w_1 ... w_{i-1}$ is called the *history* of the word w_i. Most work on language modeling therefore is directed at the estimation of $P(w_i \mid h_i)$. While this is theoretically correct, it makes it difficult to incorporate global information about the sentence into the model, e.g. length, grammaticality, etc'. For this reason, the whole-sentence maximum-entropy model was proposed in (Rosenfeld, 1997). In the WSME model the probability of a sentence is defined directly as:

$$P(s) = \frac{1}{Z} P_0(s) \cdot \exp(\sum_i \lambda_i f_i(s)) \qquad (2)$$

Where $P_0(s)$ is some baseline model, $Z \overset{def}{=} \sum_s P_0(s) \cdot \exp(\sum_i \lambda_i f_i(s))$ is a normalization constant and the $\{f_i\}$'s are *features* encoding some information about the sentence. Most generally, a feature is a function from the set of word sequences to *R*, the set of real numbers. However, in most applications, as in our work, the features are taken to be binary. Lastly, the $\{\lambda_i\}$'s are real coefficients encoding the relative importance of their corresponding features. In the WSME framework the set of features $\{f_i\}$ is given ahead of training by the modeler, and learning consists of estimating the coefficients $\{\lambda_i\}$. This is done by stipulating the constraints

$$E_p(f_i) = E_{\tilde{p}}(f_i) \overset{def}{=} \frac{1}{N} \sum_{j=1}^{N} f_i(s_j) \qquad (3)$$

where $\tilde{p}$ is the empirical distribution defined by the training set $\{s_1, ... s_N\}$.[2] If these constraints are consistent then there is a unique solution in $\{\lambda_i\}$ that satisfies them. This solution is guaranteed to be the one closest to P_0 in the Kullback-Leblier sense among all solutions satisfying (3). It is also guaranteed to be the maximum likelihood solution for the exponential family. For more details, see (Chen and Rosenfeld, 1999a).

2.2 Self-supervised boosting

A different approach to learning the same sort of model as in (2) was proposed in (Welling et al., 2003). Here, instead of having all the features pre-given, they are *learned* one at a time along with their corresponding coefficients. Welling et al. show that adding a new feature to (2) can be

[2] Sometimes a smoothed version of (3) is used instead (e.g. Chen and Rosenfeld, 1999b).

interpreted as gradient ascent on the log-likelihood function, and show that the optimal feature is the one that best discriminates real data from data sampled from the current model. To see this, let

$$E(s) = \ln(P_0(s)) + \sum_i \lambda_i f_i(s) \quad (4)$$

denote the *energy* associated with sentence s.[3] Equation (2) can now be rewritten as -

$$P(s) = \frac{1}{Z}\exp(E(s)) \quad (5)$$

where Z is a normalization constant as before. The derivative of the log-likelihood with respect to an arbitrary parameter β is then –

$$\frac{\partial L}{\partial \beta} = -\frac{1}{N}\sum_{i=1}^{N}\frac{\partial E(s_i)}{\partial \beta} + \sum_{s\in S} P(s)\frac{\partial E(s)}{\partial \beta} \quad (6)$$

where $\{s_1, \ldots s_N\}$ is once again the training corpus, and the second sum runs over the set of all word sequences.

Now, suppose we change the energy function by adding an infinitesimal multiple of a new feature f^*. The log-likelihood after adding the feature can be approximated by –

$$L(E(s) + \varepsilon f^*(s)) \approx L(E(s)) + \varepsilon \frac{\partial L}{\partial \varepsilon} \quad (7)$$

where the derivative of L is taken at $\varepsilon = 0$. Because the optimal feature is the one that maximizes the increase in log-likelihood, we are searching for a feature that maximizes this derivative. Using equation (6) and noting that $\frac{\partial E}{\partial \varepsilon} = f^*$ we have –

$$\frac{\partial L}{\partial \varepsilon} = -\frac{1}{N}\sum_{i=1}^{N} f^*(s_i) + \sum_{s\in S} P(s) f^*(s) \quad (8)$$

This expression cannot be computed in practice, because the set of all word sequences S is infinite. The second term however can approximated using samples $\{u_i\}$ from the current model –

$$\frac{\partial L}{\partial \varepsilon} \approx -\frac{1}{N}\sum_{i=1}^{N} f^*(s_i) + \frac{1}{N}\sum_{i=1}^{N} f^*(u_i) \quad (9)$$

[3] In (Welling et al., 2003) the term for P_0 does not appear, which is equivalent to taking the uniform distribution as the baseline model.

In other words, given a set of N samples $\{u_i\}$ from the model, the optimal feature to add is one that gives high scores to sampled sentences and low ones to real sentences. By labeling real sentences with 0 and sampled sentences with 1, the task of learning the feature translates into the task of training a classifier to discriminate between these two classes of sentences.

In the remainder of the paper we will use *feature* and *classifier* interchangeably.

2.3 Rejection sampling

Self-supervised boosting was presented as a general method for density estimation, and was not tested in the context of language modeling. Rather, Welling at al. demonstrated its effectiveness in modeling hand-written digits and on synthetic data. In both cases essentially linear classifiers were used as features. As these are computationally very efficient, the authors could use a variant of Gibbs sampling for generating negative samples. Unfortunately, as shown in (Okanohara and Tsujii, 2007), with the represetation of sentences that we use, linear classifiers cannot discriminate real sentences from sentences sampled from a trigram, which is the model we use as a baseline, so here we resort to a non-linear large-margin classifier (see section 3 for details). While large-margin classifiers consistently out-perform other learning algorithms in many NLP tasks, their non-linear variations are also notoriously slow when it comes to computing their decision function – taking time that can be linear in the size of their training data. This means that MCMC techniques like Gibbs sampling quickly become intractable, even for small corpora, as they require performing very large numbers of classifications. For this reason we use a different sampling scheme which we refer to as *rejection sampling*. This allows us to sample from the true model distribution while requiring a drastically smaller number of classifications, as long as the current model isn't too far removed from the baseline.

We will start by describing the sampling process, and then show that the probability distribution it samples from has the form of equation (2). To sample a sentence from the current model, we generate one from the baseline model, and then pass it through each of the classifiers in the model. If a given classifier classifies the sentence as a model sentence, then it is rejected with a certain probability associated with this classifier. Only if a sentence is accepted by all classifiers is it taken as a sample sentence. Otherwise, the sampling process is restarted.

Let us derive an expression for the probability of a sentence *s* generated in this manner. To simplify notation, assume that at this point we added but a single feature *f* to the baseline model P_0, and let p_{rej} stand for the rejection probability associated with it. Furthermore, let p_- stand for the accuracy of *f* in classifying sentences sampled from P_0 (negative samples). Formally,

$$p_- = E_{P_0}(f) \tag{10}$$

First let's assume that $f(s)=1$. The probability for generating s is a sum of the probabilities of two disjoint outcomes – the probability of generating s as the first sentence and having it survive the rejection, plus the probability of generating in the first iteration some sentence s' such that $f(s')=1$, rejecting that, and then generating s in one of the subsequent iterations. Formally, this means that –

$$P_1(s) = (1-p_{rej})P_0(s) + p_- p_{rej} P_1(s) \tag{11}$$

Rearranging, we have –

$$P_1(s) = \frac{1-p_{rej}}{1-p_- p_{rej}} P_0(s) \tag{12}$$

Similarly, the probability for a sentence s for which $f(s)=0$ is the probability of generating s as the first sentence, plus the probability of generating some other sentence s' for which $f(s')=1$, rejecting it, and then generating s in a future iteration. Formally,

$$P_1(s) = P_0(s) + p_- p_{rej} P_1(s) \tag{13}$$

and hence –

$$P_1(s) = \frac{1}{1-p_- p_{rej}} P_0(s) \tag{14}$$

Letting $Z = 1 - p_- p_{rej}$, and letting $\lambda = \ln(1-p_{rej})$, we have, for all s –

$$P_1(s) = \frac{1}{Z} P_0(s) \cdot \exp(\lambda \cdot f(s)) \tag{15}$$

This process can be trivially generalized for N features. Let –

$$p_-^i = E_{P_{i-1}}(f_i) \tag{16}$$

stand for f_i's accuracy in classifying sentences generated from P_{i-1}, and let p_{rej}^i be the rejection probability associated with the i'th feature. Sampling from the model then proceeds by sampling a sentence *s* from P_0. For each $1 \le i \le N$, in order, if $f_i(s)=1$, then we attempt to reject *s* with probability p_{rej}^i. If *s* survives all the rejection attempts, it is returned as the next sample. Using similar arguments as before it's possible to show that if we take $\lambda_i = \ln(1-p_{rej}^i)$ and –

$$Z = \prod_{i=1}^{N} (1 - p_-^i p_{rej}^i) \tag{17}$$

then the probability of a sentence *s* sampled by this process is given by equation (2). Conversely, this shows that rejection sampling can be used for obtaining negative samples from the model given in (2) by taking $p_{rej}^i = 1 - \exp(\lambda_i)$, as long as $0 < \exp(\lambda_i) \le 1$. In section 3 we show that in our experimental setup, rejection sampling brings about enormous savings in the number of classifications necessary during training, as compared with Gibbs sampling.

2.4 Adding a new feature

Given the current model P_i and a new feature f_{i+1}, we wish to find the optimal λ_{i+1}, or equivalently its optimal rejection probability p_{rej}^{i+1}. In the WSME framework, the weights of the features are set in such a way that the expected value of the features on sentences sampled from the model equals their expected value on real sentences. A possible way to set the weight of a new feature is therefore to set p_{rej}^{i+1} such that the constraint:

$$E_{P_{i+1}}(f_{i+1}) = E_{\tilde{p}}(f_{i+1}) \tag{18}$$

is satistfied, where $\tilde{p}$ is once again the empirical distribution defined by the training set. Intuitively, this means that the new feature could no longer be used to discriminate between sentences sampled from P_{i+1} and real sentences. However, setting

p_{rej}^{i+1} in this manner may violate the constraints (18) associated with the features already existing in the model, thus hampering the model's performance. Therefore, we set the new feature's rejection probability by directly searching for the one that minimizes an estimate of P_{i+1}'s perplexity on a set of held out real sentences. To do this, we first sample a new set of sentences from P_i, independently of the set that was used for training f_{i+1}, and use it to estimate p_{-}^{i+1}. For any arbitrarily determined p_{rej}^{i+1}, this enables us to calculate an estimate for the normalization constant Z (equation 17), and therefore an estimate for P_{i+1}. We do this for a range of possible values for p_{rej}^{i+1} and pick the one that leads to the largest reduction in perplexity on the held out data.[4]

3 Experimental work

We tested our approach on the ATIS natural language corpus (Hemphill et al., 1990). We split the corpus into a training set of 11,000 sentences, a held-out set containing 1,045 sentences, and a test set containing 1,000 sentences which were reserved for measuring perplexity. The corpus was pre-processed so that every word appearing less than three times was replaced by a special UNK symbol. The resulting lexicon contained 603 word types.

Our learning framework leaves open a number of design choices:

1. **Baseline language model**: For P_0 we used a trigram with modified kneser-ney smoothing [Chen and Goodman, 1998], which is still considered one of the best smoothing methods for n-gram language models.

2. **Sentence representation**: Each sentence was represented as the collection of unigrams, bigrams and trigrams it contained. A coordinate was reserved for each such n-gram which appeared in the data, whether real or sampled. The value of the n'th coordinate in the vector representation of sentence s was set to the number of times the corresponding n-gram appeared in s.

3. **Type of classifiers**: For our features we used large-margin classifiers trained using the online algorithm described in (Crammer et al., 2006). The code for the classifier was generously provided by Daisuke Okanohara. This code was extensively optimized to take advantage of the very sparse sentence representation described above. As shown in (Okanohara and Tsujii, 2007), using this representation, a linear classifier cannot distinguish sentences sampled from a trigram and real sentences. Therefore, we used a 3rd order polynomial kernel, which was found to give good results. No special effort was otherwise made in order to optimize the parameters of the classifiers.

4. **Stopping criterion**: The process of adding features to the model was continued until the classification performance of the next feature was within 2% of chance performance.

We refer to the language model obtained by this approach as the *boosted* model to distinguish it from the baseline model. To estimate the boosted model's perplexity we needed to estimate the normalization constant Z in equation (2). Since this constant is equal to $E_{P_0}(\exp(\sum_i \lambda_i f_i))$ it can be estimated from a large-enough sample from P_0. We used 10,000,000 sentences generated from the baseline trigram and took the upper bound of the 95% confidence interval of the sample mean as an upper bound for Z. This means the perplexity estimates we report are upper bounds for the real model perplexity with 95% confidence.[5]

The algorithm converged after 21 features were added to the model. Figure 1 presents the model's perplexity on the test set estimated after each iteration. The perplexity of the final model is 9.02. In comparison, the perplexity of the modified kneser-ney smoothed trigram on this corpus is 10.18. This is an 11.4% improvement relative to the baseline model.

[4] Interestingly, in practice both methods result in near identical rejection probabilities, within a precision of 0.0001. This indicates that satisfying the constraint (18) for the new feature is more important, in terms of perplexity, than preserving the constraints of the previous features, insofar as those get violated.

[5] Alternatively we could have used our estimate for $P_N(s)$ described in section 2.4. A large sample of sentences would still be necessary though, to get a good estimate for equation (16).

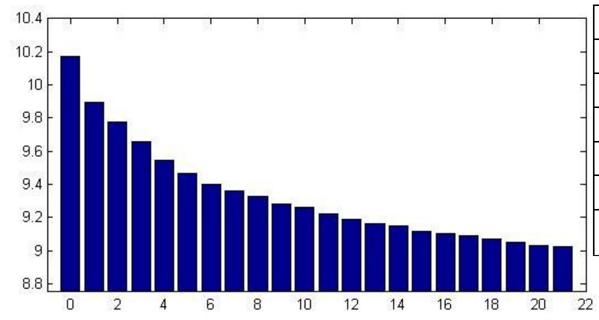

Figure 1. Model perplexity during training. The x-axis denotes the number of features added to the model. The final perplexity after 21 features is 9.02.

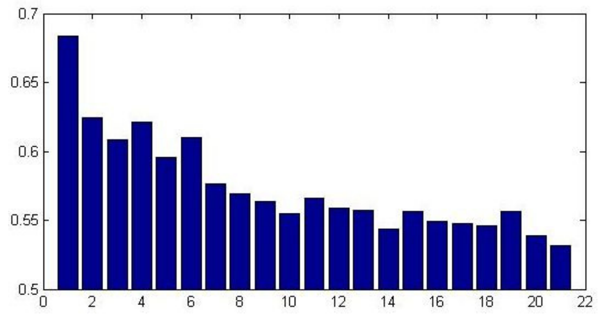

Figure 2. Classifier accuracy during training, assessed on held-out data. 0.5 signifies chance performance.

please list costs in at pittsburgh
what type of airplane is have an early morning
what types of aircraft is that a meal
what not nineteen forty two
between boston and atlanta on august fifteenth
which airlines fly american flying on
what is the flight leaving pittsburgh after six p m

Table 1. A sample of sentences generated by the baseline model, a trigram smoothed with modified kneser-ney smoothing.

what is the cost of flight d l three seventeen sixty five
what time does flight at eight thirty eight a m and six p m
what does fare code q w mean
what kind of aircraft will i be flying on
flights from philadelphia on saturday
what is the fare for flight two nine six
what is the cost of coach transcontinental flight u a three oh two from denver to san francisco

Table 2. A sample of sentences generated by the final model

Figure 2 shows the accuracy of the trained features on held-out data. The held-out data was composed of equal parts real and model sentences, so 50% accuracy is chance performance. As might have been expected, the classifiers start out with a relatively high accuracy of 68%, which dwindles down to little over 50% as more features are added to the model. Not surprisingly, there is a strong correlation between the accuracy of a feature and the reduction in perplexity it engenders (spearman correlation coefficient r=0.89, $p<10^{-5}$.)

In tables 1 and 2 we show a representative sample of sentences from the baseline model and from the final model. As the baseline model is a trigram, it cannot capture dependencies that span a range longer than two words. Hence sentences that start out seemingly in one topic and then veer off to another are common. The global information available to the features used by the boosted model greatly reduces this phenomenon. To get a quantitative sense of this, we generated 200 sentences from each model and submitted them for grammaticality testing by a proficient (though non-native) English speaker. Of the trigram-generated sentences, 86 were deemed grammatical (43%), while of those generated by the boosted model 132 were grammatical (66%). This difference is statistically significant with $p<10^{-5}$.

Finally, let us quantify the computational savings obtained from using rejection sampling. Let |V| stand for the lexicon size (here |V|=603) and |L| for the average sentence length (|L|=14). In Gibbs sampling, a sentence is sampled by starting out with a random sequence of words. For each word position, the current word is replaced with each word in the lexicon, and the probability of the resulting sentence is calculated. Then one of the words is randomly selected for this position in proportion to the calculated probabilities. The sentence has to be scanned in this manner several times for the sample to approximate the model distribution. Assuming we perform only 3 scans for each sentence, Gibbs sampling would have thus required us to classify $3|V||L| \approx 25,000$ sentences *per sampled sentence*. Given that in each iteration we generate 12,045 sentences, and that in the *n'th* iteration each sentence has to be classified by *n* features, this gives a total of roughly

$7 \cdot 10^{10}$ classifications after 21 iterations. In contrast, using rejection sampling, we used only $6.7 \cdot 10^{7}$ classifications in total – a difference of over three orders of magnitude.

4 Discussion

In this work we presented a method that enables using discriminative learning methods for refining generative language models. Utilizing large-margin classifiers that are trained to discriminate real sentences from model sentences we showed that sizeable improvements in perplexity over a state-of-the-art smoothed trigram are possible.

Our method bears some similarity to the recently developed *Contrastive Estimation* method (Smith and Eisner, 2004). Contrastive estimation (CE) was proposed as a means for training log-linear probabilistic models. As all training methods, contrastive estimation pushes probability mass unto positive samples. Unlike other methods, CE takes this probability mass from the 'neighborhood' of each positive sample. For example, given a real sentence *s*, CE might give it more probability by taking away probability from similar sentences which are likely to be ungrammatical, for instance sentences that are formed by taking *s* and switching the order of two adjacent words in it. This is intuitively similar to our approach – effectively, our model gives probability mass to positive samples, taking it away from sentences classified as model sentences. A major difference between the two approaches, however, is that in CE the definition of the sentence's neighborhood must be specified in advance by the modeler. In our work, the 'neighborhood' is determined automatically and dynamically as learning proceeds, according to the capabilities of the classifiers used.

The sentence representation we chose for this work is rather simple, and was intended primarily to demonstrate the efficacy of our approach. In future work we plan to experiment with richer representations, e.g. including long-range n-grams (Rosenfeld, 1996), class n-grams (Brown et al., 1992), grammatical features (Amaya and Benedy, 2001), etc'.

The main computational bottleneck in our approach is the generation of negative samples from the current model. Rejection sampling allowed us to use computationally intensive classifiers as our features by reducing the number of classifications that had to be performed during the sampling process. However, if the boosted model strays too far from the baseline P_0, these savings will be negated by the very large sentence rejection probabilities that will ensue. This is likely to be the case when richer representations as suggested above are used, necessitating a return to Gibbs sampling. Therefore, in future work we plan to experiment with classifiers whose decision function is cheaper to compute, such as neural networks and decision trees. Another possible direction would be using the recently proposed Deep Belief Network formalism (Hinton et al., 2006). DBNs utilize semi-linear features which are stacked recursively and thus very efficiently model non-linearities in their data. These have been used in the past for language modeling (Mnih and Hinton, 2007), but not within the whole-sentence framework.

Acknowledgements

We would like to thank Daisuke Okanohara for his generosity in providing the code for the large-margin classifier. The author is supported by the Yeshaya Horowitz association through the center of Complexity Science.

References

Fredy Amaya and Jose Miguel Benedi, 2001, Improvement of a whole sentence maximum entropy model using grammatical features. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics.*

Peter. F. Brown, Vincent. J. Della Pietra, Peter. V. deSouza, Jenifer. C. Lai and Robert. L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, Dec. 1992.

Stanley F. Chen and Joshua. Goodman. 1998. An empirical study of smoothing techniques for language modeling. *Technical Report TR–10–98*, Center for Research in Computing Technology, Harvard University

Stanley F. Chen and Ronald Rosenfeld. 1999a. Efficient sampling and feature selection in whole sentence maximum entropy language models. In *Proceedings of ICASSP'99. IEEE.*

Stanley F. Chen and Ronald Rosenfeld. 1999b. A Gaussian prior for smoothing maximum entropy models. *Technical Report CMUCS-99-108*, Carnegie Mellon University.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research*.

Jianfeng Gao, Hao Yu, Wei Yuan, and Peng Xu. 2005. Minimum sample risk methods for language modeling. In *Proc. of HLT/EMNLP*.

Charles T. Hemphill, John J. Godfrey and George R. Doddington. 1990. The ATIS Spoken Language Systems Pilot Corpus. *The workshop on speech and natural language*, Morgan Kaufmann.

Geoffrey E. Hinton, Simon Osindero and Yee-Whye Teh, 2006. A fast learning algorithm for deep belief nets, *Neural Computation*, 18(7):1527–1554.

Andriy Mnih and Geoffrey E. Hinton. 2007. Three new graphical models for statistical language modeling. In *Proceedings of the 24th international conference on Machine Learning*.

Daisuke Okanohara and Jun'ichi Tsujii. 2007. A discriminative language model with pseudo-negative samples. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*.

Brian Roark, Murat Saraclar, and Michael Collins. 2007. Discriminative n-gram language modeling. *Computer Speech and Language*, 21(2):373–392.

Ronald Rosenfeld. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer speech and language*, 10:187–228

Ronald Rosenfeld. 1997. A whole sentence maximum entropy language model. In *Proc. of the IEEE Workshop on Automatic Speech Recognition and Understanding*.

Noah A. Smith and Jason Eisner. 2005. Contrastive estimation: training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting of the Association of Computational Linguistics*.

Max Welling., Richard Zemel and Geoffrey E. Hinton. 2003. Self-Supervised Boosting. *Advances in Neural Information Processing Systems*, 15, MIT Press, Cambridge, MA