Computer Speech and Language (2001) **15**, 403–434
doi:10.1006/csla.2001.0174
Available online at http://www.idealibrary.com on IDEAL®

A bit of progress in language modeling

Joshua T. Goodman†

Machine Learning and Applied Statistics Group, Microsoft Research, One Microsoft Way, Redmond, WA 98052, U.S.A.

Abstract

In the past several years, a number of different language modeling improvements over simple trigram models have been found, including caching, higher-order *n*-grams, skipping, interpolated Kneser–Ney smoothing, and clustering. We present explorations of variations on, or of the limits of, each of these techniques, including showing that sentence mixture models may have more potential. While all of these techniques have been studied separately, they have rarely been studied in combination. We compare a combination of all techniques together to a Katz smoothed trigram model with no count cutoffs. We achieve perplexity reductions between 38 and 50% (1 bit of entropy), depending on training data size, as well as a word error rate reduction of 8.9%. Our perplexity reductions are perhaps the highest reported compared to a fair baseline.

1. Introduction

1.1. Overview

Language modeling is the art of determining the probability of a sequence of words. This is useful in a large variety of areas including speech recognition, optical character recognition, handwriting recognition, machine translation, and spelling correction (Church, 1988; Brown *et al.*, 1990; Kernighan, Church & Gale, 1990; Hull, 1992; Srihari and Baltus, 1992). The most commonly used language models are very simple (e.g. a Katz-smoothed trigram model). There are many improvements over this simple model however, including caching, clustering, higher-order *n*-grams, skipping models, and sentence-mixture models, all of which we will describe. Unfortunately, these more complicated techniques have rarely been examined in combination. It is entirely possible that two techniques that work well separately will not work well together, and, as we will show, even possible that some techniques will work better together than either one does by itself. In this paper, we will first examine each of the aforementioned techniques separately, looking at variations on the technique, or its limits. Then we will examine the techniques in various combinations, and compare to a Katz smoothed trigram with no count cutoffs. On a small training data set, 100 000 words, we can obtain up to a 50% perplexity reduction, which is 1 bit of entropy. On larger data sets, the improvement declines, going down to 41% on our largest data set, 284 000 000 words. On a similar large set without punctuation, the reduction is 38%. On that data set, we achieve an 8.9% word

†E-mail: joshuago@microsoft.com

0885–2308/01/040403 + 32 $35.00/0

error rate reduction. These are perhaps the largest reported perplexity reductions for a language model, vs. a fair baseline.

The paper is organized as follows. First, in this section, we will describe our terminology, briefly introduce the various techniques we examined, and describe our evaluation methodology. In the following sections, we describe each technique in more detail, and give experimental results with variations on the technique, determining for each the best variation, or its limits. In particular, for caching, we show that trigram caches have nearly twice the potential of unigram caches. For clustering, we find variations that work slightly better than traditional clustering, and examine the limits. For n-gram models, we examine up to 20-grams, but show that even for the largest models, performance has plateaued by 5- to 7-grams. For skipping models, we give the first detailed comparison of different skipping techniques, and the first that we know of at the 5-gram level. For sentence mixture models, we show that mixtures of up to 64 sentence types can lead to improvements. We then give experiments comparing all techniques, and combining all techniques in various ways. All of our experiments are done on three or four data sizes, showing which techniques improve with more data, and which get worse. In the concluding section, we discuss our results.

There is also an extended version of this paper (Goodman, 2001*a*) that goes into much more detail than this version. The extended version contains more tutorial information, more details about the experiments presented here, interesting implementation details, and a few additional experiments and proofs. It is meant to be a reasonable introduction to the field of language modeling.

1.2. Technique introductions

The goal of a language model is to determine the probability of a word sequence $w_1 \dots w_n$, $P(w_1 \dots w_n)$. This probability is typically broken down into its component probabilities:

$$P(w_1 \dots w_i) = P(w_1) \times P(w_2|w_1) \times \cdots \times P(w_i|w_1 \dots w_{i-1}).$$

Since it may be difficult to compute a probability of the form $P(w_i|w_1 \dots w_{i-1})$ for large i, we typically assume that the probability of a word depends on only the two previous words, the *trigram* assumption:

$$P(w_i|w_1 \dots w_{i-1}) \approx P(w_i|w_{i-2}w_{i-1})$$

which has been shown to work well in practice. The trigram probabilities can then be estimated from their counts in a training corpus. We let $C(w_{i-2}w_{i-1}w_i)$ represent the number of occurrences of $w_{i-2}w_{i-1}w_i$ in our training corpus, and similarly for $C(w_{i-2}w_{i-1})$. Then, we can approximate:

$$P(w_i|w_{i-2}w_{i-1}) \approx \frac{C(w_{i-2}w_{i-1}w_i)}{C(w_{i-2}w_{i-1})}.$$

Unfortunately, in general this approximation will be very noisy, because there are many three word sequences that never occur. Consider, for instance, the sequence *"party on Tuesday"*. What is $P(\textit{Tuesday}|\textit{party on})$? Our training corpus might not contain any instances of the phrase, so $C(\textit{party on Tuesday})$ would be 0, while there might still be 20 instances of the phrase *"party on"*. Thus, we would predict $P(\textit{Tuesday}|\textit{party on}) = 0$, clearly an underestimate. This kind of 0 probability can be very problematic in many applications of language models. For instance, in a speech recognizer, words assigned 0 probability cannot be recognized no matter how unambiguous the acoustics.

Smoothing techniques take some probability away from some occurrences. Imagine that we have in our training data a single example of the phrase *"party on Stan Chen's birthday"*. Typically, when something occurs only once, it is greatly overestimated. In particular,

$$P(Stan|party\ on) \ll \frac{1}{20} = \frac{C(party\ on\ Stan)}{C(party\ on)}.$$

By taking some probability away from some words, such as *"Stan"* and redistributing it to other words, such as *"Tuesday"*, zero probabilities can be avoided. In a smoothed trigram model, the extra probability is typically distributed according to a smoothed bigram model, etc. While the most commonly used smoothing techniques, Katz smoothing (Katz, 1987) and Jelinek–Mercer smoothing (Jelinek & Mercer, 1980) (sometimes called deleted interpolation), work fine, even better smoothing techniques exist. In particular, we have previously shown (Chen & Goodman, 1999) that versions of Kneser–Ney smoothing (Ney, Essen & Kneser, 1994) outperform all other smoothing techniques. In the appendix of the extended version of this paper, we give a proof partially explaining this optimality. In Kneser–Ney smoothing, the backoff distribution is modified: rather than a normal bigram distribution, a special distribution is used. Using Kneser–Ney smoothing instead of more traditional techniques is the first improvement we used.

The most obvious extension to trigram models is to simply move to *higher-order n-grams*, such as 4-grams and 5-grams. We will show that, in fact, significant improvements can be gained from moving to 5-grams. Furthermore, in the past, we have shown that there is a significant interaction between smoothing and n-gram order (Chen & Goodman, 1999): higher-order n-grams work better with Kneser–Ney smoothing than with some other methods, especially Katz smoothing. We will also look at how much improvement can be gained from higher order n-grams, examining up to 20-grams.

Another simple extension to n-gram models is *skipping* models (Huang *et al.*, 1993; Rosenfeld, 1994; Ney *et al.*, 1994), in which we condition on a different context than the previous two words. For instance, instead of computing $P(w_i|w_{i-2}w_{i-1})$, we could instead compute $P(w_i|w_{i-3}w_{i-2})$. This latter model is probably not as good, but can be combined with the standard model to yield some improvements.

Clustering (also called *classing*) models attempt to make use of the similarities between words. For instance, if we have seen occurrences of phrases like "*party on Monday*" and "*party on Wednesday*", then we might imagine that the word "*Tuesday*", being similar to both "*Monday*" and "*Wednesday*", is also likely to follow the phrase "*party on*." The majority of the previous research on word clustering has focused on how to get the best clusters. We have concentrated our research on the best way to *use* the clusters, and will report results showing some novel techniques that work slightly better than previous methods.

Caching models (Kuhn, 1988; Kuhn & De Mori, 1990; Kuhn &De Mori, 1992) make use of the observation that if you use a word, you are likely to use it again. They tend to be easy to implement and to lead to relatively large perplexity improvements, but relatively small word-error rate improvements. We show that by using a trigram cache, we can get almost twice the improvement as from a unigram cache.

Sentence mixture models (Iyer & Ostendorf, 1999; Iyer, Ostendorf & Rohlicek, 1994) make use of the observation that there are many different sentence types, and that making models for each type of sentence may be better than using one global model. Traditionally, only four to eight types of sentences are used, but we show that improvements can be obtained by going to 64 mixtures, or perhaps more.

1.3. Evaluation

In this section, we first describe and justify our use of perplexity or entropy as an evaluation technique. We then describe the data and experimental techniques used in the experiments in the following sections.

We will primarily measure our performance by the entropy of test data as given by the model (which should be called cross-entropy):

$$\frac{1}{N}\sum_{i=1}^{N} -\log_2 P(w_i|w_1 \ldots w_{i-1}).$$

Entropy has several nice properties. First, it is the average number of bits that would be required to encode the test data using an optimal coder. Also, assuming the test data is generated by some random process, a perfect model of this process would have the lowest possible entropy, so the lower the entropy, the closer we are, in some sense, to this true model. Sometimes, we will also measure perplexity, which is simply $2^{entropy}$, and corresponds to the weighted average number of choices for each word. Several alternatives to entropy have been shown to correlate better with speech recognition performance, but they are typically speech-recognizer-specific and much harder to compute in our framework.

All of our experiments were performed on the NAB (North American Business news) corpus (Stern, 1996). We performed most experiments at four different training data sizes: 100 000 words, 1 000 000 words, 10 000 000 words, and the whole corpus—except 1994 Wall Street Journal (WSJ) data—approximately 284 000 000 words. In all cases, we performed parameter optimization on a separate set of heldout data, and then performed testing on a set of test data. None of the three data sets overlapped. The heldout and test sets were always every 50th sentence from two non-overlapping sets of 1 000 000 words, taken from the 1994 section. In the appendix, we describe implementation tricks we used; these tricks made it possible to train very complex models on very large amounts of training data, but made it hard to test on large test sets. For this reason, we used only 20 000 words total for testing or heldout data. On the other hand, we did not simply want to use, say, a 20 000 word contiguous test or heldout set, since this would only constitute a few articles, and thus risk problems from too much homogeneity; thus we chose to use every 50th sentence from non-overlapping 1 000 000 word sets. All of our experiments were done using the same 58 546 word vocabulary. End-of-sentence, end-of-paragraph, and end-of-article symbols were included in perplexity computations, but out-of-vocabulary words were not.

It would have been interesting to try our experiments on other corpora, as well as other data sizes. In our previous work (Chen & Goodman, 1999), we compared both across corpora and across data sizes. We found that different corpora were qualitatively similar, and that the most important differences were across training data sizes. We therefore decided to concentrate our experiments on different training data sizes, rather than on different corpora.

Our toolkit is unusual in that it allows all parameters to be jointly optimized. In particular, when combining many techniques, there are many interpolation and smoothing parameters that need to be optimized. We used Powell's algorithm (Press, Flannery, Teukolsky & Vetterling, 1988) over the heldout data to jointly optimize all of these parameters.

2. Smoothing

There are many different smoothing techniques that can be used, and the subject is a surprisingly subtle and complicated one. Those interested in smoothing should consult our previous

work (Chen & Goodman, 1999), where detailed descriptions and detailed comparisons of almost all commonly used smoothing algorithms are done. We will limit our discussion here to four main techniques: simple interpolation, Katz smoothing, Backoff Kneser–Ney smoothing, and Interpolated Kneser–Ney smoothing. In this section, we describe those four techniques, and recap previous results, including the important result that Interpolated Kneser–Ney smoothing, or minor variations on it, outperforms all other smoothing techniques.

The simplest way to combine techniques in language modeling is to simply interpolate them together. For instance, if one has a trigram model, a bigram model, and a unigram model, one can use

$$P_{\texttt{interpolate}}(w|w_{i-2}w_{i-1}) = \lambda P_{\texttt{trigram}}(w|w_{i-2}w_{i-1}) + (1-\lambda)[\mu P_{\texttt{bigram}}(w|w_{i-1}) + (1-\mu)P_{\texttt{unigram}}(w)]$$

where λ and μ are constants such that $0 \leq \lambda, \mu \leq 1$. Given its simplicity, simple interpolation works surprisingly well, but other techniques, such as Katz smoothing, work even better.

Katz smoothing (Katz, 1987) is based on the Good–Turing formula (Good, 1953). Notice that if a particular word sequence (i.e. "*party on Stan*") occurs only once (out of perhaps a billion words) it is probably significantly overestimated—it probably just showed up by chance, and its true probability is much less than one billionth. It turns out that the same thing is true to a lesser degree for sequences that occurred twice, and so on. Let n_r represent the number of n-grams that occur r times, i.e.

$$n_r = |\{w_{i-n+1} \ldots w_i | C(w_{i-n+1} \ldots w_i) = r\}|.$$

Good proved that under some very weak assumptions that for any n-gram that occurs r times, we should *discount* it, pretending that it occurs $disc(r)$ times where

$$disc(r) = (r+1)\frac{n_{r+1}}{n_r}$$

[$disc(r)$ is more typically written as r^*]. In language modeling, the estimate $disc(r)$ will almost always be less than r. This will leave a certain amount of probability "left-over." In fact, letting N represent the total size of the training set, this left-over probability will be equal to $\frac{n_1}{N}$; this represents the amount of probability to be allocated for events that were never seen.

For a given context, Katz smoothing uses one of two formulae. If the word sequence $w_{i-n+1} \ldots w_i$ has been seen before, then Katz smoothing uses the discounted count of the sequence, divided by the counts of the context $w_{i-n+1} \ldots w_{i-1}$. On the other hand, if the sequence has never been seen before, then we back off to the next lower distribution, $w_{i-n+2} \ldots w_i$. Basically, we use the following formula:

$$P_{\texttt{Katz}}(w_i|w_{i-n+1} \ldots w_{i-1}) = \begin{cases} \frac{disc(C(w_{i-n+1} \ldots w_i))}{C(w_{i-n+1} \ldots w_{i-1})} & \text{if } C(w_{i-n+1} \ldots w_i) > 0 \\ \alpha(w_{i-n+1} \ldots w_{i-1}) \times P_{\texttt{Katz}}(w_i|w_{i-n+2} \ldots w_{i-1}) & \text{otherwise} \end{cases}$$

where $\alpha(w_{i-n+1} \ldots w_{i-1})$ is a normalization constant chosen so that the probabilities sum to 1.[1]

[1] Chen and Goodman (1999), as well as the appendix of the extended version of this paper, give the details of our implementation of Katz smoothing. Briefly, we also smooth the unigram distribution using additive smoothing; we discount counts only up to k, where we determine k to be as large as possible, while still giving reasonable discounts according to the Good–Turing formula; we add pseudo-counts β for any context with no discounted counts. Tricks are used to estimate n_r.

Katz smoothing is one of the most commonly used smoothing techniques, but it turns out that other techniques work even better. Chen and Goodman (1999) performed a detailed comparison of many smoothing techniques and found that a modified interpolated form of Kneser–Ney smoothing (Ney *et al.*, 1994) consistently outperformed all other smoothing techniques. The basic insight behind Kneser–Ney smoothing is the following. Consider a conventional bigram model of a phrase such as $P_{\texttt{Katz}}(\textit{Francisco}|\textit{on})$. Since the phrase *San Francisco* is fairly common, the conventional unigram probability (as used by Katz smoothing or techniques like deleted interpolation) $\frac{C(\textit{Francisco})}{\sum_w C(w)}$ will also be fairly high. This means that using, for instance, a model such as Katz smoothing, the probability

$$\begin{aligned} P_{\texttt{Katz}}(\textit{on Francisco}) &= \begin{cases} \frac{\textit{disc}(C(\textit{on Francisco}))}{C(\textit{on})} & \text{if } C(\textit{on Francisco}) > 0 \\ \alpha(\textit{on}) \times P_{\texttt{Katz}}(\textit{Francisco}) & \text{otherwise} \end{cases} \\ &= \alpha(\textit{on}) \times P_{\texttt{Katz}}(\textit{Francisco}) \end{aligned}$$

will also be fairly high. But, the word *Francisco* occurs in exceedingly few contexts, and its probability of occurring in a new one is very low. Kneser–Ney smoothing uses a modified backoff distribution based on the number of contexts each word occurs in, rather than the number of occurrences of the word. Thus, a probability such as $P_{\texttt{KN}}(\textit{Francisco}|\textit{on})$ would be fairly low, while for a word like *Tuesday* that occurs in many contexts, $P_{\texttt{KN}}(\textit{Tuesday}|\textit{on})$ would be relatively high, even if the phrase *on Tuesday* did not occur in the training data. Kneser–Ney smoothing also uses a simpler discounting scheme than Katz smoothing: rather than computing the discounts using Good–Turing, a single discount, D, (optimized on held-out data) is used. In particular, Backoff Kneser–Ney smoothing uses the following formula (given here for a bigram) where $|\{v|C(vw_i) > 0\}|$ is the number of words v that w_i can occur in the context of

$$P_{\texttt{BKN}}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i)-D}{C(w_{i-1})} & \text{if } C(w_{i-1}w_i) > 0 \\ \alpha(w_{i-1}) \frac{|\{v|C(vw_i)>0\}|}{\sum_w |\{v|C(vw)>0\}|} & \text{otherwise} \end{cases} .$$

Again, α is a normalization constant such that the probabilities sum to 1. The formula can be easily extended to higher order n-grams in general. For instance, for trigrams, both the unigram and bigram distributions are modified.

Chen and Goodman (1999) showed that methods like Katz smoothing and Backoff Kneser–Ney smoothing that backoff to lower order distributions only when the higher order count is missing do not do well on low counts, such as one counts and two counts. This is because the estimates of these low counts are fairly poor, and the estimates ignore useful information in the lower order distribution. *Interpolated* models always combine both the higher order and the lower order distribution, and typically work better. In particular, the formula for Interpolated Kneser–Ney smoothing is

$$P_{\texttt{IKN}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})} + \lambda(w_{i-1}) \frac{|\{v|C(vw_i) > 0\}|}{\sum_w |\{v|C(vw) > 0\}|}$$

where $\lambda(w_{i-1})$ is a normalization constant such that the probabilities sum to 1. Chen and Goodman (1999) proposed one additional modification to Kneser–Ney smoothing, the use of multiple discounts, one for one counts, another for two counts, and another for three or more counts. This formulation, Modified Kneser–Ney smoothing, typically works slightly better than Interpolated Kneser–Ney. However, in our experiments on combining techniques, it would have nearly tripled the number of parameters our system needed to search, and in a pilot study, when many techniques were combined, it did not work better than Interpolated

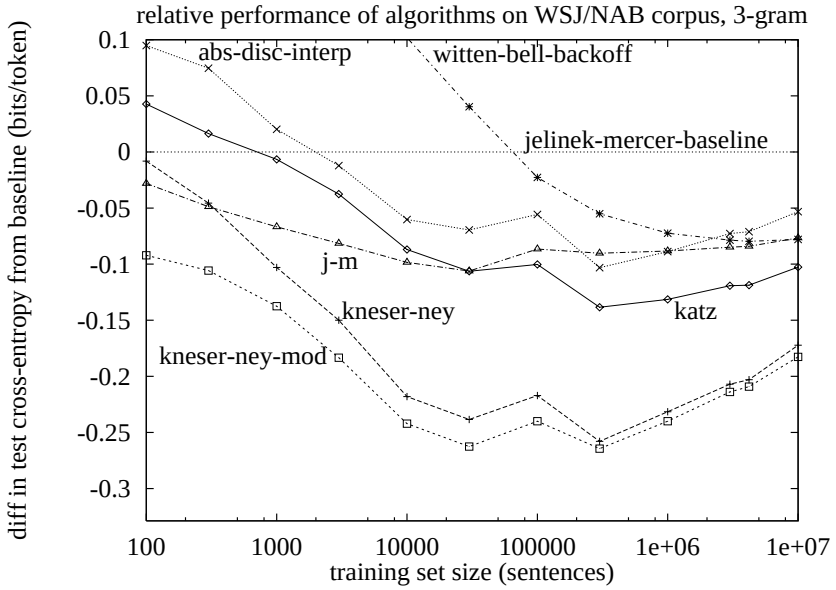

Figure 1. Smoothing results across data sizes.

Kneser–Ney. Thus, in the rest of this paper, we use Interpolated Kneser–Ney instead of Modified Kneser–Ney. In the appendix of the long version of this paper, we give a few more details about our implementation of our smoothing techniques, including standard refinements used for Katz smoothing. We also give arguments justifying Kneser–Ney smoothing, and example code, showing that interpolated Kneser–Ney smoothing is easy to implement.

In Figure 1, we repeat results from Chen and Goodman (1999). These are the only results in this paper not run on exactly the same sections of the corpus for heldout, training, and test as the rest of the paper, but we expect them to be very comparable. The baseline used for these experiments was a simple version of Jelinek–Mercer smoothing, using a single bucket; that version is identical to the first smoothing technique we described, simple interpolation. Kneser–Ney smoothing is the interpolated version of Kneser–Ney smoothing used throughout this paper, and Kneser–Ney mod is the version with three discounts instead of a single discount. Katz smoothing is essentially the same as the version in this paper. j-m is short for Jelinek–Mercer smoothing, sometimes called deleted interpolation elsewhere; abs-disc-interp is the interpolated version of absolute discounting. Training set size was measured in sentences, rather than in words, with about 20 words per sentence. Notice that Jelinek–Mercer smoothing and Katz smoothing cross, one being better at lower data sizes, the other at higher sizes. This was part of our motivation for running all experiments in this paper on multiple data sizes. On the other hand, in those experiments, which were done on multiple corpora, we did not find any techniques where one technique worked better on one corpus, and another worked better on another one. Thus, we feel reasonably confident in our decision not to run on multiple corpora. Chen and Goodman (1999) give a much more complete comparison of these techniques, as well as much more in depth analysis. Chen and Goodman (1998) gives a superset that also serves as a tutorial introduction.

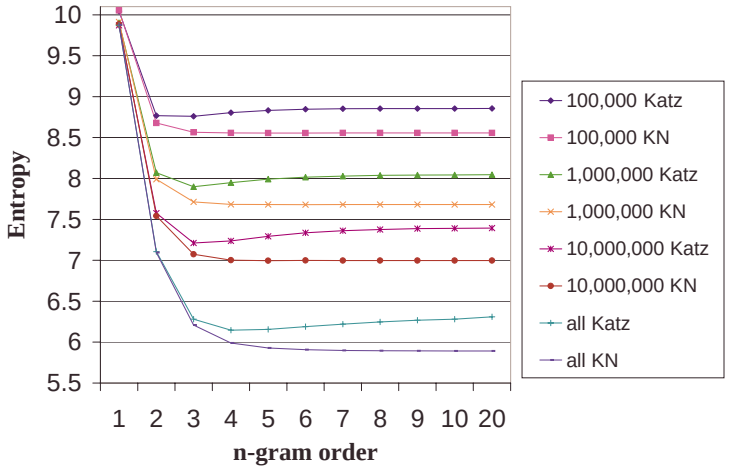

Figure 2. *n*-Gram order vs. entropy.

3. Higher-order n-grams

While the trigram assumption has proven, in practice, to be reasonable, there are many cases in which even longer contexts can be helpful. It is thus natural to relax the trigram assumption, and, rather than computing $P(w_i|w_{i-2}w_{i-1})$, use a longer context, such as $P(w_i|w_{i-4}w_{i-3}w_{i-2}w_{i-1})$, a 5-gram model. In many cases, no sequence of the form $w_{i-4}w_{i-3}w_{i-2}w_{i-1}$ will have been seen in the training data, and the system will need to backoff to or interpolate with four grams, trigrams, bigrams, or even unigrams, but in those cases where such a long sequence has been seen, it may be a good predictor of w_i.

Some earlier experiments with longer contexts showed little benefit from them. This turns out to be partially due to smoothing. As shown by Chen and Goodman (1999), some smoothing methods work significantly better with higher-order *n*-grams than others do. In particular, the advantage of Interpolated Kneser–Ney smoothing is much larger with higher-order *n*-grams than with lower-order ones.

We performed a variety of experiments on the relationship between *n*-gram order and perplexity. In particular, we tried both Katz smoothing and Interpolated Kneser–Ney smoothing on *n*-gram orders from one to 10, as well as 20, and over our standard data sizes. The results are shown in Figure 2.

As can be seen, and has been previously observed (Chen & Goodman, 1999), the behavior for Katz smoothing is very different from the behavior for Kneser–Ney smoothing. Chen and Goodman determined that the main cause of this difference was that backoff smoothing techniques, such as Katz smoothing, or even the backoff version of Kneser–Ney smoothing (we use only interpolated Kneser–Ney smoothing in this work), work poorly on low counts, especially one counts, and that as the *n*-gram order increases, the number of one counts increases. In particular, Katz smoothing has its best performance around the trigram level, and actually gets worse as this level is exceeded. Kneser–Ney smoothing, on the other hand, is essentially monotonic even through 20-grams.

The plateau point for Kneser–Ney smoothing depends on the amount of training data available. For small amounts, 100 000 words, the plateau point is at the trigram level, whereas when using the full training data, 280 million words, small improvements occur even into

the 6-gram (0.02 bits better than 5-gram) and 7-gram (0.01 bits better than 6-gram). Differences of this size are interesting, but not of practical importance. The difference between 4-grams and 5-grams, 0.06 bits, is perhaps important, and so, for the rest of our experiments, we often use models built on 5-gram data, which appears to give a good tradeoff between computational resources and performance.

Note that, in practice, going beyond trigrams is often impractical. The tradeoff between memory and performance typically requires heavy pruning of 4-grams and 5-grams, reducing the potential improvement from them. Throughout this paper, we ignore memory-performance tradeoffs, since this would overly complicate already difficult comparisons. We seek instead to build the single best system possible, ignoring memory issues, and leaving the more practical, more interesting, and very much more complicated issue of finding the best system at a given memory size, for future research [and a bit of past research, too (Goodman & Gao, 2000)]. Note that many of the experiments done in this section could not be done at all without the special tool described briefly at the end of this paper, and in more detail in the appendix of the extended version of this paper.

4. Skipping

As one moves to larger and larger *n*-grams, there is less and less chance of having seen the exact context before; but the chance of having seen a similar context, one with most of the words in it, increases. Skipping models (Huang *et al.*, 1993; Ney *et al.*, 1994; Rosenfeld, 1994; Martin, Hamacher, Liermann, Wessel & Ney, 1999; Siu & Ostendorf, 2000) make use of this observation. There are also variations on this technique, such as techniques using lattices (Saul & Pereira, 1997; Dupont & Rosenfeld, 1997), or models combining classes and words (Blasig, 1999).

When considering a 5-gram context, there are many subsets of the 5-gram we could consider, such as $P(w_i|w_{i-4}w_{i-3}w_{i-1})$ or $P(w_i|w_{i-4}w_{i-2}w_{i-1})$. Perhaps we have never seen the phrase "*Show John a good time*" but we have seen the phrase "*Show Stan a good time.*" A normal 5-gram predicting P(*time*|*show John a good*) would back off to *P(time | John a good)* and from there to P(*time*|*a good*), which would have a relatively low probability. On the other hand, a skipping model of the form $P(w_i|w_{i-4}w_{i-2}w_{i-1})$ would assign high probability to P(*time*|*show* ____ *a good*).

These skipping 5-grams are then interpolated with a normal 5-gram, forming models such as

$$\lambda P(w_i|w_{i-4}w_{i-3}w_{i-2}w_{i-1}) + \mu P(w_i|w_{i-4}w_{i-3}w_{i-1}) + (1-\lambda-\mu)P(w_i|w_{i-4}w_{i-2}w_{i-1})$$

where, as usual, $0 \leq \lambda \leq 1$ and $0 \leq \mu \leq 1$ and $0 \leq (1-\lambda-\mu) \leq 1$.

Another (and more traditional) use for skipping is as a sort of poor man's higher order *n*-gram. One can, for instance, create a model of the form

$$\lambda P(w_i|w_{i-2}w_{i-1}) + \mu P(w_i|w_{i-3}w_{i-1}) + (1-\lambda-\mu)P(w_i|w_{i-3}w_{i-2}).$$

In a model of this form, no component probability depends on more than two previous words, like a trigram, but the overall probability is 4-gram-like, since it depends on w_{i-3}, w_{i-2}, and w_{i-1}. We can extend this idea even further, combining in all pairs of contexts in a 5-gram-like, 6-gram-like, or even 7-gram-like way, with each component probability never depending on more than the previous two words.

We performed two sets of experiments, one on 5-grams and one on trigrams. For the 5-gram skipping experiments, all contexts depended on at most the previous four words,

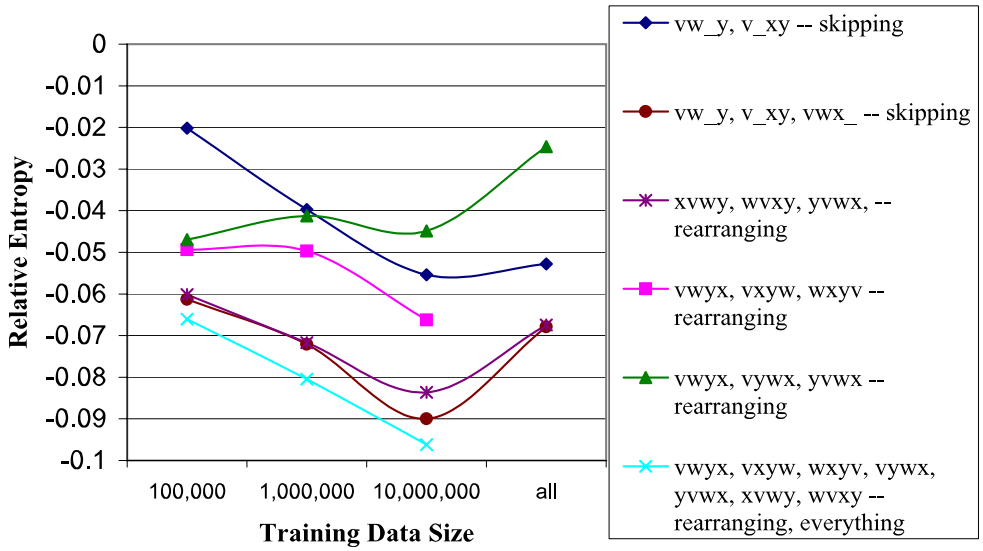

Figure 3. 5-Gram skipping techniques vs. 5-gram baseline.

w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1}, but used the four words in a variety of ways. We tried six models, all of which were interpolated with a baseline 5-gram model. For readability and conciseness, we define a new notation, letting $v = w_{i-4}$, $w = w_{i-3}$, $x = w_{i-2}$ and $y = w_{i-1}$, allowing us to avoid numerous subscripts in what follows. The results are shown in Figure 3.

The first model interpolated dependencies on vw_y and v_xy. This simple model does not work well on the smallest training data sizes, but is competitive for larger ones. Next, we tried a simple variation on this model, which also interpolated in $vwx\_$. Making that simple addition leads to a good-sized improvement at all levels, roughly 0.02 to 0.04 bits over the simpler skipping model. Our next variation was analogous, but adding back in the dependencies on the missing words. In particular, we interpolated together $xvwy$, $wvxy$, and $yvwx$; that is, all models depended on the same variables, but with the interpolation order modified. For instance, by $xvwy$, we refer to a model of the form $P(z|vwxy)$ interpolated with $P(z|vw\_y)$ interpolated with $P(z|w\_y)$ interpolated with $P(z|y)$ interpolated with $P(z|y)$ interpolated with $P(z)$. All of these experiments were done with Interpolated Kneser–Ney smoothing, so all but the first probability use the modified backoff distribution. This model is just like the previous one, but for each component starts the interpolation with the full 5-gram. We had hoped that in the case where the full 5-gram had occurred in the training data, this would make the skipping model more accurate, but it did not help at all.[2]

We also wanted to try more radical approaches. For instance, we tried interpolating together $vwyx$ with $vxyw$ and $wxyv$ (along with the baseline $vwxy$). This model puts each of the four preceding words in the last (most important) position for one component. This model does not work as well as the previous two, leading us to conclude that the y word is by far the most important. We also tried a model with $vwyx$, $vywx$, $yvwx$, which puts the y word in each possible position in the backoff model. This was overall the worst model, reconfirming the intuition that the y word is critical. However, as we saw by adding $vwx\_$ to vw_y and

[2]In fact, it hurt a tiny bit, 0.005 bits at the 10 000 000 word training level. This turned out to be due to technical smoothing issues.

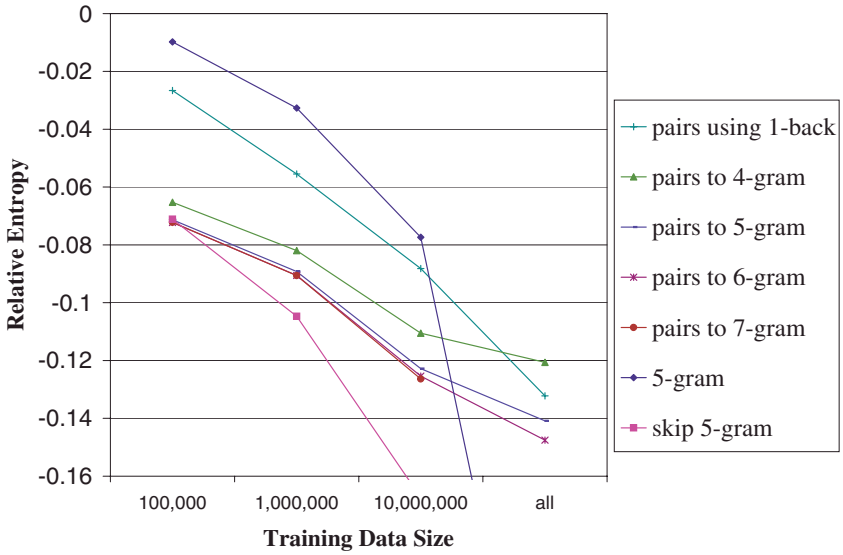

Figure 4. Trigram skipping techniques vs. trigram baseline.

v_xy, having a component with the x position final is also important. This will also be the case for trigrams.

Finally, we wanted to get a sort of upper bound on how well 5-gram models could work. For this, we interpolated together $vwyx$, $vxyw$, $wxyv$, $vywx$, $yvwx$, $xvwy$ and $wvxy$. This model was chosen as one that would include as many pairs and triples of combinations of words as possible. The result is a marginal gain—less than 0.01 bits—over the best previous model.

We do not find these results particularly encouraging. In particular, when compared to the sentence mixture results that will be presented later, there seems to be less potential to be gained from skipping models. Also, while sentence mixture models appear to lead to larger gains the more data that is used, skipping models appear to get their maximal gain around 10 000 000 words. Presumably, at the largest data sizes, the 5-gram model is becoming well trained, and there are fewer instances where a skipping model is useful but the 5-gram is not.

We also examined trigram-like models. These results are shown in Figure 4. The baseline for comparison was a trigram model. For comparison, we also show the relative improvement of a 5-gram model over the trigram, and the relative improvement of the skipping 5-gram with vw_y, v_xy and $vwx\_$. For the trigram skipping models, each component never depended on more than two of the previous words. We tried five experiments of this form. First, based on the intuition that pairs using the 1-back word (y) are most useful, we interpolated xy, wy, vy, uy and ty models. This did not work particularly well, except at the largest sizes. Presumably at those sizes, a few appropriate instances of the 1-back word had always been seen. Next, we tried using all pairs of words through the 4-gram level: xy, wy and wx. Considering its simplicity, this worked very well. We tried similar models using all 5-gram pairs, all 6-gram pairs and all 7-gram pairs; this last model contained 15 different pairs. However, the improvement over 4-gram pairs was still marginal, especially considering the large number of increased parameters.

The trigram skipping results are, relative to their baseline, much better than the 5-gram skipping results. They do not appear to have plateaued when more data is used and they are much more comparable to sentence mixture models in terms of the improvement they get. Furthermore, they lead to more improvement than a 5-gram alone does when used on small amounts of data (although, of course, the best 5-gram skipping model is always better than the best trigram skipping model). This makes them a reasonable technique to use with small and intermediate amounts of training data, especially if 5-grams cannot be used.

5. Clustering

5.1. Using clusters

Next, we describe our clustering techniques, which are a bit different (and, as we will show, slightly more effective) than traditional clustering (Brown, DellaPietra, deSouza, Lai & Mercer, 1992; Ney *et al.*, 1994). Consider a probability such as $P(\textit{Tuesday}|\textit{party on})$. Perhaps the training data contains no instances of the phrase *"party on Tuesday"*, although other phrases such as *"party on Wednesday"* and *"party on Friday"* do appear. We can put words into classes, such as the word *"Tuesday"* into the class *WEEKDAY*. Now, we can consider the probability of the word *"Tuesday"* given the phrase *"party on"*, and also given that the next word is a *WEEKDAY*. We will denote this probability by $P(\textit{Tuesday}|\textit{party on WEEKDAY})$. We can then decompose the probability

$$\begin{aligned} &P(\textit{Tuesday}|\textit{party on}) \\ &\quad = P(\textit{WEEKDAY}|\textit{party on}) \times P(\textit{Tuesday}|\textit{party on WEEKDAY}). \end{aligned}$$

When each word belongs to only one class, which is called hard clustering, this decompostion is a strict equality. Notice that, because we are using hard clusters, if we know w_i, we also know W_i, meaning that $P(w_{i-2}w_{i-1}W_i w_i) = P(w_{i-2}w_{i-1}w_i)$. With this fact,

$$\begin{aligned} P(W_i|w_{i-2}w_{i-1}) \times P(w_i|w_{i-2}w_{i-1}W_i) &= \frac{P(w_{i-2}w_{i-1}W_i)}{P(w_{i-2}w_{i-1})} \times \frac{P(w_{i-2}w_{i-1}W_i w_i)}{P(w_{i-2}w_{i-1}W_i)} \\ &= \frac{P(w_{i-2}w_{i-1}W_i w_i)}{P(w_{i-2}w_{i-1})} \\ &= \frac{P(w_{i-2}w_{i-1}w_i)}{P(w_{i-2}w_{i-1})} \\ &= P(w_i|w_{i-2}w_{i-1}). \end{aligned} \tag{1}$$

The extended version of the paper gives a slightly more detailed derivation.

Now, although Equation (1) is a strict equality, when smoothing is taken into consideration, using the clustered probability will be more accurate than the non-clustered probability. For instance, even if we have never seen an example of *"party on Tuesday"*, perhaps we have seen examples of other phrases, such as *"party on Wednesday"* and thus, the probability $P(\textit{WEEKDAY}|\textit{party on})$ will be relatively high. And although we may never have seen an example of *"party on WEEKDAY Tuesday"*, after we backoff or interpolate with a lower order model, we may be able to accurately estimate $P(\textit{Tuesday}|\textit{on WEEKDAY})$. Thus, our smoothed clustered estimate may be a good one. We call this particular kind of clustering *predictive clustering*. (On the other hand, we will show that if the clusters are poor, predictive clustering can also lead to degradation.)

Note that predictive clustering has other uses as well as for improving perplexity. Predictive clustering can be used to significantly speed up maximum entropy training (Goodman, 2001*b*), by up to a factor of 35, as well as to compress language models (Goodman & Gao, 2000).

Another type of clustering we can do is to cluster the words in the contexts. For instance, if *"party"* is in the class *EVENT* and *"on"* is in the class *PREPOSITION*, then we could write

$$P(\textit{Tuesday}|\textit{party on}) \approx P(\textit{Tuesday}|\textit{EVENT PREPOSITION})$$

or more generally

$$P(w|w_{i-2}w_{i-1}) \approx P(w|W_{i-2}W_{i-1}). \tag{2}$$

Combining Equation (2) with Equation (1) we get

$$P(w|w_{i-2}w_{i-1}) \approx P(W|W_{i-2}W_{i-1}) \times P(w|W_{i-2}W_{i-1}W). \tag{3}$$

Since Equation (3) does not take into account the exact values of the previous words, we always (in this work) interpolate it with a normal trigram model. We call the interpolation of Equation (3) with a trigram *fullibm* clustering. We call it fullibm because it is a generalization of a technique invented at IBM (Brown *et al.*, 1992), which uses the approximation $P(w|W_{i-2}W_{i-1}W) \approx P(w|W)$ to get

$$P(w|w_{i-2}w_{i-1}) \approx P(W|W_{i-2}W_{i-1}) \times P(w|W) \tag{4}$$

which, when interpolated with a normal trigram, we refer to as *ibm* clustering. Given that fullibm clustering uses more information than regular ibm clustering, we assumed that it would lead to improvements. As will be shown, it works about the same, at least when interpolated with a normal trigram model.

Alternatively, rather than always discarding information, we could simply change the backoff order, called *index* clustering:

$$P_{\texttt{index}}(\textit{Tuesday}|\textit{party on}) = P(\textit{Tuesday}|\textit{party EVENT on PREPOSITION}). \tag{5}$$

Here, we abuse notation slightly to use the order of the words on the right side of the | to indicate the backoff/interpolation order. Thus, Equation (5) implies that we would go from $P(\textit{Tuesday}|\textit{party EVENT on PREPOSITION})$ to $P(\textit{Tuesday}|\textit{EVENT on PREPOSITION})$ to $P(\textit{Tuesday}|\textit{on PREPOSITION})$ to $P(\textit{Tuesday}|\textit{PREPOSITION})$ to $P(\textit{Tuesday})$. Notice that since each word belongs to a single cluster, some of these variables are redundant. For instance, in our notation

$$C(\textit{party EVENT on PREPOSITION}) = C(\textit{party on})$$

and

$$C(\textit{EVENT on PREPOSITION}) = C(\textit{EVENT on}).$$

We generally write an index clustered model as $P(w_i|w_{i-2}W_{i-2}w_{i-1}W_{i-1})$.

There is one especially noteworthy technique, *fullibmpredict*. This is the best performing technique we have found (other than combination techniques.) This technique makes use of the intuition behind predictive clustering, factoring the problem into prediction of the cluster, followed by prediction of the word given the cluster. In addition, at each level, it smoothes this prediction by combining a word-based and a cluster-based estimate. It is not interpolated with a normal trigram model. It is of the form

$$\begin{aligned} P_{\texttt{fullibmpredict}}(w|w_{i-2}w_{i-1}) = \; & (\lambda P(W|w_{i-2}w_{i-1}) + (1-\lambda)P(W|W_{i-2}W_{i-1})) \times \\ & (\mu P(w|w_{i-2}w_{i-1}W) + (1-\mu)P(w|W_{i-2}W_{i-1}W). \end{aligned}$$

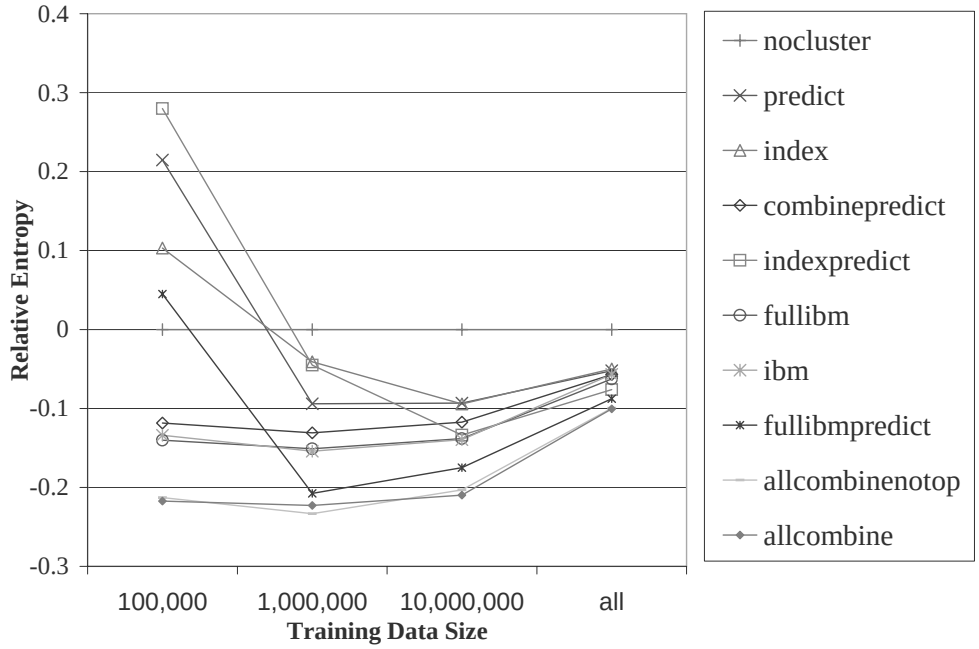

Figure 5. Comparison of nine different clustering techniques, Kneser–Ney smoothing.

There are many variations on these themes. As it happens, none of the others works much better than ibm clustering, so we describe them only very briefly. One is *indexpredict*, combining index and predictive clustering:

$$P_{\texttt{indexpredict}}(w_i|w_{i-2}w_{i-1}) = P(W_i|w_{i-2}W_{i-2}w_{i-1}W_{i-1}) \times P(w_i|w_{i-2}W_{i-2}w_{i-1}W_{i-1}W_i).$$

Another is *combinepredict*, interpolating a normal trigram with a predictive clustered trigram:

$$P_{\texttt{combinepredict}}(w_i|w_{i-2}w_{i-1}) = \lambda P(w_i|w_{i-1}w_{i-2}) + (1-\lambda)P(W_i|w_{i-2}w_{i-1}) \times P(w_i|w_{i-2}w_{i-1}W_i).$$

Finally, we wanted to get some sort of upper bound on how much could be gained by clustering, so we tried combining all these clustering techniques together, to get what we call *allcombinenotop*, which is an interpolation of a normal trigram, a fullibm-like model, an index model, a predictive model, a true fullibm model, and an indexpredict model. A variation, *allcombine*, interpolates the predict-type models first at the cluster level, before interpolating with the word level models. Exact formulae are given in the extended version of this paper.

In Figure 5, we show a comparison of nine different clustering techniques, all using Kneser–Ney smoothing. The clusters were built separately for each training size. Notice that the value of clustering decreases with training data; at small data sizes, it is about 0.2 bits for the best clustered model; at the largest sizes, it is only about 0.1 bits. Since clustering is a technique for dealing with data sparseness, this is unsurprising. Next, notice that ibm clustering consistently works very well. Of all the other techniques we tried, only four others worked as well or better: fullibm clustering, which is a simple variation; allcombine and allcombinenotop, which interpolate in a fullibm; and fullibmpredict. Fullibmpredict works

very well—as much as 0.05 bits better than ibm clustering. However, it has a problem at the smallest training size, in which case it is worse. We believe that the clusters at the smallest training size are very poor, and that predictive style clustering gets into trouble when this happens, since it smoothes across words that may be unrelated, while ibm clustering interpolates in a normal trigram model, making it more robust to poor clusters. All of the models that use predict clustering and do not interpolate an unclustered trigram are actually worse than the baseline at the smallest training size.

In the extended version of this paper, we also show results compared to a Katz smoothed model. The results are similar, with some interesting exceptions: in particular, indexpredict works well for the Kneser–Ney smoothed model, but very poorly for the Katz smoothed model. This shows that smoothing can have a significant effect on other techniques, such as clustering. The other result is that across all nine clustering techniques, at every size, the Kneser–Ney version always outperforms the Katz smoothed version. In fact, the Kneser–Ney smoothed version also outperformed both interpolated and backoff absolute discounting versions of each technique at every size.

There are two other ways to perform clustering, which we will not explore here. First, one can cluster groups of words—complete contexts—instead of individual words. That is, to change notation for a moment, instead of computing

$$P(w|\textit{word-cluster}(w_{i-2})\textit{word-cluster}(w_{i-1}))$$

one could compute

$$P(w|\textit{context-cluster}(w_{i-2}w_{i-1})).$$

For instance, in a trigram model, one could cluster contexts like *"New York"* and *"Los Angeles"* as *"CITY"*, and *"on Wednesday"* and *"late tomorrow"* as *"TIME"*. There are many difficult issues to solve for this kind of clustering. Another kind of conditional clustering one could do is to empirically determine, for a given context, the best combination of clusters and words to use, the *varigram* approach (Blasig, 1999).

5.2. Finding clusters

A large amount of previous research has focused on how best to find the clusters (Brown *et al.*, 1992; Kneser & Ney, 1993; Pereira, Tishby & Lee, 1993; Ueberla, 1995; Bellegarda, Butzberger, Chow, Coccaro & Naik, 1996; Yamamoto & Sagisaka, 1999). Most previous research has found only small differences between different techniques for finding clusters. One result, however, is that automatically derived clusters outperform part-of-speech tags (Niesler, Whittaker & Woodland, 1998), at least when there is enough training data (Ney *et al.*, 1994). We did not explore different techniques for finding clusters, but simply picked one we thought would be good, based on previous research.

There is no need for the clusters used for different positions to be the same. In particular, for a model like ibm clustering, with $P(w_i|W_i) \times P(W_i|W_{i-2}W_{i-1})$, we will call the W_i cluster a *predictive* cluster, and the clusters for W_{i-1} and W_{i-2} *conditional* clusters. The predictive and conditional clusters can be different (Yamamoto & Sagisaka, 1999). For instance, consider a pair of words like *a* and *an*. In general, *a* and *an* can follow the same words, and so, for predictive clustering, belong in the same cluster. But, there are very few words that can follow both *a* and *an*—so for conditional clustering, they belong in different clusters. We have also found in pilot experiments that the optimal number of clusters used for predictive and conditional clustering are different; in this paper, we always optimize both the number of conditional and predictive clusters separately, and reoptimize for each technique at each

training data size. This is a particularly time consuming experiment, since each time the number of clusters is changed, the models must be rebuilt from scratch. We always try numbers of clusters that are powers of 2, e.g. 1, 2, 4, etc., since this allows us to try a wide range of numbers of clusters, while never being more than a factor of 2 away from the optimal number. Examining charts of performance on heldout data, this seems to produce numbers of clusters that are close enough to optimal.

The clusters are found automatically using a tool that attempts to minimize perplexity. In particular, for the conditional clusters we try to minimize the perplexity of training data for a bigram of the form $P(w_i|W_{i-1})$, which is equivalent to maximizing

$$\prod_{i=1}^{N} P(w_i|W_{i-1}).$$

For the predictive clusters, we try to minimize the perplexity of training data of $P(W_i|w_{i-1}) \times P(w_i|W_i)$. In the full version of this paper, we show that this is equivalent to maximizing the perplexity of $P(w_{i-1}|W_i)$,[3] which is very convenient, since it means we can use the same tool to get both conditional and predictive clusters, simply switching the order of the input pairs. We give more details about the clustering algorithm used in Section 9.

6. Caching

If a speaker uses a word, it is likely that he will use the same word again in the near future. This observation is the basis of caching (Kuhn, 1988; Kupiec, 1989; Kuhn & De Mori, 1990; Kuhn &De Mori, 1992; Jelinek, Merialdo, Roukos & Strauss, 1991). In particular, in a unigram cache, we form a unigram model from the most recently spoken words (all those in the same article if article markers are available, or a fixed number of previous words if not.) This unigram cache can then be linearly interpolated with a conventional n-gram.

Another type of cache model depends on the context. For instance, we could form a smoothed bigram or trigram from the previous words, and interpolate this with the standard trigram. In particular, we use

$$P_{\texttt{trigram-cache}}(w|w_1 \ldots w_{i-2}w_{i-1}) =$$
$$\lambda P_{\texttt{Smooth}}(w|w_{i-2}w_{i-1}) + (1-\lambda)P_{\texttt{tricache}}(w|w_1 \ldots w_{i-1})$$

where $P_{\texttt{tricache}}(w|w_1 \ldots w_{i-1})$ is a simple interpolated trigram model, using counts from the preceding words in the same document.

Yet another technique is to use conditional caching. In this technique, we weight the trigram cache differently depending on whether or not we have previously seen the context or not. The exact formulae are given in the extended version of the paper, but basically, we only interpolate the trigram cache $P_{\texttt{tricache}}(w|w_{i-2}w_{i-1})$ if we have at least seen w_{i-1} in the cache. Alternatively, we can interpolate a unigram, bigram, and trigram cache, and use the bigram cache only if we have seen w_{i-1} and the trigram only if we have seen the pair $w_{i-2}w_{i-1}$. In addition, the actual formulae we used allowed the caches to have a variable weight, depending on the amount of context, but the optimal parameters found set the variable factor very near zero.

Figure 6 gives results of running each of these five cache models. All were interpolated with a Kneser–Ney smoothed trigram. Each of the n-gram cache models was smoothed using simple interpolation, for technical reasons. As can be seen, caching is potentially one of the

[3] As suggested to us by Lillian Lee.

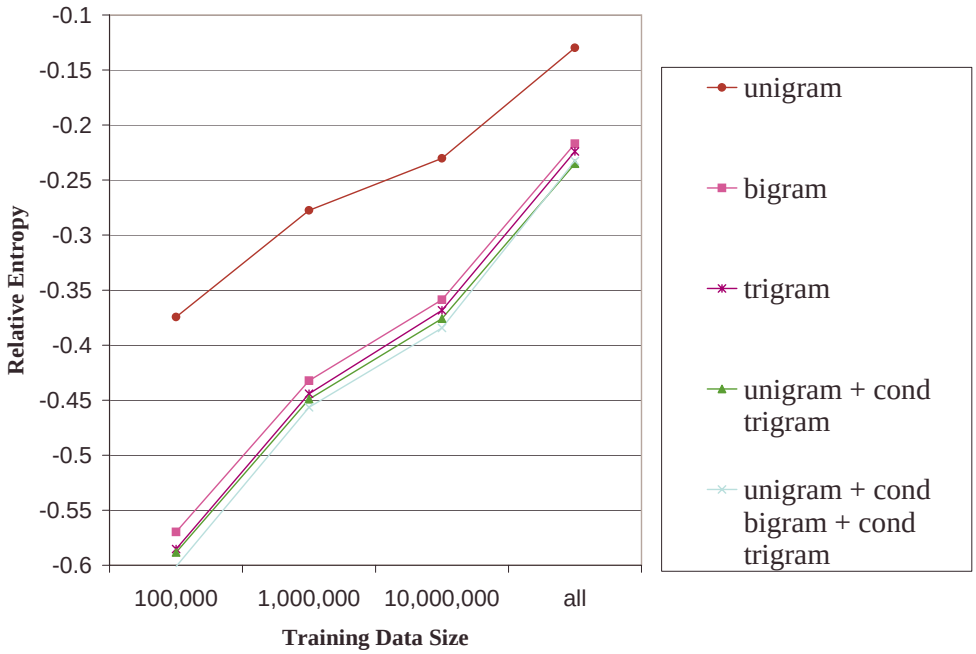

Figure 6. Five different cache models interpolated with trigram compared to trigram baseline.

most powerful techniques we can apply, leading to performance improvements of up to 0.6 bits on small data. Even on large data, the improvement is still substantial, up to 0.23 bits. On all data sizes, the *n*-gram caches perform substantially better than the unigram cache, but which version of the *n*-gram cache is used appears to make only a small difference.

It should be noted that all of these results assume that the previous words are known exactly. In a speech recognition system, however, many product scenarios do not include user correction. It is then possible for a cache to "lock-in" errors. For instance, if the user says "recognize speech" and the system hears "wreck a nice beach" then, later, when the user says "speech recognition" the system may hear "beach wreck ignition", since the probability of "beach" will be significantly raised. Thus, getting improvements from caching in a real product is potentially a much harder problem.

7. Sentence mixture models

Iyer and Ostendorf (1999) and Iyer *et al.* (1994) observed that within a corpus, there may be several different sentence types; these sentence types could be grouped by topic, or style, or some other criterion. No matter how they are grouped, by modeling each sentence type separately, improved performance can be achieved. For instance, in WSJ data, we might assume that there are three different sentence types: financial market sentences (with a great deal of numbers and stock names), business sentences (promotions, demotions, mergers), and general news stories. We can compute the probability of a sentence once for each sentence type, then take a weighted sum of the probabilities across sentence types. Because long-distance correlations within a sentence (lots of numbers, or lots of promotions) are captured by such a model, the overall model is better. Of course, in general, we do not know the

sentence type until we have heard the sentence. Therefore, instead, we treat the sentence type as a hidden variable.

Let s_j denote the condition that the sentence under consideration is a sentence of type j. Then the probability of the sentence, given that it is of type j can be written as

$$\prod_{i=1}^{N} P(w_i|w_{i-2}w_{i-1}s_j).$$

Sometimes, the global model (across all sentence types) will be better than any individual sentence type. Let s_0 be a special context that is always true:

$$P(w_i|w_{i-2}w_{i-1}s_0) = P(w_i|w_{i-2}w_{i-1}).$$

Let there be S different sentence types ($4 \leq S \leq 8$ is typical); let $\sigma_0 \ldots \sigma_S$ be sentence interpolation parameters optimized on heldout data subject to the constraint $\sum_{j=0}^{S} \sigma_j = 1$. The overall probability of a sentence $w_1 \ldots w_n$ is equal to

$$\sum_{j=0}^{S} \sigma_j \prod_{i=1}^{N} P(w_i|w_{i-2}w_{i-1}s_j). \tag{6}$$

Equation (6) can be read as saying that there is a hidden variable, the sentence type; the prior probability for each sentence type is σ_j. We compute the probability of a test sentence once for each sentence type, and then sum these probabilities according to the prior probability of that sentence type.

The probabilities $P(w_i|w_{i-2}w_{i-1}s_j)$ may suffer from data sparsity, so they are linearly interpolated with the global model $P(w_i|w_{i-2}w_{i-1})$, using interpolation weights optimized on heldout data.

Sentence types for the training data were found by using the same clustering program used for clustering words; in this case, we tried to minimize the sentence-cluster unigram perplexities. That is, let $s(i)$ represent the sentence type assigned to the sentence that word i is part of. (All words in a given sentence are assigned to the same sentence type.) We tried to put sentences into clusters in such a way that $\prod_{i=1}^{N} P(w_i|s(i))$ was maximized. This is a much simpler technique than that used by Iyer and Ostendorf (1999). We assume that their technique results in better models than ours.

We performed a fairly large number of experiments on sentence mixture models. We sought to study the relationship between training data size, n-gram order, and number of sentence types. We therefore ran a number of experiments using both trigrams and 5-grams, at our standard data sizes, varying the number of sentence types from one (a normal model without sentence mixtures) to 128. All experiments were done with Kneser–Ney smoothing. The results are shown in Figure 7. Note, however, that we do not trust results for 128 mixtures because there may not have been enough heldout data to correctly estimate the parameters; see the extended version of this paper for details.

The results are very interesting for a number of reasons. First, we suspected that sentence mixture models would be more useful on larger training data sizes, and indeed they are; with 100 000 words, the most improvement from a sentence mixture model is only about 0.1 bits, while with 284 000 000 words, it is nearly 0.3 bits. This bodes well for the future of sentence mixture models: as computers get faster and larger, training data sizes should also increase. Second, we had suspected that because both 5-grams and sentence mixture models attempt to model long distance dependencies, the improvement from their combination would be less than the sum of the individual improvements. As can be seen in Figure 7, for 100 000 and

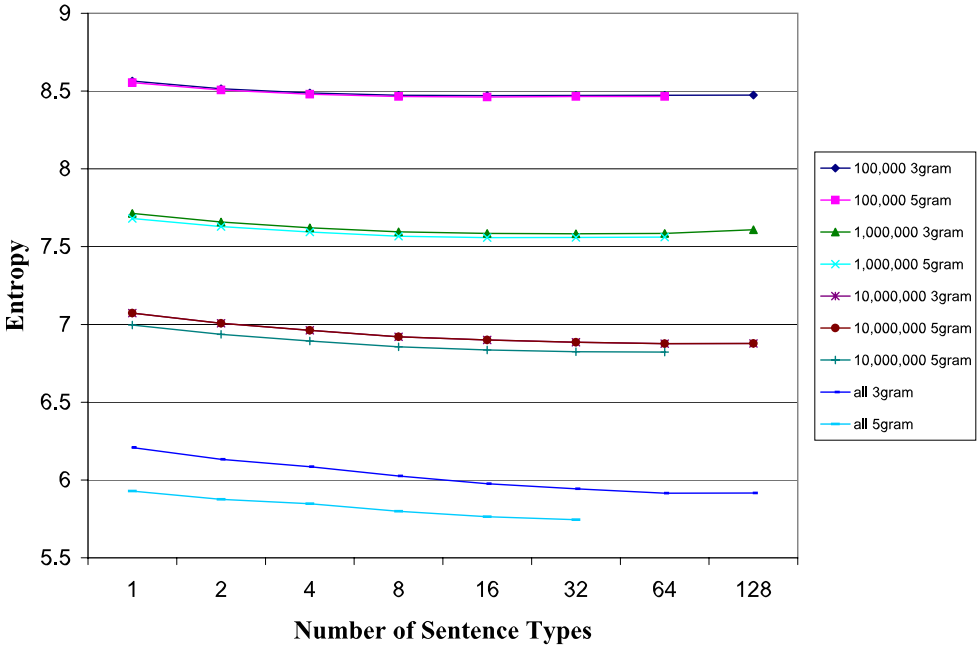

Figure 7. Number of sentence types vs. entropy.

1 000 000 words of training data, the difference between trigrams and 5-grams is very small anyway, so the question is not very important. For 10 000 000 words and all training data, there is some negative interaction. For instance, with four sentence types on all training data, the improvement is 0.12 bits for the trigram, and 0.08 bits for the 5-gram. Similarly, with 32 mixtures, the improvement is 0.27 on the trigram and 0.18 on the 5-gram. So, approximately one-third of the improvement seems to be correlated.

Iyer and Ostendorf (1999) reported experiments on both five-mixture components and eight components and found no significant difference, using 38 million words of training data. However, our more thorough investigation shows that indeed there is substantial room for improvement by using larger numbers of mixtures, especially when using more training data, and that this potential extends to at least 64 sentence types on our largest size. This is an important result, leading to almost twice the potential improvement of using only a small number of components.

We think this new result is one of the most interesting in our research. In particular, the techniques we used here were relatively simple, and many extensions to these techniques might lead to even larger improvements. For instance, rather than simply smoothing a sentence type with the global model, one could create sentence types and supertypes, and then smooth together the sentence type with its supertype and with the global model, all combined. This would alleviate the data sparsity effects seen with the largest numbers of mixtures.

Our sentence mixture model results are encouraging, but disappointing when compared to previous results. While Iyer and Ostendorf (1999) achieve about 19% perplexity reduction and about 3% word error rate reduction with five mixtures, on similar data we achieve only about 9% and (as we will show later) 1.3% reductions with four mixtures. In the extended version of this paper, we speculate on the potential causes of the differences. We suspect that

our clustering technique, much simpler than theirs, or differences in the exact composition of the data sets, account for the differences.

Sentence mixture models can also be useful when combining very different language model types. For instance, Jurafsky, Wooters, Segal, Fosler, Tajchman and Morgan (1995) uses a sentence mixture model to combine a stochastic context-free grammar (SCFG) model with a bigram model, resulting in marginally better results than either model used separately. The model of Jurafsky *et al*. is actually of the form:

$$\begin{aligned} &P(w_i|w_1 \dots w_{i-1}) \\ &\quad = P(\text{SCFG}|w_1 \dots w_{i-1}) \times P(w_i|w_1 \dots w_{i-1}, \text{SCFG}) \\ &\qquad + P(\text{bigram}|w_1 \dots w_{i-1}) \times P(w_i|w_1 \dots w_{i-1}, \text{bigram}) \end{aligned}$$

which turns out to be equivalent to a model in the form of Equation (6). Charniak (2001), as discussed in Section 10.4, uses a sentence level mixture model to combine a linguistic model with a trigram model, achieving significant perplexity reduction.

8. Combining techniques

In this section, we present additional results on combining techniques. While each of the techniques we have presented works well separately, we will show that some of them work together synergistically, and that some of them are partially redundant. For instance, we have shown that the improvement from Kneser–Ney modeling and 5-gram models together is larger than the improvement from either one by itself. Similarly, as we have already shown, the improvement from sentence mixture models when combined with 5-grams is only about two-thirds of the improvement of sentence mixture models by themselves, because both techniques increase data sparsity. In this section, we systematically study three issues: what effect does smoothing have on each technique; how much does each technique help when combined with all of the others; and how does each technique affect word error rate, separately and together.

There are many different ways to combine techniques. The most obvious way to combine techniques is to simply linearly interpolate them, but this is not likely to lead to the largest possible improvement. Instead, we try to combine concepts. To give a simple example, recall that a fullibmpredict clustered trigram is of the form:

$$\begin{aligned} &(\lambda P(W|w_{i-2}w_{i-1}) + (1-\lambda)P(W|W_{i-2}W_{i-1})) \times \\ &(\mu P(w|w_{i-2}w_{i-1}W) + (1-\mu)P(w|W_{i-2}W_{i-1}W). \end{aligned}$$

One could simply interpolate this clustered trigram with a normal 5-gram, but of course it makes much more sense to combine the concept of a 5-gram with the concept of fullibmpredict, using a clustered 5-gram:

$$\begin{aligned} &(\lambda P(W|w_{i-4}w_{i-3}w_{i-2}w_{i-1}) + (1-\lambda)P(W|W_{i-4}W_{i-3}W_{i-2}W_{i-1})) \times \\ &(\mu P(w|w_{i-4}w_{i-3}w_{i-2}w_{i-1}W) + (1-\mu)P(w|W_{i-4}W_{i-3}W_{i-2}W_{i-1}W). \end{aligned}$$

We will follow this idea of combining concepts, rather than simply interpolating throughout this section. This tends to result in good performance, but complex models.

Our overall combination technique is somewhat complicated. At the highest level, we use a sentence mixture model, in which we sum over sentence-specific models for each sentence type. Within a particular sentence mixture model, we combine different techniques with predictive clustering. That is, we combine sentence-specific, global, cache, and global skipping

models first to predict the cluster of the next word, and then again to predict the word itself given the cluster.

For each sentence type, we wish to linearly interpolate the sentence-specific 5-gram model with the global 5-gram model, the three skipping models, and the two cache models. Since we are using fullibmpredict clustering, we wish to do this based on both words and clusters. Let $\lambda_{1,j} \ldots \lambda_{12,j}$, $\mu_{1,j} \ldots \mu_{12,j}$ be interpolation parameters. Then, we define the following two very similar functions. First,[4]

$$\begin{aligned}
&\mathit{sencluster}_j(W, w_{i-4} \ldots w_{i-1}) \\
&\quad = \lambda_{1,j} P(W|w_{i-4}w_{i-3}w_{i-2}w_{i-1}s_j) + \lambda_{2,j} P(W|W_{i-4}W_{i-3}W_{i-2}W_{i-1}s_j) + \\
&\qquad \lambda_{3,j} P(W|w_{i-4}w_{i-3}w_{i-2}w_{i-1}) + \lambda_{4,j} P(W|W_{i-4}W_{i-3}W_{i-2}W_{i-1}) + \\
&\qquad \lambda_{5,j} P(W|w_{i-4}w_{i-3}w_{i-1}) + \lambda_{6,j} P(W|W_{i-4}W_{i-3}W_{i-1}) + \\
&\qquad \lambda_{7,j} P(W|w_{i-4}w_{i-2}w_{i-1}) + \lambda_{8,j} P(W|W_{i-4}W_{i-2}W_{i-1}) + \\
&\qquad \lambda_{9,j} P(W|w_{i-4}w_{i-3}w_{i-2}) + \lambda_{10,j} P(W|W_{i-4}W_{i-3}W_{i-2}) + \\
&\qquad \lambda_{11,j} P_{\texttt{unicache}}(W) + \lambda_{12,j} P_{\texttt{tricache}}(W|w_{i-2}w_{i-1}).
\end{aligned}$$

Next, we define the analogous function for predicting words given clusters:

$$\begin{aligned}
&\mathit{senword}_j(w, w_{i-4} \ldots w_{i-1}, W) \\
&\quad = \mu_{1,j} P(w|w_{i-4}w_{i-3}w_{i-2}w_{i-1}Ws_j) + \mu_{2,j} P(w|W_{i-4}W_{i-3}W_{i-2}W_{i-1}Ws_j) + \\
&\qquad \mu_{3,j} P(w|w_{i-4}w_{i-3}w_{i-2}w_{i-1}W) + \mu_{4,j} P(w|W_{i-4}W_{i-3}W_{i-2}W_{i-1}W) + \\
&\qquad \mu_{5,j} P(w|w_{i-4}w_{i-3}w_{i-1}W) + \mu_{6,j} P(w|W_{i-4}W_{i-3}W_{i-1}W) + \\
&\qquad \mu_{7,j} P(w|w_{i-4}w_{i-2}w_{i-1}W) + \mu_{8,j} P(w|W_{i-4}W_{i-2}W_{i-1}W) + \\
&\qquad \mu_{9,j} P(w|w_{i-4}w_{i-3}w_{i-2}W) + \mu_{10,j} P(w|W_{i-4}W_{i-3}W_{i-2}W) + \\
&\qquad \mu_{11,j} P_{\texttt{unicache}}(w|W) + \mu_{12,j} P_{\texttt{tricache}}(w|w_{i-2}w_{i-1}W).
\end{aligned}$$

Now, we can write out our probability model:

$$\begin{aligned}
&P_{\mathit{everything}}(w_1 \ldots w_N) \\
&\quad = \sum_{j=0}^{S} \sigma_j \prod_{i=1}^{N} \mathit{sencluster}_j(W_i, w_{i-4} \ldots w_{i-1}) \times \mathit{senword}_j(w_i, w_{i-4} \ldots w_{i-1}, W_i). \qquad (7)
\end{aligned}$$

Clearly, combining all of these techniques together is not easy, but as we will show, the effects of combination are very roughly additive, and the effort is worthwhile.

We performed several sets of experiments. In these experiments, when we perform caching, it is with a unigram cache and conditional trigram cache; when we use sentence mixture models, we use four mixtures; when we use trigram skipping, it is w_y and $wx\_$; and when we use 5-gram skipping it is vw_y interpolated with v_xy and $vwx\_$. Our word error rate experiments were done without punctuation, so, to aid comparisons, we perform additional entropy experiments in this section on "all-no-punc", which is the same as the "all" set, but without punctuation.

In the first set of experiments, we used each technique separately, and Katz smoothing. The results are shown in Figure 8. Next, we performed experiments with the same techniques, but

[4]This formula is actually an oversimplification because the values $\lambda_{11,j}$ and $\lambda_{12,j}$ depend on the amount of training data in a linear fashion, and if the context w_{i-1} does not occur in the cache, then the trigram cache is not used. In either case, the values of the λs have to be renormalized for each context so that they sum to 1.

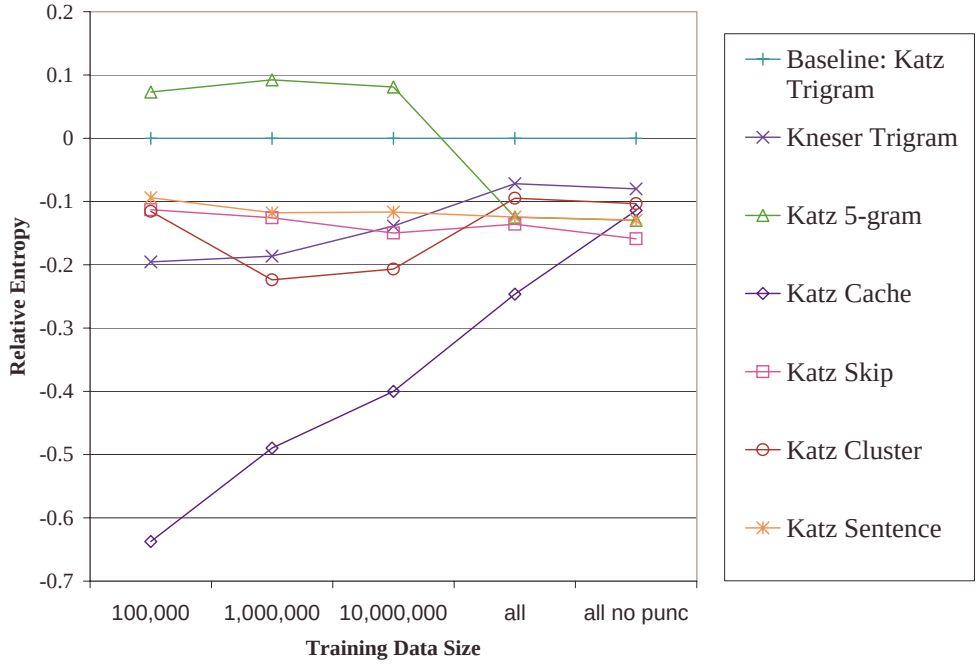

Figure 8. Relative entropy of each technique vs. Katz trigram baseline.

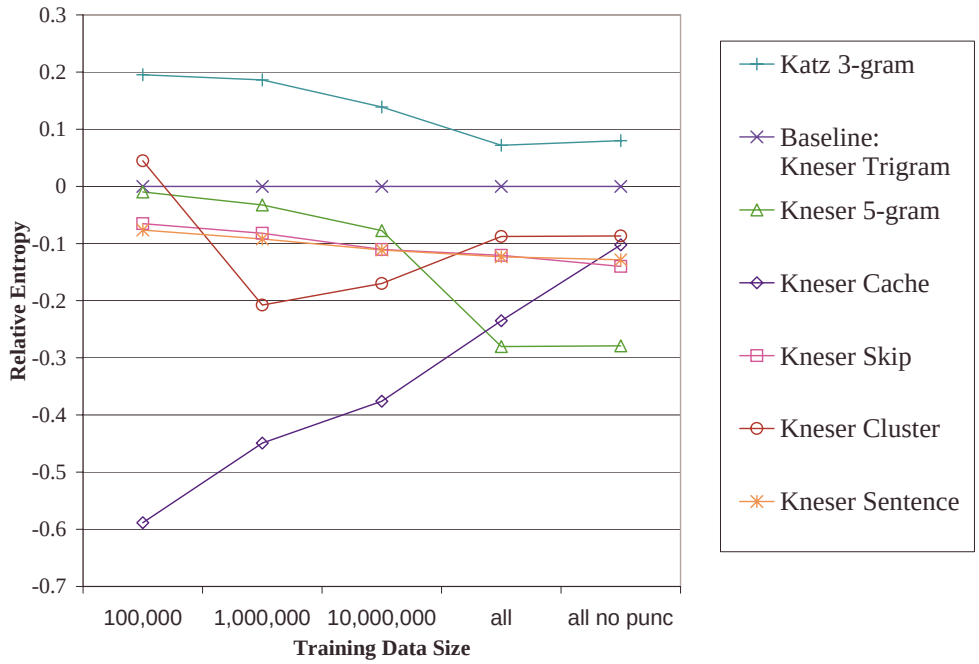

Figure 9. Relative entropy of each technique vs. Kneser–Ney trigram baseline.

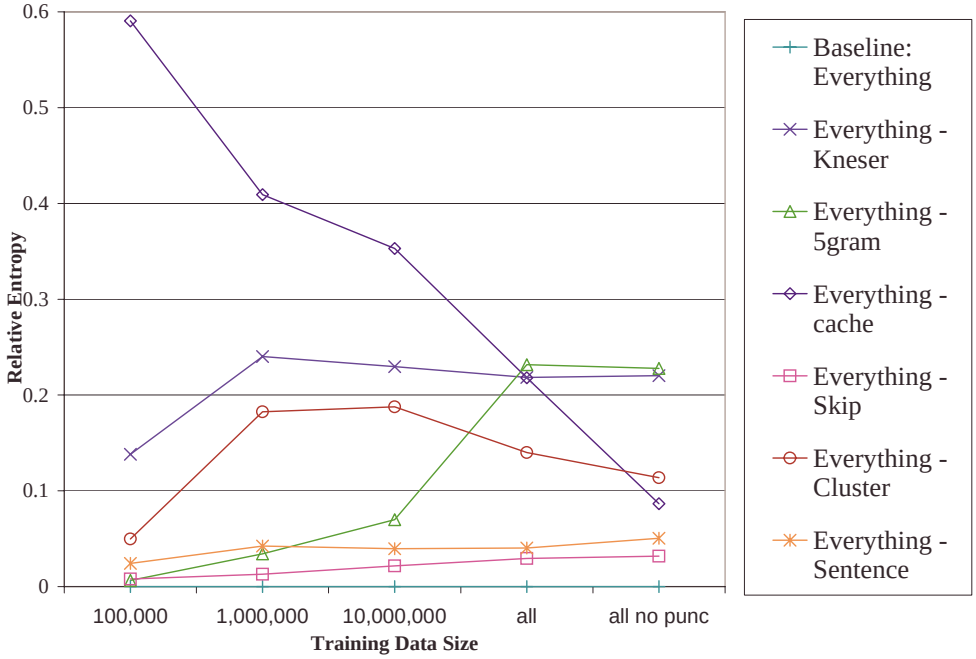

Figure 10. Relative entropy of *removing* each technique vs. all techniques combined baseline.

with Kneser–Ney smoothing; the results are shown in Figure 9. The results are similar for all techniques independent of smoothing, except 5-grams, where Kneser–Ney smoothing is clearly a large gain; in fact, without Kneser–Ney smoothing, 5-grams actually hurt at small and medium data sizes. This is a wonderful example of synergy, where the two techniques together help more than either one separately. Caching is the largest gain at small and medium data sizes, while, when combined with Kneser–Ney smoothing, 5-grams are the largest gain at large data sizes. Caching is still key at most data sizes, but the advantages of Kneser–Ney smoothing and clustering are clearer when they are combined with the other techniques.

In the next set of experiments, shown in Figure 10, we tried removing each technique from the combination of all techniques [Equation (7)]. The baseline is all techniques combined—"Everything", and then we show performance of, for instance, everything except Kneser–Ney, everything except 5-gram models, etc. In Figure 10 we show all techniques together vs. a Katz smoothed trigram. We add one additional point to this graph. With 100 000 words, our Everything model was at 0.91 bits below a normal Katz model, an excellent result, but we knew that the 100 000 word model was being hurt by the poor performance of fullibmpredict clustering at the smallest data size. We therefore interpolated in a normal 5-gram at the word level, a technique indicated as "Everything + normal 5-gram." This led to an entropy reduction of 1.0061—1 bit. This gain is clearly of no real-world value—most of the entropy gains at the small and medium sizes come from caching, and caching does not lead to substantial word error rate reductions. However, it does allow a nice title for the paper! Interpolating the normal 5-gram at larger sizes led to essentially no improvement.

We also performed word error rate experiments rescoring 100-best lists of WSJ94 dev and eval, about 600 utterances. The one-best error rate for the 100-best lists was 10.1%

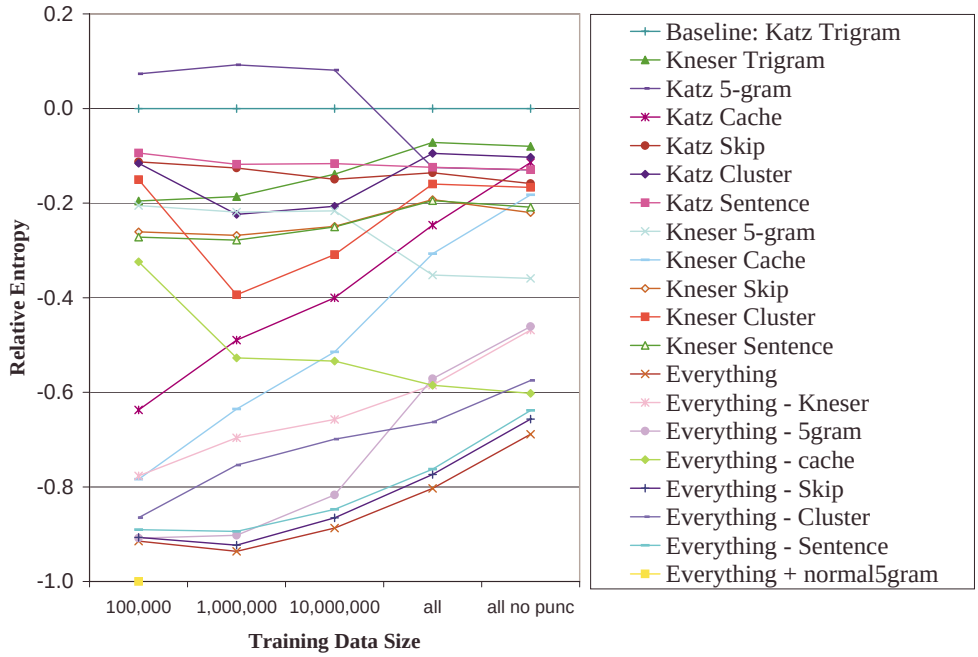

Figure 11. All techniques together vs. Katz trigram baseline.

(our recognizer's models were slightly worse than even the baseline used in rescoring) and the 100-best error rate (minimum possible from rescoring) was 5.2%. We were not able to obtain word error rate improvements by using caching (when the cache consisted of the output of the recognizer), and were actually hurt by the use of caching when the interpolation parameters were estimated on correct histories, rather than on recognized histories. Figure 12 shows word error rate improvement of each technique, either with Katz smoothing, Kneser–Ney smoothing, or removed from Everything, except caching. The most important single factor for word error rate was the use of Kneser–Ney smoothing, which leads to a small gain by itself, but also makes skipping, and 5-grams much more effective. Clustering also leads to significant gains. In every case except clustering, the Kneser–Ney smoothed model has lower word-error rate than the corresponding Katz smoothed model. The strange clustering result (the Katz entropy is higher) might be due to noise, or might be due to the fact that we optimized the number of clusters separately for the two systems, optimizing perplexity, perhaps leading to a number of clusters that was not optimal for word error rate reduction. Overall, we get an 8.9% word error rate reduction over a Katz smoothed baseline model. This is very good, although not as good as one might expect from our perplexity reductions. This is probably due to our rescoring of n-best lists rather than integrating our language model directly into the search, or rescoring large lattices.

9. Implementation notes

Actually implementing the model described here is not straightforward. We give here a few notes on the most significant implementation tricks, some of which are reasonably novel, and in the appendix of the extended paper give more details. First, we describe our parameter

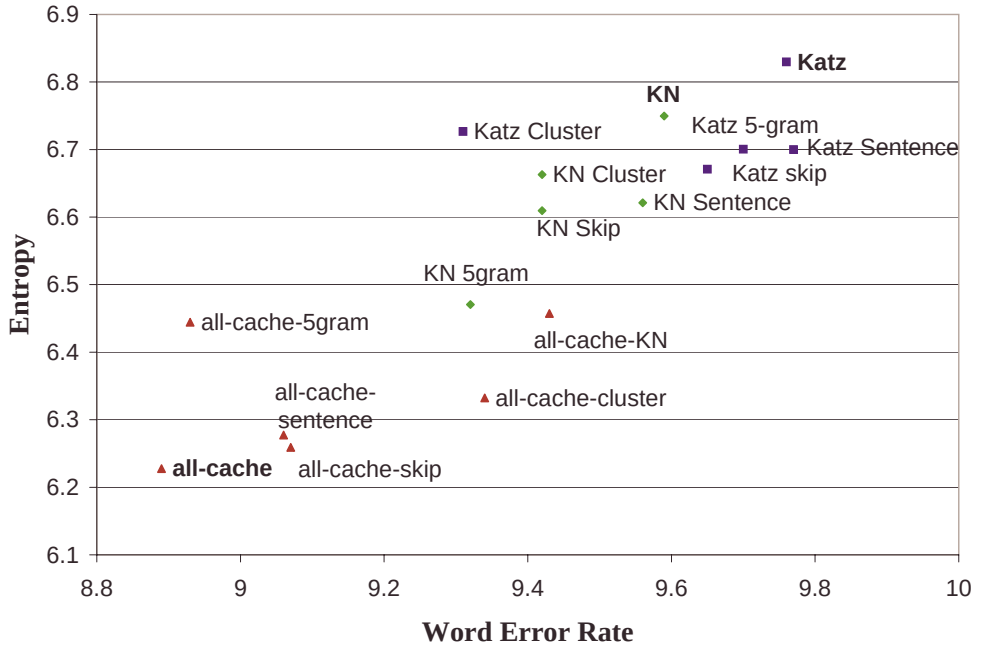

Figure 12. Word error rate vs. entropy.

search technique. Next, we discuss techniques we used to deal with the very large size of the models constructed. Then, we consider architectural strategies that made the research possible. Finally, we give a few hints on implementing our clustering methods.

The size of the models required for this research is very large. In particular, many of the techniques have a roughly multiplicative effect on data sizes: moving to 5-grams from trigrams results in at least a factor of two increase; fullibmpredict clustering results in nearly a factor of four increase; and the combination of sentence-mixture models and skipping leads to about another factor of four. The overall model size then, is, very roughly, 32 times the size of a standard trigram model. Building and using such a complex model would be impractical.

Instead, we use a simple trick. We first make a pass through the test data (either text, or *n*-best lists), and the heldout data (used for parameter optimization), and determine the complete set of values we will need for our experiments. Then, we go through the training data, and record only these values. This drastically reduces the amount of memory required to run our experiments, reducing it to a manageable 1.5 gigabytes approximately. Another trick we use is to divide the test data into pieces—the less test data there is, the fewer values we need to store. The appendix of the extended paper describes some ways that we verified that this "cheating" resulted in the same results that a non-cheating model would have achieved.

Careful design of the system was also necessary. In particular, we used a concept of a "model", an abstract object, with a set of parameters, that could return the probability of a word or class given a history. We created models that could compose other models, by interpolating them at the word level, the class level, or the sentence level, or even by multiplying them together as done in predictive clustering. This allowed us to compose primitive models that implemented caching, various smoothing techniques, etc., in a large variety of ways.

Both our smoothing techniques and interpolation require the optimization of free parameters. In some cases, these free parameters can be estimated from the training data by leaving-one-out techniques, but better results are obtained by using a Powell search of the parameters, optimizing the perplexity of heldout data (Chen & Goodman, 1999), and that is the technique used here. This allowed us to optimize all parameters jointly, rather than, say, optimizing one model, then another, then their interpolation parameters, as is typically done. It also made it relatively easy to, in essentially every experiment in this paper, find the optimal parameter settings for that model, rather than use suboptimal guesses or results from related models.[5]

Although all smoothing algorithms were reimplemented for this research (reusing only a small amount of code), the details closely follow Chen and Goodman (1999). This includes our use of additive smoothing of the unigram distribution for both Katz smoothing and Kneser–Ney smoothing. That is, we found a constant b which was added to all unigram counts; this leads to better performance in small training-data situations, and allowed us to compare perplexities across different training sizes, since no unigram received zero counts, meaning zero probabilities were never returned.

There is no shortage of techniques for generating clusters, and there appears to be little evidence that different techniques that optimize the same criterion result in a significantly different quality of clusters. We note, however, that different algorithms may require significantly different amounts of run time. In particular, agglomerative clustering algorithms may require significantly more time than top-down, splitting algorithms. Within top-down, splitting algorithms, additional tricks can be used, including the techniques of Buckshot (Cutting, Karger, Pedersen & Tukey, 1992). We also use computational tricks adapted from Brown *et al*. (1992). Many more details about the clustering techniques used are given in the appendix of the extended version of this paper.

10. Other techniques

In this section, we briefly discuss several other techniques that have received recent interest for language modeling; we have performed a few experiments with some of these techniques. These techniques include maximum entropy models, latent semantic analysis, parsing based models, and neural network based models. Rosenfeld (2000) gives a much broader, different perspective on the field, as well as additional references for the techniques discussed in this paper.

10.1. Maximum entropy models

Maximum entropy models (Darroch & Ratcliff, 1972) have received a fair amount of attention since their introduction for language modeling by Rosenfeld (1994), who achieved excellent results—up to 39% perplexity reductions. Maximum entropy models allow arbitrary constraints to be combined. For instance, rather than simply linearly interpolating the components of a skipping model, the components of a clustering model, and a baseline n-gram model, each of these models can be expressed as a constraint on how often we expect words to occur in some context. We can then build a model that satisfies all of these constraints, while still being as smooth as possible. This is a highly seductive quality. In addition, recent improvements in maximum entropy smoothing (Chen & Rosenfeld, 1999*b*) and

[5]The only exception was that for Katz smoothed "everything" models we estimated the number of clusters from simple Katz clustered models; large Katz smoothed models are extremely time consuming because of the need to find the αs after each potential parameter change.

maximum entropy training speedups (Goodman, 2001*b*), are sources for optimism. Unfortunately, typically maximum entropy models are compared only to trigram models, rather than to comparable *n*-gram models that combine the same information using simple interpolation. Our own pilot experiments comparing maximum entropy models to similar *n*-gram models were negative.

The most substantial gain in maximum entropy models comes from word triggers. In these models, a word such as "school" increases its own probability, as well as the probability of similar words, such as "teacher." Rosenfeld (1994) gets approximately a 25% perplexity reduction by using word triggers, although the gain is reduced to perhaps 7–15% when combining with a model that already contains a cache. Tillmann and Ney (1996) achieve about a 7% perplexity reduction when combining with a model that already has a cache, and Zhang, Black, Finch and Sagisaka (2000) reports an 11% reduction.

A recent variation of the maximum entropy approach is the whole sentence maximum entropy approach (Rosenfeld, Chen & Zhu, 2001). In this variation, the probability of whole sentences is predicted, instead of the probabilities of individual words. This allows features of the entire sentence to be used, e.g. coherence (Cai, Rosenfeld & Wasserman, 2000) or parsability, rather than word level features. However, training whole sentence maximum entropy models is particularly complicated (Chen & Rosenfeld, 1999*a*), requiring sampling methods such as Monte Carlo Markov Chain techniques, and we personally do not think that there are many important features of a sentence that cannot be rephrased as features of individual words.

10.2. Latent semantic analysis

Bellegarda (2000) shows that techniques based on Latent Semantic Analysis (LSA) are very promising. LSA is similar to Principle Components Analysis (PCA) and other dimensionality reduction techniques, and seems to be a good way to reduce the data sparsity that plagues language modeling. The technique leads to significant perplexity reductions (about 20%) and word error rate reductions (about 9% relative) when compared to a Katz trigram on 42 million words. It would be interesting to formally compare these results to conventional caching results, which exploit similar long term information. Bellegarda gets additional improvement over these results by using clustering techniques based on LSA; the perplexity reductions appear similar to the perplexity reductions from conventional IBM-style clustering techniques.

10.3. Neural networks

There has been some interesting recent work on using Neural Networks for language modeling, by Bengio, Ducharme and Vincent (2000). In order to deal with data sparsity, they first map each word to a vector of 30 continuous features, and then the probability of various outputs is learned as a function of these continuous features. The mapping is learned by backpropagation in the same way as the other weights in the network. The best result is about a 25% perplexity reduction over a baseline deleted-interpolation style trigram. We performed experiments on the same data set as Bengio *et al.* We found that their techniques appeared to outperform clustered models somewhat (about 5% lower perplexity) and we think they have a fair amount of potential. It remains to be seen how similar those techniques are to normal clustering techniques.

10.4. Structured language models

One of the most interesting and exciting new areas of language modeling research has been structured language models (SLMs). The first successful structured language model was the work of Chelba and Jelinek (1998), and other more recent models have been even more successful (Charniak, 2001). The basic idea behind structured language models is that a properly phrased statistical parser can be thought of as a generative model of language. Furthermore, statistical parsers can often take into account longer distance dependencies, such as between a subject and its direct or indirect objects. These dependencies are likely to be more useful than the previous two words, as captured by a trigram model. Chelba is able to achieve an 11% perplexity reduction over a baseline trigram model, while Charniak achieves an impressive 24% reduction. We hypothesized that much of the benefit of a structured language model might be redundant with other techniques, such as skipping or clustering. With the help of Chelba, we performed experiments on his model. It turned out to be hard to answer, or even rigorously ask, the question of how redundant one model was with another, but based on our experiments, in which we compared how much entropy reduction we got by combining models, vs. how much we might expect, approximately one-half of the model appears to be in common with other techniques, especially clustering. Details are given in the extended version of this paper.

11. Conclusion

11.1. Previous combinations

There has been relatively little previous research that attempted to combine more than two techniques, and even most of the previous research combining two techniques was not particularly systematic. Furthermore, one of the two techniques typically combined was a cache-based language model. Since the cache model is simply linearly interpolated with another model, there is not much room for interaction.

A few previous papers do merit mentioning. The most recent is that of Martin *et al.* (1999). They combined interpolated Kneser–Ney smoothing, classes, word-phrases, and skipping. Unfortunately, they do not compare to the same baseline we use, but instead compare to what they call interpolated linear discounting, a poor baseline. However, their improvement over Interpolated Kneser–Ney is also given; they achieve about 14% perplexity reduction over this baseline, vs. our 34% over the same baseline. Their improvement from clustering is comparable to ours, as is their improvement from skipping models; their improvement from word-phrases, which we do not use, is small (about 3%); thus, the difference in results is due mainly to our implementation of additional techniques: caching, 5-grams, and sentence-mixture models. Their word error rate reduction over Interpolated Kneser–Ney is 6%, while ours is 7.3%. We assume that the reason our word error rate reduction is not proportional to our perplexity reduction is twofold. First, 4% of our perplexity reduction came from caching, which we did not use in our word error rate results. Second, they were able to integrate their simpler model directly into a recognizer, while we needed to rescore n-best lists, reducing the number of errors we could correct.

Another piece of work well worth mentioning is that of Rosenfeld (1994). In that work, a large number of techniques are combined, using the maximum entropy framework and interpolation. Many of the techniques are tested at multiple training data sizes. The best system interpolates a Katz-smoothed trigram with a cache and a maximum entropy system. The maximum entropy system incorporates simple skipping techniques and triggering. The

best system has perplexity reductions of 32–39% on data similar to ours. Rosenfeld gets approximately 25% reduction from word triggers alone (p. 45), a technique we do not use. Overall, Rosenfeld's results are excellent, and would quite possibly exceed ours if more modern techniques had been used, such as Kneser–Ney smoothing the trigram model (which is interpolated in), using smaller cutoffs made possible by faster machines and newer training techniques, or smoothing the maximum entropy model with newer techniques. Rosenfeld achieves about a 10% word error rate reduction.

There is surprisingly little other work combining more than two techniques. The only other noteworthy research we are aware of is that of Weng, Stolcke and Sankar (1997), who performed experiments combining multiple corpora, 4-grams, and a class-based approach similar to sentence-mixture models. Combining all of these techniques leads to an 18% perplexity reduction from a Hub4-only language model. This model was trained and tested on a different text genre than our models, and so no comparison to our work can be made.

11.2. Discussion

We believe our results—a 50% perplexity reduction on a very small data set, and a 41% reduction on a large one (38% for data without punctuation)—are the best ever reported for language modeling, as measured by improvement from a fair baseline, a Katz smoothed trigram model with no count cutoffs. We also systematically explored smoothing, higher order *n*-grams, skipping, sentence mixture models, caching, and clustering.

Our most important result is perhaps the superiority of Interpolated Kneser–Ney smoothing in every situation we have examined. We previously showed (Chen & Goodman, 1998) that Kneser–Ney smoothing is always the best technique across training data sizes, corpora types, and *n*-gram order. We have now shown that it is also the best across clustering techniques, and that it is one of the most important factors in building a high performance language model, especially one using 5-grams.

We have carefully examined higher-order *n*-grams, showing that performance improvements plateau at about the 5-gram level, and we have given the first results at the 20-gram level, showing that there is no improvement to be gained past 7-grams.

We have systematically examined skipping techniques. We examined trigram-like models, and found that using pairs through to the 5-gram level captures almost all of the benefit. We also performed experiments on 5-gram skipping models, finding a combination of three contexts that captures most of the benefit.

We carefully explored sentence mixture models, showing that much more improvement can be had than was previously expected by increasing the number of mixtures. In our experiments, increasing the number of sentence types to 64 allows nearly twice the improvement over a small number of types.

Our caching results show that caching is by far the most useful technique for perplexity reduction at small and medium training data sizes. They also show that a trigram cache can lead to almost twice the entropy reduction of a unigram cache.

Next, we systematically explored clustering, trying nine different techniques, finding a new clustering technique, fullibmpredict, that is slightly better than standard ibm clustering, and examining the limits of improvements from clustering.

Our word error rate reduction of 8.9% from combining all techniques except caching is also very good.

Finally, we put all the techniques together, leading to a 38–50% reduction in perplexity, depending on training data size. The results compare favorably to other recently reported

combination results (Martin *et al.*, 1999), where, essentially using a subset of these techniques from a comparable baseline (absolute discounting), the perplexity reduction is half as much. Our results show that smoothing can be the most important factor in language modeling, and its interaction with other techniques cannot be ignored.

In some ways, our results are a bit discouraging. The overall model we built is so complex, slow and large that it would be completely impractical for a product system. Despite this size and complexity, our word error rate improvements are modest. To us, this implies that the potential for practical benefit to speech recognizers from language model research is limited. On the other hand, language modeling is useful for many fields beyond speech recognition, and is an interesting test bed for machine learning techniques in general.

Furthermore, parts of our results are very encouraging. First, they show that progress in language modeling continues to be made. For instance, one important technique in our system, sentence mixture models, is only a few years old, and, as we showed, its potential has only been partially tapped. Similarly, the combination of so many techniques is also novel. Furthermore, our results show that the improvements from these different techniques are roughly additive: one might expect an improvement of 0.9 bits for the largest training size based on simply adding up the results of Figure 9, and instead the total is about 0.8 bits—very similar. This means that further incremental improvements may also lead to improvements in the best models, rather than simply overlapping or being redundant.

As we noted in Section 10, there are many other promising language modeling techniques currently being pursued, such as maximum entropy models, neural networks, latent semantic analysis, and structured language models. Figuring out how to combine these techniques with the ones we have already implemented should lead to even larger gains, but also yet more complex models.

I would like to thank the entire Microsoft Speech.Net Research Team for their help, especially Milind Mahajan, X. D. Huang, Alex Acero, Ciprian Chelba, as well as Jianfeng Gao. I would also like to thank the anonymous reviewers, Sarah Schwarm, Roland Kuhn, Eric Brill, Hisami Suzuki, and Shaojun Wang for their comments on drafts of this paper. I would like to especially thank Stanley Chen for useful discussions; in addition, small amounts of text and code used for this implementation and paper irrespectively were originally coauthored with Stanley Chen.

References

Bellegarda, J. R., Butzberger, J. W., Chow, Y.-L., Coccaro, N. B. & Naik, D. (1996). A novel word clustering algorithm based on latent semantic analysis. *International Conference on Acoustics, Speech and Signal Processing*, volume 1, pp. 172–175.

Bellegarda, J. R. (2000). Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, **88**, 1279–1296.

Bengio, Y., Ducharme, R. & Vincent, P. (2000). A neural probabilistic language model. Technical Report 1178, Département d'informatique et recherche opérationnelle, Université de Montréal.

Blasig, R. (1999). Combination of words and word categories in varigram histories. In *International Conference on Acoustics, Speech and Signal Processing*, volume 1, pp. 529–532.

Brown, P. F., Cocke, J., DellaPietra, S. A., DellaPietra, V. J., Jelinek, F., Lafferty, J. D., Mercer, R. L. & Roossin, P. S. (1990). A statistical approach to machine translation. *Computational Linguistics*, **16**, 79–85.

Brown, P. F., DellaPietra, V. J., deSouza, P. V., Lai, J. C. & Mercer, R. L. (1992). Class-based *n*-gram models of natural language. *Computational Linguistics*, **18**, 467–479.

Cai, C., Rosenfeld, R. & Wasserman, L. (May 2000). Exponential language models, logistic regression, and semantic coherence. *Proceedings of NIST/DARPA Speech Transcription Workshop*.

Charniak, E. (2001). Immediate-head parsing for language models. In *ACL-01*. pp. 116–123.

Chelba, C. & Jelinek, F. (1998). Exploiting syntactic structure for language modeling. *Proceedings of the 36th Annual Meeting of the ACL*, pp. 225–231.

Chen, S. & Rosenfeld, R. (1999*a*). Efficient sampling and feature selection in whole sentence maximum entropy language models. *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, **1**, pp. 549–552.

Chen, S. F. & Goodman, J. (October 1999). An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, **13**, 359–394.

Chen, S. F. & Goodman, J. T. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University, 1998. Available from `http://www.research.microsoft.com/~joshuago`.

Chen, S. F. & Rosenfeld, R. (1999*b*). A gaussian prior for smoothing maximum entropy models. Technical Report CMU-CS-99-108, Computer Science Department, Carnegie Mellon University.

Church, K. (1988). A stochastic parts program and noun phrase parser for unrestricted text. *Proceedings of the Second Conference on Applied Natural Language Processing*, pp. 136–143.

Cutting, D. R., Karger, D. R., Pedersen, J. R. & Tukey, J. W. (1992). Scatter/gather: A cluster-based approach to browsing large document collections. *SIGIR 92*, pp. 318–329.

Darroch, J. N. & Ratcliff, D. (1972). Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, **43**, 1470–1480.

Dupont, P. & Rosenfeld, R. (1997). Lattice based language models. Technical Report CMU-CS-97-173, School of Computer Science, Carnegie Mellon University. Pittsburgh, PA.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, **40**, 237–264.

Goodman, J. (2001*a*). A bit of progress in language modeling, extended version. Technical Report, Microsoft Research. Draft available from `http://www.research.microsoft.com/~joshuago/longcombine.ps`.

Goodman, J. (2001*b*). Classes for fast maximum entropy training. *International Conference on Acoustics, Speech and Signal Processing*.

Goodman, J. & Gao, J. (2000). Language model size reduction by pruning and clustering. *International Conference on Spoken Language Processing*, **3**, pp. 110–113.

Huang, X., Alleva, F., Hon, H.-W., Hwang, M.-Y., Lee, K.-F. & Rosenfeld, R. (1993). The SPHINX-II speech recognition system: An overview. *Computer, Speech, and Language*, **2**, 137–148.

Hull, J. (1992). Combining syntactic knowledge and visual text recognition: A hidden Markov model for part of speech tagging in a word recognition algorithm. *AAAI Symposium: Probabilistic Approaches to Natural Language*, pp. 77–83.

Iyer, R. & Ostendorf, M. (1999). Modeling long distance dependence in language: Topic mixtures versus dynamic cache models. *IEEE Transactions on Acoustics, Speech and Audio Processing*, **7**, 30–39.

Iyer, R., Ostendorf, M. & Robin, R. J. (1994). Language modeling with sentence-level mixtures. *DARPA-HLT*, pp. 82–86.

Jelinek, F., Merialdo, B., Roukos, S. & Strauss, M. (1991). A dynamic lm for speech recognition. *Proceedings of ARPA Workshop on Speech and Natural Language*, pp. 293–295.

Jelinek, F. & Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. *Proceedings of the Workshop on Pattern Recognition in Practice*, pp. 381–397. North-Holland. Amsterdam, The Netherlands.

Jurafsky, D., Wooters, C., Segal, J., Fosler, E., Tajchman, G. & Morgan, N. (1995). Using a stochastic context-free grammar as a language model for speech recognition. *International Conference on Acoustics, Speech and Signal Processing*, pp. 189–192.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the langauge model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, **ASSP-35**, 400–401.

Kernighan, M. D., Church, K. W. & Gale, W. A. (1990). A spelling correction program based on a noisy channel model. *Proceedings of the Thirteenth International Conference on Computational Linguistics*, pp. 205–210.

Kneser, R. & Ney, H. (1993). Improved clustering techniques for class-based statistical language modeling. *Eurospeech 93*, volume 2, pp. 973–976.

Kuhn, R. (1988). Speech recognition and the frequency of recently used words: A modified Markov model for natural language. *12th International Conference on Computational Linguistics*, Budapest, Hungary, pp. 348–350.

Kuhn, R. & De Mori, R. (1990). A cache-based natural language model for speech reproduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**, 570–583.

Kuhn, R. & De Mori, R. (1992). Correction to a cache-based natural language model for speech reproduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**, 691–692.

Kupiec, J. (1989). Probabilistic models of short and long distance word dependencies. *Proceedings of ARPA Workshop on Speech and Natural Language*, pp. 290–295.

Martin, S., Hamacher, C., Liermann, J., Wessel, F. & Ney, H. (1999). Assessment of smoothing methods and complex stochastic language modeling. *6th European Conference on Speech Communication and Technology*, Budapest, Hungary, volume 5, pp. 1939–1942.

Ney, H., Essen, U. & Kneser, R. (1994). On structuring probabilistic dependences in stochastic language modeling. *Computer, Speech, and Language*, **8**, 1–38.

Niesler, T. R., Whittaker, E. W. D. & Woodland, P. C. (1998). Comparison of part-of-speech and automatically derived category-based language models for speech recognition. *International Conference on Acoustics, Speech and Signal Processing*, volume 1, pp. 177–180.

Pereira, F., Tishby, N. & Lee, L. (1993). Distributional clustering of english words. *Proceedings of the 31st Annual Meeting of the ACL*, pp. 183–190.

Press, W. H., Flannery, B. P., Teukolsky, S. A. & Vetterling, W. T. (1988). *Numerical Recipes in C*. Cambridge University Press, Cambridge.

Rosenfeld, R. (1994). *Adaptive statistical language modeling: a maximum entropy approach*. PhD Thesis, Carnegie Mellon University.

Rosenfeld, R. (2000). Two decades of statistical language modeling: Where do we go from here? *Proceedings of the IEEE*, **88**, 1270–1278.

Rosenfeld, R., Chen, S. F. & Zhu, X. (2001). Whole-sentence exponential language models: a vehicle for linguistic-statistical integration. *Computer Speech and Language*, to appear.

Saul, L. & Pereira, F. (1997). Aggregate and mixed-order Markov models for statistical language processing. *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pp. 81–89.

Siu, M. & Ostendorf, M. (2000). Variable n-grams and extensions for conversational speech language modeling. *IEEE Transactions on Speech and Audio Processing*, **8**, 63–75.

Srihari, R. & Baltus, C. (1992). Combining statistical and syntactic methods in recognizing handwritten sentences. *AAAI Symposium: Probabilistic Approaches to Natural Language*, pp. 121–127.

Stern, R. M. (1996). Specification of the 1995 ARPA hub 3 evaluation: Unlimited vocabulary NAB news baseline. *Proceedings of the DARPA Speech Recognition Workshop*, pp. 5–7.

Tillmann, C. & Ney, H. (1996). Statistical language modeling and word triggers. *Proceedings of the International Workshop "Speech and Computer" (SPECOM 96)*, pp. 22–27.

Ueberla, J. P. (1995). More efficient clustering of n-grams for statistical language modeling. *Eurospeech-95*, pp. 1257–1260.

Weng, F., Stolcke, A. & Sankar, A. (1997). Hub4 language modeling using domain interpolation and data clustering. *1997 DARPA Speech Recognition Workshop*, pp. 147–151.

Yamamoto, H. & Sagisaka, Y. (1999). Multi-class composite n-gram based on connection direction. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Phoenix, Arizona.

Zhang, R., Black, E., Finch, A. & Sagisaka, Y. (2000). Integrating detailed information into a language model. *International Conference on Acoustics, Speech and Signal Processing*, pp. 1595–1598.

(Received 30 April 2001 and accepted for publication 9 August 2001)