

Adaptive Statistical Language Modelling

by

Raymond Lau

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

at the Massachusetts Institute of Technology

May 1994

Copyright © 1994, Raymond Lau. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce
and to distribute copies of this thesis document in whole or in part,
and to grant others the right to do so.

Author _____
Department of Electrical Engineering and Computer Science
May, 1994

Certified by _____
Victor Zue
Thesis Supervisor

Certified by _____
Salim Roukos, IBM Thomas J. Watson Research Center
Company Supervisor

Accepted by _____
F. R. Morgenthaler
Chair, Department Committee on Graduate Students

Adaptive Statistical Language Modelling

by

Raymond Lau

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 1994 in partial fulfillment of the requirements for the
Degree of Master of Science in Electrical Engineering and Computer Science

ABSTRACT

The *trigram statistical language model* is remarkably successful when used in such applications as speech recognition. However, the trigram model is *static* in that it only considers the previous two words when making a prediction about a future word. The work presented here attempts to improve upon the trigram model by considering additional contextual and longer distance information. This is frequently referred to in the literature as *adaptive* statistical language modelling because the model is thought of as adapting to the longer term information. This work considers the creation of topic specific models, statistical evidence from the presence or absence of *triggers*, or related words, in the document history (*document triggers*) and in the current sentence (*in-sentence triggers*), and the incorporation of the *document cache*, which predicts the probability of a word by considering its frequency in the document history. An important result of this work is that the presence of *self-triggers*, that is, whether or not the word itself occurred in the document history, is an extremely important piece of information.

A *maximum entropy* (ME) approach will be used in many instances to incorporate information from different sources. Maximum entropy considers a model which maximizes entropy while satisfying the constraints presented by the information we wish to incorporate. The *generalized iterative scaling* (GIS) algorithm can be used to compute the maximum entropy solution. This work also considers various methods of smoothing the information in a maximum entropy model. An important result is that smoothing improves performance noticeably and that Good-Turing discounting is an effective method of smoothing.

Thesis Supervisor: Victor Zue

Title: Principal Research Scientist, Department of Electrical Engineering and Computer Science

Table of Contents

Acknowledgments	5
1.0 Introduction and Background	6
1.1 Language Models	6
1.2 Trigram Statistical Language Model.....	7
1.3 Measuring Language Model Performance: Perplexity	8
1.4 Beyond Trigrams: Adaptive Language Models	8
2.0 Experimental Preliminaries	11
2.1 The Wall Street Journal Corpus	11
2.2 Adaptation Within a Document.....	12
3.0 Topic Specific Language Models	13
3.1 Identifying a Suitable Set of Topics	13
3.1.1 Measuring the Closeness of Documents.....	14
3.1.2 Clustering Documents	15
3.2 Topic Specific Trigram Models.....	18
4.0 Maximum Entropy Models	21
4.1 Principle of Maximum Entropy	21
4.2 Maximum Entropy Solution and Constraint Functions	23
4.3 Maximum Likelihood Maximum Entropy	24
4.4 Generalized Iterative Scaling.....	26
4.4.1 Computational Issues Involved with GIS	27
4.4.2 Partial Updating of Constraints	29
5.0 Maximum Entropy Trigram Model	30
5.1 N-gram Constraints	30
5.2 Performance of ME N-gram Model	32
5.3 Improvement Through Smoothing	33
5.4 Fuzzy Maximum Entropy with Quadratic Penalty	33
5.5 Good-Turing Discounting	36
5.6 Results of Smoothing Experiments.....	37
6.0 Self-Triggers	39
6.1 Self-Trigger Constraints	39
6.2 Experimental Results	40
6.3 Analysis of Trained Parameters	41
7.0 Triggers	43
7.1 Selection of Triggers	43

Acknowledgments

First and foremost, I would like to thank Salim Roukos at IBM's Thomas J. Watson Research Center and Ronald Rosenfeld at Carnegie Mellon University. The work presented here was conducted while I was a visitor at TJ Watson during the summer and fall of 1993 and it was an extension of my previous work on trigger language models done at TJ Watson the preceding summer. Salim was my supervisor and mentor throughout my stays at TJ Watson. Roni Rosenfeld and I worked together on my earlier trigger language model exploration at IBM. Without Roni, many of the implementation issues related to maximum entropy would have not been resolved as efficiently. I would also like to thank Peter Brown and Robert Mercer presently at Renaissance Technology and Stephen Della Pietra and Vincent Della Pietra presently at TJ Watson. Peter, Bob, Stephen, Vincent and Salim all jointly conceived and developed the maximum entropy framework for building statistical models at TJ Watson. Finally, I cannot forget Victor Zue, my thesis supervisor, who took time out of his busy schedule to offer his valuable input.

Raymond Lau

MIT, Cambridge, MA

May, 1994

1.0 Introduction and Background

1.1 Language Models

Language models predict the probability of a word's occurrence given information about the context in which the word is expected to occur. Language models are useful in speech recognition, machine translation and natural language systems. For example, in a speech recognition system, we are given some acoustic evidence, a , and are asked to predict the spoken word, w , or a word sequence. A typical approach would be to choose w such that $Pr(w | a)$ is maximized over all possible w . $Pr(w | a)$ may be rewritten using Bayes' rule as $Pr(w|a) = \frac{Pr(a|w)Pr(w)}{Pr(a)}$. Here, $Pr(a)$ does not depend on the word w so to find the word w which maximizes this expression, we need to maximize $Pr(a | w) Pr(w)$ where $Pr(a | w)$ is the probability that we receive the acoustic evidence a given that the word w was spoken and $Pr(w)$ is the *a priori* probability that the word w occurs. The language model's task is to provide the probability distribution for $Pr(w)$. Language models may also be used to prune the list of possible words which need to be considered by the speech recognition system by eliminating words with low probability. As another example, consider a natural language system. Language models may be used to correct improperly formed sentences by scoring alternative sequences of words.

Statistical language models provide this probability distribution based upon statistical evidence gathered from a large sample of training text. For example, a simple zero order model may recognize that the word "the" occurs much more frequently in English training text than any other word. A statistical language model can take advantage of this information when making its predictions.

1.2 Trigram Statistical Language Model

A very successful statistical language model is the *trigram language model* (Bahl et al. [1]). In the trigram model, the probability of a word is predicted by constructing a *Markov model* whose state is based on the previous two words. The transition probabilities for the Markov model are estimated from a large sample of training text. For example, in a very simple trigram, the probability of a word, w , given the previous two words, w_{-1} and w_{-2} may be given by:

$$Pr(w|w_{-1}w_{-2}) = \frac{c(w_{-2}w_{-1}w)}{c(w_{-2}w_{-1})} \quad (\text{EQ 1})$$

In reality, we do not have enough training text to see a large sampling of trigrams for any given bigram key, w_{-1} and w_{-2} . If we assigned all trigrams $w_{-2}w_{-1}w$ which we have never seen in the training data the probability of zero, our model would have a very poor performance. Instead, we also include bigrams in our model to assist in cases where we have never seen the trigram. Because bigrams only involve two words instead of three, we will see a much larger sampling of bigrams than trigrams in the training data. Similarly, we also want to incorporate unigrams into our model. The question then is, how do we incorporate all three sources of information into one model? In the deleted interpolation trigram model, we linearly interpolate the three probability distributions. The probability of a word, w , given the previous two words, w_{-1} and w_{-2} is:

$$Pr(w|w_{-1}w_{-2}) = \lambda_0 + \lambda_1 \frac{c(w)}{N} + \lambda_2 \frac{c(w_{-1}w)}{c(w_{-1})} + \lambda_3 \frac{c(w_{-2}w_{-1}w)}{c(w_{-2}w_{-1})} \quad (\text{EQ 2})$$

Here, N is the number of words in the training corpus, the $c()$'s are the counts of occurrences in the training corpus. The λ 's are parameters optimized over some heldout text not seen during accumulation of the $c()$'s. The idea here is that the heldout text approximates unseen

test data, on which our model will ultimately be measured, so it should provide good estimates for the smoothing parameters. We will refer primarily to this deleted interpolation trigram model as simply the trigram model. The model given in (EQ 2) has a natural interpretation as a *hidden* Markov model, which has three hidden states corresponding to the unigram, bigram and trigram distributions for each context. The trigram model has been highly successful because it performs well and is simple to implement and train.

1.3 Measuring Language Model Performance: Perplexity

Ideally, we would like to measure a language model's performance by considering the decrease in error rate in the system in which the model is incorporated. However, this is often difficult to measure and is not invariant across different embodiments. A frequently used measure of a language model's performance is the *perplexity* (Bahl [1] et al., section IX) which is based on established information theoretic principles. Perplexity is defined as:

$$PP(T) = 2^{H(\tilde{P}(T), P(T))} \quad (\text{EQ 3})$$

where $H(\tilde{P}(T), P(T))$, the cross entropy of $\tilde{P}(T)$ (the observed distribution of the text T) with respect to $P(T)$ (the model's predicted distribution of the text) is defined as:

$$H(\tilde{P}(T), P(T)) = - \sum_{x \in T} \tilde{P}(x) \log P(x) \quad (\text{EQ 4})$$

Although lower perplexity does not necessarily result in lower error rate, there is a clear and definite correlation (Bahl et al. [1], section IX).

1.4 Beyond Trigrams: Adaptive Language Models

The trigram model is rather naive. Our natural intuition tells us the likelihood of a word's appearance depends on more than the previous two words. For example, we would reasonably expect that the context of the text has a large impact on the probability of a word's

appearance; if we were looking at a news story about the stock market, the probability of financial terms should be increased. Much of this information is not captured by the trigram model. Also, the trigram model is *static* in that it does not take into account the style or patterns of word usage in the source text. For example, in a speech recognition system, the speaker may use a very limited vocabulary, and yet the trigram model's estimates are based on a training corpus which may not match the speaker's more limited vocabulary. The trigram model is unable to adapt to the speaker's vocabulary and style, nor to topic information which is contained in a document.¹

An improvement upon trigram model was proposed in Kuhn and De Mori [15] and Jelinek et al. [13]. The *cache* model takes into account the relative dynamic frequencies of words in text seen so far and incorporates this information into the prediction by maintaining a cache of recently used words. This results in up to a 23% improvement in perplexity as reported in Jelinek et al. [13]. However, the manner in which the cache component was combined with the trigram component was a simplistic linear interpolation. In Della Pietra et al. [7], a more disciplined approach was pursued. The static trigram model was considered the initial probability distribution. The dynamic cache distribution was then imposed as a constraint on the probability distribution. The final probability distribution was the one which satisfied the cache constraint while minimizing the *Kullback-Leibler distance* or *discrimination of information* (Kullback [16]) from the initial trigram distribution. Improvements of up to 28% in perplexity were reported, albeit on a very small test set of 321 words.

1. Strictly speaking, models which depend only upon a limited amount of information such as the current document or the current session are not truly adaptive in that they can be considered a function of the current document or session. However, the term adaptive language modelling is frequently used to refer to such longer context models so we will continue to use the term here.

Another method in which the trigram model can be improved is to take *triggers*, or related words, into account. A trigger pair, $A \rightarrow B$, is a sequence such that the appearance of A in the current document history significantly affects the probability of seeing B . Triggers in the document history may help establish the context of the current document. For example, in the *Wall Street Journal corpus* the presence of the word “brokers” in a document makes the appearance of the word “stock” approximately five times higher than normal. The additional presence of the word “investors” increases the probability of “stock” by another 53%. In earlier work by the author and others, initial attempts to model triggers were made and improvements of between 11% and 25% have been reported (Lau [17], Lau et al. [18], Lau et al. [19], Rosenfeld [21], Rosenfeld and Huang [22]).

A third approach to adaptive language modelling involves trying to identify the topic or domain of the current document and selecting an appropriately trained model for that topic or domain as opposed to relying on a general model trained on all topics and domains. Such an attempt was made at IBM (Roukos [23]). To identify the domain of a test document, the unigram distribution of the words seen so far is compared to the average unigram distributions of the domains in the training data. The attempt was somewhat successful when it came to decoding test documents from a domain which existed in the training data. However, in that attempt, widely disparate domains such as newspaper text and insurance documents were considered and the domains of the documents used for training were known in advance so there was no need to cluster documents into topics or domains through statistical means. Whether a similar attempt will be successful in modelling topics within a narrower domain such as the text of the *Wall Street Journal* and in situations where unsupervised clustering of documents into topics must be used will be investigated in the present work.

2.0 Experimental Preliminaries

2.1 The Wall Street Journal Corpus

For the present work, the text from the Wall Street Journal years 1987 through 1989 will be used as both the training and test corpora. The Wall Street Journal task, perhaps because it is sponsored by ARPA, appears to be the one of primary interest within the adaptive language modelling community². Furthermore, the Wall Street Journal data made available by ARPA includes both sentence and document boundaries, necessary for our model of adaptation. We will consider a Wall Street Journal article to be the document we are dealing with. Articles tend to be on a narrow enough topic to benefit from an adaptive language model. We will use the data as processed by Doug Paul of MIT Lincoln Laboratories. We will use the non-verbalized punctuation version of the data along with the standard ARPA 20K vocabulary. By non-verbalized punctuation, we mean that text does not include punctuation, such as periods and commas, as words. The text itself consists of roughly 39 million words. For the purposes of our experiments, we will separate the text into three corpora: a training corpus of 37 million words, a held-out corpus of one million words and a test corpus of 870 thousand words. The selection of the corpora will be chronological, that is, the earliest 37 MW will be used for the training corpus, etc. This results in a tougher test corpus than a random selection of documents for the corpora because topics in the text evolves over time and the topics in the test corpus may be substantially different from those in the earlier training and held-out corpora. However, we feel that this is the more realistic since our speech recognition

2. As a matter of fact, in the recent November '93 evaluation conducted by ARPA, language model adaptation was one of the specialized areas evaluated.

system, or whatever other system in which the language model will be used, will be confronted with data from the present and future, not from the past.

2.2 Adaptation Within a Document

We will use the document as the basic unit we are adapting to. That is, in our test environment, we consider the beginning of a document a clean slate and our goal is to improve the language model performance as we see more and more text of a document. This is not the sole possibility. An alternative might be to assume that each testing session proceeds in chronological order so that temporal relationships between documents may be exploited. The reason we made this choice is because the amount of data needed to statistically learn relationships across document boundaries is much greater. While our 37 million word training corpus has over 82 thousand documents, we will typically run our experiments with a much smaller subset of the training corpus consisting of five million words and roughly ten thousand documents due to computational limitations. We do not feel that the smaller subset is sufficiently large to explore temporal relationships across document boundaries.

3.0 Topic Specific Language Models

One possibility for building an adaptive statistical language model is to divide the documents into different topics and to have different models for each specific topic. Especially in the Wall Street Journal data, we would naturally expect dramatic differences between an article on politics and one on the equity markets. After having built topic specific models, we can then have one super-model which attempts to identify the specific topic of a document and uses the information provided by the topic specific models to make its predictions. Since the trigram model has been quite successful, it would be natural to build different trigram models for each topic and then interpolate amongst them in some manner as to weigh the topic models according to the probability of the candidate document being a given topic. An added benefit of topic specific models is that they can be used as the basis for further adaptive extensions such as triggers. For example, a topic identification can be used to select amongst various sets of triggers to consider for a given word.

3.1 Identifying a Suitable Set of Topics

Before we can build topic specific language models, we clarify exactly what we mean by “topics” and we must decide on how to arrive at a suitable set of topics. In some cases, such as in earlier experiments performed at IBM on building topic specific models (Roukos [23]), the documents are already tagged with topics selected by humans. In such a case, we can just use those topics as is, or perhaps clean them up slightly by consolidating small topics. Documents can then be clustered into the given topics (*supervised clustering*). However, the Wall Street Journal data which we have does not carry any topic information, so we must find some other way of clustering the documents into topics.

While it is possible to label the documents with topics by hand or to hand pick prototypes for each topic and perform a statistical clustering around those prototypes, we choose to follow a purely statistical approach of *unsupervised* (without human supervision) clustering. Our motivation here is that the success of statistical models such as the trigram model provides ample anecdotal evidence that statistical techniques may discover phenomenon not evident to us humans.

3.1.1 Measuring the Closeness of Documents

In order to cluster our documents statistically, we need some measure of the closeness between two documents or perhaps between a document and a prototype. This measure does not have to be a true metric. It does not even have to be symmetric. However, it should reflect what we as humans consider as topic similarity. One possible measure is the *cross entropy* between the unigram probability distributions of two documents (or of a document and a prototype). However, preliminary attempts at using this measure met with poor results.

Another possible measure which proved effective in previous topic identification experiments (Roukos [23]) is the *Kullback-Leibler distance* between the unigram probability distributions of two documents (or of a document and a prototype). More specifically, let $D(A,B)$ represent the distance from document A to document or prototype B according to this measure. Then:

$$D(A, B) = \sum_{w \in V} \frac{c_A(w)}{N_A} \log \frac{(c_A(w))/N_A}{(c_B(w))/N_B} \quad (\text{EQ 5})$$

Here, $c_A(w)$ and $c_B(w)$ are the number of times w occurs in A and B respectively and N_A and N_B are the number of words in A and B respectively and V is the vocabulary. Finally, $\log(0/0)$ is defined to be 1. One of the problems with this measure is that if a word occurs in A and not in B or vice-versa, $D(A,B)$ will be zero. To circumvent this problem, we smooth the

probabilities by adding a small factor which depends on the unigram probability of a word in the entire training corpus. Namely, instead of $\frac{c_X(w)}{N_X}$, we will use $\lambda \frac{c_X(w)}{N_X} + (1-\lambda) P_u(w)$. Here, $P_u(w)$ is the unigram probability of w over all training data. Every word in our vocabulary occurs at least once in the total training corpus so this term is never zero. For λ we select some convenient value such as 0.99. This smoothed measure is the one that we will use.

3.1.2 Clustering Documents

Once we have selected a distance measure, we still need to come up with an algorithm from clustering the documents into topics in an unsupervised manner. There are two basic approaches to this problem, namely top down and bottom up. For the purpose of generating approximately ten topics, a random sample of approximately 2000 documents should be adequate. An initial attempt at a bottom up approach which first compared all possible pairs of documents and a symmetrized variant of the final distance measure given in section 3.1.1 took an excessive amount of computing time (over six hours on an IBM RS/6000 PowerServer 580, a 62 MIPS machine), forcing us to abandon such an approach. Instead, we will pursue a top-down approach.

We randomly select ten documents as the prototypes for the clusters. We then iteratively place each document into the cluster it was closest to until no documents change clusters. An issue which needs to be explored here is how to update the prototype of a cluster to reflect the addition of a new document or the removal of an existing document. One possibility is to let the prototype's unigram distribution be the average unigram distribution of all the documents in the cluster. However, when we attempt this approach, we run into the following problem. Clusters with a large number of documents tend to have "smoother" unigram distributions. So, a few of the random seeds which are similar in their unigram distributions to a large number of documents became magnets, draw more and more

documents into their clusters. In the end, we have only three extremely large topics. Examination of sample documents in these topics reveals no noticeable similarity of the documents within each topic. To alleviate this problem, we instead choose a real document in a given cluster which is closest to the average unigram distribution for the cluster. Since the prototype for each cluster is a single representative document, each prototype is of equal “sharpness” and the aforementioned problem does not occur.

Running this procedure on 2000 documents randomly selected from our 5 MW test set converges in three iterations and results in eight clusters with more than a handful of documents. (Two of the clusters have fewer than ten documents. The documents within them are folded into the other eight clusters using the same distance measure.) Five documents are randomly selected from each cluster and examined. We obtained a quick synopsis of the random documents for the clusters by randomly choosing five documents from each cluster and examining them. A summary of the five random documents selected from each cluster is given in Figure 1. The topics appear to successfully discern various financial topics such as economics in cluster one and rules and regulations in cluster four. However, non-financial subjects were not well separated by the topics obtained. Almost all non-financial documents were placed into cluster zero which ended up with 682 documents. Nevertheless, we feel that the clusters look good enough subjectively to be useful.

Cluster 0:

1. Iran Contra.
2. Regional dispute/unrest in some country. Elections. Fight for political control.
3. U.S. and Soviet involvement in Iran. Gulf geopolitics.
4. Summaries: sale of jets to Saudis, political control in the Phillipines, NASA and S. Korean developments.
5. Japanese trade relations with U.S.

Cluster 1:

1. Decrease in bank reserves.
2. Economic and trade statistics about England.
3. New construction statistics.
4. Budget surplus, national debt and deficit.
5. Leading economic indicators, other economic statistics.

Cluster 2:

1. Appointment of a corporate officer.
2. Appointment of a corporate director.
3. Summaries: new product, earnings, joint-ventures, OTC activity.
4. Corporate acquisition.
5. Plant closing.

Cluster 3:

1. New stock issue.
2. Corporate acquisition.
3. Stock repurchase.
4. Tender offer.
5. Sale of unit.

Cluster 4:

1. New procedures adopted by NYSE and SEC.
2. New regulations adopted by London Stock Exchange.
3. Appellate court ruling blocking action related to financial markets.
4. Court issued injunction against postponing shareholder meeting.
5. Investigation into insider trading activities.

Cluster 5:

1. IRS issues a new withholding form.
2. Problems in oil industry and revenue bonds relating to same.
3. Debt problems in Argentina.
4. Recovery and outlook of the semiconductor industry.
5. Appointment of executive VP and general counsel at a bank.

Cluster 6:

1. Airline industry - traffic and revenue.
2. New stock issue.
3. New Treasury issue.
4. Merger and acquisition offer.
5. Sale of unit considered.

Cluster 7:

1. Elimination of calories and cholesterol. Health fads and legitimate concerns.
2. Studies on cholesterol.
3. Naval ship in Gulf hit by Iraqi missile.
4. Gene located for disease.
5. Refinery removed due to hazards.

FIGURE 1. Synopsis of Five Random Documents in Each Cluster

3.2 Topic Specific Trigram Models

The trigram model is a natural candidate for which we can build topic specific models and evaluate their performance on test data. If this effort proves successful, we can then add other information, such as triggers, to the topic specific trigram.

Once we have topic specific trigram models, there is still the issue of identifying which topic specific model or which combination of topic specific models to use when decoding test data. For an initial experiment, we can just tag each test document with its closest topic as determined by the distance measure used to create the topic clusters. This is more optimistic than a real decoding scenario because normally, we would not have the ability to “look ahead” and access the entire document. We normally have access only to the document history as decoded so far. Any results obtained from this experiment will be admittedly an upper bound on the actual performance to be expected. Nevertheless, this experiment will help us to evaluate whether we should continue with the topic specific adaptation idea.

When we perform the experiment as described, we obtain perplexities which are even worse than using a single universal (non-topic specific) trigram model. One possible explanation might be because we do not have adequate data within each topic. So, instead, we can try smoothing the topic specific models with the universal model. We accomplish this by using linearly interpolating the topic specific model with the universal model using weights optimized over held-out data. The perplexity results of this experiment are shown in Table 1. The entire 37 MW training set was used to create the universal trigram model and the eight topic specific trigram models.

The topic specific trigram models appear to offer an improvement. However, it is worth noting that the least improvement occurs with the largest topics, namely topic zero and topic five. A possible explanation for this behavior is that documents without truly discerning

characteristics tend to be clustered together into larger topics so that models specific to such topics have little to offer given the non-uniqueness of the topics. Another possibility is that we may have just been unlucky with the construction of topics zero and five. However, when another experiment to further subdivide cluster zero into three topics is performed, we observe only a marginal improvement. (The overall perplexity using models for the subtopics smoothed with the universal model is 228.83. This compares to the 236.13 shown in the Table 1 for topic zero.)

TABLE 1. Smoothed Topic Specific vs. Universal Trigram Model

Topic Cluster	Number of Words	PP with Universal Model	PP With Topic Specific Model
0	382435	249.56	236.13
1	17802	43.86	32.53
2	23745	39.92	33.15
3	57155	62.14	52.18
4	28210	122.14	106.06
5	206728	120.38	116.64
6	91827	41.05	32.59
7	0	-	-
Overall	807902	136	124

The upper bound on the improvement we can expect from the topic specific trigram models is from a perplexity of 136 to 124, only 8.8%. We do not feel that an 8.8% improvement is enough to justify continuing with using topic specific models. Perhaps the topic selection algorithm can be improved but a related result arrived at by Martin Franz at IBM suggests otherwise (Franz [8]). Franz had Wall Street Journal data for years 1990 through 1992 obtained directly from Dow Jones, the publisher of the Wall Street Journal. Unlike the ARPA supplied data, his data set includes human derived subject classifications for each article. He built several trigram language models specific to a small number of subject classifications along with a universal model. Then, considering the most optimistic case, he took test documents for the subjects he had chosen and decoded them with the appropriate

interpolation of subject specific and universal language model. He reports an improvement in perplexity of around 10% only. This is not conclusive evidence that better statistically derived topics will not result in a much greater improvement. As already mentioned, what makes sense to a model need not make sense to humans at all, so human derived topics need not be considered sacrosanct. However, since the motivation behind building topic specific language models is because we feel that language properties are related to what we as humans would consider topics, the failure of Franz's experiment leaves us with little inspiration on making further progress in this direction.

4.0 Maximum Entropy Models

All of the remaining models which we will discuss are maximum entropy models. An extensive theoretical derivation of a framework for statistical models based on the maximum entropy principle (Good [11]) can be found in Della Pietra and Della Pietra [5]. The author's prior efforts (Lau [17], Lau et al. [18] and Lau et al. [19]) also involves the creation of maximum entropy statistical language models. In this chapter, we recap the salient details of the maximum entropy framework.

4.1 Principle of Maximum Entropy

In a maximum entropy formulation, we are initially given some a priori information about the source being modelled. This information may be an existing model distribution or it may be the uniform distribution which corresponds to no prior information. We then define a series of linear constraint functions to present the information we wish to model. The principle of maximum entropy calls for a distribution which satisfies the linear constraint functions, hence capturing the information we are modelling, and which is “closest” to the initial distribution. Closest is defined in terms of Kullback-Leibler distance. Let Q be our initial distribution and Y be the set of possible futures being predicted. Then the maximum entropy solution is the distribution P which satisfies the constraint functions and which minimizes the Kullback-Leibler distance $D(P \parallel Q)$:

$$D(P \parallel Q) = \sum_{y \in Y} P(y) \log \frac{P(y)}{Q(y)} \quad (\text{EQ 6})$$

The name maximum entropy³ comes from the special case where Q is the uniform distribution, hence, P is the distribution which satisfies the constraints and which maximizes the entropy function $H(P)$:

$$H(P) = - \sum_{y \in Y} P(y) \log P(y) \quad (\text{EQ 7})$$

The advantages of the maximum entropy approach include:

- It makes the least assumptions about the distribution being modelled other than those imposed by the constraints and given by the prior information.
- The framework is completely general in that almost any consistent piece of probabilistic information can be formulated into a constraint.
- The case of linear constraints, that is, constraints on the marginals, corresponds to no higher order interaction (Good [11]).
- If the constraints are consistent, that is there exists a probability function which satisfies them, then amongst all probability functions which satisfy the constraints, there is an *unique* maximum entropy one (Csiszár [2], Theorem 3.2 in particular and Della Pietra and Della Pietra [5]).
- The method of *generalized iterative scaling*⁴ (Darroch and Ratcliff [3]) provides an iterative algorithm for finding the maximum entropy solution. Using this algorithm, we have the flexibility to add new constraints after the model has already partially converged provided that the new constraints are not inconsistent with the existing ones and we can also remove existing constraints.

3. Where the prior distribution is not uniform, the maximum entropy solution is sometimes referred to in the literature as minimum discriminant information, a technically more accurate name.

4. Generalized iterative scaling is sometimes referred to in the literature as iterative proportional fitting.

- Fuzzy maximum entropy (Della Pietra and Della Pietra [5]) allows us to further “smooth” the maximum entropy distribution.

Some of the disadvantages of a maximum entropy approach include:

- There are no known bounds on the number of iterations required by generalized iterative scaling. Also, each iteration of generalized iterative scaling is computationally expensive.
- Good-Turing discounting (Good [12]) may be useful in smoothing low count constraints (Katz [14]). However, it introduces inconsistencies so a unique maximum entropy solution may no longer exist and generalized iterative scaling may not converge.

4.2 Maximum Entropy Solution and Constraint Functions

If we let x denote the current history and y denote a future, in the maximum entropy approach the solution is of the form:

$$Pr(x, y) = Q(x, y) e^{\sum_i \lambda_i f_i(x, y)} \quad (\text{EQ 8})$$

Here, $Pr(x,y)$ is the desired maximum entropy probability distribution of (x, y) , $Q(x,y)$ is the prior or initial distribution, the $f_i(x,y)$ are the *constraint functions* and the λ_i are the *model parameters* or *factors* to be determined by generalized iterative scaling or gradient search. The constraints themselves are expressed as:

$$Ef_i \equiv \sum_{x, y} Pr(x, y) f_i(x, y) = d_i \quad (\text{EQ 9})$$

In other words, we want the *model expectation*, Ef_i , to be equal to some *desired expectation*, d_i . The desired expectation may be set equal to the *observed expectation*, $\tilde{E}f_i$, which can be computed using the expression in (EQ 9), but with $Pr(x,y)$ being replaced by the *observed*

probability, $\tilde{Pr}(x,y)$. Note that the definition of $f_i(x,y)$ is arbitrary. For concreteness, we cite an example of a possible constraint function we may use in a language model. We can define a constraint function for trigrams as follows:

$$f_{w_1 w_2 \rightarrow w_3}(x, y) = \begin{cases} 1 & \text{if } x \text{ ends in } w_1 w_2 \text{ and } y = w_3 \\ 0 & \text{otherwise} \end{cases} \quad (\text{EQ 10})$$

Thus, in a maximum entropy model with such trigram constraints, we expect the model to predict w_3 when the history ends in $w_1 w_2$ with the same probability as observed during training of the model parameters.

4.3 Maximum Likelihood Maximum Entropy

The maximum entropy model as stated so far gives a joint distribution of histories and futures. For our purposes, the word being predicted is the future and the document history is the history. What we really want is a conditional distribution on words given a history. The conditional probability is given by:

$$Pr(w|h) = \frac{Pr(h, w)}{\sum_{w' \in V} Pr(h, w')} \quad (\text{EQ 11})$$

Here, V is the vocabulary and h is the current document history. As noted in Lau [17], this model is inadequate for most language modelling applications because we lack sufficient data to reliably estimate the event space of (h,w) pairs. More specifically, since we cannot hope to observe more than an insignificant fraction of the possible $O(|V|^{|h|})$ histories, we must rely on a partition of the histories according to broad features. However, using such a technique does not allow us to adequately rule out ridiculous histories such as a repetition of the word “the” 500 times. We cannot hope to come up with classifying features specific enough to rule out all such outrageous histories. Instead, we modify the model slightly in a

manner suggested by colleagues at IBM. This modification was referred to as maximum likelihood maximum entropy in Lau [17] and we will continue to use that term here.

The definition of Ef_i in (EQ 9) can be rewritten as:

$$Ef_i \equiv \sum_h Pr(h) \sum_w Pr(w|h) f_i(h, w) \quad (\text{EQ 12})$$

Now, we will replace $Pr(h)Q(h)$ with $\tilde{Pr}(h)$, the observed probability of the histories in the *training data*:

$$Ef_i \equiv \sum_h \tilde{Pr}(h) \sum_w Pr(w|h) f_i(h, w) \quad (\text{EQ 13})$$

This modification says that we are constraining the expectations to match the specific numerical values observed from the histories in the training data. Here, by definition, the probability distribution of the histories matches the observed training data distribution of histories as far as the constraint functions which are non-zero for a given history are concerned. An interesting result of this modification is that the new model has the following equivalent interpretation:

Consider the exponential form of the model proposed by maximum entropy in (EQ 8) with our desired constraint functions. Now, find the set of factors, λ_i , which maximize the likelihood of generating the data observed. This equivalence is the duality property shown in Della Pietra and Della Pietra [5] and derived constructively in Lau [17].

Finally, we note that (EQ 13) may be rewritten in a more useful form:

$$Ef_i \equiv \frac{1}{|TR|} \sum_{h \in TR} \sum_w Pr(w|h) f_i(h, w) \quad (\text{EQ 14})$$

This equation is of practical importance. Here, TR denotes the training data. The summation over $h \in TR$ means that we go through the positions, which correspond to the events (h,w) , in the training data and accumulate the conditional expectations evaluated at each position. *This*

expression is particularly important because it allows us to practically evaluate the model expectations, Ef_i , without summing over all $O(|V|^{h+1})$ possible history-word events! Many of these histories never occur in the training data and this formulation makes it explicitly clear that we only need to consider those which do occur.

The other important equation, derived from (EQ 9), is the one defining how we can compute the desired expectations, which we are setting to be equal to the observed training data expectations:

$$d_i = \tilde{E}f_i \equiv \frac{1}{|TR|} \sum_{(h, w) \in TR} f_i(h, w) \quad (\text{EQ 15})$$

4.4 Generalized Iterative Scaling

The method of generalized iterative scaling (Darroch and Ratcliff [3]) can be used to find the factors, λ_j , of the model. Generalized iterative scaling is an iterative algorithm and is guaranteed to converge to the correct solution, provided that the desired constraint expectations are consistent. The GIS algorithm as applied to our model may be summarized as follows:

- (i) Compute the desired expectations, d_j .
- (ii) Initialize the factors, λ_j , to some arbitrary initial value. (For concreteness, we may consider them all to be initially zero.)
- (iii) Compute the model expectations, Ef_j , using the current set of factors.
- (iv) Update the factors by:

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} + \ln \frac{\tilde{E}f_i}{Ef_i} \quad (\text{EQ 16})$$

- (v) If not converged, go to (iii). (Convergence may be determined, for example, by the movement in the factors or by the change in perplexity from the previous iteration.)

The GIS algorithm requires that for any event, (h,w) , the constraint functions sum to one. When constraint functions do not naturally present themselves in such a manner, we can avoid the problem by artificially rescaling the constraint functions so that the sum to one requirement is met. To handle the varying number of constraints which may be active, we will add a *filler* constraint for each future word⁵.

4.4.1 Computational Issues Involved with GIS

While the statement of the GIS algorithm itself is very simple and concise, many computational issues arise in creating a feasible and efficient implementation. Here we will address generally the issues involved in implementing GIS for training maximum entropy language models. Lau [17] goes into quite some detail on describing the exact mechanics of implementing GIS for a trigger and n-gram language model.

The GIS algorithm can be broken down into three modules:

Initiator - The initiator sets up the files needed for the model and computes the observed or desired values of the constraint functions.

Slave - The slave module is executed once per iteration. It computes the model expectations for the constraint functions given the current model factors. Since these model expectations are summed over the training data, we can divide the training data into segments and run multiple slaves in parallel, summing the results of all the slaves at the end of the iteration. This last step can be performed by the updater. This ability to parallelize the

5. Defining a different filler constraint for each word was suggested as a possible improvement over having one universal filler constraint in Lau [17].

execution of the slave is important because the bulk of the computation takes place in the slave since the slave must examine every event in the training set.

Updater - The updater takes the model expectations computed by the slave together with the desired expectations computed by the initiator and updates the model parameters. If parallel slaves were used, the updater also combines the results from the multiple slaves.

As already mentioned, most of the work is done in the slave where the model expectations, Ef_i , defined in (EQ 14) must be updated for each event in the training data. If we examine the expression in (EQ 14), we note that potentially each *active* constraint's expectation may need to be incremented for each event. By active constraint, we mean constraint's whose constraint function evaluates to a non-zero value for an event. This is potentially disastrous for constraints such as each word's filler constraint which are always active. However, because of the sum to one requirement for the constraints, we can compute the filler constraint analytically by subtraction at the end of the iteration. For some other constraints which are always active, such as unigram constraints if we choose to include them, we can circumvent the problem by observing that such constraints have a "default" value for most events. Hence we only need to explicitly accumulate over positions where the value is not the default value and analytically compute the correct value at the end of an iteration. A similar technique can be used to maintain the denominator in (EQ 11). (The expression defined there is the model probability needed for computing the model expectations.) Namely, we consider the changes from one event to the next and add the changes into the denominator rather than explicitly recomputing the entire denominator. Note that the changes to the terms in the denominator correspond exactly to those constraints whose expectation needs to be updated from their default values so this leads to a convenient implementation.

4.4.2 Partial Updating of Constraints

Empirical evidence suggests that the model parameters (λ_i s) corresponding to constraint functions (f_i s) with larger relative values converge more quickly to their final values under generalized iterative scaling. Lau [17] discusses a technique for emphasizing n-gram constraints to improve convergence rates for maximum entropy models with n-gram constraints. The idea is to rescale the constraint functions so that the most powerful information, the n-gram constraints, have functions with a higher relative weight. Observing that each f_i is always used with the corresponding λ_i , we can arbitrarily scale f_i by a constant, k , as long as we scale the corresponding λ_i by $1/k$. In our models with n-gram constraints, we will want to make use of this technique to improve training convergence times. We can take this technique to the extreme and only update certain constraints during initial iterations. This corresponds to scaling the constraints we do not wish to update by a factor of zero so that they drop out. Frequently, we will want to make use of this technique and run the first few iterations updating the n-gram constraints only.

5.0 Maximum Entropy Trigram Model

We mentioned that one of the advantages of a maximum entropy model is that we can combine information from various sources in the framework. Undoubtedly, given the success of the trigram model, we will want to include trigram constraints in at least some of the models we will investigate. In Lau [17], a maximum entropy trigram model achieved a perplexity of 185 on test data as compared to 170 achieved by a deleted interpolation trigram model. However, it was observed that the deleted interpolation model was an order of magnitude larger since it included all unigrams, bigrams and trigrams whereas the maximum entropy model only included bigrams and trigrams with a count of three or more and unigrams with a count of two or more. The reason we did not try including more lower count constraints in the model was the prohibitive cost of computation involved.

In this chapter, we explore the maximum entropy trigram model once again. This time, we have switch to the more standardized ARPA 20K vocabulary with non-verbalized punctuation. We will present a closed form solution to the maximum entropy trigram model, alleviating any computational bottlenecks. We will also examine improvements which allow the maximum entropy model to actually surpass the deleted interpolation model.

5.1 N-gram Constraints

To capture the trigram model's information within the maximum entropy framework, we use an *n-gram constraint*. First, let's define three sets:

$$S_{123} = \{ w_1 w_2 w_3 : \text{count } w_1 w_2 w_3 \geq \text{threshold} \}$$

$$S_{23} = \{ w_1 w_2 w_3 : w_1 w_2 w_3 \notin S_{123} \text{ and count } w_2 w_3 \geq \text{threshold} \}$$

$$S_3 = \{ w_1 w_2 w_3 : w_1 w_2 w_3 \notin \{ S_{123} \cup S_{23} \} \text{ and count } w_3 \geq \text{threshold} \}$$

$$S_0 = \{ w_1 w_2 w_3 : w_1 w_2 w_3 \notin \{ S_{123} \cup S_{23} \cup S_3 \} \}$$

Then our constraint functions $f(h,w)$ will be indicator functions for the above sets with $w = w_3$ and the last two words in $h = w_1 w_2$. Several observations are worth making here. At any point, at most one constraint is active for a given future word because the sets are non-overlapping. This corresponds to the *no-overlap* formulation given in Lau [17]. The alternative is to define set S_{23} without regard to S_{123} and to define set S_3 without regard to S_{23} . One of the benefits of this approach is that convergence of GIS is much faster for any model which includes the no-overlap form of n-gram constraints than the other form because there are fewer active constraints active for any given event. Our experience with GIS shows that the fewer the number of active constraints, the faster the convergence rate. This was borne out in Lau [17] as far as n-gram constraints are concerned.

Another benefit is that in a model which includes *only* n-gram constraints, the no-overlap formulation gives us a model which can be computed in closed form so multiple iterations of GIS are not needed. To see this, consider the computation of the model probability given in (EQ 11). For each event $w_1 w_2 w_3$, we can express the model probability as:

$$Pr(w_3 | w_1 w_2) = \frac{e^{\lambda_{w_1 w_2 w_3}}}{Z(w_1 w_2)} \quad (\text{EQ 17})$$

There is at most one constraint active for any given event, so we only have one λ to worry about. We should also recognize that since the denominator in (EQ 11) sums over all possible w_3 , it is a function of $w_1 w_2$ only. Now, if we further consider that there is one more equation, namely that everything sums to one for a given $w_1 w_2$ then we observe that there are as many equations as there are unknowns. (There is one equation for each $w_3 \in V$ of the type given in (EQ 17) set equal to the observed training data expectation and the sum to one equation. There

is one unknown for each $w_3 \in V$, the λ s, and there is one $Z(w_1w_2)$.) Thus, a closed form solution exists.

5.2 Performance of ME N-gram Model

To evaluate the performance of our maximum entropy n-gram model, we construct a deleted interpolation trigram model out of the small 5 MW training set and evaluated its performance on our 870 KW test set as a baseline for further comparisons. This deleted interpolation model achieves a perplexity of 225.1. Next we construct two maximum entropy n-gram models with different cutoff thresholds for the definitions of the sets S_{123} , S_{23} and S_3 . With a threshold of 3 for each set, our model achieves a perplexity of 253.5. With a threshold of two for each set, our model achieves a perplexity of 247.9. Franz and Roukos [9] reports that a threshold of one, which corresponds to all n-grams as in the deleted interpolation case, does not perform well. One possible reason for this is because of the unreliability of count one data and the strict constraints imposed by the maximum entropy framework based on this unreliable data. As a final comparison, we construct a deleted interpolation model which uses only unigram, bigram and trigram counts of at least three. This model achieves a perplexity of 262.7. Note that this final model has roughly the same amount of information as the maximum entropy model with a threshold of three. It seems that maximum entropy performs better than deleted interpolation when the two models are of roughly the same size. However, we cannot directly build a maximum entropy model with a threshold of one in order to capture the same amount of information as available in the full trigram model. Doing so would cause performance to deteriorate significantly (Franz and Roukos [9]). Thus, we cannot match the full deleted interpolation model with the machinery we have developed so far. The results of the experiments involving the various N-gram models are summarized in Table 2:

TABLE 2. Deleted Interpolation vs. Maximum Entropy Trigram Model

Model	Amount of Information	Perplexity
DI	3.5M	225.1
ME w/threshold of 2	790K	247.9
ME w/threshold of 3	476K	253.5
DI w/threshold of 3	790K	262.7

5.3 Improvement Through Smoothing

One possible way of incorporating the lower count trigram information in the maximum entropy model is to first smooth the information. The favorable maximum entropy results reported in Lau et al. [18] actually included smoothing of the n-gram constraints. The exact improvement due to smoothing were never isolated and examined in that case. Similarly, experimental evidence witnessed by colleagues at IBM in applying maximum entropy modelling to the problem of statistical machine translation suggests that smoothing may be very important (Della Pietra and Della Pietra [6]). In the following section, we pursue two approaches to smoothing the n-gram data.

5.4 Fuzzy Maximum Entropy with Quadratic Penalty

Della Pietra and Della Pietra [5] introduces a framework which they have dubbed *fuzzy maximum entropy*. Fuzzy maximum entropy is a more generalized formulation of maximum entropy where the constraints (EQ 9) are allowed to be relaxed. Strict equality is no longer demanded. Instead, as we deviate from a strict equality, we must pay a *penalty*. We no longer seek the solution which minimizes the Kullback-Leibler distance given in (EQ 6). Instead, we minimize the sum of the Kullback-Leibler distance and the *penalty function*, $U(c)$.

We require that the penalty function be convex. The special case of an infinite well penalty function

$$U(c) = \begin{cases} 0 & \text{if } c_i = d_i \\ \infty & \text{otherwise} \end{cases} \quad (\text{EQ 18})$$

corresponds to the standard maximum entropy setup. Here, an exact equality of the constraint expectation, c_i , and the desired value, d_i , results in a zero penalty. Otherwise, we have an infinite penalty.

One way to think of the fuzzy maximum entropy framework in the context of smoothing is that we can make use of penalty functions which penalize deviations in more reliably observed constraints to a greater degree than deviations in less reliably observed constraints. Thus, we admit a solution where the less reliably observed constraints are relaxed in order to improve overall smoothness.

We consider the following quadratic penalty function:

$$U(c) = \frac{1}{2} \sum_{i,j} (c_i - d_i) \sigma_{ij}^{-1} (c_j - d_j) \quad (\text{EQ 19})$$

Here, we can think of σ as the covariance matrix. Unfortunately, computation of σ is nontrivial. As a matter of fact, we have to consider all possible pairs of constraints. Instead we will consider only the diagonal entries of σ , that is, we will only consider the variance of each constraint.

An intuitive interpretation of the quadratic penalty function with a diagonal covariance matrix is the following. Instead of requiring a strict equality to the desired value, d_i , of a constraint, we can think of a Gaussian distribution centered at the desired value. The width of the Gaussian is determined by the variance. The variance of the i^{th} constraint function is given by $\sigma_{ii} = \tilde{E}(f_i^2) - (\tilde{E}f_i)^2$ where as before \tilde{E} denotes the observed value summed over all events in

the training data and N is the size of the training data. The constraints with a high number of observations would have a low variance and hence a smaller Gaussian.

In Della Pietra and Della Pietra [5], a variant of the generalized iterative scaling algorithm which can handle fuzzy maximum entropy is developed. For the case of a quadratic penalty with a diagonal covariance matrix, the update step in GIS is replaced with the following at the end of iteration k :

$$e^{\delta\lambda_i^{(k+1)}} E f_i = c_i^{(k+1)} \sigma_{ii} \quad (\text{EQ 20})$$

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} + \delta\lambda_i^{(k+1)} \quad (\text{EQ 21})$$

$$c_i^{(k+1)} = c_i^{(k)} - \lambda_i^{(k+1)} \sigma_{ii} \quad (\text{EQ 22})$$

Initially, $c_i^{(0)} = d_i$. To solve (EQ 20) for the update value $\delta\lambda_i^{(k+1)}$, we must use a numerical root finding algorithm such as the Newton-Raphson method.

Some of the benefits of this approach to smoothing are:

- Smoothness here has a precise definition, namely, entropy. This definition is rather natural for the case of a perfectly smooth distribution, the uniform distribution, corresponds to that of maximum entropy.
- The generalized iterative scaling modified as described is guaranteed to converge.
- There is very little additional computation required *per iteration* as compared to the regular GIS algorithm. (However, we may need more iterations. Also, the special case of non-overlapping constraints such as the n-grams only case we are considering can no longer be solved in closed form.)

A disadvantage is:

- The choice of a penalty function is subjective. We have selected our particular function mainly on the basis of computational convenience although we feel that the selection is reasonable.

5.5 Good-Turing Discounting

Another possible approach to smoothing is to apply *Good-Turing discounting* to the desired constraint values, d_i . The use of Good-Turing discounting with the back-off trigram model was discussed in Katz [14]. We use the same formulation here. Let r represent the “count” of a constraint, that is, $r = d_i * N$ where N is the size of the training text. Note that r is guaranteed to be an integer when we define the desired constraint values to be the observed values in the training data as given by (EQ 14). Then, the discounted count D_r is given by:

$$D_r = \frac{(r+1) \frac{n_{r+1}}{n_r} - (k+1) \frac{n_{k+1}}{n_1}}{1 - \frac{(k-1) n_{k+1}}{n_1}}, \quad \text{for } (1 \leq r \leq k) \quad (\text{EQ 23})$$

Here, n_i denotes the total number of constraints whose count equals i and k is the maximum count for which we are discounting. As in Katz [14], we will use $k = 5$ and we will omit all singleton constraints, that is, we will set $D_1 = 0$. Also, because we expect the constraint functions from sets S_{123} , S_{23} and S_3 (defined in section 5.1) to have varying reliability for a given count, we will compute a set of discounted counts separately for each set.

Some of the advantages of using Good-Turing discounting are:

- It was successful with the back-off trigram model, which resembles our no-overlap n -gram constraint formulation somewhat.

- It is computationally easy to implement, requiring only a minor adjustment to the GIS update step.

Some disadvantages of this method include:

- GIS is no longer guaranteed to converge since the discounted constraints imposed are no longer consistent with the observed training data. In reality, we expect that GIS will eventually diverge given discounted constraints.
- There is no easy way to apply discounting to overlapping constraints because we no longer have counts for partitions of the event space. However, even if we combine n-grams with other information in an overlapping manner, we can still discount just the n-gram constraints because those are non-overlapping with each other.

5.6 Results of Smoothing Experiments

We build two maximum entropy n-grams only language models using the fuzzy maximum entropy with quadratic penalty and Good-Turing discounting smoothing methods discussed. These models use the same training and test data used to build the unsmoothed maximum entropy n-grams only model and the deleted interpolation trigram model. For the quadratic penalty fuzzy maximum entropy model, we choose to include constraints with a count of two or greater only because the inclusion of count one constraints would be a serious computational bottleneck. The training perplexities after each iteration are given in Table 3. For the model with Good-Turing discounting, we will discount counts less than or equal to five and we will omit singletons to insure a fair comparison with the fuzzy maximum entropy experiment. The discounted counts for the sets S_{123} , S_{23} and S_3 are given in Table 4. Note that both of these models have the same amount of information as the threshold of two maximum entropy n-grams only model examined in section 5.2. A comparison of the performance of the

smoothed models on test data is given in Table 5. As we can see, smoothing of the maximum entropy model certainly helps performance. Good-Turing discounting appears to be more helpful than fuzzy maximum entropy. As a matter of fact, the discounted maximum entropy model with thresholds of two on the constraints marginally outperformed the deleted interpolation model which includes singletons.

TABLE 3. Training Perplexities for Fuzzy ME N-grams Model

Iteration	Perplexity
1	20010
2	105.1
3	103.5
4	103.1
5	102.9
6	102.9

TABLE 4. Discounted Counts for Good-Turing Discounting of ME N-grams Model

Count	Disc. Count for S_{123}	Disc. Count for S_{23}	Disc. Count for S_3
2	0.975	1.204	1.852
3	1.942	2.143	2.715
4	2.884	3.190	3.926
5	3.817	4.074	4.722

TABLE 5. Performance of Smoothed ME N-grams Models

Model	Size of Model	Perplexity
Deleted Interpolation	3.5M	225.1
ME w/threshold of 2	790K	247.9
Fuzzy ME w/threshold of 2	790K	230.3
Good-Turing Discounted ME w/threshold of 2	790K	220.9

6.0 Self-Triggers

The success of the cache in Kuhn and De Mori [15] suggests that the appearance of a word has a substantial impact on the probability of a future appearance of the same word. Preliminary experimentation in Lau [17] and Lau et al. [18] reveal the same interesting phenomenon. We will refer to the effect of a word on its own future probability as the *self-triggering effect*. In the present work, we will try to determine the improvement attributable to self-triggers within a maximum entropy model.

6.1 Self-Trigger Constraints

For our experiment, we will only attempt to model whether a word has or has not already occurred in the current document. To do this, we introduce a *self-trigger constraint* for each word of the form:

$$f_{w \rightarrow w}(h, w) = \begin{cases} 1 & \text{if } w \text{ occurs in the document history } h \\ 0 & \text{otherwise} \end{cases} \quad (\text{EQ 24})$$

We include along with these self-trigger constraints the no-overlap n-gram constraints discussed earlier and a filler constraint for each word:

$$f_{-w}(h, w) = 1 - f_{w \rightarrow w}(h, w) \quad (\text{EQ 25})$$

The purpose of the filler constraint is the following. Recall that GIS requires that all constraints sum to one for each event. Without the filler constraint, the sum is either one if the self-trigger constraint function is zero, or two if the self-trigger constraint function is one. (There is exactly one n-grams constraint function which is one for a given event.) However,

with the filler constraint, the sum is always two so we can just mechanically scale by a factor of 1/2 to implement GIS.

6.2 Experimental Results

We train our maximum entropy model on our 5MW training corpus. For the n-gram constraints, we will only include constraints whose count is at least two and we will make use of the successful Good-Turing discounting technique here as well. For the self-trigger constraints, we will likewise require a count of two. However, because most words occur more than once per document, almost all words (19K out of 20K) has corresponding self-trigger constraints. To speed up the convergence rate, we will only update n-gram constraints on the first iteration as described in section 4.4.2. The training perplexities after each iteration are given in Table 6.

TABLE 6. Training Perplexities for Self-Trigger Experiment

Iteration	Perplexity
1	20010
2	107.5
3	93.9
4	88.2
5	85.5
6	84.2
7	converged

This model achieves a test set perplexity of 178.5, a 20.7% improvement over the deleted interpolation trigram model. The other worthwhile comparison is the deleted interpolation trigram model combined with a dynamic cache similar to that of Kuhn and De Mori [15]. The dynamic cache makes a prediction based on the frequency of occurrence of the current unigram, bigram and trigram in the document history so far. It is combined with the deleted interpolation model through simple linear interpolation. Such a model has a perplexity

of 186.3 on our test set. While this is a 17.2% improvement over the trigram model alone, our maximum entropy model gives another 4.2% improvement beyond the 186.3.

6.3 Analysis of Trained Parameters

We decide to examine this model more closely. In Table 7, we give some sample α s for several words in the self-triggered and untriggered states. Here in the self-triggered case, we define $\alpha \equiv \exp[\lambda_{w \rightarrow w}]$ and in the untriggered case, $\alpha \equiv \exp[\lambda_{\neg w}]$. In our model, the probability of a word is given by:

$$Pr(w|h) = \frac{\exp[\lambda_{w_1 w_2 w} + \lambda_{w \rightarrow w} f_{w \rightarrow w}(h, w) + \lambda_{\neg w} f_{\neg w}(h, w)]}{Z(h)} \quad (\text{EQ 26})$$

Here $\lambda_{w_1 w_2 w}$ denotes the appropriate λ from the n-gram constraint and $Z(h)$ denotes the appropriate normalization factor, the denominator in (EQ 11), for the given history. Note that by our definitions in (EQ 24) and (EQ 25), at most one of $f_{w \rightarrow w}(h, w)$ and $f_{\neg w}(h, w)$ is non-zero depending on whether the word w is self-triggered or not. The exponentials of the respective λ s are the α s we have defined and shown in Table 7. We may express (EQ 26) as:

$$Pr(w|h) = \alpha \times \frac{\exp[\lambda_{w_1 w_2 w}]}{Z(h)} \quad (\text{EQ 27})$$

where α is the appropriate α for w and its self-triggered state. Thus, a larger α for the self-triggered state indicates a larger probability boost. As we can see from Table 7, the model appears to be doing something “right.” Note however that although the α s for the self-triggered state may be many times larger than those for the untriggered state, the actual boost in probability is less than what the ratio of the α s for the two states would suggest. For example, although the α s for the word “STOCKS” shows a 15.7 times boost for the self-triggered state vs. the untriggered state, it does not necessarily mean that the probability for “STOCKS” in the self-triggered state is 15.7 times larger. That would be the case if *only* the

word STOCKS were self-triggered. However, as other words are triggered, the boost is lessened because the other boosted α s play a role in the normalization factor $Z(h)$. The maximum entropy model merely says that with the α s given, the average probability in the training data will be matched.

TABLE 7. Some α 's for Self-Trigger and N-grams ME model

Word	No Self-Trigger α	Self-Trigger α
STOCKS	0.326	5.112
POLITICAL	0.463	3.653
FOREIGN	0.462	3.097
BONDS	0.338	3.357
MARKET	0.453	2.013
COMPUTER	0.425	4.341
EARNINGS	0.501	3.097

7.0 Triggers

In our previous work (Lau [17], Lau et al. [18] and Lau et al. [19]), we explored the possibility of modelling triggers within a maximum entropy framework and we met with moderate success. A trigger for a word is a related word whose presence in the document history may influence its probability of appearance. For example, upon seeing the word “PILOTS,” the probability of the word “AIRLINE” is increased. In this case, “PILOTS” is a trigger for “AIRLINE.” In the previous chapter, we discussed self-triggers which can be thought of as the special case of the word itself being the trigger. The present work attempts to expand upon our previous work with triggers by trying to pinpoint exactly what benefit triggers yield, as opposed to self-triggers, the cache, etc.

7.1 Selection of Triggers

It would be nice if we employ every possible trigger-word pair which occurs in the training data and let the model weed out the useless pairs. However, a significant portion of the $|V|^2$ possible trigger-word pairs occurs in the training data, making this an infeasible objective. Another possibility might be to select a reasonable set of initial trigger-word pairs and then modify the set by adding in pairs which improve performance, and removing pairs which contribute little to performance. Della Pietra and Della Pietra [5] presents methods for finding the best “feature” (constraint) from a set of features and for finding the gain from the inclusion of a given feature. Unfortunately, even those techniques do not avail us here because they require running an iteration of GIS for each determination, a prohibitively expensive

computation. Thus, we are presently resigned to a simple selection of a static set of reasonable trigger-word pairs.

We will use the same technique we have used in our previous work. Namely, we will use the *average mutual information* between a trigger candidate and a word as the primary criterion. The average mutual information between a trigger, t , and a word, w , is given by:

$$\begin{aligned}
 I(t;w) = & Pr(t, w) \log \frac{Pr(w|t)}{Pr(w)} + Pr(t, \bar{w}) \log \frac{Pr(\bar{w}|t)}{Pr(\bar{w})} \\
 & + Pr(\bar{t}, w) \log \frac{Pr(w|\bar{t})}{Pr(w)} + Pr(\bar{t}, \bar{w}) \log \frac{Pr(\bar{w}|\bar{t})}{Pr(\bar{w})}
 \end{aligned} \tag{EQ 28}$$

The probabilities are taken over the entire training set. A high average mutual information indicates that the appearance (or lack of appearance) of w is highly correlated with the appearance (or lack of appearance) of t . Furthermore, because we are considering the average mutual information over the entire training corpus, highly correlated pairs which occur only rarely will nevertheless have a low mutual information. This is what we want because a big win on a pair which occurs once out of millions of positions is pretty useless. However, average mutual information is a very simplistic measure because it assumes that the only information we have in predicting the appearance of a word w is the presence or absence of a trigger t in the document history. It does not consider what added information the presence or absence of t provides given that we have other information such as the trigram or even other triggers. As already mentioned, we do have the machinery to compute the specific gain due to a trigger-word pair in the maximum entropy framework but that machinery is computationally infeasible presently so we are resigned to this simple measure.

Using average mutual information, we can select the set of the highest ranking n triggers for a given word. For our experiments, we also want to select *positive* trigger-word pairs, that is, pairs where the presence of t increases the probability of w . To classify a

candidate pair as positive or negative, we can use the *cross-product ratio* (Gokhale [10]) which was also used in our previous work:

$$Cr(t, w) = \frac{c(t, w) c(\neg t, \neg w)}{c(\neg t, w) c(t, \neg w)} \quad (\text{EQ 29})$$

Here $c(t, w)$ is the count of the joint appearance of t and w , etc. A cross-product ratio greater than one indicates a positive correlation.

Using this criteria, we select the top seven triggers for each word. We include the self-trigger in our selection process. Some sample words and triggers for them are given in Table 8. The triggers are listed in order of decreasing average mutual information. In most (over 19K of our 20K vocabulary) cases, the self-trigger is the highest ranking trigger. However, exceptions do exist as in the case of “FOREIGN.” The triggers appear reasonable by subjective criteria. We take our candidate trigger list and evaluate its coverage of the test data. Considering only the first occurrence of each word, approximately 141K positions in the test set were preceded by a trigger whereas approximately 309K positions were not. Needless to say, non-first occurrences are covered by the self-trigger. First occurrence positions represented approximately 47.5% of our test set.

TABLE 8. Top Seven Triggers for Several Words

Triggers	Word
STOCKS	STOCKS
INDEX	
INVESTORS	
MARKET	
DOW	
AVERAGE	
INDUSTRIAL	
POLITICAL	POLITICAL
PARTY	
PRESIDENTIAL	
POLITICS	
ELECTION	
PRESIDENT	
CAMPAIGN	
CURRENCY	FOREIGN
DOLLAR	
JAPANESE	
DOMESTIC	
EXCHANGE	
JAPAN	
TRADE	
BONDS	BONDS
BOND	
YIELD	
TREASURY	
MUNICIPAL	
TREASURY'S	
YIELDS	

7.2 Trigger Constraints

We incorporate triggers into our maximum entropy model with the following constraint functions:

$$f_{t_i^w \rightarrow w}(h, w) = \begin{cases} 1 & t_i^w \text{ occurs in the document history } h \\ 0 & \text{otherwise} \end{cases} \quad (\text{EQ 30})$$

Here t_i^w denotes the i 'th trigger for word w . The corresponding filler constraint function for each word is:

$$f_{-w}(h, w) = T_w - \sum_i f_{t_i^w \rightarrow w}(h, w) \quad (\text{EQ 31})$$

T_w denotes the number of triggers for word w and the summation is over all triggers for w . We combine these constraints along with the n-gram constraints in our model. Note that our model does not take into consideration the frequency of a trigger in the document history nor the distance between a word and its trigger.

7.3 Experimental Results

We train our model as described. As before, we use Good-Turing discounting with the n-gram constraints. For the trigger constraints, we require a threshold count of two. This results in a model with approximately 98K trigger constraints or an average of five triggers per word considering that we have 20K words in our vocabulary. For the GIS training process, we will again use the technique of only updating n-gram constraints on the first iteration to improve convergence. We observe that convergence is much slower than in the self-trigger model (Table 9). We give the α s for several words and their various triggers in Table 10. Once again, the α s look “reasonable” as far as the expected triggering effect is concerned. We do make the following observations though. The triggers for the sample words are listed by order of their average mutual information ranking. That measure does not do the greatest of jobs as evidenced by the lack of a dominant trend of decreasing α s for the trigger list of each word. However, a perfectly reasonable explanation for this may be that triggers tend to co-occur. For example, in the case of the word “POLITICAL,” the trigger “PARTY” may tend to always co-

occur with the word “PRESIDENTIAL.” So although “PARTY” and “PRESIDENTIAL” may have lower α s than “POLITICS,” when they co-occur, the combined α is much larger. Such behavior is not accounted for by the average mutual information measure. Similarly, the seemingly negative trigger “PRESIDENT” for “POLITICAL” may really be a positive trigger but tends to co-occur with a trigger with a high α . We have not pursue this hypothetical explanation much further.

TABLE 9. Training Perplexities for Trigger ME Model

Iteration	Perplexity
1	20010
2	107.5
3	99.4
4	94.8
5	91.6
6	89.2
7	87.4
8	86.1
9	85.0
10	84.1
11	83.4
12	82.9
	stopped training

Despite the seemingly promising α s we have sampled, the trigger model does not result in a significant improvement over the self-trigger model. Recall that the self-trigger model had a test set perplexity of 178.5. The trigger model has a test set perplexity of 177.6, an almost insignificant difference. (However, when we test the two models on another test set, we also obtained a marginal gain for the trigger model so we feel that the improvement, although small, is real. In that test, the self-trigger model perplexity is 135.4 and the trigger model perplexity is 133.5.) It appears that triggers offer little additional benefit beyond the self-trigger model. To be perfectly fair, we lack the patience to train the trigger model for a

few more iterations until absolute convergence — the progress made on the training perplexity is so small that we do not see any breakthrough progress to be made here.

TABLE 10. Some α s for Trigger ME Model

Trigger	α	Trigger	α
Word =	STOCKS	Word =	POLITICAL
filler	0.868	filler	0.84
STOCKS	4.738	POLITICAL	3.939
INDEX	1.913	PARTY	1.575
INVESTORS	1.656	PRESIDENTIAL	1.454
MARKET	1.6	POLITICS	1.588
DOW	1.27	ELECTION	1.194
AVERAGE	1.219	PRESIDENT	0.738
INDUSTRIAL	0.911	CAMPAIGN	1.346
Word =	FOREIGN	Word =	BONDS
filler	0.956	filler	1.004
CURRENCY	2.489	BONDS	4.723
DOLLAR	1.028	BOND	2.514
JAPANESE	1.533	YIELD	1.850
DOMESTIC	2.467	TREASURY	1.416
EXCHANGE	1.270	MUNICIPAL	1.937
JAPAN	1.216	TREASURY'S	1.825
TRADE	1.616	YIELDS	1.503

7.4 Limiting Triggering Distance

One possible explanation for the poor improvement attributable to triggers as compared to self-triggers is that as we get into a document, more and more words are triggered, lessening the impact of triggers on particular words. For example, when we look at the performance of the self-trigger model vs. the trigger model by position within a document, we observe that the trigger model tends to do better at the beginning of documents but worse further on into a document, lending credibility to this hypothesis. This phenomenon is shown

in Figure 2 which gives the percentage improvement of the trigger model over the self-trigger model depending on the position within each document. Also shown in the same figure is the overall percentage of text which falls into each position within the documents. (For example, 4.4% of the text occur within the first 20 document positions.) Furthermore, we observe that most documents have some 5000+ triggered words by the time we get to the end of the document.

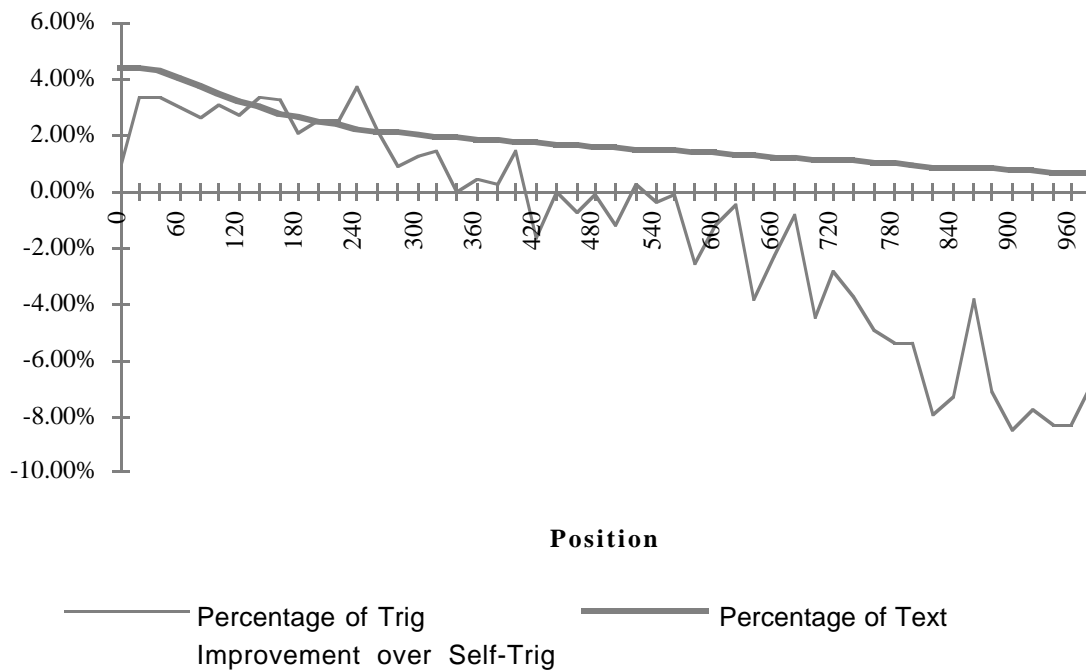


FIGURE 2. Comparison of Self-Trigger and Trigger Model Performance Based on Position within Document on Test Data

To avoid this potential problem, we propose the following modification to our trigger model. Let us limit ourselves to the last n sentences when looking for triggers rather than the entire document history. Table 11 shows the trigger coverage of first occurrences, that is, the

percentage of first occurrences with a trigger in the last sentences. Since we are considering first occurrences only, the effect of self-triggers is not reflected in this table. We observe that if we only consider the last five sentences, we have 90% of the coverage we get with the full document history.

TABLE 11. Trigger Coverage when Triggers are Limited to Last n Sentences

Last n Sentences	Trigger Coverage of First Occurrences
∞ (anywhere in history)	31%
5	29%
4	28%
3	26%
2	23%
1	19%
0 (same sentence only)	12%

We build a model where we only consider triggers in the last five sentences. Since this model differs only a little from the previous trigger model, we take advantage of the following trick to speed up the training process. We take the model parameters from the previous trigger model as the starting point for the training. Given this, the training perplexities are shown in Table 12.

TABLE 12. Training Perplexities for Last 5 Sentence Trigger ME Model

Iteration^a	Perplexity
1	93.8
2	84.9
3	84.2
4	83.6
5	83.2
6	82.9
7	82.7
8	82.4
	stopped training

a. Parameters from Trigger ME Model were used as the starting point for the training.

With the last five sentence trigger ME model, the number of triggered words at the end of most documents falls from 5000+ to somewhere between 3000 and 5000 typically. While an improvement, it is not as dramatic as we would have like. When we test this model on test data, we obtain a disappointing perplexity of 177.4, statistically indistinguishable from the standard trigger ME model. It appears that curtailing the distance of triggers' effects to five sentences instead of the entire document does not improve performance.

8.0 Triggers and Cache

The cache in Kuhn and De Mori [15] has proven to be quite successful although our self-trigger ME model seems to fare better. However, the self-trigger model does not consider the frequency of appearance of a word in the history as the cache does. A natural question to ask is what would happen if we incorporate the full cache into the ME framework? Since it is not much more work to include the trigger constraints as well, we will do so.

8.1 Cache in ME Framework

We do not know of a way to formulate the cache as constraint functions within the maximum entropy framework. However, recall that ME allows for inclusion of a prior distribution, Q (section 4.1). Thus far, we have assumed this distribution to be uniform. We can allow the cache to be this distribution. This approach has the advantage that the cache is a real dynamic cache. That is, if a word, or maybe a bigram, were used with a much higher frequency in test data but not in the training data, we can capture this effect. This contrasts with the self-trigger model, for example, where if a word has not been seen triggering itself in training data, repeated occurrences of that word in test data will not increase its probability.⁶ However, this is also a disadvantage in that the prior distribution in the training data may differ from that in the test data, rendering our faith in the model to ruins. Nevertheless, we decide to train the model and see how it behaves.

6. However, we can include a generic self-trigger constraint that is not specific to any word. Then, this constraint will come into play in the hypothetical scenario mentioned.

8.2 Experimental Results

We build the model as described. For the cache, we included three dynamic distributions and one static distribution. The dynamic distributions were the observed unigram, bigram and trigram frequencies. The static distribution was just an overall unigram distribution of the training data. We require the static distribution to avoid giving an event a zero probability (e.g. the first word in a document). The four distributions are linearly interpolated as follows:

$$Pr_{dyn}(w|h) = 0.1Pr_{stat-1g}(w) + 0.5f_{1g}(w|h) + 0.25f_{2g}(w|h) + 0.15f_{3g}(w|h) \quad (\text{EQ 32})$$

Here, the f s are the dynamic probabilities. For example, $f_{3g}(w|h)$ is the count of w_1w_2w divided by the count of w_1w_2 in the history h with w_1w_2 being the last two words in h . The weights for each distribution were optimized over our held-out data set. However, the optimization considered only the dynamic distribution by itself, that is, not within a ME model. With the trigger cache model, we obtained a perplexity on test data that is 4.1% better than the deleted interpolation model with the standard cache. Recall that the self-trigger model is 3.5% better than deleted interpolation with the cache. As with the trigger model, we find the additional improvement marginal and not worthwhile.

9.0 In-Sentence Bigrams

Recall from Table 11 that 12% of the first occurrences were covered by triggers within the same sentence. This suggests that perhaps a model which includes trigger-word pairs limited to the same sentence may be helpful especially since word relationships within a sentence are probably more robust than trigger-word relationships across many sentences. In this chapter, we explore a model with such *in-sentence bigram* information in addition to self-trigger and n-gram information. The formulation of the in-sentence bigram constraints is analogous to that of the trigger constraints. However, since we will be including self-trigger constraints, our in-sentence bigram constraints will only be active for words which have not yet occurred. We feel that the self-trigger is adequate for words already seen.

9.1 Selection of In-Sentence Bigrams

Since we are limiting ourselves to be within a sentence, we will have to have many more in-sentence bigram pairs than we did trigger-word pairs. We may very well want to consider all possible pairs if possible. Unfortunately, this is too many pairs to consider so we have to trim the list down somehow.

First, we consider that the most frequent words, like “the,” would not be good candidates for bigram pairs. We feel that for these words, the n-gram model provides much more information than we can hope to obtain from in-sentence bigrams. The most frequent ten words are shown in Figure 3.

We also require a certain amount of average mutual information between the words in each in-sentence bigram pair. We find that a threshold of one microbit leaves a manageable number of pairs to work with.

Finally, we require that the pairs have a high enough count in the training data. Table 13 gives some statistics on what happens as we vary the threshold from three to five. We decide to use a threshold of four.

THE
OF
TO
A
AND
IN
THAT
FOR
ONE
IS

FIGURE 3. Top Ten Words in Training Data

TABLE 13. In-Sentence Bigram Constraints with Varying Thresholds

Threshold	Number of Constraints	Maximum Number of Constraints Active	Average Number of Constraints Active
3	565771	51	3.74
4	434212	51	3.21
5	338772	50	2.81

9.2 Experimental Results

With up to 51 constraints active at one time, we expect very slow convergence since each constraint will have a very small weight. Needless to say, we will use the technique of updating only n-gram constraint parameters on the first iteration. We will go further and only update n-gram and self-trigger constraints on the second and third iterations in an attempt to

get us more converged on the more power constraints before working on the in-sentence bigrams. As we can see from Table 14, convergence is indeed extremely slow for this model.

TABLE 14. Training Perplexities for In-Sentence Bigram Model

Iteration	Perplexity
1	20010
2	107.5
3	96.0
4	90.6
5	87.6
6	85.8
7	84.4
8	83.3
9	82.4
10	81.6
11	80.9
12	80.3
13	79.8
14	79.4
15	78.9
16	78.6
17	78.2
18	77.9
19	77.6
20	77.3
21	77.1
22	77.0
23	76.8
24	76.7
	stopped training

When we evaluate this model on test data, we discover a very disappointing result. The test set perplexity is 174.8. While this is measurably better than the self-trigger model which has a test set perplexity of 178.5, we went through a lot of extra work in terms of training and we have a bigger model, an extra 434K constraints, to gain that 2% advantage. It would appear

that in-sentence bigrams do not capture information which is powerful and robust enough to win big over the self-trigger model.

10.0 Conclusions, Related Work and Future Directions

10.1 Summary and Conclusions

In this work, we explored various adaptive statistical language models and their performance in the Wall Street Journal domain. We started by looking at the possibility of creating topic specific models. We looked into a method of statistically clustering documents into topics making use of the Kullback-Leibler distance between unigram distributions, building deleted interpolation trigram models for them and evaluating their performance on test data. Our actual initial evaluation was rigged insofar as we identified the topics for the documents in the test set before hand. Nevertheless, the improvement in perplexity was a meager 8.8% leading us to abandon further work along these lines.

Next, we introduced the maximum entropy framework for building statistical language models. We have discovered that a ME n-grams only model with Good-Turing discounting performs better than the deleted interpolation trigram model even though our ME model was smaller in size. Without the discounting or with an alternate method of smoothing known as fuzzy maximum entropy, the results were less favorable.

With the discounted ME n-grams model as our backbone, we then added self-trigger information. This results in a significant improvement, 20.7%, over the deleted interpolation trigram model. Furthermore, this model even surpasses the trigram model with the cache.

Our next step was to add triggers to the model. Our triggers were selected on the basis of average mutual information between the candidate trigger and word. We took the top seven triggers, including self-trigger, for each word. This model results in an almost insignificant

improvement over the self-trigger model. We hypothesized that the poor performance may be due to the large number of words being triggered and hence a dilution of the triggering effect. Examination of the model in action supported this theory, so we tried limiting the triggering effect to five sentences instead of the entire document. This model was even more disappointing. For all intents and purposes, this addition did not improve performance.

We considered the incorporation of the cache into the trigger model. Unfortunately, we did not know how to directly model the cache as maximum entropy constraints, so we used the cache as the prior distribution in the ME framework. This did not lead to any significant improvement.

Finally, we considered in-sentence bigrams. We constructed a model with n-grams, self-triggers and in-sentence bigrams. Selection of the in-sentence bigrams involved setting a minimum average mutual information between the words in the in-sentence bigram and requiring a count of four or more in the training set. Our in-sentence bigram model was 2% better than our self-trigger model. However, we added quite a bit more information and paid a much higher price in terms of training time to achieve that gain.

The performance of the various pure ME models and of the benchmark deleted interpolation models is summarized in Table 15. We feel that maximum entropy is a viable framework for constructing language models. However, we have not found any information, except for n-grams and self-triggers, to be particularly useful in terms of improving performance.

TABLE 15. Comparison of Various Maximum Entropy and Deleted Interpolation Models

Model	Amount of Information	Perplexity
ME in-sent bigram + self-trig w/threshold of 2	1.2M	174.8
ME 7 trigs in last 5 sent w/threshold of 2	888K	177.4
ME 7 trigs w/threshold of 2	888K	177.6
ME self-trigger w/threshold of 2	809K	178.5
DI w/dynamic cache	3.5M + cache model	186.3
DI	3.5M	225.1
Fuzzy ME w/threshold of 2	790K	230.1
ME w/threshold of 2	790K	247.9
ME w/threshold of 3	476K	253.5
DI w/threshold of 3	790K	262.7

10.2 Related Work

The work discussed here builds upon previous work of the author’s along with others. We feel that it is appropriate to compare the results here with the ones there. Lau et al. [18] reported results for the trigger model which are slightly better than those obtained here. However, there are two differences between the experiments. Whereas the comparison is against the deleted interpolation trigram model here, it was against a backoff trigram model there. The deleted interpolation model is generally slightly better, making it harder to beat. Also, the test set used there was slightly “easier” than the one here. In both cases, training data was taken from early ‘87. There, the test set was taken half from late ‘88 and half from late ‘89 whereas here, the entire set is from late ‘89. Considering that topics change over time, it is not surprising that the results there were slightly better. Lau [17] and Lau et al. [19] reported somewhat worse results than those here. In that work, rawer Wall Street Journal data was used as opposed to the cleaned up data provided by Doug Paul of Lincoln Labs. Furthermore, a 50K case sensitive vocabulary was used as compared with the standard 20K case insensitive

vocabulary here and verbalized punctuation (VP) was used as opposed to non-verbalized punctuation (NVP). With VP, we would expect less of an improvement because none of the additional information we have added is expected to help with predicting the punctuation.

Ronald Rosenfeld is pursuing further work involving maximum entropy and adaptive language models at Carnegie Mellon. He has recently reported (Rosenfeld [20]) improvements of 32% and more over the backoff trigram model. He attributes much of his further improvement to the inclusion of constraints on what he calls “distance-2” trigrams and bigrams, which are basically trigrams which use the second to last two positions and bigrams which use the second to last position as the context as opposed to the last two and last positions, respectively. We observe that this may be analogous to including a subset of four-gram information. Our experiments here are limited to a small 5 MW training set. We believe that the improvements will carry over to the full training set. Roni has confirmed such in his work. Furthermore, he reports that when implemented into CMU’s speech recognizer, the perplexity improvements do translate into actual error rate reduction.

10.3 Future Directions

While the maximum entropy framework is very general and appears to be quite effective given that the information being modelled is robust and powerful, there is still the problem of the immense computational bottleneck posed by the GIS training process. More progress needs to be made in finding a faster way to train ME models.

Other than self-triggers, we have not hit upon any other great adaptive element to include in our model. Even Rosenfeld’s further dramatic improvement mentioned in section 10.2 is primarily due to n-gram like information. We need to continue to explore other information elements which can be exploited.

Although the longer distance information we have experimented with qualifies as “adaptive” in the literature, we must reemphasize that many of our models are not truly adaptive in that these models attempt to “learn” relationships between the current history and the next word on training data only. Not much work has been done in truly adaptive models which may learn changing language characteristics over time although many speech recognition and some pen recognition systems employ a user cache which basically boosts the probabilities of n-grams frequently employed by a user.

References

- [1] Bahl, L. R., Jelinek, F., and Mercer, R. L., “A Maximum Likelihood Approach to Continuous Speech Recognition,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(2): 179-190, 1983.
- [2] Csiszár, I., “I-Divergence Geometry of Probability Distributions and Minimization Problems,” in *The Annals of Probability*, 3(1): 146-158, 1975.
- [3] Darroch, J. N. and Ratcliff, D., “Generalized Iterative Scaling for Log-Linear Models,” in *The Annals of Mathematical Statistics*, 43(5): 1470-1480, 1972.
- [4] Della Pietra, S., *personal communications*, 1993.
- [5] Della Pietra, S. and Della Pietra, V., *Statistical Modelling by Maximum Entropy, monograph to appear*, 1993.
- [6] Della Pietra, S. and Della Pietra, V., *personal communications*, 1993.
- [7] Della Pietra, S., Della Pietra, V., Mercer, R., and Roukos, S., “Adaptive Language Modeling Using Minimum Discriminant Estimation,” in *Proceedings of ICASSP ‘92*, I-633-636, San Francisco, March 1992.
- [8] Franz, M., *personal communications*, 1993.
- [9] Franz, M. and Roukos, S., *personal communications*, 1993.
- [10] Gokhale, D. V. and Kullback, S., *The Information in Contingency Tables*, Marcel Dekker, Inc., New York, 1978.
- [11] Good, I. J., “Maximum Entropy for Hypothesis Formulation, Especially for Multidimensional Contingency Tables,” in *The Annals of Mathematical Statistics*, 34: 911-934, 1963.
- [12] Good, I. J., “The Population Frequencies of Species and the Estimation of population Parameters,” in *Biometrika*, 40(3, 4): 237-264, 1953.
- [13] Jelinek, F., Merialdo, B., Roukos, S., and Strauss, M., “A Dynamic Language Model for Speech Recognition,” in *Proceedings of the Speech and Natural Language DARPA Workshop*, 293-295, February 1991.

- [14] Katz, Slava M., “Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer,” in *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35(3): 400-401, March 1987.
- [15] Kuhn, R. and De Mori, R., “A Cache-Based Natural Language Model for Speech Recognition,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6): 570-583, 1990.
- [16] Kullback, S., *Information Theory and Statistics*, John Wiley & Sons, New York, 1959.
- [17] Lau, R., *Maximum Likelihood Maximum Entropy Trigger Language Model*, S.B. thesis, Massachusetts Institute of Technology, May 1993.
- [18] Lau, R., Rosenfeld, R., and Roukos, S., “Adaptive Language Modelling Using the Maximum Entropy Approach,” in *ARPA Human Language Technologies Workshop*, March 1993.
- [19] Lau, R., Rosenfeld, R., and Roukos, S., “Trigger-based Language Models Using Maximum Likelihood Estimation of Exponential Distributions,” in *Proceedings ICASSP '93*, Minneapolis, April 1993.
- [20] Rosenfeld, R., *personal communications*, 1994.
- [21] Rosenfeld, R., *Adaptive Statistical Language Modelling: A Maximum Entropy Approach*, Ph.D. thesis proposal, Carnegie Mellon University, October 1992.
- [22] Rosenfeld, R. and Huang, X., “Improvements in Stochastic Language Modeling,” in *Proceedings of the Speech and Natural Language DARPA Workshop*, Morgan Kaufmann Publishers, San Mateo, CA, February 1992.
- [23] Roukos, S., *personal communications*, 1993.

7.2	Trigger Constraints.....	46
7.3	Experimental Results	47
7.4	Limiting Triggering Distance.....	49
8.0	Triggers and Cache	53
8.1	Cache in ME Framework	53
8.2	Experimental Results	54
9.0	In-Sentence Bigrams.....	55
9.1	Selection of In-Sentence Bigrams.....	55
9.2	Experimental Results	56
10.0	Conclusions, Related Work and Future Directions	59
10.1	Summary and Conclusions.....	59
10.2	Related Work.....	61
10.3	Future Directions.....	62
	References.....	64