The Cost of Natural Language Modeling

A Dissertation Proposal

by Matthew V. Mahoney
Computer Science Dept., Florida Tech.

Abstract

The purpose of this research is to estimate the size (memory requirement) of the language model needed for artificial intelligence (AI). Recognizing that AI is unsolved and may remain so for a while, my approach is to compare model size with performance (error rate) in a text prediction task, and extrapolate to the level of human performance. The ability to model language, i.e. estimate the probability of observing any given text string, is central to all natural language processing tasks, including passing the Turing test for AI. Knowing the size of the model would allow us to estimate when (or if) the AI problem will be solved, and how much it would cost.

My work will involve two major areas of research. First, I plan to build a language model that integrates the known character, lexical, semantic, and syntactic constraints, components that are usually studied in isolation, or at most, two at a time. Second, I plan to refine the estimated entropy of written English, a number now known so poorly that we cannot even say whether the AI problem has already been solved.

1. Introduction

Natural language would be an ideal user interface to computer systems, but very few software developers today would even consider the idea. Since the 1950's, researchers have known about problems like ambiguity and complexity in natural languages such as English, but severely underestimated the enormity of these hurdles. Even then, we had systems such as Green et. al.'s (1961) expert on 1959 baseball statistics that could handle input such as "How many games did the Yankees win in July?" At the time it was believed that AI could be solved with just a little more effort, for example, Newell and Simon's (1961) logic proving program described in a paper titled "GPS: A Program that Simulates Human Thought." Such views now seem silly in retrospect.

The problem is: we still don't know how big the problem is. History doesn't help very much in predicting whether it will ever be solved. In some fields, such as speech recognition, we have been making slow but steady progress for decades (Rudnicky, Hauptmann and Lee, 1994), ever since Uhr and Vossler (1963) demonstrated a speech system based on neural pattern recognition. Still, on standard benchmarks such as *switchboard*, we have not been able to do better than a 35% word error rate (Stolcke, 1997). In information retrieval, progress on the TREC benchmark (Harman, 1995) has been stuck at a 50% error rate (average of recall and precision) since 1992. The situation is even worse for systems without standard evaluation techniques, such as language translation (Knight, 1997). We do not seem to have made much progress since the ill-fated U.S. government Russian-English project of 1959-62 (Borko, 1967). Alta Vista (www.altavista.digital.com) translates "The spirit is willing, but the flesh is weak" to Spanish and back as "The alcohol is arranged, but the meat is weak".

1.1. AI as a Language Modeling Problem

Some, such as Penrose (1990) argue that AI cannot be solved: intelligence requires a consciousness or self awareness that is not computable on a Turing machine (a mathematical abstraction of a computer with infinite memory). I am more prone to agree with Turing's (1950) view that AI is strictly behavioral: if the average person is unable to distinguish a machine from a human, based only on written communication with it, then the machine has AI. Turing proposed a test for AI which he called the *imitation game*, now better known as the *Turing test*. He offered the following hypothetical example:

> Q: Please write me a sonnet on the subject of the Forth Bridge.
> A: Count me out on this one. I never could write poetry.
> Q: Add 34957 to 70764.
> A: (Pause about 30 seconds and then give as answer) 105621.
> Q: Do you play chess?
> A: Yes.
> Q: I have K at my K1, and no other pieces. You have only K at K6 and R at R1. It is your move. What do you play?
> A: (After a pause of 15 seconds) R-R8 mate.

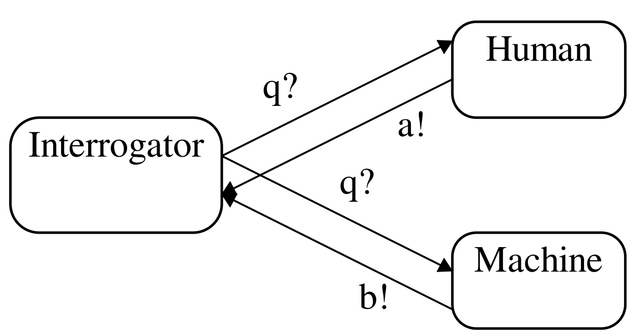

Fig. 1.1. The Turing test for AI. The machine passes if the interrogator cannot distinguish it from the human, based on the responses (a and b) to a series of questions (q).

The problem is equivalent to language modeling: given a string *x* (such as the above dialogue), what is the probability P(x) of observing x in human communication? Turing made two basic assumptions in designing the imitation game: first, that humans are intelligent, and second, that they recognize intelligence in others, even when restricted to written communication. In language modeling terms, the first assumption says that humans generate strings with probability P(x). The second says that given a string *x*, humans can use P(x) to evaluate whether the source was intelligent (i.e. likely, given that the source is human).

If a machine knows P(x), then it could pass the Turing test. That is because knowing P(x) for any x implies knowing $P(a|q) = P(qa)/P(q)$, the probability of *a* given *q*, for any concatenation of two strings, $x = qa$. If an interrogator asked a question *q* and got two responses, *a* and *b*, one human and one from a machine, then he or she could evaluate whether $P(a|q) < P(b|q)$, indicating that *a* is the machine, or $P(a|q) > P(b|q)$, indicating that *b* is the machine. If the machine knows P, then $P(a|q) = P(b|q)$, and no distinction is possible. The machine passes the Turing test.

1.2. Applications of Language Models to AI

The previous result is theoretical, because no machine has passed the Turing test. In practice, less than perfect language models are used to correct (partially) the output of most natural language systems. In speech recognition, optical character recognition (OCR), handwriting recognition, language translation, and proofreading, it is often the case that the system must decide between two or more outputs because of noise or ambiguity in the input. For instance, an ambiguous utterance could be interpreted as *recognize speech* or *wreck a nice beach*, or a raster-scanned image could be *mail* or *rnai1*. Often the choice is obvious to humans but obscure to machines. A good language model could reduce the error rate by evaluating the probability of each choice, i.e., P(*mail*) and P(*rnai1*), and choosing the text string with the highest probability.

The basic approach, outlined by Knight (1997) is to decompose a system into a front end model and a language model. Any of the systems above can be described by an overall model P(y|x), where x is the input (speech, scanned text, pen movements, foreign text, or misspelled text), and y is the output text. The output is the value of y that maximizes P(y|x), the probability of y given x. Knight's approach is to decompose the model using Bayes law:

$$P(y|x) = P(x|y)P(y)/P(x)$$

where P(x|y) is the "reverse" model (speech synthesizer, printer, etc.), and P(y) is the language model. Since we are only interested in maximizing y, we can drop the fixed term P(x), and maximize P(x|y)P(y). In speech recognition, P(x|y) is called the *acoustic* model, and is normally trained by having a speaker read some text (y) aloud, and recording the resulting sounds (x). The language model P(y) is typically a trigram model; the probability of the next word depends only on the previous two words, and is estimated by counting word triples in a large corpus. Nearly all work in speech recognition uses approach similar to this.

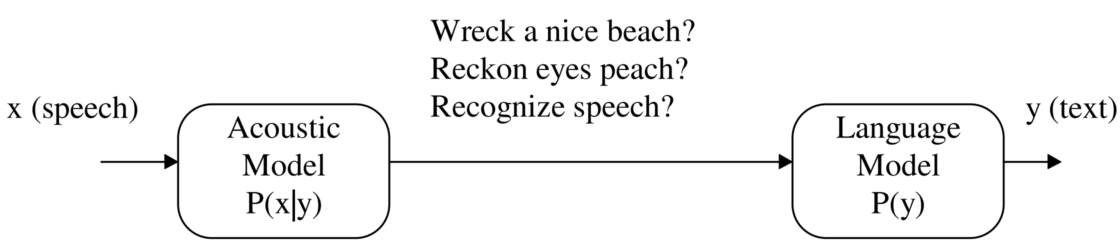

Fig. 1.2. Decomposing a speech recognition system, P(y|x), into an acoustic model and a language model.

In language translation, say, French to English, decomposing the model allows us to train P(y) on the large quantity of readily available monolingual text which would otherwise be of no use (Knight 1997). In this model, x = French, y = English, and the decomposition of P(y|x) is to P(x|y)P(y) as before. There is no disadvantage to decomposition, as it is no harder to create the bilingual model P(y|x) (French to English) as it is to produce P(x|y) (English to French).

Teahan et. al. (1998) showed how a language model could increase OCR accuracy from 96.3% to 96.9% in one experiment, using essentially the same approach. The reverse model P(x|y) gave the probability that the scanner would make an error, for instance, P(*thc* | *the*), reading *the* as *thc*. The language model P(y) was based on 5-grams, i.e. character probabilities are based on only the preceding four characters and estimated by counting 5-grams in a large corpus. Teahan and Cleary (1997) outlined how decomposition could be used to correct spelling and speech recognition errors and solve simple substitution ciphers using similar language models.

1.3. Proposal Outline

The purpose of this dissertation proposal is to investigate the cost of natural language processing (NLP). Specifically, we want to know the size (or total entropy) of the knowledge base needed to pass the Turing test for AI, currently an unsolved problem. Other costs of an AI system, such as speed, memory, and development effort, would obviously depend on the goal of the project and on its implementation, but without a size estimate, we could not even begin to estimate these other costs. The size estimate is currently unknown, probably because it is implicitly assumed that we cannot know how hard a problem is until we know how to solve it.

My approach does not involve solving the AI problem. Rather, I investigate the empirical relationship between knowledge base size and performance in a language modeling task, and project these results to the level of human performance. Language modeling, the ability to estimate the probability of observing any given string in natural language text, is central to all NLP tasks including passing the Turing test, and is measurable using text prediction or compression tests. Current research with a large number of statistical language models suggests that 10^8 to 10^{10} bits (training set size after compression) are needed to solve AI, but there are two major sources of uncertainty. First, the large number of disparate models and training sets evaluated by different researchers makes it difficult to observe a trend, and second, techniques for measuring human performance in modeling tasks are poor. My research addresses both of these issues. First, I develop a language model that integrates character, lexical, semantic, and syntactic constraints (which are usually studied in isolation) and observe the effects of model size on compression of a single corpus at various sizes. Second, I refine techniques used to estimate the entropy of text relative to humans, and use them to measure the same test corpus. This is illustrated below.

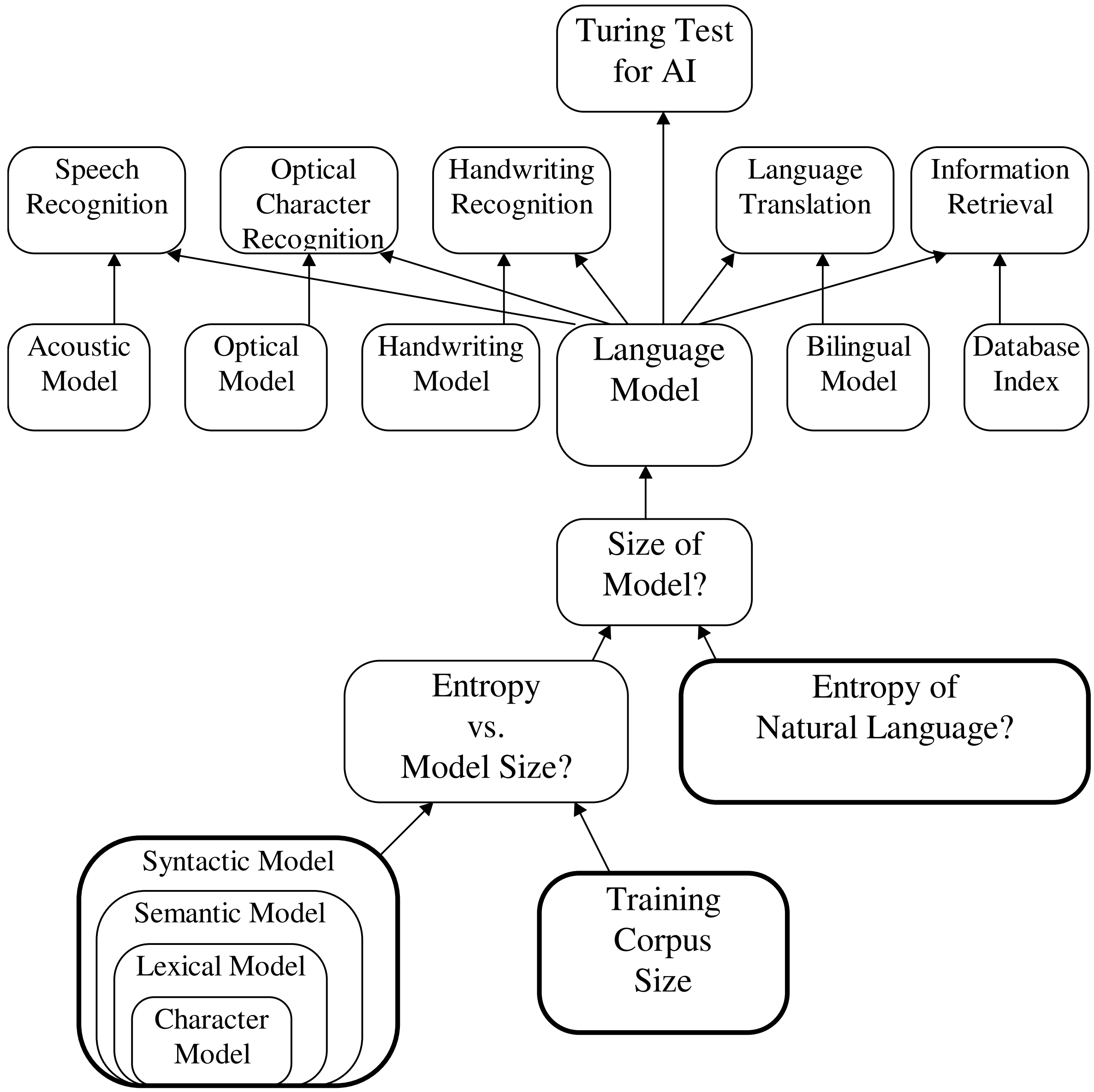

Fig. 1.3. Outline of the proposal. The problems to be solved are shown in bold outline. Knowing the size (memory requirement) of the human language model would allow us to better esimate the cost of solving many AI problems.

This paper is organized as follows. In section 2, I review related work in model evaluation techniques (human and machine) and machine performance as a function of model size. In section 3, I outline the approach, which is to build a model and measure its performance (compression ratio) as model size is varied, and compare these results with humans in a text prediction task. Because of the complexity of the language model, I develop it in four steps, each corresponding to a stage of language development in children: character, lexical, semantic, and syntactic (sections 4-7). In section 4, I develop a neural network implementation of a character or n-gram model, which knows nothing about words. I show that this implementation is practical, i.e. competitive with current techniques in data compression, in speed, memory, and compression ratio, but unlike current techniques, can be extended to higher level models. In section 5, I develop a lexical model, which learns how to segment the input text into words (even if spaces are removed) using nothing more than n-gram statistics. In section 6, I develop a semantic model, which learns word meanings by the tendency of related words to occur near each other in the input. In section 7, I develop a syntactic model, which learns a grammar from the tendency of parts of speech (nouns, prepositional phrases, etc.) to appear in the same immediate context. In section 8, I compare the now completed model with humans in a text prediction task, using improved techniques to relate prediction with compression. In section 9, I conclude by summarizing potential contributions and the work remaining to be done to complete the dissertation.

2. Related Work in Model Evaluation and Estimates of Model Size

We have so far seen that AI can be reduced to a language modeling problem for purely text based systems, and also that language models are of great practical importance in AI systems that accept input other than text. Next we attack the problem of estimating the size or memory requirements of the model needed to achieve AI.

In sections 2.1 and 2.2 we estimate upper and lower bounds by examining the smallest known working model – the human mind – and sampling some of the largest members of the set of non-working models – which would be every machine model ever built. This narrows our range to about 10^8 to 10^{14} bits. Then in 2.3 we review evaluation techniques for statistical models that learn from a training set of a corpus of natural language text, the most successful type of model to date. Statistical models have well defined measures of both performance (perplexity, entropy, or compression ratio – all equivalent measures), and model size (training set size after compression). In 2.4 we review equivalent performance measures for humans (but not size measures – that is what we are trying to find out) In 2.5 we plot performance against model size for a large number of statistical models, and compare their performance with humans. These results suggest a human model size of around 10^8 to 10^{10} bits.

2.1. Approximations of the Human Model – An Upper Bound on Size

One approach to estimating the size of the human language model is to ask how much people can know. This can neither exceed the memory capacity of the human brain, nor can it exceed the amount of information inherited in our genes plus all of the input received over a lifetime. Although we can estimate these numbers, the results unfortunately differ by a factor of a million depending on which approach we take.

2.1.1. Memory Capacity of the Human Brain

The cerebral cortex has about 10^{10} neurons, each with "hundreds" or "thousands" of synapses connecting them (Crick and Asanuma, 1986; Kuffler, Nicholls, and Martin, 1984). Most neural network models use some variation of Hebb's (1949) rule, in which memory is stored in the synapse (Rumelhart and McClelland, 1986), although this model has not been verified experimentally in the brains of vertebrates (see Beardsley, 1999). In most simulations, each synapse is represented by a few bits of memory, although Hopfield (1982) found that associative networks (fully connected and symmetric) store 0.15 bits per synapse. These data suggests that the human brain has a memory capacity of roughly 10^{12} to 10^{14} bits.

2.1.2. Memory in Psycholinguistic Models

Osterson, Stob, and Weinstein (1986) define language learning in the formal sense (where a language is a discrete set of strings rather than a probability distribution) as the ability to derive an algorithm for deciding the language after a finite set of positive examples. They argue that their model is appropriate because children appear to learn natural language that way, without negative examples. If we accept this view, then the human language model can be no larger than the amount of information to which a person is exposed in a lifetime. If we count only listening and reading, and estimate 150 words per minute, 3 hours per day for 20 years, then the model size for an adult would be about 2×10^8 words. If this model is stored in compressed form using Shannon's (1950) estimate of 0.6 to 1.3 bits per character at 5.5 characters per word (including spaces), then we have a model size of about 10^9 bits. Landauer (1986) also estimated the capacity of human long term memory to be 10^9 bits based on known rates of learning and forgetting.

In a more detailed study of language learning, Landauer and Dumais (1997) noted that only a fourth of the typical vocabulary of a high school graduate (40,000 to 100,000 common words, plus at least an equal number of proper nouns) appears in speech, and that almost all of the spoken vocabulary is learned by age 12. Children at this age continue to add to their vocabulary at the rate of 10 to 15 words per day, reading an average of 50 70-word paragraphs, about 15 minutes in school and 15 minutes elsewhere. This leads to an estimate of 1.2×10^8 bits if we ignore both speech and nonverbal learning.

We might get around the disparity between the neurophysiological and psycholinguistic approaches by arguing that the latter does not take into account any knowledge that might be inborn. But this cannot exceed the information content of the human genome, which is 6×10^9 bits (2 bits per base pair), or 2×10^8 bits if we exclude the 97% that does not encode protein (see Waterman, 1995).

2.2 Some Large AI Applications – A Lower Bound on Size

Having estimated an upper bound perhaps as high as 10^{14} bits for a working model, we can look at what doesn't work to estimate a lower bound. For many models, this is difficult because information on size is often unavailable, and even when it is, we can't really say that the model "doesn't work". All of these models are useful but there is usually no standard way of evaluating or comparing them. A few examples will illustrate the problem.

System	Model Type	Size	Reference
Webster's Dictionary	Lexical, semantic	55,000 entries, 8M characters	Webster, 1964.
Wordnet lexicon	Lexical, semantic	96,500 entries, (37 MB download)	Miller et. al. 1993, www.cogsci.princeton.edu/~wn/
Toshiba English-Japanese translator	Lexical, syntactic	130,000 words, 100,000 language rules	Newsbytes, 1988
Cyc knowledge system	Semantic	$>10^6$ first-order logic rules	Guha, 1994; Whitten, 1994, www.cyc.com
LSA - TOEFL test taker	Semantic	4.6M words	Landauer and Dumais, 1997
Ask Jeeves Internet search engine	Semantic	$>10^8$ web pages	www.ask.com

Table 2.2. Some large language models

Webster's is a 1700 page printed dictionary (lexical model) with definitions (semantic model). If we use Shannon's (1950) estimate of 0.6 to 1.3 bits per character (bpc), nominally 1 bpc, for the entropy or information content of written English, then we have 8×10^6 bits or 145 bits per word.

Wordnet is a machine readable dictionary with a hierarchy of synonyms, antonyms, hypernyms/hyponyms (*is-a* relations) and meronyms (*has-a* relations). The download size is 37M bytes as a *zip* file. Zip files compress text to about 3 bpc, so the theoretical compression is probably closer to 10^8 bits. This is about 7 times as much space per word, about 1000 bits.

No information on the performance or total size of the Toshiba translator was available. If we assume that each word and each rule is equivalent to a dictionary entry, then its size is 3×10^7 to 2×10^8 bits.

Cyc is probably the most ambitious attempt ever at encoding all human knowledge by hand, taking 11 years (1984 to 1995).to develop, although it has still not been incorporated into any products. It encodes common sense knowledge, for example, that brooms have handles or that countries have capitols, using an extension of first-order logic called *cycl*. Its current size is not disclosed. Whitten reported that the size passed one million hand-coded rules in 1990. If we assume 2 million rules, each equal to a dictionary entry, then the size is 10^8 to 10^9 bits.

LSA ranks word pair similarity according to how often they appear in the same region of text. On the synonym portion of the TOEFL (Test of English as a Foreign Language), it scored 51.5 correct out of 80 multiple choice questions, compared to an average score of 51.6 for students taking the test, a score that qualifies for admission to most universities. However, LSA does not solve the AI problem because it does not model syntax -- it is a "bag of words" model. LSA will choose "*the dog barked*" over "*the dog said*" because *barked* is closer to *dog*, but it sees nothing wrong with *"barked dog the*". The training text was Grolier's Encyclopedia, 4.6M words, or 2.5×10^7 bits.

Ask Jeeves is an Internet search engine that accepts natural language questions (although it flunked my informal Turing test). It is a meta-engine, combining results from several other search engines. It's potential knowledge base is the entire Internet. Peterson (1998) estimated that search engines cover 4% to 34% of 320M pages as of Dec. 1997. If we assume 1000 characters per page and disregard the historical doubling of the size of the Internet every year since 1985, then the model size would be 3×10^{11} bits.

2.3. Evaluating Machine Language Models

Our effort to estimate an upper bound on the AI problem (10^8 to 10^{14} bits) is confounded by our lack of understanding of the human brain, and how much nonverbal knowledge contributes to language. Do we know that the sky is blue because we saw it or because we read it? For the lower bound (10^6 to 10^{11} bits), we have no way to compare different AI applications. Just because a search engine fails the Turing test doesn't mean it is inferior to humans. Try searching the Internet without one. Could a search engine pass the test by using a different algorithm?

If we can isolate the language model from the application, then the standard performance test is entropy (in the data compression community), or perplexity (in the speech recognition community), the two groups doing most of the research in modeling. Given a corpus x of length n, and a model P, the entropy is estimated as $H = 1/n \log_2 1/P(x)$, where n is in characters and H is expressed in bits per character. Shannon (Shannon and Weaver, 1949) showed that the entropy is the lower bound to which x can be compressed without loss of data, and that this occurs only when the model is exact. In practice, compression to within one bit of the optimal value can be achieved by arithmetic encoding (Bell, Witten, and Cleary, 1989), so text compression is a good measure. Perplexity is another measure of entropy, $PP = 2^H$, where n is usually given in words rather than characters. In either case, lower values of H or PP (or higher values of P(x)) are better.

One might object at this point that text compression has nothing to do with AI, because it doesn't simulate something that people can do well. The reason that people can't compress text is because both the compressor and decompressor have to agree on a coding that depends on the model, P. No two people use exactly the same model, and no single person uses exactly the same model at two different points in time. Compression is not considered AI only because we cannot make exact copies of the human mind.

Another objection to entropy as a measure of goodness is that duplicating the human model is not the optimal strategy for passing the Turing test. Suppose that an interrogator asks a question *q* and gets two responses, *a* and *b*. If the interrogator chooses *a* as human and *b* as machine on the basis of $P(a|q) > P(b|q)$, then the optimal machine response to question *q* is not the random distribution $P(b|q)$, but the single most likely response for a human, $\max_b P(b|q)$. These are two different models. Minimizing entropy finds $P(b|q)$, the exact model, but minimizing error rate find $\max_b P(b|q)$, a deterministic model, which is the one we want.

However, an interrogator is free to choose any strategy for picking the human, not just $P(a|q) > P(b|q)$. An exact model will always succeed regardless of strategy, because the human and machine would be indistinguishable. Whether this is the right goal is a different story. We do not want a computer to inject random errors (such as the response to Turing's addition problem) in order to fool an interrogator. We want machines to always give the right answers.

Bimbot, El-Beze, and Jardino (1997) proposed character guessing accuracy as an alternative measure. Given a string $x = x_1x_2...x_n$ of n characters, we have $P(x) = P(x_1)P(x_2|x_1)P(x_3|x_1x_2)...P(x_n|x_1x_2...x_{n-1})$ by the chain rule. Then the model P could be evaluated by the average number of guesses per character needed to predict the string x, proceeding to the next character after each correct guess. However, this measure discards information about the model, as different distributions could result in the same guessing sequence.

Iyer, Ostendorf, and Meteer (1997) tested the widely held assumption that entropy/perplexity is the best measure of goodness of a language model for speech recognition. They did indeed find a strong correlation with word error rate when the training data and test data are taken from the same domain, which is the usual case, but for cross domain tests they found that a decision tree using more detailed model features gave a better fit.

2.3. Evaluating the Human Model

Minimizing the entropy of a language model is guaranteed to find one that will pass the Turing test, although it may not be the best model. In practice, reducing entropy reliably improves word error rate in speech recognition. These are both good reasons to use entropy to evaluate machine language models, which we can do by inserting the model into a data compressor and measuring the compression ratio (bpc, bits per character). How do we evaluate the human language model, if people can't do text compression?

Shannon (1950) used a text prediction task to estimate the entropy of written English. He took random 100-character long samples from *Jefferson the Virginian*, using a 27-character alphabet (A-Z and space) and asked subjects to guess the next character. Subjects were allowed to use dictionaries and character frequency tables (1-, 2-, and 3-grams) as aids. Shannon proved that the entropy bounds (in bpc) are:

$$\Sigma_r\, r(p_r - p_{r+1}) \log_2 r \leq H \leq \Sigma_r\, p_r \log_2 1/p_r \qquad (2.3)$$

where p_r is the probability that it will take r tries to guess the next character. Shannon found that $p_1 = 0.8$, $p_2 = 0.07$, etc., meaning that subjects guessed correctly on the first try 80% of the time, on the second try 7% of the time, and so on. Using this method, he obtained bounds of 0.6 to 1.3 bpc. Shannon found similar (but slightly higher) entropies for newspaper articles, poetry, and technical literature.

The reason for the uncertainty in Shannon's estimate is that converting a probability distribution to a ranking discards information. Cover and King (1978) and Tan (1981) eliminated this uncertainty by having subjects assign the probabilities directly in a gambling game, but there is another problem: people do not bet rationally (Schwartz and Reisberg, 1991, pp. 552 ff.). People tend to overestimate the probability of unlikely events, which would explain the popularity of both lotteries and insurance. McDonald (1998) showed that people will accept supernatural explanations for unusual events because, when confronted with a choice of explanations including some very unlikely ones, they tend to assign equal probabilities to each. If this behavior occurs when assigning probabilities to letters, then the entropy would be overestimated. Cover and King's measurement of 1.3 bpc was obtained by combining the results of several subjects; individual results ranged from 1.3 bpc to 1.7 bpc, outside the bounds established by Shannon.

2.4. Size vs. Performance in Statistical Language Models

We now have the tools needed to estimate the size of the human language model. Given a statistical model, we can evaluate both its performance (compression ratio) and its size (training text after compression). Performance is measured in bits per character (bpc), and is estimated by compressing (or estimating the probability of) a test string *y* which is held out from the training corpus.

$$\text{performance} = H = 1/|y| \log_2 1/P(y) \qquad (2.4.1)$$

where $|y|$ is the length of the test string, and P(y) is estimated by the model.

The *size* of a language model is defined here as the lower bound on the amount of memory needed to represent the knowledge or algorithm used to estimate P. Because software costs more than hardware, we are more interested in how much *information* a model uses, as opposed to how much *memory*. The memory requirement of a model will never be less than the information content, but the actual value will depend on the implementation. For statistical models, the amount of information cannot exceed the information contained in the training corpus, *x*. The most efficient representation of *x* is

$$\text{size} = \log_2 1/P(x) \qquad (2.4.2)$$

where the best available estimate of P is used. This is the language model itself. In other words, the size of the language model is the size of the training corpus after it is compressed with the model.

We observe that performance varies widely among different researchers, depending on the type of model used and on the training corpus, but generally improves as the models get larger. This data suggests that the best models could achieve human performance at around 10^8 to 10^{10} bits.

In the table below, we compare a large number of statistical language models, giving the training corpus size (in bytes) and entropy (in bpc). The model size is the product of these two numbers. When a series of results are reported for different models, usually only the best is shown.

When the result is reported as a word perplexity (PP), the reported figure is shown and the entropy (H) is inferred using

$$H = (\log_2 PP)/cpw + bpw \times (1 - cov) + bpw \times vocab/size$$

where *cpw* = 5.5 is the average number of characters per word token, *cov* is the coverage, the fraction of word tokens in the vocabulary (vocab), bpw = 30 is the number of bits per word (assuming 10 characters × 3 bits = 30 with a reasonable character model and assuming that the average word type is longer than the average token), and *size* is the number of characters in the training set. Thus, there are three terms: the first for coding known words, the second for spelling unknown words, and the third to encode the dictionary. The best model on written text (Rosenfeld, 1.221 bpc) has a typical breakdown: 1.106 bpc for known words (PP = 68), 0.112 bpc for unknown words (97.8% coverage), and 0.003 bpc for the dictionary (20K vocabulary, size = 210Mb).

When the coverage is not reported, an estimated value is given in parenthesis by assuming that coverage is inversely proportional to vocabulary size, and comparing to Rosenfeld's reported values of 97.8% coverage with a 20,000 words vocabulary. If the vocabulary includes the test set (indicated by a *), then the fraction of words not covered is estimated by dividing half of the vocabulary by the corpus size in words. Both of these methods assume a Zipf distribution (Zipf, 1939). Zipf discovered that if the vocabulary is ranked by frequency (in English: *the, of, and, to...*), then the n'th most frequent word has frequency k/n for some constant k (0.1 in English). Thus, the frequency of *the* is 1/10, *of* is 1/20, *and* is 1/30, and so on. At the other end, half of all words that occur at all occur only once.

For character models, the vocabulary is the alphabet size. In most cases, the text is reduced from 256 characters to 27 by converting upper case letters to lower case, and all other sequences to a single space. The corpus size is taken after this transformation. When the corpus size is reported in words, we assume 5.5 characters per word. K, M, and G indicate 10^3, 10^6, and 10^9 respectively.

Three measurements using human models are included, for which the model size is left blank. This is the number we seek.

Character models	**Corpus**	**Size**	**Vocab**	**Cov**	**PP**	**H**	**Reference**
LZ (compress)	book1	731Kb	27	1		3.151	compress 4.3d (1990)
LZ (zip)	book1	731Kb	27	1		2.956	PKZIP 2.04e (1993)
LZ (gzip -9)	book1	731Kb	27	1		2.921	gzip 1.2.4 (1993)
PPMC5 (ha -a2)	book1	731Kb	27	1		2.141	ha 0.98, Hirvola 1993
BW (szip)	book1	731Kb	27	1		2.102	szip 1.05x, Schindler 1997, 1998
Neural net, 2 layer	book1	731Kb	27	1		2.062	Mahoney, 1999
PPMZ (boa -m15)	book1	731Kb	27	1		1.962	boa 0.58b, Sutton 1998
PPMZ (rkive)	book1	731Kb	27	1		1.943	rkive 1.91b1, Taylor 1998
BW (szip)	book1	768Kb	256	1		2.345	szip 1.05x
BW	book1	768Kb	256	1		2.49	Burrows, Wheeler, 1994
	Hector	103Mb	256	1		2.01	
PPM*	book1	768Kb	256	1		2.40	Cleary, Teahan, Witten, 1995
Neural net, 3 layer	Münchner	600Kb	80	1		2.89	Schmidhuber, Heil, 1996
PPM5	Malone	46Kb	27	1		2.402	Teahan, Cleary, 1996
	Malone	6.6Mb	27	1		1.598	
PPM5+bigrams	Malone	6.6Mb	27	1		1.488	Teahan, Cleary, 1997
PPM5	WSJ	15.4Mb	27	1		1.602	Teahan, Cleary, 1997
Symbol ranking	Calgary	3.1Mb	256	1		3.1	Fenwick, 1997
Lexical Models	**Corpus**	**Size**	**Vocab**	**Cov**	**PP**	**H**	**Reference**
WDLZW	text?	62Kb		1		2.88	Jiang, Jones, 1992
Bigram	LOB	6Mb	50K*	0.96	513	2.104	Ney, Essen, Kneser, 1995

Trigram	WSJ	250Mb	45K	0.997	144.3	1.325	Kneser, Ney, 1995
Trigram	WSJ	250Mb	58K	(.99)	132	1.341	Seymore, Rosenfeld, 1996
5-gram scaled	NAB	1.32Gb	64K	0.995	127.9	1.301	Kneser, 1996
n-gram+phrases	SWB speech	11Mb	(22K*)	(.995)	78	1.226	Ries, Buø, Waibel, 1996
4-gram scaled	WSJ	250Mb	20K	(.978)	84.4	1.284	Ristad, Thomas, 1997
Trigram	BNC	550Mb	65K	(.99)	165.5	1.398	Clarkson, Robinson, 1997
Trigram+fixed distance bigrams	WSJ	25Mb	20K	(.978)	138.6	1.437	Martin, Ney, Zaplo, 1999
Semantic models	**Corpus**	**Size**	**Vocab**	**Cov**	**PP**	**H**	**Reference**
Bigram+topic +cache	LOB	6Mb	(50K*)	(.96)	384	2.028	Kneser, Steinbiss, 1993
Phrase bigrams +5 topics	RR speech	330Kb	762*	(.994)	15.7	0.823	Giachen, 1995
Function/content trigram	Vermo-bil speech	281Kb	5442	(.95)	60.3	1.927	Geutner, 1996
Trigram+distant bigrams	WSJ	210Mb	20K	0.978	68	1.221	Rosenfeld, 1996
Trigram+distant bigrams	WSJ	25Mb	20K	(.978)	144.0	1.446	Simons, Ney, Martin, 1997
Trigram+topic +cache	BNC	550Mb	65K	(.99)	115.9	1.303	Clarkson, Robinson, 1997
Bigram+LSA	WSJ	250Mb	23K	(.98)	102	1.325	Bellegarda, 1998
Trigram+topic	AP, BN WSJ, SWB	99Mb			159.6		Kalai, Chen, Blum, Rosenfeld, 1999
Trigram+cache +topic (IR)	WSJ	210Mb	60K	(.99)	104.5	1.283	Mahajan, Beeferman, Huang, 1999
Trigram+topic	SWB speech	11.5Mb	22K*	(.995)	73.5	1.211	Khudanpur, Wu, 1999
Syntactic models	**Corpus**	**Size**	**Vocab**	**Cov**	**PP**	**H**	**Reference**
Trigram+POS	Office	137Mb	20K	0.966	320	1.702	Jelinek, Mercer, Roikos, 1990
Bigram+distant bigram+CFG	ARM speech		500*		11.3		Wright, Jones, Lloyd-Thomas, 1994
Tagged	Malone	6.6Mb		1		1.433	Teahan, Cleary, 1998
	WSJ	5.63Mb		1		1.490	
Human models	**Corpus**	**Size**	**Vocab**	**Cov**	**PP**	**H**	**Reference**
Character ranking	Malone	?	27	1		.6-1.3	Shannon, 1950
Gambling	Malone	?	27	1		<1.3	Cover, King, 1978
Gambling	Malay	?	27	1		<1.3	Tan, 1981

Table 2.4. Statistical language models

2.4.1. Types of Language Models

The language models have been divided into four categories: character, lexical, semantic, and syntactic. In order, they deal with letters, words, meaning, and structure.

A *character* model exploits the nonuniform distribution of *n-grams*, character sequences of length n. For example, in English, *e* is more common than *u*, but *qu* is more common than *qe*. The n-gram model is:

$$P(x_i|x_1x_2...x_{i-1}) = P(x_i|x_{i-n+1}...x_{i-1})$$

In other words, the i'th character x_i depends only on the last n − 1 characters of the history, $x_{1-n+1}...x_{i-1}$. These last n − 1 characters are called the *context* of x_i. For most models, $n \le 5$.

A *lexical* model is an n-gram model where the symbols x_i are words rather than characters. In English, words are usually separated by spaces, but a more appropriate unit is the *morpheme* (in speech) or *term* (in writing), the smallest unit of text whose meaning stands alone. For example, *roses* is two terms, *rose*, and *-s* denoting plural. *New York* is one term. Term boundaries are sometimes ambiguous, for instance *mailbox* or *Silicon Valley*. The most common models are the *bigram* (n = 2) and *trigram* (n = 3).

A *semantic* model exploits the tendency of related terms, such as *fire...heat* or *rose...garden* to appear close together in running text, but not necessarily adjacent. Such pairs are sometimes called *triggers*, as one word triggers the other. A *self trigger* or *cache* model exploits the tendency of words to trigger themselves. We distinguish between *content words*, mostly nouns and verbs, that obey semantic constraints, and the small number of high frequency *function words*, such as *the, what, and*, that play a syntactic role. Information retrieval systems (search engines) use a semantic model by matching content words between the query and document, after removing the function words (called *stop terms*).

A *syntactic* model exploits the tendency of words that appear in the same context (preceding and following word) to cluster in categories or *part-of-speech* (POS), such as nouns, verbs, etc., which obey n-gram constraints. In a *hidden Markov model* (HMM), we decompose the next-word probability into a POS state transition probability and an output probability, for example, P(*dog* | *the big*) ≈ P(noun | det adj)P(*dog* | noun). A *context free grammar* (CFG) allows POS symbols to be concatenated to form new symbols, for instance, *NP → det adj noun*, or *S → NP verb NP*. Most CFGs are limited to describing the structure of sentences from words, but we also consider extensions to lower levels (*syllable → consonant vowel*), and higher levels (*paragraph → indentation topic-sentence example example example*).

2.4.2. Language Model Descriptions

The individual language models from the table are as follows:

- LZ -- Limpel-Ziv data compression (Bell, Witten, and Cleary, 1989), used in the popular .Z (UNIX compress), .zip, and .gz (gzip) formats. Frequently occurring character n-grams, such as the 3-gram *the* in English, are replaced by pointers to the previous occurrence (or to dictionary entries built during compression). Decompression is very fast, but compression is poor because of less than optimal encoding of the pointers.
- PPM-- Prediction by Partial Match data compression (Bell, Witten, and Cleary, 1989). The input x is encoded by an arithmetic encoder, which selects a code within one byte of the optimal $\log_2 1/P(x)$ bits. The model P(x) is an n-gram model, $P(x_i|x_1x_2 ...x_{i-1}) = P(x_i|x_{i-n+1}...x_{i-1}) \approx N(x_{i-n+1}...x_i)/N(x_{i-n+1}...x_{i-1})$, where the current character depends on only the context of the previous n − 1 characters, and the probability is estimated by counting occurrences of x_i in that context.
- PPMC -- PPM using escape method C, which falls back to an n − 1 model when the predicted character does not occur in the current context at length n (Bell, Witten, and Cleary, 1989). The probability reserved for unseen characters is t/(t + m), where t different character types are seen in m occurrences of the context. For instance, if the input is "*the the that th_*", then in context *th*, t = 2 (*e* and *a*), m = 3 (*e, e, a*), P(all but *e* or *a*|*th*) = 2/(2 + 3) = 2/5, P(*e*|*th*) = 2/5, P(a|*th*) = 1/5. For improved escape methods, see (Witten and Bell, 1991; Ney, Essen, and Kneser, 1995; Kneser and Ney, 1995; Åberg, Shtarkov, and Smeets 1997; Bordel, Torres, and Vidal, 1995; Cleary and Teahan 1995).
- PPMC5 -- PPM using a maximum n-gram model size of n = 5. Used by *ha*, the top rated compression program on the Calgary corpus as of 1994 (Data Compression FAQ, 1998).
- PPMZ -- PPM in which the optimal escape probability is adaptively adjusted to a large set of context features (Bloom, 1998). Used by *boa*, the top rated program on the Calgary corpus as of July 1999 by Gilchrist (1999), and by *rkive*, the top rated program on the text portion (*book1*, 256 character set) as of Sept. 1998 (since exceeded).
- PPM* -- PPM with an unbounded context size. $P(x_i|x_{i-n+1}...x_{i-1})$ is calculated using the largest n for which the context $x_{i-n+1}...x_{i-1}$ occurred at least once before.
- BW -- Burrows and Wheeler (1994) compression. An unbounded context model in which the input is sorted by context, and the permuted character stream is compressed using adaptive 1-gram methods. Used by *szip*, the top rated program on the large (2-4 Mb) text file portion of the Canterbury corpus (Bell, 1998).

- Neural net -- A neural network is trained to predict the next character given the preceding $n - 1$ characters. See Rumelhart and McClelland (1986) for general techniques. For compression, the output probabilities are fed to an arithmetic encoder as in PPM.
- PPM+bigrams -- Frequently occurring character pairs, such as *th*, *er*, *in*, are replaced with a single character, followed by PPM compression.
- Symbol ranking -- A predictor outputs the number of guesses it needed to guess each character using an n-gram model.
- WDLZW -- Word Division LZW. A dictionary-based LZ compressor in which the dictionary contains only whole words (separated by spaces).
- Trigram -- A PPM3 model where the symbols are words instead of letters.
- Bigram -- A PPM2 word level model.
- Scaled -- A word-level n-gram model where counts of higher level n-grams are selectively discarded to save memory (Seymore and Rosenfeld, 1996).
- Fixed-distance bigram -- Bigram model that skips a fixed number of words. A distance-1 bigram model is $P(w_i|w_1...w_{i-1}) = P(w_i|w_{i-2})$.
- Phrases -- Frequently occurring word pairs are added to the vocabulary and treated as single words. See also (Heeman and Dammani, 1997).
- Topic -- The training corpus is divided into sets by topic or genre, each producing a language model. Predictions from the different models are interpolated in favor of models that gave the most accurate predictions in the immediate past. See also (Gorin and Riccardi, 1999).
- Topic (IR) -- The language model is selected by matching words in the recent history to words in the topic set using information retrieval (IR) techniques.
- Distant bigram (or trigger) -- A bigram model $P(w_i|w_1...w_{i-t}...w_{i-1}) = P(w_i \mid t, w_{i-t})$, where t is a variable distance. Typically w_i and w_{i-t} are associated in meaning (*fire, heat*), and the association weakens as the distance increases or crosses a document boundary.
- Cache (or self-trigger) -- After a word is observed in input, its unigram (1-gram) probability is raised for awhile.
- LSA -- A distant bigram model in which word pair associations (w_i, w_t) are represented in a compressed form by latent semantic analysis (LSA). The idea is to construct a word matrix **W**, where W_{it} is the probability that random instances of w_i and w_t will be found in the same document (or within a window). The matrix **W** (typically 10K to 20K dimensions) is projected into a smaller dimensional space **S** (typically 100-200 dimensions), oriented along the longest eigenvectors of **W** to minimize the loss of information. The dimensions of **S** represent semantic attributes.
- Function/Content Trigram -- Instead of the trigram model $P(w_i|w_{i-2}w_{i-1})$, the context is the last function word and the last content word. Function words are the most frequent 100-200 words in the vocabulary and play a mainly syntactic role (*the, of, and...*). Content words are all others (nouns, verbs, adjectives).
- POS -- New words are matched by context (preceding and following words) to known words in order to infer trigram statistics. This method tends to classify words by their part of speech (POS), but on a finer scale (for example, *third* is matched with *first*).
- Tagged -- Part of speech tags (noun, preposition, etc.) are attached to each word prior to PPM compression of the alternating sequence of tags and words.
- CFG -- The sentence probability is the sum of the probabilities of all possible parses, where the parse probability is the product of the probabilities assigned to each rule in a probabilistic context free grammar (CFG). An example rule might be P(*noun-phrase* $\rightarrow$ *det adj noun*) = 0.23.

The human models are entropy estimates of the indicated corpus using human subjects. The measurement techniques are described in more detail in section 2.3 and summarized here:

- Character ranking -- Subjects are asked to guess the next character in a text sample, continuing until correct. Entropy bounds are inferred from the distribution of the number of trials to success.
- Gambling -- Subjects assign odds and bet on the next character in a gambling game. Entropy is computed directly from the assigned probabilities.

2.4.3. Corpora

The corpora are as follows:

- *book1* -- from the Calgary corpus (a widely used benchmark for data compression), *Far from the Madding Crowd*, by Hardy.
- *Hector* -- A 103 MB collection of English text from a variety of sources.
- *Münchner* -- A collection of 40 German newspaper articles from *Münchner Merkur.*
- *Malone* -- *Jefferson the Virginian* (1948) and 5 subsequent books by Dumas Malone.
- *WSJ* -- Wall Street Journal newspaper articles, 1987-1992.
- *LOB* -- A collection of British English from a variety of sources, 90% written and 10% transcribed speech.
- *NAB* -- Transcribed TV news broadcasts.
- *SWB* -- *Switchboard*, transcribed interactive speech on a variety of topics.
- *RR* -- Transcribed spoken dialogs from a train scheduling task.
- *Vermobil* -- Transcribed interactive German speech on a variety of topics.
- *BNC* -- British text on a variety of topics.
- *AP, BN* -- Associated Press and Broadcast News.
- *Office* -- A collection of office correspondence, insurance documentation and email.
- *ARM* -- Transcribed Airborne Reconnaissance Mission reports.
- *Malay* -- Malaysian poetry.

2.4.4. Comparison of Size and Entropy for Written Corpora

The following graph shows entropy plotted against corpus size for the best result from each corpus size in the table above. Also included are results from a single experiment on the Hector corpus from 1 Kb to 103 Mb reported by Burrows and Wheeler (1994). Interactive speech is excluded, since we have observed that it has a significantly lower entropy than text prepared for publication (including transcribed read speech such as *NAB*).

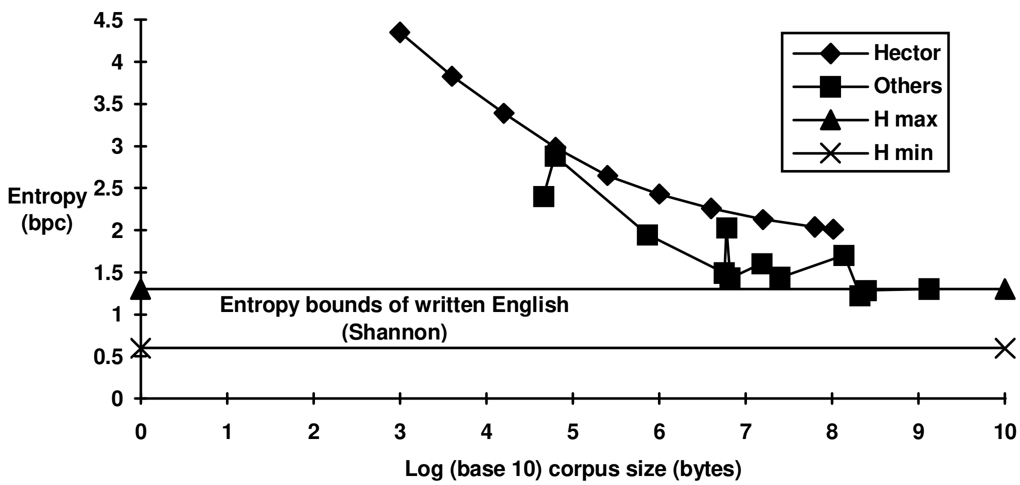

Fig. 2.4. Entropy of published English text by size using (1) Hector corpus with 256-character alphabet (adding about 0.25 bpc) and Burrows-Wheeler compressor, (2) various models and corpora with a 27-character alphabet or word-level modeling.

3. Proposal Overview

Figure 2.4 suggests that in order to achieve human performance (using Shannon's estimate of 0.6 to 1.3 bpc), requires 10^8 to 10^{10} bits of memory (corpus size after compression), consistent with Turing's prediction of 10^9 bits. However two major sources of uncertainty remain.

1. The trend is unclear because we are using many different models and corpora.
2. The true entropy of written English is not known very precisely.

The first problem is due to the nature of the research. Most researchers were not trying to maximize entropy or study the effects of model size, but rather were evaluating one aspect of the modeling problem. For example, a researcher studying the effects of syntactic constraints might use a simple trigram model as a baseline, ignoring any improvements that might be realized with a better lexical or semantic model. Most of these studies examined only one or two of the four major language model components, and none used all four.

The Burrows and Wheeler data on the Hector corpus is the type of data we are seeking. For example, we observe a bend in the data at $10^{5.5}$ bytes, a distinct peak in the second derivative. This is a character level model. Do higher level models have a similar behavior? We don't know.

The second problem is the uncertainty in the estimated entropy of written English, which was discussed in section 2.3. People are not very good at assigning numerical probabilities directly to strings or characters, but are good at comparing strings and deciding which is the most likely. The trick is to devise a test that involves comparison but narrows the bounds in equation 2.3.

My proposal is to build a series of language models, progressively adding character, lexical, semantic, and syntactic components one at a time, and compare their performances as model size is varied with humans in a prediction task on the same test set. To get a better measure of human performance, I plan to use variations of Shannon's ranking method that reduce the range of uncertainty, and to determine the empirical relation between ranking performance and entropy. If successful, the results should look something like this:

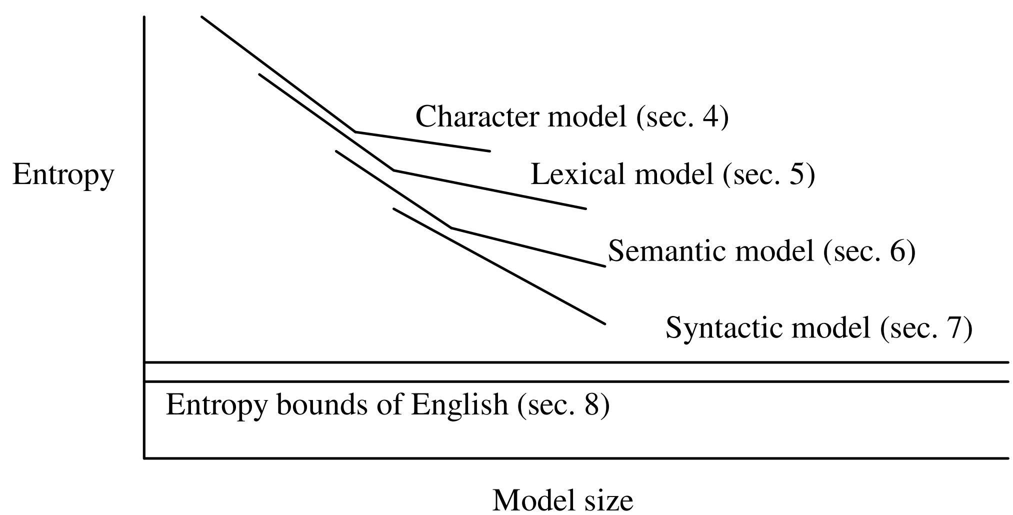

Fig. 3. Hypothetical results from a full language model study.

In the remainder of this section I describe the language model. Then in sections 4 through 7 I describe each of the components and the work to be done. In section 8 I describe improved methods of measuring the entropy of English.

3.1. Language Model Development Order

My approach to building the language model is incremental. This allows any problems discovered at one level to be addressed before proceeding to the next. In section 2.4.1, I categorized the four major types of models, and gave examples of each in 2.4.2. To summarize, the four levels are:

1. Character – The model knows about letter frequencies but nothing about words.
2. Lexical – The model knows about words but nothing about their meaning or usage.
3. Semantic – The model knows the meanings of words but not how to form sentences.
4. Syntactic – The model also knows about sentence structure.

Choosing the correct order in which to add each component is critical. Many artificial language models, such as compilers, model syntax (parsing) before semantics (extraction and code generation). This does not work in English. For example, it is not possible to parse the following sentences without knowing the meanings of the words.

- I ate pizza with pepperoni.
- I ate pizza with Sam.
- I ate pizza with a fork.

On the other hand, there are plenty of successful models that use semantics but no syntax. Both LSA and information retrieval systems (search engines) are "bag of words" models. They make no use of word order. They use the fact that semantically related words, like *pizza* and *pepperoni*, or *ate* and *fork*, often occur near each other in text, within the same paragraph or document.

The reason that syntax requires semantics but not vice versa may be that there is no human analogue of a syntactic language model that lacks semantics. Children begin to learn content words like *ball* or *mommy*, and their meanings, at around 12 months. They do not start using function words that play a syntactic role, like *the* and *of*, until around age 2 or 3, even though they have a higher frequency (Papalia and Wendkos-Olds, 1990; see also Hörmann, 1979).

Following this approach for the other levels, it would seem unlikely that there is a lexical level in human language development – a stage where we can parse language into words without knowing what the words mean or how they are used. Yet that is exactly what Jusczyk (1996) found in infants listening to speech. Speech is continuous – there are no "spaces" to divide speech into words. Yet infants are able to segment speech into words even before they have learned to speak or understand even a single word. After hearing words like *doctor* or *candle* in continuous speech, they later pay attention longer to other sentences containing those words than they do to sentences containing *dock* or *can*. At 7.5 months, infants depend on the first syllable being stressed, but by 10.5 months they are able to correctly segment words like *beret* or *guitar*.

Again this agrees with statistical models of English. Hutchens and Alder (1998) showed that the instantaneous entropy (unpredictability) of English is higher at the beginning of words, and used this fact to deduce that words are separated by spaces with just a simple 3-gram character model. In section 5 I describe an experiment that shows that a 5-gram model can identify word boundaries with 75% accuracy in text when the spaces are removed. This is with *no training* on text with spaces.

3.2. Completeness

I identified four language levels, and argued that they should be added in a certain order, but are there any more? Looking at Turing's hypothetical dialog in section 1.2, it does not seem possible that a language model could add numbers or play chess. In theory, a large enough corpus would contain a previous example of the relevant questions and their answers, but this is not really a satisfactory answer, because the model size would have to grow exponentially with context size. It is more likely that we would find rules for playing chess or adding numbers in a smaller corpus, but it is not clear how a model could use these rules to solve the problems. Both problems require some spatial reasoning, and it would probably be fair to modify the Turing test slightly to disallow human competitors from working out problems with pencil and paper. We could also argue that arithmetic and chess have already been solved algorithmically, so are not very interesting, but we still have not solved the problem of translating natural language questions into a form that these algorithms could understand.

My answer is that we do not need to solve the general problem of AI because that is not our goal. A syntactic level model would simulate the language skills of a child, perhaps around age 2 or 3. This would give us enough data to estimate the size of an adult model. We would not expect a 3 year old child to add numbers or play chess.

3.3. The Model Implementation

A language model is a probability distribution over a set of strings, P(x), $x = x_1x_2...x_i$. The model I describe is on-line. After each input character x_i, the model is updated and it outputs a prediction, the probability distribution $P(x_{i+1}|x_1x_2...x_i)$ for each character in the alphabet.

The model is implemented as a set of memories, or fuzzy sets of symbols. A *symbol* is defined recursively as:

- A letter of the alphabet (*a, b, c*...)
- A concatenation of symbols (*ca, cat*, *the cat*..)
- A fuzzy set of symbols ({*cat, dog*}, animals, nouns, noun phrases...)

Thus, a symbol can be a letter, syllable, word, or part of speech. These are organized into 5 memories or fuzzy sets, V, B, I, S, and L. Each memory except L associates a membership value between 0 and 1 with every symbol. L is a fuzzy set of unordered pairs of symbols. The magnitude of each set (sum of membership values) is bounded. These sets are (see fig. 3.3).

- V – The vocabulary of all symbols and their unigram frequencies.
- B – Input/output buffer. During update, it contains the current input symbol at each level (letter, syllable, word, part of speech). During prediction, it contains the probability assignments for each symbol.
- I – Immediate memory. Contains the previous input symbol from B.
- S – Short term memory. Contains an exponentially decaying recent history of inputs from B.
- L – Long term memory. Permanent store of symbol pairs, reflecting their probability of appearing simultaneously in S.

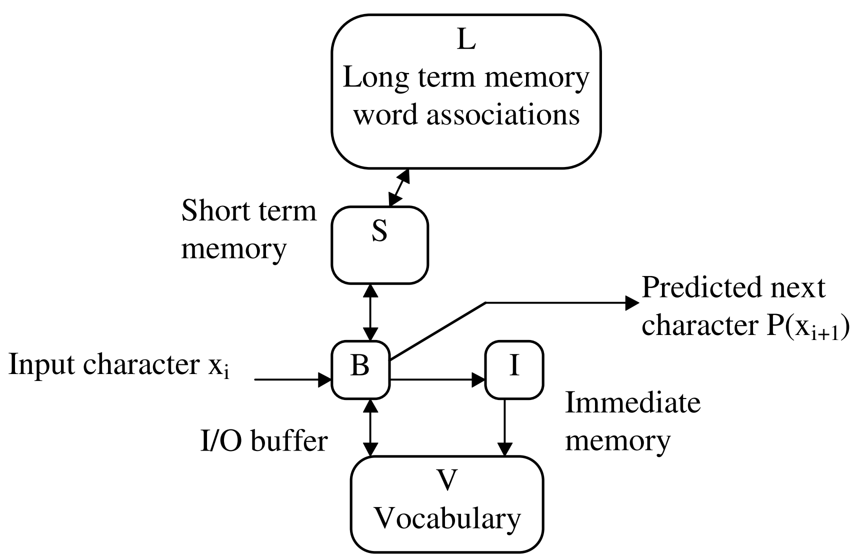

Fig. 3.3. A full language model.

The update and prediction algorithms are easiest to describe by building the model up in layers: character, lexical, semantic, and syntactic.

3.4. Character Model

A bigram character uses only B, I, and V. Symbols consist only of single characters and pairs of characters (*a, b, aa, ab*, ...). During update, B contains the current input character, I contains the previous character, and V contains unigram and bigram frequencies. Thus, if the input is *ab*, then I contains *a* and B contains *b*. The update steps are:

U1. Put input b in B.
U2. Add B and V
U3. If *a* is in I and *b* is in B, then add *ab* to V.
U4. Move B to I.

The output (B) is a prediction of the next character. Thus, if the input is *a*, and *ab* was previously observed, then *b* should be predicted. The unigram frequencies are also taken into account. Thus, if *c* was observed frequently but never after *a*, then there is still a good chance it could occur. The output steps are:

P1. Clear B.
P2. Move V to B.
P3. If *a* is in I, and *ab* is in B, then add *b* to B.

Since all of the memories are bounded, it is implicit in each step that if we add elements, then we reduce the membership values of other elements by an equal amount. The output B contains both character and bigram predictions (so its magnitude is 2), but we would normally use only the character probabilities to evaluate the model.

A bigram model can be extended to an n-gram model by adding a chain of $n - 1$ immediate memories in place of I. However, for our subsequent models, bigrams will be sufficient.

3.5. Lexical Model

In a lexical model, symbols may be more than two characters long. The update algorithm learns which pairs of symbols occur most frequently and adds them to the vocabulary, thus building up syllables, words, and phrases. The update steps of the character model are extended as follows:

U5. If *a* is in I, *b* is in B and *ab* is already in V, then add *ab* to B.

This input feedback allows larger symbols to be built. Thus, if the input is *abc*, and *ab* is already a symbol, then on the next step when *c* is received, *ab* is moved to I, and step U3 adds *abc* as a symbol. Output symbols are expanded into character predictions as they were with bigrams without any further changes to the model.

3.6. Semantic Model

The semantic model uses short and long term memory. The short term memory (or cache) models the property that words (specifically content words – those that convey meaning) tend to cluster in running text. In other words, a word such as *semantic* is likely to appear again if it appeared recently in running text, even if it rarely appears as a whole. This is implemented by the update step:

U6. Add B to S.

and for output,

P4. Add S to B

Long term memory stores semantic associations between words. It uses the property that associated words like *semantic* and *meaning* frequently occur near each other. Thus if both are found in short term memory at the same time, they are likely to be related, and therefore one predicts the other. Because semantics is transitive, we can add the predicted word back to short term memory to recall indirect associations. Thus, if *semantic* and *meaning* are associated, and so are *meaning* and *content*, then *semantic* predicts *content*, even if they never occurred together before.

The second part of the semantic model adds long term memory, L., and the following two update steps.

U7. If *a* and *b* are in S, then add (*a*, *b*) and (*b*, *a*) to L (learning associations).
U8. If *a* is in S and (*a*, *b*) is in L, then add *b* to S. (recalling associations).

The recalled associations are output in step P4.

3.7. Syntactic Model

The syntactic model is also implemented in two stages. First we add a "no-repeat" rule, to disallow strings like *the the* or *aa*. Since the previous input is in I, we have

P5. Remove I from B.

Then we add a context free grammar. The syntactic model extends the symbol set to include sets of sequences, such as {*dog, cat*} or {*and, or, but*}, or {*a*: *a* is a noun}. These word classes can be concatenated just like other symbols in the lexical model, so we could have *the-noun* or *NP = det adj noun* or *sentence = NP verb NP*. Symbols don't have to be words, so we might also have categories such as vowels or digits. The model would discover syntactic categories by observing symbols in the same immediate context, as in *the cat* and *the dog*, or *and sentence* and *but sentence*. We add the update rules:

U9. If *b* is in B and {*a, b*} is in V, then add {*a, b*} to B (generalize the input).
U10. If *b* is in I, *c* is in B, and *ac* is in V, then add {*a, b*} to V (post-context *c*).
U11. If *c* is in I, *b* is in B and *ca* is in V, then add {*a*, *b*} to V (pre-context *c*).

and the output rule,

P6. If {*a*, *b*} is in B, then add *a* and *b* to B (specialization).

Rules U9 and P6 are inverses. The first generalizes (*cat* → *noun*), and the second specializes (*noun* → *cat, dog,...*). Rule U10 groups *a* with *b* when they both occur before *c*, as in *ac bc*.. Rule U11 groups *a* with *b* when they both occur after *c*, as in *ca cb*.

3.8. Feasibility

The language model has two important characteristics that suggest it is feasible. First, it runs in O(n) time. This is because all of the memory sizes are fixed, and none of the steps depend on the amount of previous input. The significance of this is that we observe that the human model also appears to be O(n). People seem to process language at a fairly constant rate of about 150 words per minute.

The second characteristic is that the model can be implemented as a neural network. In the connectionist model (Feldman and Ballard 1982), neurons would represent letters, words, and syntactic categories (i.e. the set V), and the connections between them would represent associations (the set L). The other memories, B, I, and S, represent activation levels with various decay rates, along the idea of Little and Shaw (1975). The significance of this is that the human language model is also implemented with a type of neural network.

3.9. Size of the Model

A major factor in the memory requirements of the model is the size of the vocabulary. If there are *n* symbols, then we could potentially need to store n^2 bigrams in V and n^2 pairs in L. Counting only words in an adult vocabulary, we have about $n = 10^5$.

We can reduce the size of L by using LSA (Bellegarda et. al. 1996; Landauer and Dumais, 1997). Essentially, the $n \times n$ matrix L, where L_{ij} is the association between words *i* and *j*, is replaced by a smaller $n \times m$ matrix A, where $L_{ij} = A_i \bullet A_j / |A_i||A_j|$, the cosine of the angle between the vector rows A_i and A_j corresponding to words *i* and *j*. LSA not only saves space, but improves performance for a given training set size. Landauer and Dumais found that for n = 60,000, that m = 300 gave the best performance on the TOEFL test in section 2.2. Bellegarda et. al. found that for n = 23,000, that m = 100 to 200 was optimal for perplexity (table 2.4). In both cases, $m \approx n^{1/2}$. Landauer and Dumais also point out that LSA can be implemented with a neural network, essentially adding a layer of m hidden units between the n input units and n output units that represent the words in the vocabulary.

For V, we will probably save space because the bigram matrix will be sparse, but it remains to be seen how much. I believe that the syntactic model will play an important role in filling in the missing statistics for rarely observed bigrams, just as LSA fills in for rarely observed word associations.

4. Character Model

In this section I describe the first layer of the language model, the character model. The goal is to build an efficient (*O*(n)) on-line character (n-gram) model using techniques that can be extended to higher level models. Specifically,

1. What is the most efficient on-line method of combining evidence from non-independent sources (different contexts) to arrive at a joint probability (for the next character)?
2. How do we combine older but more reliable statistics with newer but sparser statistics for nonstationary sources (sources whose statistics change over time)?

4.1. Background

A central problem in on-line language modeling is in combining the evidence presented by different contexts, h, to arrive at an overall probability for the next input character, w. For n-gram models, there are n contexts of length 0 through $n - 1$, each presenting a different probability, $P(w|h) \approx N(hw)/N(h)$, where N is a count. This problem has been extensively studied in PPM models (Bell, Witten, and Cleary, 1989, Witten and Bell, 1991; Ney, Essen, and Kneser, 1995; Kneser and Ney, 1995; Åberg, Shtarkov, and Smeets 1997; Bordel, Torres, and Vidal, 1995; Cleary and Teahan 1995; Bloom, 1998). PPM models are among the best known n-gram models. However, none of these address the more general problem of combining evidence that occurs in higher level models. This is addressed by the *maximum entropy* (ME) approach, which finds the most likely joint distribution over a set of random variables subject to arbitrary constraints (Rosenfeld, 1996; Manning and Schütze, 1999). The approach is widely used, mostly in semantic models for combining local n-gram and distant bigram statistics (Rosenfeld, 1996; Khudanpur, Wu, 1999; Lau, Rosenfeld, Roukos, 1993; Martin, Ney, Zaplo, 1999; Simons, Ney, Martin, 1997). The problem is that the ME solution, *generalized iterative scaling* (GIS) is too slow for practical data compression programs. Neural networks can also be used (for example, Schmidhuber and Heil, 1996), but again, past solutions have been too slow. As a result, all of the data compression programs rated by Gilchrist (1999) are character level models.

We seek a fast, on-line, and general method of combining constraints from different contexts. The method should be on-line so that it can adapt to a nonstationary source. Although we are testing a character model, the method should be general so that it can be extended to higher level models.

4.2. Approach

In (Mahoney, 1999), I show that a specific but useful form of ME is equivalent to a 2-layer neural network, which can be used to predict the input one bit at a time instead of one character at a time. If $\mathbf{x} = x_1, x_2, ..., x_n$ are Boolean functions of the history (1 if a context is present, otherwise 0), y is the next bit, and all of the constraints have the form of known probabilities $P(y|x_i)$, then

$$P(y|\mathbf{x}) = g(\Sigma_i\, w_i x_i), \text{ where } g(x) = 1/(1 + e^{-x}) \qquad (4.1)$$

and the weights w_i are found (as in GIS) by iteratively adjusting them in the direction that reduces the error $E = y - p(y|\mathbf{x})$, the difference between the expected and actual outputs during training. The adjustment is as follows:

$$w_i = w_i + \Delta w_i, \text{ where } \Delta w_i = \eta x_i E \qquad (4.2)$$

where η is a parameter (the *learning rate*), typically set *ad hoc* to around 0.1 to 0.5. The value does not affect the solution, only the number of passes over the training set ($\mathbf{x}$, y) needed for the w_i to converge. I also show in (Mahoney, 1999) that for the case of x_i independent and a stationary source, that convergence can be done in one pass (necessary for on-line compression). The learning rate is

$$\eta = 1/\sigma^2, \text{ where } \sigma_i^2 \approx N_i(0)N(1)/N_i \qquad (4.3)$$

and where $N_i(0)$ and $N_i(1)$ are the number of times that $y = 0$ or $y = 1$ respectively when x_i is active ($x_i = 1$), and $N_i = N_i(0) + N_i(1)$. In other words, σ_i^2 is the variance of y in context x_i.

For an n-gram model, only n inputs are active at any time, and there is only one output (as opposed to one output for each character in the alphabet). Thus, the model is fast because only 8n connections are used to predict each 8-bit character, and only 8n weights need to be updated. For bit level prediction, the n-gram context is the last n – 1 characters plus the bits of the current character seen so far.

However, (4.3) does not apply to the more general case of non-independent inputs (certainly the case) and a nonstationary source. If n inputs are fully correlated, then they have the same effect as one connection with weight nw_i, so $\eta = 1/n\sigma^2$. For the general case, the learning rate should be somewhere in between.

Equation (4.3) has the effect of decreasing the learning rate over time so that all training examples have an equal effect on w_i regardless of their order. For a nonstationary source, we should weight the training data in favor of the most recent examples. One way to do this is to add a fixed minimum learning rate as in (4.2). The result is

$$\Delta w_i = (\eta_S + \eta_L/\sigma^2)x_iE \tag{4.4}$$

where η_S and η_L are parameters, called the *short* and *long* term learning rates. The long term rate "remembers" all of the training data, but the short term rate "forgets" by exponentially decaying the data at a rate η_S.

Also, the approximation of $\sigma_i^2 \approx N_i(0)N_i(1)/N_i$ in (4.3) is due to the zero count problem, which might result in an infinite learning rate. The solution is to start the counts at a small value $d > 0$.

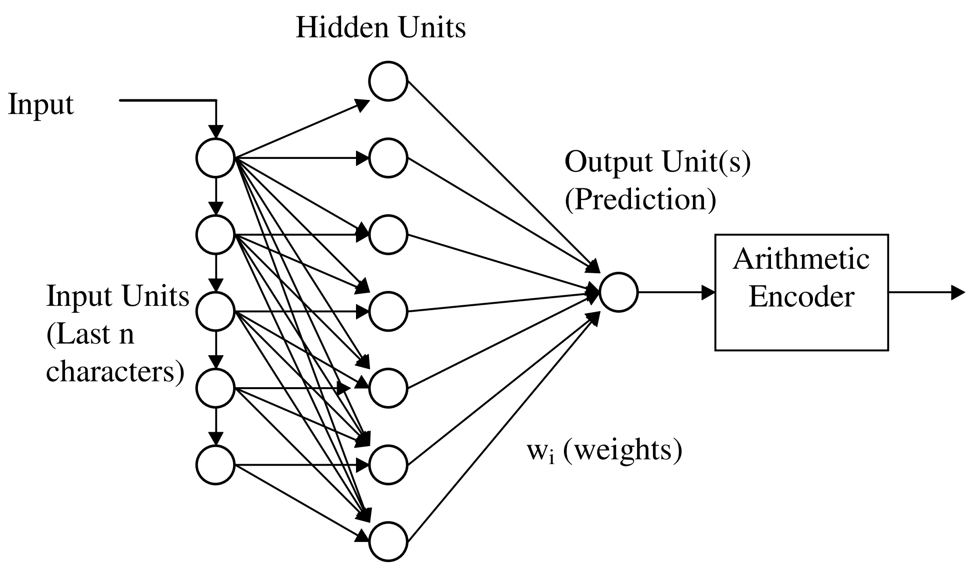

Fig. 4.2. A neural network used for text compression. In (Schmidhuber and Heil, 1996) there was one output unit for each of the 80 characters in the printable alphabet, corresponding to the probability that that character is next. The network was trained off-line by back propagation. In (Mahoney, 1999), prediction was done one bit at a time, so there was only one output unit. The hidden units were fixed, computed by a hashing function. The output layer was trained after each bit prediction.

4.3. Procedure

An n-gram data compressor was built by using the above neural network and an arithmetic encoder. The inputs x_i are all 0 except for n inputs that depend on each of the contexts of length 0 through n – 1. To save memory, the index *i* of the active input x_i was computed as a hash function of the context. Alternative models were evaluated by compression on a text corpus. The steps were:

1. Investigate alternatives to (4.1) empirically, for example, interpolation or binary PPM.
2. Assuming (4.1), investigate alternatives to (4.4) empirically, for example, (4.2).
3. Assuming (4.1) and (4.4), find the values of η_S, η_L, and d that maximize compression.
4. Optimize for speed.

4.4. Results

Results are complete and reported in (Mahoney, 1999). To summarize, a 4M by 1 6-gram model was built using *Alice in Wonderland* (152Kb 8-bit characters) for development, then tested on *book1* (table 2.4.1) and the Calgary corpus with no further changes. It was found during development that the n = 1 (size 0) context did not aid compression, so was dropped. Equation (4.1) was found superior to interpolation (averaging context probabilities) and binary PPM (choosing the longest context that occurred at least once). Equation (4.4) was found superior to about 20 *ad hoc* methods tested, mostly with the idea of reducing the learning rate over time. The optimal values found were d = 0.5, $\eta_L = 0.35$, and $\eta_S = (0.08, 0.15, 0.2, 0.2, 0.2)$ for context lengths 1, 2, 3, 4, and 5. With no further tuning, the network compressed the Calgary corpus (3.141 Mb) to 2.142 bpc in 537 seconds on a 100 MHz 80486, compared to 1.904 bpc in 720 seconds for *boa*, the top rated program by Gilchrist (1999). Both programs use 15-16 Mb of memory.

4.5. Conclusions

It was shown that neural network language modeling is a viable alternative to PPM and BW compression, and better (but slower) than LZ. Unlike current methods, neural networks can be extended to higher level models. The value $\eta_L \approx n^{-1/2}$ suggests that the contexts are partially correlated, as expected. The different values of η_S may be due to the cache effect, which affects words but not letters.

An implementation (C++ source and Windows executable) is available at (Mahoney, 1999). I plan to investigate methods of adapting the parameters η_S and η_L to the data, but the use of *ad hoc* parameters seems to be unavoidable. Some improvement might also be possible by using checksums to reduce hash collisions.

5. Lexical Model

We now add the second layer of the model. The goal is to build an efficient (*O*(n)) on-line lexical model that discovers the vocabulary of a language, given only a character n-gram model. The goal is twofold:

1. *Statistical* -- What partitioning of the input text into words achieves the best word-level n-gram compression? How big should the vocabulary be? How much memory is needed?
2. *Linguistic* -- Does this partitioning agree with our intuitive notion of a "word", i.e. the smallest unit of text that has meaning when standing alone? Does the model find syllables? Phrases? Root words and suffixes? Does the model work on agglutinative languages (no spaces between words)?

5.1. Background

A number of researchers have shown that n-gram compression (or perplexity) improves when the symbols are word rather than letters. For example, Teahan and Cleary (1997) observed improvements of 0.06 to 0.08 bpc. One explanation is that for character models, there is little improvement beyond n = 5, yet most word models use trigrams, n = 3, which is about twice as much context when word length is taken into account.

Most lexical models use simple language-dependent rules to split text into words, for instance, dividing the text at spaces and punctuation. Even poor lexical models yield some improvement. Teahan and Cleary (1996) found that replacing the 100 most common letter bigrams (*th, er, in*...) with a single character improves PPM compression, but curiously, not if either of the characters is a space. Going the other way, Hwang (1997) found that compression got worse (though that wasn't his goal) when Korean words were split into syllables or phonemes.

Parsing text into terms -- the smallest units of meaning -- is not simple, because a term may be part of a word or more than one word. In information retrieval systems, we want a search for *valley* to match *valleys* (two terms: *valley* and *-s*), but we don't want a search for *Silicon Valley* (one term) to match *silicon* and *valley* separately. *Silicon Valley* is a single term because its meaning is different from its components. It does not mean *valley of silicon*.

Complex stemmers such as Porter's (1980) suffix stripping algorithm (with about 200 English rules) still miss some cases (*Christmas* is not plural). The problem is more severe with Finnish verbs and German nouns, where many morphemes are run together, and in Chinese, where there are no spaces between ideograms (syllables). Although there are about 1.8 Chinese ideograms per word (Huang, Luo, and Tang, 1997), the concept of "word" is poorly defined, and there is no generally accepted lexicon (Yang, Ho, Chien, Lee, 1998). For example, the Chinese word for *computer* is made of two ideograms meaning *brain* and *electricity*. A similar problem exists for English words like *sunrise* and *mailbox*.

Since Jusczyk (1996) found that infants can segment continuous speech into words before they are old enough to speak or understand them, one might surmise that it is possible to infer a lexical model without any semantic knowledge of the language. Indeed, Hutchens and Alder (1998) found that during compression using a simple trigram character model that the instantaneous entropy was higher at word boundaries. The model also inferred that the space character served as a word separator, with no prior knowledge of this fact. Using a similar approach, I found that in text without spaces, that a simple 5-gram model could find word boundaries with 75% accuracy (see *results* below). Nevill-Manning, Witten, and Olsen (1996) developed a simple grammar induction algorithm for compressing structured data, that replaces any symbol sequence that occurs more than once with a new symbol. When Nevill-Manning, and Witten (1997) applied the algorithm to text, they observed that most of the symbols were words.

A lexical model partitions the text into sequences and replaces each with a symbol. Later we will be interested in the semantic and syntactic properties of these symbols, but right now we are only interested in the entropy of this meaningless symbol sequence. For a corpus of n characters, a brute force search for the optimal partitioning would require trying all 2^{n-1} of them. This is not feasible, so a heuristic approach is required. An "optimal" greedy approach that runs in $O(n^2)$ time is to try combining each adjacent pair of symbols and choosing the one that gives the best compression, stopping when there is no more improvement. There are many faster algorithms that do about as well, for example, the forward-backwards algorithm (Yang, Ho, Chien, Lee, 1998), variable context length methods (Kneser, 1996), and nonuniform Markov models (Ristad, Thomas, 1997). Klakow (1998) compared three approaches: pair frequency, mutual information, and change in unigram entropy, and found that all worked about the same. Ries, Buø, and Waibel (1996) investigated faster versions of the "optimal" method that collected a large number of pairs during each pass instead of just the best one.

5.2. Procedure

The approach is to build a lexical model as described in section 3.5 and to compare it with standard (character level) text compression methods. Roughly, the idea is to build a character bigram model, and then when a bigram is observed frequently, above some threshold T, then add a new symbol to represent it. Then T will be adjusted for maximum compression. Another approach is to bound the number of symbols, and save only those that have the highest frequency. Compression will be evaluated as a function of the bound size. I will also perform a qualitative evaluation by sampling the symbols formed in each implementation to see if they are mostly words.

5.3. Preliminary results

I have improved on the work of Hutchens and Alder (1998) in showing that word boundaries can be inferred from an n-gram model, even when spaces are removed, by increasing n from 3 to 5 and examining entropy going both forwards and backwards. I reduced the text of *Alice in Wonderland* (152Kb) to a 26 character alphabet by changing upper case letters to lower case and removing all other characters including spaces. Then, using a 5-gram model, each character boundary was tested and a space was inserted between characters x_4 and x_5 if $H(x_5|x_1x_2x_3x_4) + H(x_4|x_5x_6x_7x_8) > T$, where T is a threshold, and $H(x) = \Sigma_x P(x) \log_2 1/P(x)$ is the entropy of the distribution of x. Probabilities were determined by simple counting without smoothing, i.e. $P(x_5|x_1...x_4) = N(x_1...x_5)/N(x_1...x_4)$, and defining $0 \log 1/0 = 0$. Then I adjusted T so that recall (fraction of original word boundaries found) equaled precision (fraction of spaces inserted that are correct). A baseline of 22% is possible by random guessing. Recall and precision increased with n up to n = 5, the largest value that was practical to test using a 26^n counter array. For n = 5 and T = 4.3 bits, recall and precision were 75%.

6. Semantic Model

We now add the third layer of the language model, which we break down into two sublayers. The first sublayer is the cache model, which adds a short term memory to exploit the tendency of content words to cluster in small regions of text. The second sublayer is the semantic model, which adds a long term memory to store word associations, also called distant bigrams or triggers.

6.1. Cache Model

The goal is to add an efficient (*O*(n)) word-caching technique to the lexical model of section 5. A cache model raises P(w) for a word w as a function of the time *t* since it was last seen. The goal is twofold:

1. *Statistical* -- Which methods improve text compression the most? How is P(w) affected by *t* and the previous value of P(w)? How much memory should the cache use?
2. *Linguistic* -- Does a cache model human short term memory? Does a cache distinguish between function and content words?

6.1.1. Background

Adding a cache to a language model can improve it substantially. For example, Clarkson and Robinson (1997) reported a 14% perplexity reduction in a trigram word model and 6% reduction in a trigram plus topic mixture model by adding an exponentially decaying cache with a decay rate of 0.005. Mahajan, Beeferman, and Huang (1999) reported a 15% perplexity reduction over a trigram model.

A cache works because content words appear in clusters, so the first occurrence of a word signals an increased likelihood of its reappearance. However, current cache models do not discriminate between content words and function words, which tend to be uniformly distributed throughout the text (stationary).

A cache is analogous to human short-term memory, which has a capacity of 7 ± 2 "chunks", independent words or digits (Schwartz and Reisberg, 1991). In contrast, the optimal cache size found by Clarkson and Robinson is about 200 words (1/0.005). Another difference is that human short term memory seems to use a lexical model, while long term memory uses a semantic model. For instance, in recall tests where people are asked to remember long lists of words, they tend to make the fewest mistakes at the beginning and at the end of the list. At the beginning, which is stored in long term memory, people tend to confuse words with semantically related words, like *cows* for *cattle*. At the end, which is stored in short term memory, errors are usually lexical, like *rattle* for *cattle* (Schwartz and Reisberg, 1991). This model is at odds with the previous observation that sublexical symbols tend to be stationary and should not be cached.

6.1.2. Procedure

The approach is to extend the lexical model of section 5 with a cache or short term memory model, S, and add update rule U6 (add B to S), and output rule P4 (add S to B). The model will be compared with previous versions for text compression as the size of S and decay rate are varied. I will also investigate whether it is worthwhile to make the decay rate proportional to word frequency in order to favor content words over function words and sublexical symbols.

6.1.3. Preliminary results

A cache effect was observed in section 4, and tended to favor longer n-grams. See also section 7.1.

6.2. Distant Bigram Model

The goal is to add an efficient (*O*(n)) distant-bigram model to the cache model of 6.1. A distant bigram is a word pair, $w_1 \ldots w_2$, with a variable distance, *t*, between them. The goal is twofold:

1. *Statistical* -- Which methods improve text compression the most? How does *t*, $P(w_1)$, and $P(w_2)$ affect $P(w_2|w_1, t)$? Is the distant bigram relation symmetric, $P(w_1...w_2) = P(w_2...w_1)$? Is it transitive? Do they form a hierarchical equivalence relation, and if so, how broad and how deep? How much memory is needed?
2. *Linguistic* -- Do distant bigrams capture the meanings of words? Does the model derive a reasonable ontology? Does this model human long term memory? What role does short term memory play in learning and recall?

6.2.1. Background

There are a number of methods of exploiting distant bigram constraints. Topic models (Kneser and Steinbiss, 1993; Giachen, 1995; Kalai, Chen, Blum, Rosenfeld, 1999; Khudanpur, Wu, 1999) divide the corpus into subsets by topic to create a set of submodels. These are linearly interpolated adaptively to favor those that have given the best performance in the recent past. The idea is to use a training corpus that more closely resembles the text being predicted, but this also results in less statistics being available. To overcome this, the submodels are usually combined with the full model by interpolation (weighted averaging).

Mahajan, Beeferman, and Huang (1999) used information retrieval (IR) techniques instead of adaptation to pick the submodel weights. In their experiments, they found that the standard *tf-idf* term weighting used in IR to match queries to documents also worked well to match context to training sets. In the *tf-idf* model, a match for term *w* is weighted by log 1/P(w) in order to favor low-frequency words (content words). They also found that stemming (removing suffixes) improved performance as it does in IR, but unlike IR, removing stop terms (function words) did not help. Nor did query expansion help. Query expansion adds terms from the top ranked documents back to the query, which has the effect of applying the transitive property of semantic equivalence.

Geutner (1996) used a trigram model in which the context was the last function word and last content word, instead of the last two words. This produced only a slight improvement over the standard trigram model, and then only when blended slightly with a mostly trigram model (interpolation weight of 0.2). On the other hand, Rosenfeld (1996) found substantial improvement by modeling three distant-bigrams per word and combining with a trigram model using a maximum entropy approach. His result of 1.22 bpc is the best known for written English.

A distant bigram model is an n by n matrix **W**, where W_{ik} is the strength of the association between words w_i and w_k, and there are n words in the vocabulary. For reasonable vocabulary sizes, this matrix is huge. Bellegarda et. al. (1996) used two methods, latent semantic analysis (LSA) and clustering, both to reduce the memory requirement, and to smooth the normally sparse data to reduce entropy.

In LSA, the row (or column) vectors of **W**, which represent words, are projected into a space of fewer dimensions, **S**, oriented along the longest eigenvectors of **W** so as to minimize the loss of information. This is illustrated below.

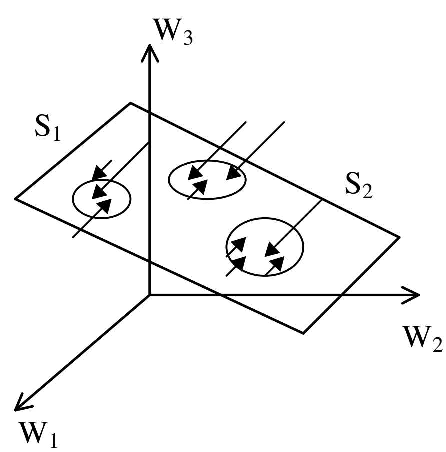

Fig. 6.2.1. Latent semantic analysis (LSA) followed by clustering. Words in a word-word association matrix **W** are projected into a semantic space **S** of fewer dimensions, oriented so as to minimize the lengths of the projection vectors. Then the projected words are grouped into clusters in **S**.

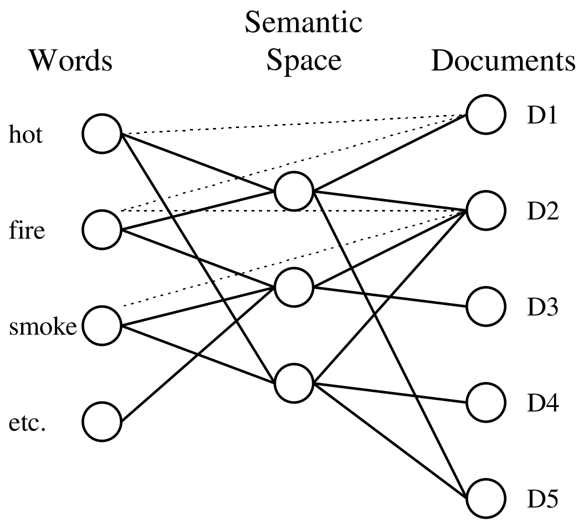

Fig. 6.2.2. A neural network representation of LSA (due to Landauer and Dumais, 1997). The network is trained to associate words (first layer) with the documents, paragraphs, or text windows containing them (third layer). The effect of inserting the middle layer (semantic space) is to associate words such as *hot* and *smoke* that do not occur in the same documents (dotted lines), but that do occur with the same third word (*fire*).

In one experiment, Bellegarda et. al. produced a document matrix **D**, where D_{ij} is the number of times word *i* appears in document *j*. They used a 20,000 word vocabulary on the NAB document set, excluding 300 function words, then constructed $\mathbf{W} = DD^t$, where W_{ik} is the number of times w_i and w_k appear in the same document. They projected the rows of **W** into a 125 dimensional space **S** by LSA (they report that 100-200 dimensions works best), then grouped the projected words into 500 clusters. They give the following two example clusters, which illustrate the close association between distant-bigram statistics and the meanings of words.

1. abstract, art, artist, artist's, canvas, curator, decorative, drawings, exhibit, exhibition, exhibitions, gallery, galleries, Gogh, Henri, museum, museum's museums, painted, painter, painters, painting, paintings, photographs, Picasso, poems, Pollock, Pons, portraits, retrospective, Revere, sketches.

2. appeal, appeals, appellate, argued, arguments, attorney's, circuit, confessed, count, courts, criminal, decide, decision, indict, indictments, judge, judge's, judges, leniency, misdemeanor, office's, prosecuted, prosecution, prosecutions, overturned, prosecutor, prosecutorial, prosecutors, ruled, ruling, rulings, witness.

In a later experiment, Bellegarda (1998) evaluated a bigram plus distant bigram on a set of 87,000 documents from the WSJ corpus and 23,000 word vocabulary. With no smoothing, the model size was $23{,}000^2$ = 529M entries with bigram and trigram perplexities of 215 and 142. Applying LSA with 125 dimensions reduced the model to $23{,}000 \times 125$ = 2.3M entries and reduced bigram perplexity to 147. Then applying clustering with 100 clusters reduced the model size to 100×125 = 12.5K, and reduced bigram perplexity to 106.

While these results are impressive, one might wonder if we could do better. LSA and clustering unilaterally discard information, much like using a bigram model because trigram statistics are too sparse. The preferred method, used by PPM, is to use trigram statistics when they are available, and fall back to a bigram model otherwise. Likewise, if there are sufficient statistics on a distant bigram $w_i...w_k$, then we should use them, otherwise we fall back to cluster statistics. The model could be further improved by using a hierarchical cluster tree or ontology rather than a uniform set of clusters. Then if there were not enough statistics on $w_i...w_k$, we would fall back to the smallest clusters containing each word that did.

6.2.2. Procedure

The model of section 6.1 will be extended with a long term memory, L, containing distant bigram statistics (the matrix **W**), and the following update rules will be added:

U7. If a and b are in S, then add (a, b) and (b, a) to L (learning associations).
U8. If a is in S and (a, b) is in L, then add b to S. (recalling associations).

The effect of rule U7 is to learn associations between pairs of words that occur close together in running text. The effect of rule U8 is to recall associations, which are then output as described in section 6.1. Repeated applications of rule U7 apply the transitive property of association.

I will test (again) whether making cache (S) decay rate proportional to word frequency (in V) improves compression. The idea now is to favor learning associations between content words over function words. I will also investigate cache size, which now affects the distance range over which distant bigram statistics apply. Intuitively we expect that the strength of the association between two words varies inversely with the distance between them.

Since L is very large and will probably be sparse, I will investigate more efficient representations. The goal is to achieve the storage gains of LSA and clustering without the unilateral discarding of data. One method that I plan to explore is to store sublexical symbols in S and sublexical symbol pairs in L, that is, word fragments. For example, the word *cattle* would be stored as *ca, at, ttl, le* and would be reassembled during recall using output rules 2, 3, and 4. This would be consistent with the psychological model, where an error such as *rattle* could be modeled by dropping the symbol *ca*. A semantic relationship between *cattle* and *cow* would be stored as a large number of pairs (*ca, co*), *(ca, ow*), (*at, ow*), etc., instead of a single pair (*cattle, cow*). This would reduce the memory requirement for L, because there would be fewer sublexical components than words. If implemented as a neural network, this would be a *distributed* connectionist model, in which concepts such as *cattle* are mapped into activation patterns across a large number of neurons, and each neuron contributes to a large number of patterns. Distributed models are robust because they degrade gradually with the loss of neurons, and they solve the problem of modeling the addition of new words to the vocabulary. The distributed model should be easily implemented by discarding the variable decay rates in S.

A second variation of LSA is to use a hidden layer to represent the semantic space, but to add connections between the hidden units. The fact that it is possible to both improve compression and save space by clustering shows that LSA is not the most efficient way to represent semantic knowledge. The effect of an interconnected hidden layer should be to implement a hierarchical ontology of word relations, such as *cat - pet - animal - object*, with hidden units (rather than clusters) standing for successively more abstract concepts. Whether I can test this interpretation remains to be seen. To implement this network, the linear hidden units will be made nonlinear, and the network will be trained by back propagation.

6.2.3. Preliminary results

Although Bellegarda et. al. (1996) presented compelling evidence that distant bigrams model semantics, I wanted to confirm that semantics could be extracted from a purely statistical model. Resnik (1998) investigated whether subjective semantic similarity could be predicted using a hand-built lexicon. In earlier work by Miller and Charles (cited by Resnik), 38 subjects were asked to rate the similarity between pairs of words on a scale from 0 (no similarity, *rooster* and *voyage*), to 4 (very similar, *car* and *automobile*). Resnik tested 10 additional subjects, then developed a similarity measure (*wsim*) based on the number of edges between words in the Wordnet taxonomy tree and on unigram word frequency statistics gathered from a corpus. He found a correlation of 0.7911 between *wsim* and the original set of 38 similarity judgments.

Wordnet is a hand-built lexicon. To test whether subjective association could be predicted using a purely statistical approach, I developed a statistical similarity measure (*ssim*) based on the probability that the two words would be found on the same web page relative to their individual frequencies. The measure is:

$$\text{ssim}(w_1, w_2) = \ln P(w_1, w_2)/(P(w_1)P(w_2))^k \qquad (6.2.1)$$

For k = 1, ssim is 0 if w_1 and w_2 are independent. Probabilities were obtained by entering the queries w_1, w_2, and w_1w_2 into the *Hotbot* search engine (www.hotbot.com), which reports the number of matching pages found. These were divided by the total number of pages indexed, 110 million (about a third of the Internet, the best among search engines as of Dec. 1997, see Peterson, 1998). Results are shown below.

w_1	$N(w_1)$	w_2	$N(w_2)$	$N(w_1, w_2)$	Subj	wsim	ssim
car	951850	automobile	118610	59060	3.9158	8.0411	4.0525
gem	53210	jewel	46500	2320	3.7692	14.9286	4.6361
journey	233660	voyage	49380	11275	3.7692	6.7537	4.6774

boy	**408040**	**lad**	**14830**	**5405**	**3.7058**	**8.4240**	**4.5875**
coast	**445230**	**shore**	**157350**	**34970**	**3.6583**	**10.8076**	**4.0057**
asylum	**32330**	**madhouse**	**2570**	**108**	**3.6079**	**15.6656**	**4.9673**
magician	**18400**	**wizard**	**128820**	**1650**	**3.5000**	**13.6656**	**4.3382**
midday	**17170**	**noon**	**159880**	**2430**	**3.4575**	**12.3925**	**4.5785**
furnace	**24980**	**stove**	**30340**	**1600**	**3.0038**	**1.7135**	**5.4477**
food	**1196860**	**fruit**	**184670**	**77280**	**2.8758**	**5.0076**	**3.6496**
bird	**225920**	**cock**	**253030**	**3720**	**2.8729**	**9.3139**	**1.9683**
bird	**225920**	**crane**	**49980**	**4565**	**2.7888**	**9.3139**	**3.7949**
tool	**785720**	**implement**	**362050**	**72200**	**3.0438**	**6.0787**	**3.3293**
brother	**355150**	**monk**	**27480**	**4595**	**2.7325**	**2.9683**	**3.9472**
crane	**49980**	**implement**	**362050**	**1715**	**1.3925**	**2.9683**	**2.3442**
lad	**14830**	**brother**	**355150**	**3400**	**1.5642**	**2.9355**	**4.2628**
journey	**233660**	**car**	**951850**	**31855**	**1.0642**	**0.0000**	**2.7572**
monk	**27480**	**oracle**	**140690**	**830**	**1.0375**	**2.9683**	**3.1619**
food	**1196860**	**rooster**	**9170**	**2390**	**0.9337**	**1.0105**	**3.1761**
coast	**445230**	**hill**	**547540**	**47400**	**0.8346**	**6.2344**	**3.0628**
forest	**337650**	**graveyard**	**24630**	**1975**	**0.7900**	**0.0000**	**3.2628**
monk	**27480**	**slave**	**114490**	**1410**	**0.5813**	**2.9683**	**3.8979**
coast	**445230**	**forest**	**337650**	**42020**	**0.4575**	**0.0000**	**3.4258**
lad	**14830**	**wizard**	**128820**	**675**	**0.4783**	**2.9683**	**3.6601**
chord	**23500**	**smile**	**161390**	**2845**	**0.1238**	**2.3544**	**4.4130**
glass	**296840**	**magician**	**18400**	**2150**	**0.1079**	**1.0105**	**3.7682**
noon	**159880**	**string**	**400930**	**6500**	**0.0633**	**0.0000**	**2.4118**
rooster	**9170**	**voyage**	**49380**	**205**	**0.0633**	**0.0000**	**3.9080**

Table 6.2.3. Web page counts for word pairs, and similarity measures based on subjective judgments (n = 48), Resnik's *wsim* measure using Wordnet, and statistical similarity (*ssim*) using *Hotbot*. (Test performed on Aug. 10, 1999).

The correlation between Resnik's *wsim* and the combined judgments of the 48 human subjects (*subj*) is 0.8001. The correlation between *ssim* and these judgments is 0.4745. On further examination, it was found that for word pairs with low probabilities, that the combined counts $N(w_1, w_2)$ are overestimated. For example, many of the pages retrieved by the query *rooster voyage* contained, for unknown reasons, only the word *rooster*, or were very large collections of unrelated text (e.g. a list of book titles). To compensate, the parameter *k* was added to equation (6.2.1). With k = 0.78, a correlation of 0.5745 was observed. While weaker than Resnik's *wsim*, it does show that semantic judgment of word similarity is at least partly a statistical property of language

7. Syntactic Model

We now complete the model by adding the ability to learn a context free grammar. We do this in two steps. The first step is minor, and adds a "no-repeat" rule, a general rule that says that words rarely repeat consecutively. The second step adds the grammar by endowing the model with the capability of learning to classify words by their context. When these categories or part-of-speech symbols are concatenated to form new symbols as in the lexical model, the result is a set of nonterminals in a context free grammar.

7.1. No-Repeat Rule

The goal is to add an efficient (O(n)) syntax rule: $P(w_1w_1) < P(w_1)^2$ to the semantic model of section 6. The rule says that symbols are usually not repeated, as in *aa* or *the the*. The goal is twofold:

1. *Statistical* -- Which methods improve compression the most? How does $P(w_1)$ affect the rule? When does the bigram $P(w_1w_1)$ override (as in *ee* or *had had*)?
2. *Linguistic* -- Which symbols are affected? Letters? Syllables? Function words? Content words? Parts of speech?

7.1.1. Background

It is commonly observed that words rarely repeat, for example, *the the*. However, I am not aware of any language models that use this rule.

7.1.2. Procedure

Output rule P5 (remove I from B) will be added to the language model and compared with models without this rule for text compression.

7.1.3. Preliminary results

To test whether a no-repeat rule would benefit a character model, the probability that two n-grams separated by a distance *t* was calculated for n = 1 through 6 and *t* = 1, 2, 4, 8,..., 64K on *Alice in Wonderland* after reduction to a 27 character alphabet (a-z and space), 137 Kb. The results are graphed below.

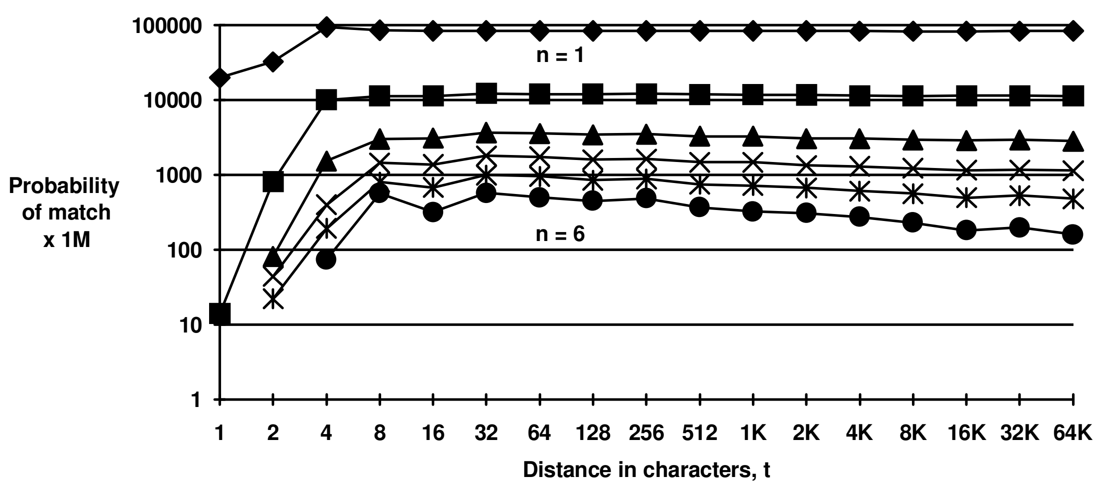

Figure 7.1.3. Probability (in parts per million) that two strings of length 1 (diamonds) to 6 (circles) separated by a distance of 1 to 64K characters will match in *Alice in Wonderland* when reduced to a 27 character alphabet.

The results clearly show a no-repeat rule for all n-grams of length 1 to 6. They also show a cache-effect (section 6.1) for characters that peaks at t = 4, and for strings of length 3 to 6 (i.e. words) that peaks at about t = 50, or about 10 words. Match probabilities continue to decline up to the length of the input file, demonstrating that text is not stationary (section 4).

7.2. Hidden Markov Model Induction

The goal is to add an efficient (O(n)) method of classifying symbols by their immediate context and applying group statistics to make individual predictions. The goal is twofold:

1. *Statistical* -- Which methods improve text compression the most? To what extent should we generalize across categories? Do the categories form a hierarchical equivalence relation, and if so, how broad and how deep? How much memory is needed?
2. *Linguistic* -- Do the categories correspond to known parts of speech such as nouns, adjective, etc.? Does it group on a finer level (mass nouns, people, colors, etc.)? At the character level, does the model group vowels, consonants, digits, etc.? Does it learn upper/lower case equivalencies?

In a hidden Markov model (HMM), n-gram statistics are smoothed by categorizing words into their parts of speech to increase the available statistics. For example, P(*store* | *to the*) =
P(*noun* | *prep det*)P(*store* | *noun*). Most work with HMMs is with manually tagged corpora such as the Penn treebank (Charniak, 1993; Manning and Schütze, 1999). Even with automatically tagged systems, the goal is usually not language modeling but some intermediate task such as word sense disambiguation or parsing.

Jelinek, Mercer, and Roikos (1990) built a model that learns the *statistical synonyms* of novel words by grouping them with known words that have the same immediate context (previous two words, next two words, or previous and next word). For example, by one of their measures, the top five synonyms of the novel word *3rd* were *third, second, first, each, d*; and for *hydroxide* they found *chloride, salt, diffusion, light, contamination* (words that follow *sodium*). They significantly reduced the contribution of novel words to perplexity using this method.

Teahan and Cleary (1997) investigated whether adding part of speech tags (*noun, det*, etc.) to text would improve compression. They found that word-level PPM on the alternating sequence of manually added tags and words (with some reordering of the history) produced the same amount of output as the words alone. Essentially, the tags were compressed for free. In a later experiment (1998), they obtained similar results with automated taggers.

I believe that both methods can be improved. The work on statistical synonyms started with each word in its own category and only classified new words. This could be generalized to classify words already in the vocabulary. The categories could be made broader by using a one word context instead of two. For tagged compression, results could be combined using ME (with a neural network as in section 4) instead of PPM. Thus, if trigram statistics (last word and last tag) were insufficient, then the pair of bigram statistics (last word or last tag) could be combined, instead of using just one or the other as in PPM.

7.2.1. Procedure

The procedure is to construct a syntactic model by adding update rules U9, U10, and U11, and output rule P6. The update rules form categories or sets of symbols when they appear in the same context. Thus, after observing the sequences *ca* and *cb*, we would form a classification $\{a, b\}$ because they both appear after the context *c*. Likewise with *ac* and *bc*, which share the same following context. Update rule U9 generalizes the current input, $a \rightarrow \{a, b\}$. Output rule P6 does the opposite, decomposing a category into its elements, e.g. $\{a, b\} \rightarrow a, b$.

As a purely qualitative test, a bigram matrix **B** will be constructed, where B_{ij} is the probability of observing the bigram w_iw_j. Then the context matrix $\mathbf{C} = \mathbf{B}\mathbf{B}^t + \mathbf{B}^t\mathbf{B}$ groups the symbols into categories, such that C_{ij} is the probability that w_i and w_j appear in the same context. The matrix will be constructed once for letters (256 character alphabet) and once for words (27 character alphabet and limited vocabulary), and the largest entries of C will be observed. For the letter model, we should observe that vowels are grouped with other vowels, consonants with consonants, spaces with newlines, periods with commas, and so on. In the word matrix, we should observe groupings by parts of speech, and perhaps finer classifications such as {*red, yellow, green*} or {*dog, cat*}.

If this works, then we proceed to add the syntax rules to the full model and evaluate the effects on text compression.

7.3. Context Free Grammar Induction

The goal is to add efficient (O(n)) context free grammar (CFG) induction to the model by applying lexical concatenation (section 5) to the POS symbols learned in the HMM model of 7.2 to form new symbols, corresponding to the nonterminals in a CFG. The goal is twofold:

1. *Statistical* -- Does text compression improve? (Unlike the previous models, unsupervised CFG induction is unsolved). How many rules are learned? How much memory is needed?
2. *Linguistic* -- Do the rules learned correspond to familiar hand-built models (for example, *NP* → *det adj noun*).

7.3.1. Background

Grammar is undoubtedly the most difficult aspect of a language to learn. It is the last stage of language development in children. It is the only aspect of NLP that has not been successfully applied to a product. It has failed to improve speech recognition (Stolcke, 1997) or information retrieval (Grossman and Frieder, 1998). The problem of modeling grammar is essentially unsolved.

Most work in grammar modeling has been directed toward solving what seem to be intermediate problems such as parsing, tagging, or word sense disambiguation. (See Charniak 1993; Manning and Schütze, 1999, for reviews). Among the rare exceptions are the work of O'Sullivan, Mark, and Miller (1994) combining a probabilistic context free grammar (PCFG) and n-gram word model, and of Wright, Jones, and Lloyd-Thomas (1994) combining n-grams, distant bigrams, and a PCFG. In the latter case, adding a hand coded PCFG with 100 rules (50% coverage) to the ARM corpus (500 word vocabulary) improved perplexity from 15.3 to 11.7. Still, the problem of unsupervised (untagged corpus) PCFG learning without a predefined set of rules is unsolved.

I believe that trying to solve intermediate problems such as parsing, etc., only make the problem harder. We observe that people learn to use grammar correctly without knowing how to diagram sentences, or even the difference between a noun and a verb. My strategy is simply to skip the parsing step. If the input fits the grammar model but is ambiguous, then we blend the predictions of all of the rules that apply to the current context. If the input is ungrammatical, then we still have the lower level models to fall back on.

7.3.2. Procedure

The model is finished! When we combine the word classification rules of section 7.2 with the symbol concatenation rules of the lexical model of section 5, we obtain the nonterminal symbols of a CFG. Thus, after frequent occurrences of the symbol pair *det, noun*, we add a new symbol, *noun-phrase*. These symbols are in turn concatenated to form more rules like *sentence* → *noun-phrase verb-phrase*. In this phase of the work, I plan only to observe the symbols in the model qualitatively to find examples of the known grammar rules of English to confirm that a CFG is indeed being learned.

8. Entropy of Written English

In sections 4 through 7 we developed a language model by successively adding character, lexical, semantic, and syntactic components. We evaluated each model on the same corpus and observed the relationship between model size and entropy. From these results, we were able to predict to some extent what the entropy would be for larger models, beyond those that were practical to test. From this projection, we would be able to estimate what model size would be needed to equal the entropy of the corpus as determined by human prediction tests. The size of that model is the number we seek: the space cost of natural language processing.

Unfortunately we are not done, because we still need to test the corpus using human subjects. In section 2.4 I described the problems with current testing techniques. Shannon (1950) proved entropy bounds from a character ranking game, but these are rather broad, 0.6 to 1.3 bpc in his tests. Cover and King (1978) had subjects assign probabilities directly in a gambling game, but the subjective difficulty of this task led to overestimates of the entropy, 1.3 to 1.7 bpc for individual subjects on the same corpus that Shannon used. We cannot accept the overlap (1.3 bpc) as the true entropy, because several models have already done better than this (e.g. 1.22 bpc by Rosenfeld, 1996), which would imply that the AI problem has already been solved.

8.0.1. Procedure

In the next three sections, I describe improvements to Shannon's character ranking tests that narrow the bounds without requiring subjects to assign probabilities. The human entropy of the test corpus used in sections 4 through 7 will be measured by recruiting subjects to play the "Shannon game", a character guessing game over the Internet. The web page will announce high scores, i.e. those needing the fewest guesses to complete the text string. High scoring contestants will also be asked if they know the source of the text, so their results can be excluded from the entropy measurements (but not from the high score list). The entropy will be estimated from the median scores of the remaining results. Test strings will be drawn randomly each time so that repeat players will not see the same text twice. Contestants will be asked their age in order to measure the correlation with entropy. (I suspect that entropy will decrease with age). The game will be implemented as a JavaScript/CGI program.

Shannon allowed subjects to use dictionaries and n-gram statistics ($n \leq 3$) as aids. To reproduce these conditions (and make the game faster), the player will be presented with up to 27 choices (A-Z and space, removing characters already guessed), ranked in order from most likely to least likely. The ranking will be done using a language model (probably a simple character model) trained on the text corpus after removing the test string. To speed up the game, the player can press a button at any time to guess all of the remaining characters in the order presented.

8.1. Comparing Character Ranking with Entropy

The goal is to find an empirical relationship between model evaluation by character ranking and by entropy in order to better estimate the entropy of English using Shannon's (1950) results.

8.1.1. Background

Recall from section 2.4 that Shannon (1950) measured the entropy of written English by having subjects try to predict successive characters in a string of text. From these experiments, Shannon obtained a set of probabilities $p_1 = 0.8$, $p_2 = 0.07$, etc., where p_r is the probability that it will take r tries to correctly guess the next character. Shannon proved that the entropy, H bpc, is bounded by

$$\Sigma_r\, r(p_r - p_{r+1}) \log_2 r \leq H \leq \Sigma_r\, p_r \log_2 1/p_r \qquad (2.4.1)$$

which is rather broad, 0.6 to 1.3 bpc in his experiments.

8.1.2. Procedure

For a machine model, we can calculate H directly by compressing the corpus. We can also obtain the p_r by having the model assign probabilities to each character, sorting them from most likely to least likely, and observing the position of the correct guess. Using this data, we could obtain an empirical relationship between the bounds and the true entropy.

Because the relationship may depend on the size and the level of the model, the experiment will be performed on each of the models in sections 4 through 7 for various corpus sizes. The most accurate data should come from the model that is closest to the human model, i.e. section 7 with the largest corpus possible. Testing the smaller models will allow us to observed any trends in the relationship between bounds and entropy as the level or size increases, in order to improve the quality of the estimate.

8.2. Entropy Estimation by Partitioned Ranking

The goal is to improve on Shannon's (1950) method of measuring the entropy of written text by reducing the mix of "easy" and "hard" parts of the test.

8.2.1. Background

The large range of entropy bounds in (2.4.1) is due to the variability in the difficulty of guessing some characters over others. The upper bound holds for the case where every letter position is equally hard to guess, i.e. subjects are always 80% accurate on every position in the string. This is not so, because letters at the beginning of a word are usually harder to guess than those near the end. The lower bound holds for the case where some positions are harder than others, but for each one, there are n equally likely choices, where n ranges from 1 to 27. This is not the case either, because some letters are always more likely than others regardless of their position. Thus, we expect that the true entropy would be somewhere in the middle of the bounds (fig. 8.2.1).

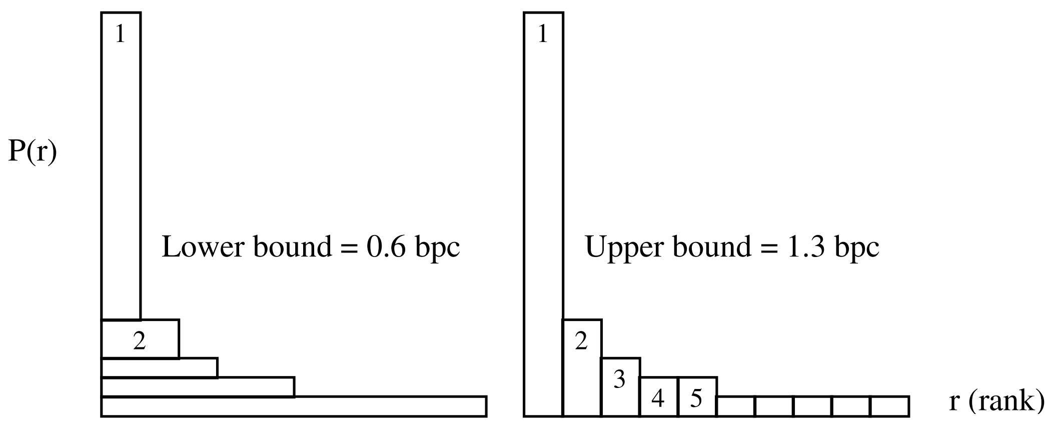

Fig. 8.2.1. Estimating entropy from a rank distribution in a character guessing game. The lower bound assumes that each character instance has a uniform distribution among one or more possibilities. The upper bound assumes that each instance has the same distribution as all of the others.

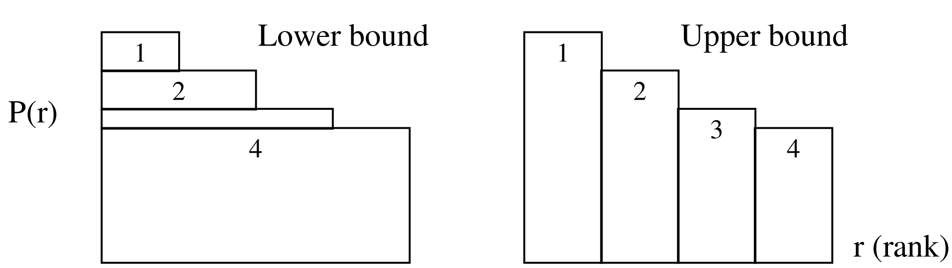

Fig. 8.2.2. The effect of reducing the distribution skew is to tighten the bounds of the entropy estimate.

If there are n equally likely choices at each position, then the subject can do no better than guess randomly. Then, $p_1 = p_2 = ... = p_n = 1/n$, and the bounds in (2.4.1) are equal:

$$\Sigma_r\, r(p_r - p_{r+1}) \log_2 r = \Sigma_r\, p_r \log_2 1/p_r = \log_2 n = H \qquad (8.2.1)$$

Thus, we would expect that any variation of the Shannon game that makes the guesses more uniform should tighten the bounds. Here are three ways to do this.

8.2.2. Procedure

1. *Partition by position*. Test data is collected using the Shannon game described in section 8. Then the results are partitioned into two sets: the "hard" part, the first letter of each word, and the "easy" part, the remainder of the text. The entropy bounds of each set will be determined using (2.4.1). Then the bounds of the total entropy is the sum of the bounds of each set, weighted by set size. Bounds will also be calculated for larger partitionings, such as the first, second, third, and remaining letters of each word.

2. *Partition by free guesses*. In a variation of the Shannon game, the player will be given a limited number of "free" guesses, which can be used before the first guess at any position in the string. The player still has to guess the letter, but the guesses won't count in calculating the score. Then the test will be partitioned into two sets, a "hard" set where a free guess was used, and an "easy" set. Then the bounds will be calculated as in part 1.

3. *Language model partitioning*. Instead of presenting the player with a choice of letters, the language model divides the hidden text into n equally likely ranges, which might extend across more than one character position. For instance, with n = 2, the choices might be "space and A-EF or EG to Z". We obtain p_1 = fraction of correct guesses, and $p_2 = 1 - p_1$. Then we apply (2.4.1), multiply by the number of guesses, and divide by the length of the test string to obtain the bounds on H.

9. Conclusion

In section 1, we saw that language modeling is critical to a wide range of natural language processing problems, all of which are unsolved, and that the best evaluation of a model is its ability to compress text. In section 2, we looked at various approaches to estimating how hard the problem is, specifically how much space is required, and chose the statistical modeling approach as the most promising. The evidence suggests a size of 10^8 to 10^{10} bits. Most of the uncertainty is due to variations in the models tested and the training text used, and our uncertain knowledge of the entropy of written language with respect to human performance.

In section 3 I described how the character, lexical, semantic, and syntactic components of a model could be integrated. The model is purely statistical -- it requires no supervised training or prior language knowledge to implement. It learns a language from nothing more than a large collection of text. In sections 4 through 7 I described how each of the four language components could be added to the model and tested before proceeding to the next level. In section 8 I described how the entropy of written language with respect to human performance could be measured more accurately.

9.1. Potential Contributions

Although the goal of this proposal is to estimate the difficulty of AI without solving it, it is clear that this research will also lead to a greater understanding of the AI problem. To wit:

- In section 4, I showed that the maximum-entropy method of combining evidence from different sources can be implemented in an on-line system using a neural network. I also showed that neural networks can be implemented efficiently for language modeling, in spite of their reputation for being computationally intensive.
- In section 5, I showed that vocabulary acquisition is possible without prior language knowledge or spaces between words. This would explain how children are able to learn to segment continuous speech even before they can use or understand any words. In my research I hope to get the same results using an on-line model. I also hope to get the same or better compression using a bigram model than a word trigram model that uses *ad hoc* text segmentation.
- In section 6.1, I hope to show that adding a cache eliminates the need for, or at least simplifies, the short term learning rate (η_S) discussed in section 4.
- In section 6.2, I showed that search engine statistics could be used to learn word associations, but other researchers have gotten better results on smaller corpora using LSA's ability to exploit indirect associations. I hope to show that LSA can be implemented as an on-line neural network (off-line has been demonstrated using singular value decomposition). I hope to improve on LSA, first by modeling associations between word fragments to solve the vocabulary building problem, and second, by adding connections between the units on the hidden layer to model the hierarchical relationships between words. I hope to show the dual role of short term memory as both a cache and a text window for learning associations. The optimal sizes for the two purposes need not be the same, but if it is, it would give strong support for the psychological model of short term memory.
- In section 7.1, I showed that English apparently has a no-repeat rule, and I hope to show that a model that exploits it gives better results, by eliminating the need for a bigram rule for each symbol in the vocabulary.
- In section 7.2, I hope to show that it is possible to learn a context free grammar from untagged text with no initial rule set, a feat never accomplished. Showing this would help explain how humans are able to do this. I also hope to show that grammar is a completely general property of natural language, that it also applies below the word level and above the sentence level.
- In section 8.1, I hope to find an empirical relationship between character ranking performance and entropy and apply them to the results of Shannon's classic 1950 measurements to better estimate the entropy of English.
- In section 8.2, I hope to find the entropy of English for a variety of corpora, and perhaps how entropy is affected by the age and education level of the test subjects. When we get close to solving AI, we may describe models as having the language skills of an X year old child.
- Overall I hope to estimate the size of the language model needed for human performance. For historical reasons, I would like to confirm or refine Turing's 1950 estimate of 10^9 bits. This would also help resolve a difficult question in psycholingistics: how much of our language knowledge comes from nonverbal sources? Since the model size determines both the hardware and software costs of an AI system, knowing the size would allow us to make intelligent decisions on

where to direct research dollars: should we build or not? Finally, by plotting the historical growth of hardware performance, and the accompanying language models, we would be able to predict when the AI problem will be solved.

9.2. Summary of Proposed Work

The proposed work is detailed in sections 4 through 8. To summarize,

- 4. Character model. The work is complete. It was found that the maximum entropy approach to combining evidence could be implemented in on-line form with reasonable efficiency as a neural network. I have submitted a paper to the FLAIRS special track on neural network applications, and will present it at the conference in Orlando in May if it is accepted. Notification is Jan. 20.
- 5. Lexical model. This remains to be built. Preliminary evidence shows the feasibility of parsing text into words with no language knowledge other than n-gram statistics. The proposed model uses hierarchical symbol building, a different but proven algorithm.
- 6. Semantic model. This remains to be built. Preliminary work shows a cache effect, and the technique is proven. Experiments with search engines show that the numerical data they return is a rather poor indicator of semantic relationships in spite of the enormous corpus size. LSA is a proven method for smaller corpora, but an on-line neural network version needs to be designed and tested. The interaction of the cache with long term memory has not been explored previously, nor have the two alternatives I have proposed to LSA: sublexical associations and a hierarchical semantic space.
- 7. Syntactic model. This remains to be built. In spite of the simplicity of the "no-repeat" rule, and preliminary data showing its effect, I know of no prior studies of its effectiveness. On the other hand, there has been an enormous effort to solve the problem of unsupervised learning of context free grammars, to no avail. I am concerned that the approach I outlined is too simple, and it does not seem to depend on the semantic level as I argued it should.
- 8. Entropy of English. This remains to be done. Best results depend on a full language model being available, but preliminary results could be obtained with the character model, which has already been built. I plan to build a prototype in order to resolve issues with recruiting test subjects over the Internet.

References

Åberg, J., Yu. M. Shtarkov, B. J. M. Smeets (1997), “Estimation of escape probabilities for PPM based on universal source coding theory”, Proc. Intl. Symposium on Information Theory, 65.

Anderson, James A., (1983) “Cognitive and Psychological Computation with Neural Models”, IEEE Transactions on Systems, Man, and Cybernetics 13(5) Sept./Oct., pp. 799-815.

Bahl, L. R., et. al. (1989), “Large vocabulary natural language continuous speech recognition”, IEEE ICASSP, Vol. 1, 465-567.

Beardsley, Tim (1999), “Getting Wired -- New observations may show how neurons form connections”, Scientific American 280(6) June, 24-25.

Bell, Timothy, Ian H. Witten, John G. Cleary (1989), “Modeling for Text Compression”, ACM Computing Surveys (21)4, pp. 557-591, Dec. 1989

Bell, Timothy (1998), Canterbury Corpus, http://corpus.canterbury.ac.nz/

Bellegarda, Jerome R., John W. Butzberger, Yen-Lu Chow, Noah B. Coccaro, Devang Naik (1996), “A novel word clustering algorithm based on latent semantic analysis”, Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, vol. 1, 172-175.

Bellegarda, Jerome R. (1998), “Exploiting both local and global constraints for multi-span statistical language modeling”, IEEE Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing, vol. 2, 677-680.

Bellegarda, Jerome R. (1999), “Speech recognition experiments using multi-span statistical language models”, IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, 717-720.

Bimbot, F., M. El-Beze, M. Jardino (1997), “An alternative scheme for perplexity estimation”, Proc. IEEE ICASSP, Vol. 2., 1483-1486.

Bloom, Charles (1998), “Solving the Problems of Context Modeling”, http://www.cco.caltech.edu/~bloom/papers/ppmz.zip

Bordel, G., I Torres, E. Vidal (1995), “QWI: A method for improved smoothing in language modelling”, IEEE ICASSP, Vol. 1, 185-188.

Borko, Harold (1967), *Automated Language Processing, The State of the Art*, New York: Wiley.

Brill, E., and Mooney, R. J. (1997), “An overview of Empirical Natural Language Processing”, AI Magazine, 18(4), 13-24.

M. Burrows and D. J. Wheeler (1994), A Block-sorting Lossless Data Compression Algorithm, Digital Systems Research Center, http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-124.html

Calgary Corpus (1993), http://www.kiarchive.ru/pub/msdos/compress/calgarycorpus.zip

Cardie, Claire (1997), “Empirical Methods of Information Extraction”, AI Magazine 18(4) 65-79

Carroll, Lewis (1865), *Alice in Wonderland*, Gutenberg Press, ftp://sunsite.unc.edu/pub/docs/books/gutenberg/etext97/alice30h.zip

Charniak, Eugene (1997), “Statistical Techniques for Natural Language Parsing”, AI Magazine 18(4) 33-43.

Charniak, Eugene (1993), *Statistical Language Learning*, Cambridge MA: MIT Press.

Clarkson, P. R., and A. J. Robinson (1997), "Language model adaptation using mixtures and an exponentially decaying cache", IEEE ICASSP, Vol. 2, 799-802.

Cleary, John G., W. J. Teahan, Ian H. Witten (1995), "Unbounded length contexts for PPM", Proc. Data Compression Conference, 52-61.

Cleary, John G., W. J. Teahan (1995), "Experiments on the zero frequency problem", Proc. Data Compression Conference, 480.

compress 4.3d for MSDOS (1990), ftp://ctan.tug.org/tex-archive/tools/compress/msdos.zip

Cover, T. M., and R. C. King (1978), "A Convergent Gambling Estimate of the Entropy of English", IEEE Transactions on Information Theory (24)4 (July) pp. 413-421.

Crick, F. H. C. and C. Asanuma (1986), Certain aspects of the anatomy and physiology of the cerebral cortex, in Rumelhart, David E., James L. McClelland, and the PDP Research Group (1986), *Parallel Distributed Processing*, vol. 2, Cambridge MA: MIT Press. p. 333-371.

Data Compression FAQ (July 25, 1998), http://www.cs.ruu.nl/wais/html/na-dir/compression-faq/.html

Della Pietra, S., V. Della Pietra, R. Mercer, S. Roukos, (1992) "Adaptive Language Modeling Using Minimum Discriminant Estimation". *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 633-636, San Francisco, Mar. 1992. Also *Proceedings of the DARPA Workshop on Speech and Natural Language*, Morgan Kaufmann, 103-106, Feb. 1992.

Feldman, J. A. and D. H. Ballard (1982), "Connectionist Models and their Properties", Cognitive Science (6)., pp. 205-254.

Fenwick, Peter (1997), "Symbol ranking text compressors", Data Compression Conference, 436.

Geutner, Petra (1996), "Introducing linguistic constraints into statistical language modeling", Proc. 4'th Intl. Conf. on Spoken Language, Vol. 1, 402-405.

Giachin, Egidio P. (1995), "Phrase bigrams for continuous speech recognition", ICASSP Vol. 1, 225-228.

Gilchrist, Jeff (Sept. 1998), Archive Comparison Test, http://www.geocities.com/SiliconValley/Park/4264/act-mcal.html

Gorin, Allen L., Giuseppe Riccardi (1999), "Spoken language variation over time and state in a natural spoken dialog system", Proc. IEEE ICASSP, Vol. 2, 721-724.

Green, Bert F. Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery (1961), Baseball: An Automatic Question Answerer, Proceedings of the Western Joint Computer Conference, 19:219-224, reprinted in *Computers and Thought,* E. A. Feigenbaum and J. Feldman eds, New York: McGraw Hill, 1963.

Grossman, David A., and Ophir Frieder (1998), *Information Retrieval: Algorithms and Heuristics,* Boston: Kluwer Academic Publishers.

Guha, R. V., and D. B. Lenat (1994), "Comparing CYC to Other AI Systems", http://www.cyc.com/tech-reports/act-cyc-406-91/act-cyc-406-91.html

gzip 1.2.4 (1993), Jean-loup-Gailly, http://www.kiarchive.ru/pub/msdos/compress/gzip124.exe

Harman D., (1995) (ed.), "Overview of the Third Text Retrieval Conference (TREC-3)", National Institute of Standards and Technology Special Publication 500-225, Gaitersberg MD 20879

Hebb, D. O. (1949), *The Organization of Behavior*, New York: Wiley.

Heeman, Peter A., Géraldine Dammani (1997), "Deriving phrase-based language models", IEEE Workshop on Automatic Speech Recognition and Understanding, 41-48.

Hirvola, H., 1993. ha 0.98, http://www.webwaves.com/arcers/msdos/ha098.zip

Hopfield, J. J. (1982), "Neural networks and physical systems with emergent collective computational abilities", Proceedings of the National Academy of Sciences (79) 2554-2558.

Hörmann, Hans (1979), *Psycholinguistics*, 2nd Ed., New York: Springer-Verlag.

Huang, Shell Ying, and Ghim Hwee Ong (1993), "Entropies of Chinese texts based on three models of Hanyu Pinyin phonetic system", Proc. Intl. Conf. on Information Engineering, Communications and Networks for the year 2000, vol. 1, 305-309.

Huang, Xiaohong, Zhensheng Luo, Jian Tang (1997), "A quick method for Chinese word segmentation", IEEE Intl. Conf. on Intelligent Processing Systems, Vol. 2, 1773-1776.

Hutchens, Jason L., and Michael D. Alder (1997), "Language Acquisition and Data Compression", 1997 Australasian Natural Language Processing Summer Workshop Notes (Feb.), pp. 39-49, http://ciips.ee.uwa.edu/au/~hutch/research/papers

Hutchens, Jason L., and Michael D. Alder (1998), "Finding Structure via Compression", Proceedings of the International Conference on Computational Natural Language Learning, pp. 79-82, http://ciips.ee.uwa.edu/au/~hutch/research/papers

Hwang, Kyuwoong (1997), "Vocabulary optimization based on perplexity", IEEE ICASSP, Vol. 2, 1419-1422.

Iyer, R., M. Ostendorf, M. Meteer (1997), Proc. IEEE Workshop on Automatic Speech Recognition and Understanding, 254-261.

Jelinek, F., R. Mercer, Salim Roukos (1990), "Classifying words for improved statistical language models", IEEE ICASSP, Vol. 1, 621-624.

Jiang, J., S. Jones (1992), "Word-based dynamic algorithms for data compression", IEE Proc. Communication, Speech, and Vision, 139(6): 582-586.

Jusczyk, Peter W. (1996), "Investigations of the word segmentation abilities of infants", 4'th Intl. Conf. on Speech and Language Processing, Vol. 3, 1561-1564.

Kalai, Adam, Stanley Chen, Avrim Blum, Ronald Rosenfeld (1999), "On-line algorithms for combining language models", IEEE Proc. Intl. Conf. on Acoustics Speech, and Signal Processing, 745-748.

Khudanpur, Sanjeev, Jun Wu (1999), "A maximum entropy language model integrating n-grams and topic dependencies for conversational speech recognition", Proc. IEEE Conf. on Acoustics, Speech, and Signal Processing, 553-556.

Klakow, Dietrich (1998), "Language-model optimization by mapping of corpora", Proc. IEEE ICASSP, Vol. 2, 701-704.

Kneser, Reinhard, Volker Steinbiss (1993), "On the dynamic adaptation of stochastic language models", IEEE ICASSP, Vol. 2, 586-589.

Kneser, Reinhard, and Hermann Ney (1995), "Improved backing-off for m-gram language modeling", IEEE ICASSP, Vol. 1, 181-184.

Kneser, Reinhard (1996), "Statistical language modeling using a variable context length", IEEE ICSLP, Vol. 1, 494-497.

Knight, Kevin, (1997), "Automatic Knowledge Acquisition for Machine Translation", AI Magazine, 18(4) 81-96.

Kuffler, Stephen W., Nicholls, John G., Martin, Robert A., (1984), *From Neuron to Brain*, 2nd ed., Sunderland MA: Sinauer Associates Inc.

Kupiec, Julian, and John Maxwell (1992), "Training Stochastic Grammars from Unlabeled Text Corpora", in *Statistically-Based Natural Language Programming Techniques*, Papers from the 1992 AAAI Workshop, Technical Report W-92-01, Menlo Park CA: AAAI Press. pp. 8-13.

Landauer, Tom (1986), "How much do people remember? Some estimates of the quantity of learned information in long term memory", Cognitive Science (10) pp. 477-493.

Landauer, T. K., & Dumais, S. T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. Psychological Review, 104, 211-240.

Landauer, T. K., Laham, D., Rehder, B., & Schreiner, M. E., (1997). How well can passage meaning be derived without using word order? A comparison of Latent Semantic Analysis and humans. In M. G. Shafto & P. Langley (Eds.), Proceedings of the 19th annual meeting of the Cognitive Science Society (pp. 412-417). Mahwah, NJ: Erlbaum.

Lau, Raymond, Ronald Rosenfeld, Salim Roukos (1993), "Trigger-based language models: a maximum entropy approach", IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, vol. 2, 45-48.

Little, W. A., and Gordon L. Shaw (1975), "A statistical theory of short and long term memory", Behavioral Biology (14) pp. 115-133.

Loebner, Hugh, (1998) Home Page of The Loebner Prize--"The First Turing Test". http://www.loebner.net/Prizef/loebner-prize.html

Luo, Xiaoqiang, and Frederick Jelenick (1999), "Probabilistic classification of HMM states for large vocabulary continuous speech recognition", IEEE ICASSP, Vol. 1, 353-356.

Mahajan, M., D. Beeferman, X. D. Huang (1999), "Improved topic-dependent language modeling using information retrieval techniques", IEEE ICASSP, Vol. 1, 541-544.

Mahoney, M. V. (1999), "Fast text compression with neural networks", submitted to FLAIRS, http://mahoney4.home.netcom.com/compression/

Manning, Christopher D., and Hinrich Schütze, (1999), *Foundations of Statistical Natural Language Processing*, Cambridge MA: MIT Press.

Martin, S. C., H. Ney, J. Zaplo (1999), "Smoothing methods in maximum entropy language modeling", Proc. IEEE ICASSP, Vol. 1, 545-548.

McClelland, James L., and David E. Rumelhart (1981), "An interactive activation model of context effects in letter perception: part 1. An account of basic findings", Psychological Review (88) pp. 375-407.

McDonald, John, (1998), 200% Probability and Beyond: The Compelling Nature of Extraordinary Claims in the Absence of Alternative Explanations. Skeptical Inquirer (22)1: 45-49, 64.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K. (1993), Introduction to WordNet: An On-line Lexical Database, ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps

Mitchell Tom M. (1997), *Machine Learning*, New York: McGraw-Hill

Nevill-Manning, Craig G., Ian H. Witten, Dan R. Olsen, Jr. (1996), "Compressing semi-structured text using hierarchical phrase identifications", IEEE Proc. Data Compression Conference, 63-72.

Nevill-Manning, Craig G., Ian H. Witten (1997), "Inferring lexical and grammatical structure from sequences", IEEE Proc. Conf. on Compression and Complexity of Sequences, 265-274.

Newell, Allen, H. A. Simon (1961), “GPS: A Program that Simulates Human Thought”, *Lernende Automaten*, Munich: R. Oldenbourg KG, reprinted in *Computers and Thought,* E. A. Feigenbaum and J. Feldman eds, New York: McGraw Hill, 1963.

Newsbytes (1988), “Automatic Language Translation System for Telecommunications”, Mar. 22, 1988, http://www.nbnn.com/pubNews/88/48342.html

Ney, H., U. Essen, R. Kneser (1995), “On the estimation of ‘small’ probabilities by leaving-one-out”, IEEE Trans. Pattern Analysis and Machine Intelligence, 17(12) 1202-1212.

O’Sullivan, J. A., Mark, K., Miller, M. I. (1994), “Markov random fields on graphs for natural languages”, Proc. Conf. Information Theory and Statistics, 47.

Osterson, Daniel N., Michael Stob, and Scott Weinstein (1986), *Systems that Learn*, Cambridge MA: MIT Press.

Papalia, Diane E., and Sally Wendkos-Olds (1990), *A Child’s World, Infancy through Adolescence*, New York: McGraw-Hill.

Penrose, Roger (1990), *The Emperor’s New Mind*, New York: Oxford University Press.

Peterson, I., (1998), “Web searches fall short”, Science News (153) p. 286, May 2.

PKZIP (1993), version 2.04e, PKWARE Inc.

Porter, M. F. (1980), “An algorithm for suffix striping”, Program (14), pp. 130-137.

Resnik, Philip (1992), “WordNet and Distributional Analysis: A Class-based Approach to Lexical Discovery”, in *Statistically-Based Natural Language Programming Techniques*, Papers from the 1992 AAAI Workshop, Technical Report W-92-01, Menlo Park CA: AAAI Press, pp. 48-56, http://www.jair.org/abstracts/resnik99a.html

Ries, Klaus, Finn Dag Buø, Alex Waibel (1996), “Class phrase models for language modeling”, ICSLP, Vol. 1, 398-401.

Ristad, E. S, Thomas, R. G (1997), “Nonuniform Markov models”, ICASSP, vol. 2, 791-794.

Rosenfeld, Ronald (1996), “A Maximum Entropy Approach to Adaptive Statistical Language Modeling”, Computer, Speech and Language, 10, http://www.cs.cmu.edu/afs/cs/user/roni/WWW/me-csl-revised.ps

Rudnicky, A. I., Hauptmann, A. G., Lee, K. (1994), “Survey of Current Speech Technology”, Communications of the ACM, 37(3), 52-57.

Rumelhart, David E., James L. McClelland, and the PDP Research Group (1986), *Parallel Distributed Processing*, Cambridge MA: MIT Press.

Schindler, Michael (1998), szip homepage, http://www.compressconsult.com/szip/

Schindler, Micheal (1997), A Fast Block-sorting Algorithm for Lossless Data Compression, 1997 Data Compression Conference, http://www.compressconsult.com/szip/

Schmidhuber, Jürgen, Stefan Heil (1996), “Sequential neural text compression”, IEEE Trans. on Neural Networks 7(1): 142-146.

Schwartz, Barry, and Daniel Reisberg (1991), *Learning and Memory*, New York: W. W. Norton and Company.

Seymore, Kristie, Ronald Rosenfeld (1996), “Scalable backoff language models”, Proc. Intl. Conf. on Spoken Language, 232-235.

Shannon, Claude, and Warren Weaver (1949), *The Mathematical Theory of Communication*, Urbana: University of Illinois Press.

Shannon, Cluade E. (1950), “Prediction and Entropy of Printed English”, Bell Sys. Tech. J (3) p. 50-64.

Simons, M., H. Ney, S. C. Martin (1997), “Distant bigram language modelling using maximum entropy”, IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, vol. 2, 787-790.

Stolcke, Andreas (1997), “Linguistic Knowledge and Empirical Methods in Speech Recognition”, AI Magazine 18(4), 25-31.

Storer, James A. (1988), *Data Compression*, *methods and theory*, Rockville MD: Computer Science Press.

Sutton, Ian (1998), boa 0.58 beta, http://webhome.idirect.com/~isutton/

Tan, C. P., (1981), “On the Entropy of the Malay Language”, IEEE Transactions on Information Theory (27)3 (May) pp. 383-384.

Taylor, Malcolm, RKIVE v1.91 beta 1 (1998) http://www.geocities.com/SiliconValley/Peaks/9463/rkive.html

Teahan, W. J., John G. Cleary (1996), “The entropy of English using PPM-based models”, IEEE Proc. Data Compression Conference, 53-62.

Teahan, W. J., John G. Cleary (1997), “Models of English text”, IEEE Proc. Data Compression Conference, 12-21.

Teahan, W. J., John G. Cleary (1998), Tag based models of English text, Proc. Data Compression Conference, 289-298.

Teahan, W. J., S. Englis, J. G. Cleary, G. Holmes (1998), “Correcting English text using PPM models”, IEEE Proc. Data Compression Conference, 289-298.

Turing, A. M., (1950) “Computing Machinery and Intelligence, Mind, 59:433-460, reprinted in *Computers and Thought,* E. A. Feigenbaum and J. Feldman eds, New York: McGraw Hill, 1963, also http://www.loebner.net/Prizef/TuringArticle.html

Uhr, Leonard, Charles Vossler (1963) “A Pattern-Recognition Program that Generates, Evaluates, and Adjusts its own Operators”, *Computers and Thought,* E. A. Feigenbaum and J. Feldman eds, New York: McGraw Hill, pp. 251-268

Wardhaugh, Ronald (1972), *Introduction to Linguistics*, New York: McGraw-Hill.

Waterman, Michael S. (1995), *Introduction to Computational Biology*, London: Chapman & Hall.

Webster’s New World Dictionary of the American Language, College Edition (1964), Cleveland: The World Publishing Co.

Whitten, David (1994) “The Unofficial, Unauthorized CYC Frequently Asked Questions Information Sheet.”, http://www.mcs.net/~jorn/html/ai/cycfaq.html

Witten, Ian H., Timothy C. Bell (1991), “The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression”, IEEE Trans. on Information Theory, 37(4): 1085-1094.

Wright, J. H., G. I. F. Jones, H. Lloyd-Thomas (1994), “A robust language model incorporating a substring parser and extended n-grams”, IEEE ICASSP, Vol. 1, 361-364.

Yang, Kae-Cheng, Tai-Hsuan Ho, Lee-Feng Chien, Lin-Shan Lee (1998), “Statistics-based segment pattern lexicon -- a new direction for Chinese language modeling”, Proc. IEEE ICASSP, Vol. 1, 169-172.

Zipf, George Kingley (1935), *The Psycho-Biology of Language, an Introduction to Dynamic Philology*, M.I.T. Press.