1996; Niesler and Woodland, 1996).

In general, using very large and rich contexts can result in very large language models. Thus these models are often pruned, removing low-probably events. Pruning is also essential for using language models on small platforms such as cellphones (Stolcke, 1998).

Finally, there is a wide body of research on integrating sophisticated linguistic structures into language modeling. Language models based on syntactic structure from probabilistic parsers are described in Ch. 14. Language models based on the current speech act in dialogue are described in Ch. 23.

## 4.10 Advanced: Information Theory Background

*I got the horse right here*
Frank Loesser, Guys and Dolls

We introduced perplexity in Sec. 4.4 as a way to evaluate $N$-gram models on a test set. A better $N$-gram model is one which assigns a higher probability to the test data, and perplexity is a normalized version of the probability of the test set. Another way to think about perplexity is based on the information-theoretic concept of **cross-entropy**. In order to give another intuition into perplexity as a metric, this section gives a quick review of fundamental facts from **information theory** including the concept of cross-entropy that underlies perplexity. The interested reader should consult a good information theory textbook like Cover and Thomas (1991).

ENTROPY

Perplexity is based on the information-theoretic notion of **cross-entropy**, which we will now work toward defining. **Entropy** is a measure of information, and is invaluable throughout speech and language processing. It can be used as a metric for how much information there is in a particular grammar, for how well a given grammar matches a given language, for how predictive a given $N$-gram grammar is about what the next word could be. Given two grammars and a corpus, we can use entropy to tell us which grammar better matches the corpus. We can also use entropy to compare how difficult two speech recognition tasks are, and also to measure how well a given probabilistic grammar matches human grammars.

Computing entropy requires that we establish a random variable $X$ that ranges over whatever we are predicting (words, letters, parts of speech, the set of which we'll call $\chi$), and that has a particular probability function, call it $p(x)$. The entropy of this random variable $X$ is then

$$H(X) = -\sum_{x\in\chi} p(x)\log_2 p(x) \tag{4.54}$$

The log can in principle be computed in any base; if we use log base 2, the resulting value of entropy will be measured in **bits**.

The most intuitive way to define entropy for computer scientists is to think of the entropy as a lower bound on the number of bits it would take to encode a certain decision or piece of information in the optimal coding scheme.

Cover and Thomas (1991) suggest the following example. Imagine that we want to place a bet on a horse race but it is too far to go all the way to Yonkers Racetrack, and we'd like to send a short message to the bookie to tell him which horse to bet on. Suppose there are eight horses in this particular race.

One way to encode this message is just to use the binary representation of the horse's number as the code; thus horse 1 would be `001`, horse 2 `010`, horse 3 `011`, and so on, with horse 8 coded as `000`. If we spend the whole day betting, and each horse is coded with 3 bits, on the average we would be sending 3 bits per race.

Can we do better? Suppose that the spread is the actual distribution of the bets placed, and that we represent it as the prior probability of each horse as follows:

| | | | |
|---|---|---|---|
| Horse 1 | $\frac{1}{2}$ | Horse 5 | $\frac{1}{64}$ |
| Horse 2 | $\frac{1}{4}$ | Horse 6 | $\frac{1}{64}$ |
| Horse 3 | $\frac{1}{8}$ | Horse 7 | $\frac{1}{64}$ |
| Horse 4 | $\frac{1}{16}$ | Horse 8 | $\frac{1}{64}$ |

The entropy of the random variable $X$ that ranges over horses gives us a lower bound on the number of bits, and is:

$$\begin{aligned} H(X) &= -\sum_{i=1}^{i=8} p(i)\log p(i) \\ &= -\tfrac{1}{2}\log\tfrac{1}{2}-\tfrac{1}{4}\log\tfrac{1}{4}-\tfrac{1}{8}\log\tfrac{1}{8}-\tfrac{1}{16}\log\tfrac{1}{16}-4(\tfrac{1}{64}\log\tfrac{1}{64}) \\ &= 2 \text{ bits} \end{aligned} \quad (4.55)$$

A code that averages 2 bits per race can be built by using short encodings for more probable horses, and longer encodings for less probable horses. For example, we could encode the most likely horse with the code `0`, and the remaining horses as `10`, then `110`, `1110`, `111100`, `111101`, `111110`, and `111111`.

What if the horses are equally likely? We saw above that if we use an equal-length binary code for the horse numbers, each horse took 3 bits to code, and so the average was 3. Is the entropy the same? In this case each horse would have a probability of $\frac{1}{8}$. The entropy of the choice of horses is then:

$$H(X) = -\sum_{i=1}^{i=8}\frac{1}{8}\log\frac{1}{8} = -\log\frac{1}{8} = 3 \text{ bits} \quad (4.56)$$

Until now we have been computing the entropy of a single variable. But most of what we will use entropy for involves *sequences*; for a grammar, for example, we will be computing the entropy of some sequence of words $W = \{w_0, w_1, w_2, \ldots, w_n\}$. One way to do this is to have a variable that ranges over sequences of words. For example we can compute the entropy of a random variable that ranges over all finite sequences of words of length $n$ in some language $L$ as follows:

$$H(w_1, w_2, \ldots, w_n) = -\sum_{W_1^n \in L} p(W_1^n)\log p(W_1^n) \quad (4.57)$$

ENTROPY RATE

We could define the **entropy rate** (we could also think of this as the **per-word entropy**) as the entropy of this sequence divided by the number of words:

$$\frac{1}{n}H(W_1^n) = -\frac{1}{n}\sum_{W_1^n \in L} p(W_1^n)\log p(W_1^n) \tag{4.58}$$

But to measure the true entropy of a language, we need to consider sequences of infinite length. If we think of a language as a stochastic process $L$ that produces a sequence of words, its entropy rate $H(L)$ is defined as:

$$\begin{aligned} H(L) &= -\lim_{n\to\infty}\frac{1}{n}H(w_1,w_2,\ldots,w_n) \\ &= -\lim_{n\to\infty}\frac{1}{n}\sum_{W\in L} p(w_1,\ldots,w_n)\log p(w_1,\ldots,w_n) \end{aligned} \tag{4.59}$$

The Shannon-McMillan-Breiman theorem (Algoet and Cover, 1988; Cover and Thomas, 1991) states that if the language is regular in certain ways (to be exact, if it is both stationary and ergodic),

$$H(L) = \lim_{n\to\infty} -\frac{1}{n}\log p(w_1w_2\ldots w_n) \tag{4.60}$$

That is, we can take a single sequence that is long enough instead of summing over all possible sequences. The intuition of the Shannon-McMillan-Breiman theorem is that a long enough sequence of words will contain in it many other shorter sequences, and that each of these shorter sequences will reoccur in the longer sequence according to their probabilities.

STATIONARY

A stochastic process is said to be **stationary** if the probabilities it assigns to a sequence are invariant with respect to shifts in the time index. In other words, the probability distribution for words at time $t$ is the same as the probability distribution at time $t+1$. Markov models, and hence $N$-grams, are stationary. For example, in a bigram, $P_i$ is dependent only on $P_{i-1}$. So if we shift our time index by $x$, $P_{i+x}$ is still dependent on $P_{i+x-1}$. But natural language is not stationary, since as we will see in Ch. 11, the probability of upcoming words can be dependent on events that were arbitrarily distant and time dependent. Thus our statistical models only give an approximation to the correct distributions and entropies of natural language.

To summarize, by making some incorrect but convenient simplifying assumptions, we can compute the entropy of some stochastic process by taking a very long sample of the output, and computing its average log probability. In the next section we talk about the why and how; *why* we would want to do this (i.e., for what kinds of problems would the entropy tell us something useful), and *how* to compute the probability of a very long sequence.

## 4.10.1 Cross Entropy for Comparing Models

CROSS ENTROPY

In this section we introduce the **cross entropy**, and discuss its usefulness in comparing different probabilistic models. The cross entropy is useful when we don't know the actual probability distribution $p$ that generated some data. It allows us to use some $m$, which is a model of $p$ (i.e., an approximation to $p$. The cross-entropy of $m$ on $p$ is defined by:

$$H(p,m) = \lim_{n\to\infty} \frac{1}{n} \sum_{W\in L} p(w_1,\ldots,w_n)\log m(w_1,\ldots,w_n) \tag{4.61}$$

That is we draw sequences according to the probability distribution $p$, but sum the log of their probability according to $m$.

Again, following the Shannon-McMillan-Breiman theorem, for a stationary ergodic process:

$$H(p,m) = \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \ldots w_n) \tag{4.62}$$

This means that, as for entropy, we can estimate the cross-entropy of a model $m$ on some distribution $p$ by taking a single sequence that is long enough instead of summing over all possible sequences.

What makes the cross entropy useful is that the cross entropy $H(p,m)$ is an upper bound on the entropy $H(p)$. For any model $m$:

$$H(p) \le H(p,m) \tag{4.63}$$

This means that we can use some simplified model $m$ to help estimate the true entropy of a sequence of symbols drawn according to probability $p$. The more accurate $m$ is, the closer the cross entropy $H(p,m)$ will be to the true entropy $H(p)$. Thus the difference between $H(p,m)$ and $H(p)$ is a measure of how accurate a model is. Between two models $m_1$ and $m_2$, the more accurate model will be the one with the lower cross-entropy. (The cross-entropy can never be lower than the true entropy, so a model cannot err by underestimating the true entropy).

We are finally ready to see the relation between perplexity and the cross-entropy we saw in Equation (4.62). Cross-entropy is defined in the limit, as the length of the observed word sequence goes to infinity. We will need an approximation to cross-entropy, relying on a (sufficiently long) sequence of fixed length. This approximation to the cross-entropy of a model $M = P(w_i|w_{i-N+1}...w_{i-1})$ on a sequence of words $W$ is:

$$H(W) = -\frac{1}{N}\log P(w_1 w_2 \ldots w_N) \tag{4.64}$$

PERPLEXITY

The **perplexity** of a model $P$ on a sequence of words $W$ is now formally defined as the exp of this cross-entropy:

$$\begin{aligned}
\text{Perplexity}(W) &= 2^{H(W)} \\
&= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}
\end{aligned}$$

$$= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}} \quad (4.65)$$

## 4.11 Advanced: The Entropy of English and Entropy Rate Constancy

As we suggested in the previous section, the cross-entropy of some model $m$ can be used as an upper bound on the true entropy of some process. We can use this method to get an estimate of the true entropy of English. Why should we care about the entropy of English?

One reason is that the true entropy of English would give us a solid lower bound for all of our future experiments on probabilistic grammars. Another is that we can use the entropy values for English to help understand what parts of a language provide the most information (for example, is the predictability of English mainly based on word order, on semantics, on morphology, on constituency, or on pragmatic cues?) This can help us immensely in knowing where to focus our language-modeling efforts.

There are two common methods for computing the entropy of English. The first was employed by Shannon (1951), as part of his groundbreaking work in defining the field of information theory. His idea was to use human subjects, and to construct a psychological experiment that requires them to guess strings of letters; by looking at how many guesses it takes them to guess letters correctly we can estimate the probability of the letters, and hence the entropy of the sequence.

The actual experiment is designed as follows: we present a subject with some English text and ask the subject to guess the next letter. The subjects will use their knowledge of the language to guess the most probable letter first, the next most probable next, and so on. We record the number of guesses it takes for the subject to guess correctly. Shannon's insight was that the entropy of the number-of-guesses sequence is the same as the entropy of English. (The intuition is that given the number-of-guesses sequence, we could reconstruct the original text by choosing the "$n$th most probable" letter whenever the subject took $n$ guesses). This methodology requires the use of letter guesses rather than word guesses (since the subject sometimes has to do an exhaustive search of all the possible letters!), and so Shannon computed the **per-letter entropy** of English rather than the per-word entropy. He reported an entropy of 1.3 bits (for 27 characters (26 letters plus space)). Shannon's estimate is likely to be too low, since it is based on a single text (*Jefferson the Virginian* by Dumas Malone). Shannon notes that his subjects had worse guesses (hence higher entropies) on other texts (newspaper writing, scientific work, and poetry). More recently variations on the Shannon experiments include the use of a gambling paradigm where the subjects get to bet on the next letter (Cover and King, 1978; Cover and Thomas, 1991).

The second method for computing the entropy of English helps avoid the single-text problem that confounds Shannon's results. This method is to take a very good stochastic model, train it on a very large corpus, and use it to assign a log-probability to a very long sequence of English, using the Shannon-McMillan-Breiman theorem:

$$H(\text{English}) \leq \lim_{n\to\infty} -\frac{1}{n}\log m(w_1 w_2 \ldots w_n) \tag{4.66}$$

For example, Brown et al. (1992a) trained a trigram language model on 583 million words of English, (293,181 different types) and used it to compute the probability of the entire Brown corpus (1,014,312 tokens). The training data include newspapers, encyclopedias, novels, office correspondence, proceedings of the Canadian parliament, and other miscellaneous sources.

They then computed the character-entropy of the Brown corpus, by using their word-trigram grammar to assign probabilities to the Brown corpus, considered as a sequence of individual letters. They obtained an entropy of 1.75 bits per character (where the set of characters included all the 95 printable ASCII characters).

The average length of English written words (including space) has been reported at 5.5 letters (Nádas, 1984). If this is correct, it means that the Shannon estimate of 1.3 bits per letter corresponds to a per-word perplexity of 142 for general English. The numbers we report earlier for the WSJ experiments are significantly lower than this, since the training and test set came from the same subsample of English. That is, those experiments underestimate the complexity of English (since the Wall Street Journal looks very little like Shakespeare, for example)

A number of scholars have independently made the intriguing suggestion that entropy rate plays a role in human communication in general (Lindblom, 1990; Van Son et al., 1998; Aylett, 1999; Genzel and Charniak, 2002; Van Son and Pols, 2003). The idea is that people speak so as to keep the rate of information being transmitted per second roughly constant, i.e. transmitting a constant number of bits per second, or maintaining a constant entropy rate. Since the most efficient way of transmitting information through a channel is at a constant rate, language may even have evolved for such communicative efficiency (Plotkin and Nowak, 2000). There is a wide variety of evidence for the constant entropy rate hypothesis. One class of evidence, for speech, shows that speakers shorten predictable words (i.e. they take less time to say predictable words is shorter) and lengthen unpredictable words (Aylett, 1999; Jurafsky et al., 2001; Aylett and Turk, 2004). In another line of research, Genzel and Charniak (2002, 2003) show that entropy rate constancy makes predictions about the entropy of individual sentences from a text. In particular, they show that it predicts that local measures of sentence entropy which ignore previous discourse context (for example the $N$-gram probability of sentence), should increase with the sentence number, and they document this increase in corpora. Keller (2004) provides evidence that entropy rate plays a role for the addressee as well, showing a correlation between the entropy of a sentence and the processing effort it causes in comprehension, as measured by reading times in eye-tracking data.

## Bibliographical and Historical Notes

The underlying mathematics of the $N$-gram was first proposed by Markov (1913), who used what are now called **Markov chains** (bigrams and trigrams) to predict whether an upcoming letter in Pushkin's *Eugene Onegin* would be a vowel or a consonant. Markov classified 20,000 letters as V or C and computed the bigram and trigram probability that a given letter would be a vowel given the previous one or two letters. Shannon (1948) applied $N$-grams to compute approximations to English word sequences. Based on Shannon's work, Markov models were commonly used in engineering, linguistic, and psychological work on modeling word sequences by the 1950s.

In a series of extremely influential papers starting with Chomsky (1956) and including Chomsky (1957) and Miller and Chomsky (1963), Noam Chomsky argued that "finite-state Markov processes", while a possibly useful engineering heuristic, were incapable of being a complete cognitive model of human grammatical knowledge. These arguments led many linguists and computational linguists to ignore work in statistical modeling for decades.

The resurgence of $N$-gram models came from Jelinek, Mercer, Bahl, and colleagues at the IBM Thomas J. Watson Research Center, influenced by Shannon, and Baker at CMU, influenced by the work of Baum and colleagues. These two labs independently successfully used $N$-grams in their speech recognition systems (Baker, 1990; Jelinek, 1976; Baker, 1975; Bahl et al., 1983; Jelinek, 1990). A trigram model was used in the IBM TANGORA speech recognition system in the 1970s, but the idea was not written up until later.

Add-one smoothing derives from Laplace's 1812 law of succession, and was first applied as an engineering solution to the zero-frequency problem by Jeffreys (1948) based on an earlier Add-K suggestion by ? (?). Problems with the Add-one algorithm are summarized in Gale and Church (1994). The Good-Turing algorithm was first applied to the smoothing of $N$-gram grammars at IBM by Katz, as cited in Nádas (1984). Church and Gale (1991) gives a good description of the Good-Turing method, as well as the proof.Sampson (1996) also has a useful discussion of Good-Turing. Jelinek (1990) summarizes this and many other early language model innovations used in the IBM language models.

A wide variety of different language modeling and smoothing techniques were tested through the 1980's and 1990's, including those we discuss as well as Witten-Bell discounting (Witten and Bell, 1991), varieties of class-based models (Jelinek, 1990; Kneser and Ney, 1993; Heeman, 1999; Samuelsson and Reichl, 1999), and others (Gupta et al., 1992). In the late 1990's, Chen and Goodman produced a very influential series of papers with a comparison of different language models (Chen and Goodman, 1996, 1998, 1999; Goodman, 2006). They performed a number of carefully controlled experiments comparing different discounting algorithms, cache models, class-based (cluster) models, and other language model parameters. They showed the advantages of Interpolated Kneser-Ney, which has since become one of the most popular current methods for language modeling. These papers influenced our discussion in this chapter, and are recommended reading if you have further interest in language modeling.

As we suggested earlier in the chapter, recent research in language modeling has focused on adaptation, and also on the use of sophisticated linguistic structures based on syntactic and dialogue structure.