CHAPTER 3

N-gram Language Models

"You are uniformly charming!" cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

Random sentence generated from a Jane Austen trigram model

Predicting is difficult—especially about the future, as the old quip goes. But how about predicting something that seems much easier, like the next few words someone is going to say? What word, for example, is likely to follow

```
Please turn your homework ...
```

Hopefully, most of you concluded that a very likely word is *in*, or possibly *over*, but probably not *refrigerator* or *the*. In the following sections we will formalize this intuition by introducing models that assign a **probability** to each possible next word. The same models will also serve to assign a probability to an entire sentence. Such a model, for example, could predict that the following sequence has a much higher probability of appearing in a text:

```
all of a sudden I notice three guys standing on the sidewalk
```

than does this same set of words in a different order:

```
on guys all I of notice sidewalk three a sudden standing the
```

Why would you want to predict upcoming words, or assign probabilities to sentences? Probabilities are essential in any task in which we have to identify words in noisy, ambiguous input, like **speech recognition**. For a speech recognizer to realize that you said *I will be back soonish* and not *I will be bassoon dish*, it helps to know that *back soonish* is a much more probable sequence than *bassoon dish*. For writing tools like **spelling correction** or **grammatical error correction**, we need to find and correct errors in writing like *Their are two midterms*, in which *There* was mistyped as *Their*, or *Everything has improve*, in which *improve* should have been *improved*. The phrase *There are* will be much more probable than *Their are*, and *has improved* than *has improve*, allowing us to help users by detecting and correcting these errors.

Assigning probabilities to sequences of words is also essential in **machine translation**. Suppose we are translating a Chinese source sentence:

他	向	记者	介绍了	主要	内容
He	to	reporters	introduced	main	content

As part of the process we might have built the following set of potential rough English translations:

he introduced reporters to the main contents of the statement
he briefed to reporters the main contents of the statement
he briefed reporters on the main contents of the statement

A probabilistic model of word sequences could suggest that *briefed reporters on* is a more probable English phrase than *briefed to reporters* (which has an awkward *to* after *briefed*) or *introduced reporters to* (which uses a verb that is less fluent English in this context), allowing us to correctly select the boldfaced sentence above.

AAC

Probabilities are also important for **augmentative and alternative communication** systems (Trnka et al. 2007, Kane et al. 2017). People often use such **AAC** devices if they are physically unable or sign but can instead using eye gaze or other specific movements to select words from a menu to be spoken by the system. Word prediction can be used to suggest likely words for the menu.

language model
LM
n-gram

Models that assign probabilities to sequences of words are called **language models** or **LMs**. In this chapter we introduce the simplest model that assigns probabilities to sentences and sequences of words, the **n-gram**. An n-gram is a sequence of N words: a 2-gram (or **bigram**) is a two-word sequence of words like "please turn", "turn your", or "your homework", and a 3-gram (or **trigram**) is a three-word sequence of words like "please turn your", or "turn your homework". We'll see how to use n-gram models to estimate the probability of the last word of an n-gram given the previous words, and also to assign probabilities to entire sequences. In a bit of terminological ambiguity, we usually drop the word "model", and thus the term **n-gram** is used to mean either the word sequence itself or the predictive model that assigns it a probability. In later chapters we'll introduce more sophisticated language models like the RNN LMs of Chapter 9.

3.1 N-Grams

Let's begin with the task of computing $P(w|h)$, the probability of a word w given some history h. Suppose the history h is "*its water is so transparent that*" and we want to know the probability that the next word is *the*:

$$P(the|its\ water\ is\ so\ transparent\ that). \tag{3.1}$$

One way to estimate this probability is from relative frequency counts: take a very large corpus, count the number of times we see *its water is so transparent that*, and count the number of times this is followed by *the*. This would be answering the question "Out of the times we saw the history h, how many times was it followed by the word w", as follows:

$$P(the|its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)} \tag{3.2}$$

With a large enough corpus, such as the web, we can compute these counts and estimate the probability from Eq. 3.2. You should pause now, go to the web, and compute this estimate for yourself.

While this method of estimating probabilities directly from counts works fine in many cases, it turns out that even the web isn't big enough to give us good estimates in most cases. This is because language is creative; new sentences are created all the time, and we won't always be able to count entire sentences. Even simple extensions of the example sentence may have counts of zero on the web (such as "*Walden Pond's water is so transparent that the*"; well, *used to* have counts of zero).

Similarly, if we wanted to know the joint probability of an entire sequence of words like *its water is so transparent*, we could do it by asking "out of all possible sequences of five words, how many of them are *its water is so transparent*?" We would have to get the count of *its water is so transparent* and divide by the sum of the counts of all possible five word sequences. That seems rather a lot to estimate!

For this reason, we'll need to introduce cleverer ways of estimating the probability of a word w given a history h, or the probability of an entire word sequence W. Let's start with a little formalizing of notation. To represent the probability of a particular random variable X_i taking on the value "the", or $P(X_i = \text{"the"})$, we will use the simplification $P(the)$. We'll represent a sequence of N words either as $w_1 \dots w_n$ or w_1^n (so the expression w_1^{n-1} means the string $w_1, w_2, ..., w_{n-1}$). For the joint probability of each word in a sequence having a particular value $P(X = w_1, Y = w_2, Z = w_3, ..., W = w_n)$ we'll use $P(w_1, w_2, ..., w_n)$.

Now how can we compute probabilities of entire sequences like $P(w_1, w_2, ..., w_n)$? One thing we can do is decompose this probability using the **chain rule of probability**:

$$
\begin{aligned}
P(X_1 ... X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1^2)\dots P(X_n|X_1^{n-1}) \\
&= \prod_{k=1}^{n} P(X_k|X_1^{k-1})
\end{aligned}
\tag{3.3}
$$

Applying the chain rule to words, we get

$$
\begin{aligned}
P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\dots P(w_n|w_1^{n-1}) \\
&= \prod_{k=1}^{n} P(w_k|w_1^{k-1})
\end{aligned}
\tag{3.4}
$$

The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words. Equation 3.4 suggests that we could estimate the joint probability of an entire sequence of words by multiplying together a number of conditional probabilities. But using the chain rule doesn't really seem to help us! We don't know any way to compute the exact probability of a word given a long sequence of preceding words, $P(w_n|w_1^{n-1})$. As we said above, we can't just estimate by counting the number of times every word occurs following every long string, because language is creative and any particular context might have never occurred before!

The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can **approximate** the history by just the last few words.

bigram

The **bigram** model, for example, approximates the probability of a word given all the previous words $P(w_n|w_1^{n-1})$ by using only the conditional probability of the preceding word $P(w_n|w_{n-1})$. In other words, instead of computing the probability

$$P(\text{the}|\text{Walden Pond's water is so transparent that}) \tag{3.5}$$

we approximate it with the probability

$$P(\text{the}|\text{that}) \tag{3.6}$$

When we use a bigram model to predict the conditional probability of the next word, we are thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1}) \tag{3.7}$$

Markov

The assumption that the probability of a word depends only on the previous word is called a **Markov** assumption. Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past. We can generalize the bigram (which looks one word into the past) to the trigram (which looks two words into the past) and thus to the **n-gram** (which looks $n-1$ words into the past).

n-gram

Thus, the general equation for this n-gram approximation to the conditional probability of the next word in a sequence is

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \tag{3.8}$$

Given the bigram assumption for the probability of an individual word, we can compute the probability of a complete word sequence by substituting Eq. 3.7 into Eq. 3.4:

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}) \tag{3.9}$$

maximum likelihood estimation

How do we estimate these bigram or n-gram probabilities? An intuitive way to estimate probabilities is called **maximum likelihood estimation** or **MLE**. We get the MLE estimate for the parameters of an n-gram model by getting counts from a corpus, and **normalizing** the counts so that they lie between 0 and 1.[1]

normalize

For example, to compute a particular bigram probability of a word y given a previous word x, we'll compute the count of the bigram $C(xy)$ and normalize by the sum of all the bigrams that share the same first word x:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \tag{3.10}$$

We can simplify this equation, since the sum of all bigram counts that start with a given word w_{n-1} must be equal to the unigram count for that word w_{n-1} (the reader should take a moment to be convinced of this):

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{3.11}$$

Let's work through an example using a mini-corpus of three sentences. We'll first need to augment each sentence with a special symbol <s> at the beginning of the sentence, to give us the bigram context of the first word. We'll also need a special end-symbol. </s>[2]

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

[1] For probabilistic models, normalizing means dividing by some total count so that the resulting probabilities fall legally between 0 and 1.

[2] We need the end-symbol to make the bigram grammar a true probability distribution. Without an end-symbol, the sentence probabilities for all sentences of a given length would sum to one. This model would define an infinite set of probability distributions, with one distribution per sentence length. See Exercise 3.5.

Here are the calculations for some of the bigram probabilities from this corpus

$P(\texttt{I}|\texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam}|\texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am}|\texttt{I}) = \frac{2}{3} = .67$
$P(\texttt{</s>}|\texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam}|\texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do}|\texttt{I}) = \frac{1}{3} = .33$

For the general case of MLE n-gram parameter estimation:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \tag{3.12}$$

Equation 3.12 (like Eq. 3.11) estimates the n-gram probability by dividing the observed frequency of a particular sequence by the observed frequency of a prefix. This ratio is called a **relative frequency**. We said above that this use of relative frequencies as a way to estimate probabilities is an example of maximum likelihood estimation or MLE. In MLE, the resulting parameter set maximizes the likelihood of the training set T given the model M (i.e., $P(T|M)$). For example, suppose the word *Chinese* occurs 400 times in a corpus of a million words like the Brown corpus. What is the probability that a random word selected from some other text of, say, a million words will be the word *Chinese*? The MLE of its probability is $\frac{400}{1000000}$ or .0004. Now .0004 is not the best possible estimate of the probability of *Chinese* occurring in all situations; it might turn out that in some other corpus or context *Chinese* is a very unlikely word. But it is the probability that makes it *most likely* that Chinese will occur 400 times in a million-word corpus. We present ways to modify the MLE estimates slightly to get better probability estimates in Section 3.4.

relative frequency

Let's move on to some examples from a slightly larger corpus than our 14-word example above. We'll use data from the now-defunct Berkeley Restaurant Project, a dialogue system from the last century that answered questions about a database of restaurants in Berkeley, California (Jurafsky et al., 1994). Here are some text-normalized sample user queries (a sample of 9332 sentences is on the website):

can you tell me about any good cantonese restaurants close by
mid priced thai food is what i'm looking for
tell me about chez panisse
can you give me a listing of the kinds of food that are available
i'm looking for a good place to eat breakfast
when is caffe venezia open during the day

Figure 3.1 shows the bigram counts from a piece of a bigram grammar from the Berkeley Restaurant Project. Note that the majority of the values are zero. In fact, we have chosen the sample words to cohere with each other; a matrix selected from a random set of seven words would be even more sparse.

Figure 3.2 shows the bigram probabilities after normalization (dividing each cell in Fig. 3.1 by the appropriate unigram for its row, taken from the following set of unigram probabilities):

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Here are a few other useful probabilities:

$P(\texttt{i}|\texttt{<s>}) = 0.25 \qquad P(\texttt{english}|\texttt{want}) = 0.0011$
$P(\texttt{food}|\texttt{english}) = 0.5 \qquad P(\texttt{</s>}|\texttt{food}) = 0.68$

Now we can compute the probability of sentences like *I want English food* or *I want Chinese food* by simply multiplying the appropriate bigram probabilities together, as follows:

	i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

$$
\begin{aligned}
P(\texttt{<s> i want english food </s>}) & \\
&= P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{i})P(\texttt{english}|\texttt{want}) \\
&\qquad P(\texttt{food}|\texttt{english})P(\texttt{</s>}|\texttt{food}) \\
&= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\
&= .000031
\end{aligned}
$$

We leave it as Exercise 3.2 to compute the probability of *i want chinese food*.

What kinds of linguistic phenomena are captured in these bigram statistics? Some of the bigram probabilities above encode some facts that we think of as strictly **syntactic** in nature, like the fact that what comes after *eat* is usually a noun or an adjective, or that what comes after *to* is usually a verb. Others might be a fact about the personal assistant task, like the high probability of sentences beginning with the words *I*. And some might even be cultural rather than linguistic, like the higher probability that people are looking for Chinese versus English food.

trigram
4-gram
5-gram

Some practical issues: Although for pedagogical purposes we have only described bigram models, in practice it's more common to use **trigram** models, which condition on the previous two words rather than the previous word, or **4-gram** or even **5-gram** models, when there is sufficient training data. Note that for these larger n-grams, we'll need to assume extra context for the contexts to the left and right of the sentence end. For example, to compute trigram probabilities at the very beginning of the sentence, we can use two pseudo-words for the first trigram (i.e., $P(\texttt{I}|\texttt{<s><s>})$).

log probabilities

We always represent and compute language model probabilities in log format as **log probabilities**. Since probabilities are (by definition) less than or equal to 1, the more probabilities we multiply together, the smaller the product becomes. Multiplying enough n-grams together would result in numerical underflow. By using log probabilities instead of raw probabilities, we get numbers that are not as small.

Adding in log space is equivalent to multiplying in linear space, so we combine log probabilities by adding them. The result of doing all computation and storage in log space is that we only need to convert back into probabilities if we need to report them at the end; then we can just take the exp of the logprob:

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4) \tag{3.13}$$

3.2 Evaluating Language Models

The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves. Such end-to-end evaluation is called **extrinsic evaluation**. Extrinsic evaluation is the only way to know if a particular improvement in a component is really going to help the task at hand. Thus, for speech recognition, we can compare the performance of two language models by running the speech recognizer twice, once with each language model, and seeing which gives the more accurate transcription.

extrinsic evaluation

Unfortunately, running big NLP systems end-to-end is often very expensive. Instead, it would be nice to have a metric that can be used to quickly evaluate potential improvements in a language model. An **intrinsic evaluation** metric is one that measures the quality of a model independent of any application.

intrinsic evaluation

For an intrinsic evaluation of a language model we need a **test set**. As with many of the statistical models in our field, the probabilities of an n-gram model come from the corpus it is trained on, the **training set** or **training corpus**. We can then measure the quality of an n-gram model by its performance on some unseen data called the **test set** or test corpus. We will also sometimes call test sets and other datasets that are not in our training sets **held out** corpora because we hold them out from the training data.

training set

test set

held out

So if we are given a corpus of text and want to compare two different n-gram models, we divide the data into training and test sets, train the parameters of both models on the training set, and then compare how well the two trained models fit the test set.

But what does it mean to "fit the test set"? The answer is simple: whichever model assigns a **higher probability** to the test set—meaning it more accurately predicts the test set—is a better model. Given two probabilistic models, the better model is the one that has a tighter fit to the test data or that better predicts the details of the test data, and hence will assign a higher probability to the test data.

Since our evaluation metric is based on test set probability, it's important not to let the test sentences into the training set. Suppose we are trying to compute the probability of a particular "test" sentence. If our test sentence is part of the training corpus, we will mistakenly assign it an artificially high probability when it occurs in the test set. We call this situation **training on the test set**. Training on the test set introduces a bias that makes the probabilities all look too high, and causes huge inaccuracies in **perplexity**, the probability-based metric we introduce below.

Sometimes we use a particular test set so often that we implicitly tune to its characteristics. We then need a fresh test set that is truly unseen. In such cases, we call the initial test set the **development** test set or, **devset**. How do we divide our data into training, development, and test sets? We want our test set to be as large as possible, since a small test set may be accidentally unrepresentative, but we also want as much training data as possible. At the minimum, we would want to pick

development test

the smallest test set that gives us enough statistical power to measure a statistically significant difference between two potential models. In practice, we often just divide our data into 80% training, 10% development, and 10% test. Given a large corpus that we want to divide into training and test, test data can either be taken from some continuous sequence of text inside the corpus, or we can remove smaller "stripes" of text from randomly selected parts of our corpus and combine them into a test set.

3.2.1 Perplexity

perplexity

In practice we don't use raw probability as our metric for evaluating language models, but a variant called **perplexity**. The **perplexity** (sometimes called *PP* for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words. For a test set $W = w_1 w_2 \dots w_N$,:

$$\begin{aligned}\mathrm{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \quad (3.14)\\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}\end{aligned}$$

We can use the chain rule to expand the probability of W:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}} \quad (3.15)$$

Thus, if we are computing the perplexity of W with a bigram language model, we get:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \quad (3.16)$$

Note that because of the inverse in Eq. 3.15, the higher the conditional probability of the word sequence, the lower the perplexity. Thus, minimizing perplexity is equivalent to maximizing the test set probability according to the language model. What we generally use for word sequence in Eq. 3.15 or Eq. 3.16 is the entire sequence of words in some test set. Since this sequence will cross many sentence boundaries, we need to include the begin- and end-sentence markers <s> and </s> in the probability computation. We also need to include the end-of-sentence marker </s> (but not the beginning-of-sentence marker <s>) in the total count of word tokens N.

There is another way to think about perplexity: as the **weighted average branching factor** of a language. The branching factor of a language is the number of possible next words that can follow any word. Consider the task of recognizing the digits in English (zero, one, two,..., nine), given that (both in some training set and in some test set) each of the 10 digits occurs with equal probability $P = \frac{1}{10}$. The perplexity of this mini-language is in fact 10. To see that, imagine a test string of digits of length N, and assume that in the training set all the digits occurred with equal probability. By Eq. 3.15, the perplexity will be

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= (\frac{1}{10}^N)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned} \tag{3.17}$$

But suppose that the number zero is really frequent and occurs far more often than other numbers. Let's say that 0 occur 91 times in the training set, and each of the other digits occurred 1 time each. Now we see the following test set: 0 0 0 0 0 3 0 0 0 0. We should expect the perplexity of this test set to be lower since most of the time the next number will be zero, which is very predictable, i.e. has a high probability. Thus, although the branching factor is still 10, the perplexity or *weighted* branching factor is smaller. We leave this exact calculation as exercise 12.

We see in Section 3.7 that perplexity is also closely related to the information-theoretic notion of entropy.

Finally, let's look at an example of how perplexity can be used to compare different n-gram models. We trained unigram, bigram, and trigram grammars on 38 million words (including start-of-sentence tokens) from the *Wall Street Journal*, using a 19,979 word vocabulary. We then computed the perplexity of each of these models on a test set of 1.5 million words with Eq. 3.16. The table below shows the perplexity of a 1.5 million word WSJ test set according to each of these grammars.

	Unigram	**Bigram**	**Trigram**
Perplexity	962	170	109

As we see above, the more information the n-gram gives us about the word sequence, the lower the perplexity (since as Eq. 3.15 showed, perplexity is related inversely to the likelihood of the test sequence according to the model).

Note that in computing perplexities, the n-gram model P must be constructed without any knowledge of the test set or any prior knowledge of the vocabulary of the test set. Any kind of knowledge of the test set can cause the perplexity to be artificially low. The perplexity of two language models is only comparable if they use identical vocabularies.

An (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in the performance of a language processing task like speech recognition or machine translation. Nonetheless, because perplexity often correlates with such improvements, it is commonly used as a quick check on an algorithm. But a model's improvement in perplexity should always be confirmed by an end-to-end evaluation of a real task before concluding the evaluation of the model.

3.3 Generalization and Zeros

The n-gram model, like many statistical models, is dependent on the training corpus. One implication of this is that the probabilities often encode specific facts about a given training corpus. Another implication is that n-grams do a better and better job of modeling the training corpus as we increase the value of N.