

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 word someone is going to say? What word, for example, is likely to follow

The water of Walden Pond is so beautifully ...

language model

LM

You might conclude that a likely word is blue, or green, or clear, but probably not refrigerator nor this. In this chapter we formalize this intuition by introducing **language models** or **LMs**, models that assign a **probability** to each possible next word. Language models can also assign a probability to an entire sentence, telling us 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

AAC

Why would we want to predict upcoming words, or know the probability of a sentence? One reason is for generation: choosing contextually better words. For example we can correct grammar or spelling errors 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 is more probable than Their are, and has improved than has improve, so a language model can help users select the more grammatical variant. Or for a speech system to recognize that you said I will be back soonish and not I will be bassoon dish, it helps to know that back soonish is a more probable sequence. Language models can also help in **augmentative and alternative communication** (Trnka et al. 2007, Kane et al. 2017). People can use AAC systems if they are physically unable to speak or sign but can instead use eye gaze or other movements to select words from a menu. Word prediction can be used to suggest likely words for the menu.

n-gram

Word prediction is also central to NLP for another reason: **large language models** are built just by training them to predict words!! As we’ll see in chapters 7-9, large language models learn an enormous amount about language solely from being trained to predict upcoming words from neighboring words.

In this chapter we introduce the simplest kind of language model: the **n-gram** language model. An n-gram is a sequence of n words: a 2-gram (which we’ll call **bigram**) is a two-word sequence of words like The water, or water of, and a 3-gram (a **trigram**) is a three-word sequence of words like The water of, or water

of Walden. But we also (in a bit of terminological ambiguity) use the word ‘n-gram’ to mean a probabilistic model that can estimate the probability of a word given the n-1 previous words, and thereby also to assign probabilities to entire sequences.

In later chapters we will introduce the much more powerful neural **large language models**, based on the **transformer** architecture of Chapter 9. But because n-grams have a remarkably simple and clear formalization, we use them to introduce some major concepts of large language modeling, including **training and test sets**, **perplexity**, **sampling**, and **interpolation**.

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 “The water of Walden Pond is so beautifully” and we want to know the probability that the next word is blue:

$$P(\text{blue}|\text{The water of Walden Pond is so beautifully}) \quad (3.1)$$

One way to estimate this probability is directly from relative frequency counts: take a very large corpus, count the number of times we see The water of Walden Pond is so beautifully, and count the number of times this is followed by blue. 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:

$$\begin{aligned} P(\text{blue}|\text{The water of Walden Pond is so beautifully}) &= \\ \frac{C(\text{The water of Walden Pond is so beautifully blue})}{C(\text{The water of Walden Pond is so beautifully})} \end{aligned} \quad (3.2)$$

If we had a large enough corpus, we could compute these two counts and estimate the probability from Eq. 3.2. But even the entire web isn’t big enough to give us good estimates for counts of entire sentences. This is because language is **creative**; new sentences are invented all the time, and we can’t expect to get accurate counts for such large objects as entire sentences. For this reason, we’ll need more clever ways to estimate the probability of a word w given a history h , or the probability of an entire word sequence W .

Let’s start with some notation. First, throughout this chapter we’ll continue to refer to **words**, although in practice we usually compute language models over **tokens** like the BPE tokens of page 21. 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 $P(\text{the})$. We’ll represent a sequence of n words either as $w_1 \dots w_n$ or $w_{1:n}$. Thus the expression $w_{1:n-1}$ means the string w_1, w_2, \dots, w_{n-1} , but we’ll also be using the equivalent notation $w_{<n}$, which can be read as “all the elements of w from w_1 up to and including w_{n-1} ”. For the joint probability of each word in a sequence having a particular value $P(X_1 = w_1, X_2 = w_2, X_3 = w_3, \dots, X_n = w_n)$ we’ll use $P(w_1, w_2, \dots, w_n)$.

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

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

Applying the chain rule to words, we get

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

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 in some corpus, because language is creative and any particular context might have never occurred before!

3.1.1 The Markov assumption

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

$$P(\text{blue}|\text{The water of Walden Pond is so beautifully}) \quad (3.5)$$

we approximate it with the probability

$$P(\text{blue}|\text{beautifully}) \quad (3.6)$$

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

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1}) \quad (3.7)$$

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

Let's see a general equation for this n-gram approximation to the conditional probability of the next word in a sequence. We'll use N here to mean the n-gram size, so $N = 2$ means bigrams and $N = 3$ means trigrams. Then we approximate the probability of a word given its entire context as follows:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1}) \quad (3.8)$$

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:

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

3.1.2 How to estimate probabilities

maximum likelihood estimation

normalize

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

For example, to compute a particular bigram probability of a word w_n given a previous word w_{n-1} , we'll compute the count of the bigram $C(w_{n-1}w_n)$ and normalize by the sum of all the bigrams that share the same first word w_{n-1} :

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

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):

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

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>`.¹

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

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

$$\begin{aligned} P(I|<s>) &= \frac{2}{3} = 0.67 & P(Sam|<s>) &= \frac{1}{3} = 0.33 & P(am|I) &= \frac{2}{3} = 0.67 \\ P(</s>|Sam) &= \frac{1}{2} = 0.5 & P(Sam|am) &= \frac{1}{2} = 0.5 & P(do|I) &= \frac{1}{3} = 0.33 \end{aligned}$$

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

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

relative frequency

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. 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 0.0004. Now 0.0004 is not the best possible estimate of the probability of *Chinese* occurring in all situations; it

¹ We need the end-symbol to make the bigram grammar a true probability distribution. Without an end-symbol, instead of the sentence probabilities of all sentences summing to one, 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.

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 3.6.

Let's move on to some examples from a real but tiny corpus, drawn 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 (Jürafksy et al., 1994). Here are some sample user queries (text-normalized, by lower casing and with punctuation striped) (a sample of 9332 sentences is on the website):

can you tell me about any good cantonese restaurants close by
 tell me about chez panisse
 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 part of a bigram grammar from text-normalized Berkeley Restaurant Project sentences. 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 eight words would be even more sparse.

	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. Each cell shows the count of the column label word following the row label word. Thus the cell in row **i** and column **want** means that **want** followed **i** 827 times in the corpus.

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 counts):

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

Here are a few other useful probabilities:

$$\begin{aligned} P(i|<\text{s}>) &= 0.25 & P(\text{english}|want) &= 0.0011 \\ P(\text{food}|\text{english}) &= 0.5 & P(</\text{s}>|\text{food}) &= 0.68 \end{aligned}$$

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:

$$\begin{aligned} P(<\text{s}> \text{ i want english food } </\text{s}>) \\ &= P(i|<\text{s}>)P(want|i)P(\text{english}|want) \\ &\quad P(\text{food}|\text{english})P(</\text{s}>|\text{food}) \\ &= 0.25 \times 0.33 \times 0.0011 \times 0.5 \times 0.68 \\ &= 0.000031 \end{aligned}$$

	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.

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.

3.1.3 Dealing with scale in large n-gram models

In practice, language models can be very large, leading to practical issues.

log probabilities

Log probabilities Language model probabilities are always stored and computed in log space as **log probabilities**. This is because probabilities are (by definition) less than or equal to 1, and so the more probabilities we multiply together, the smaller the product becomes. Multiplying enough n-grams together would result in numerical underflow. Adding in log space is equivalent to multiplying in linear space, so we combine log probabilities by adding them. By adding log probabilities instead of multiplying probabilities, we get results that are not as small. We do all computation and storage in log space, and just convert back into probabilities if we need to report probabilities at the end by taking the exp of the logprob:

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4) \quad (3.13)$$

In practice throughout this book, we'll use log to mean natural log (\ln) when the base is not specified.

trigram
4-gram
5-gram

Longer context Although for pedagogical purposes we have only described bigram models, when there is sufficient training data we use **trigram** models, which condition on the previous two words, or **4-gram** or **5-gram** models. For these larger n-grams, we'll need to assume extra contexts to the left and right of the sentence end. For example, to compute trigram probabilities at the very beginning of the sentence, we use two pseudo-words for the first trigram (i.e., $P(I|<s>|<s>)$).

Some large n-gram datasets have been created, like the million most frequent n-grams drawn from the Corpus of Contemporary American English (COCA), a curated 1 billion word corpus of American English (Davies, 2020), Google's Web 5-gram corpus from 1 trillion words of English web text (Franz and Brants, 2006), or the Google Books Ngrams corpora (800 billion tokens from Chinese, English, French, German, Hebrew, Italian, Russian, and Spanish) (Lin et al., 2012a)).