8
n-Gram Language Models

In this chapter, we continue our investigation into unsupervised learning techniques and now turn our attention to language models. You may have heard of natural language processing (NLP) and models such as GPT-3 in the news lately. The latter is quite impressive, being able to write and publish its own opinion article on a reputable news website! [1] While most of these models are trained using state-of-the-art deep learning techniques which we will discuss later on in this text, this chapter explores a key idea, which is to view language as the output of a probabilistic process, which leads to an interesting measure of the "goodness" of the model. Specifically, we will investigate the so-called *n-gram language model*.

[1] The full piece can be found at https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3

8.1 Probabilistic Model of Language

Classical linguistics focused on the syntax or the formal grammar of languages. The linguists believed that a language can be modeled by a set of sentences, constructed from a finite set of vocabularies and a finite set of grammatical rules. But this approach in language modeling had limited success in machine learning.

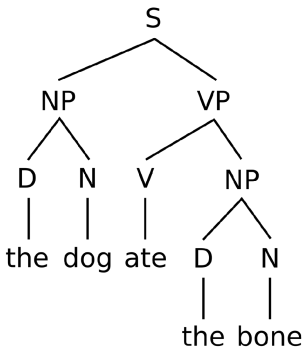

Figure 8.1: An example of a syntax tree of an English sentence.

Instead, the approach of machine learning in language model,

pioneered by Claude Shannon, has been to learn the *distribution* of pieces of text.

In other words, the model assigns a probability to all conceivable finite pieces of English text (even those that have not yet been spoken or written). For example, the sentence "how can I help you" will be assigned some probability, most likely larger than the probability assigned to the sentence "can I how you help." Note that we don't expect to find a "correct" model; all models found to date are approximations. But even an approximate probabilistic model can have interesting uses, such as the following:

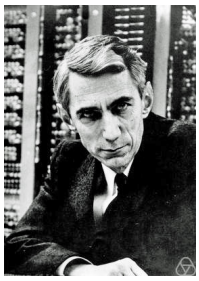

Figure 8.2: Claude Shannon, inventor of the n-gram language model in https://languagelog.ldc.upenn.edu/myl/Shannon1950.pdf. Picture from https://en.wikipedia.org/wiki/Claude_Shannon.

1. *Speech recognition:* A machine processes a recording of a human speech that sounds somewhere between "I ate a cherry" and "eye eight a Jerry." If the model assigns a higher probability score to the former, speech recognition can still work in this instance.

2. *Machine translation:* "High winds tonight" should be considered a better translation than "large winds tonight."

3. *Context sensitive spelling correction:* We can compare the probabilities of sentences that are similar to the following sentence — "Their are problems wit this sentence." — and output the corrected version of the sentence.

4. *Sentence completion:* We can compare the probabilities of sentences that will complete the following phrase — "Please turn off your ..." — and output the one with the highest probability.

8.2 *n-Gram Models*

Say we are in the middle of the process of assigning a probability distribution over all English sentences of length 5. We want to find the probability of the sentence "I love you so much." If we let X_i be the random variable that takes the value of the i-th word, the probability we are looking for is the joint probability

$$\Pr[X_1 = "I", X_2 = "love", X_3 = "you", X_4 = "so", X_5 = "much"] \quad (8.1)$$

By Chain Rule, we can split this joint probability into the product of a marginal probability and four conditional probabilities:

$$\begin{aligned}(8.1) = &\Pr[X_1 = "I"] \\ &\times \Pr[X_2 = "love" \mid X_1 = "I"] \\ &\times \Pr[X_3 = "you" \mid X_1 = "I", X_2 = "love"] \\ &\times \Pr[X_4 = "so" \mid X_1 = "I", X_2 = "love", X_3 = "you"] \\ &\times \Pr[X_5 = "much" \mid X_1 = "I", X_2 = "love", X_3 = "you", X_4 = "so"]\end{aligned} \quad (8.2)$$

If we estimate all components of the product in (8.2), we will be able to estimate the joint probability (8.1).

Now consider the *bigram model*, which has the following two assumptions:

1. The probability of a word is only dependent on the immediately previous word.
2. That probability does not depend on the position of the word in the sentence.

The first assumption says that, for example, the conditional probability

$$\Pr[X_3 = \text{"}you\text{"} \mid X_1 = \text{"}I\text{"}, X_2 = \text{"}love\text{"}]$$

can be simplified as

$$\Pr[X_3 = \text{"}you\text{"} \mid X_2 = \text{"}love\text{"}]$$

The second assumption says that

$$\Pr[X_3 = \text{"}you\text{"} \mid X_2 = \text{"}love\text{"}] = \Pr[X_{i+1} = \text{"}you\text{"} \mid X_i = \text{"}love\text{"}]$$

for any $1 \leq i \leq 5$. We abuse notation and denote any of these probabilities as $\Pr[\text{"}you\text{"} \mid \text{"}love\text{"}]$.

Applying these assumptions to (8.2), we can simplify it as

$$\begin{aligned}(8.1) = \Pr[\text{"}I\text{"}] &\times \Pr[\text{"}love\text{"} \mid \text{"}I\text{"}] \times \Pr[\text{"}you\text{"} \mid \text{"}love\text{"}] \\ &\times \Pr[\text{"}so\text{"} \mid \text{"}you\text{"}] \times \Pr[\text{"}much\text{"} \mid \text{"}so\text{"}] \end{aligned} \tag{8.3}$$

Now we are going to estimate each component of (8.3) from a large corpus of text. The estimation for the marginal probability of the word "I" is given as

$$\Pr[\text{"}I\text{"}] \approx \frac{\text{Count("I")}}{\text{total number of words}} \tag{8.4}$$

where Count refers to the number of occurrences of the substring in the text. In other words, this is the proportion of the occurrence of the word "I" in the entire corpus. Similarly, we can estimate the conditional probability of the word "love" given its previous word is "I" as

$$\Pr[\text{"}love\text{"} \mid \text{"}I\text{"}] \approx \frac{\text{Count("I love")}}{\sum_w \text{Count("I" + } w\text{)}} \tag{8.5}$$

where in the denominator, we sum over all possible vocabularies in the dictionary. This is the proportion of the word "love" occurring immediately after the word "I" out of every time some word w in the dictionary occurring immediately after the word "I." [2] In general, we

[2] Notice that there is no word occurring immediately after the word "I" when "I" is at the end of the sentence in the training corpus. Therefore, the denominator in (8.5) is equal to the Count of "I" minus the Count of "I" at the end of a sentence. This is not necessarily the case when we introduce the sentence stop tokens in Section 8.3.

can estimate the following conditional probability as

$$\Pr[w_{i+1} \mid w_i] \approx \frac{\text{Count}(w_i w_{i+1})}{\sum_w \text{Count}(w_i w)} \tag{8.6}$$

where w_i is the i-th word of the sentence. Once we calculate these estimates from the corpus, we are able to define the probability of the sentence "I love you so much."

8.2.1 *Defining n-Gram Probabilities*

We can extend the example above to a more general setting. Now we want to define the probability distribution over all sentences of length k (grammatical or not). Say we want to find the joint probability of the sentence $w_1 w_2 \dots w_k$ where w_i is the i-th word of the sentence. We will employ a n-gram model which has two assumptions:

1. The probability of a word is only dependent on the immediately previous $n-1$ words. [3]
2. That probability does not depend on the position of the word in the sentence.

[3] If $n = 1$, the model is called a *unigram model*, and the probability is not dependent on any previous word. When $n = 2$ and $n = 3$, the model is respectively called a *bigram* and a *trigram model*.

By a similar logic from the earlier example, we abuse notation and denote the joint probability of the sentence $w_1 w_2 \cdots w_k$ as $\Pr[w_1 w_2 \dots w_k]$; the marginal probability of the first word being w_1 as $\Pr[w_1]$; and so on. We can apply the Chain Rule again to define the n-gram model.

Definition 8.2.1 (n-Gram Model)**.** *An n-gram model assigns the following probability to the sentence $w_1 w_2 \dots w_k$ if $n > 1$:* [4]

$$\begin{aligned}\Pr[w_1 w_2 \dots w_k] &= \Pr[w_1] \Pr[w_2 \mid w_1] \cdots \Pr[w_k \mid w_1 w_2 \dots w_{k-1}] \\ &= \Pr[w_1] \times \prod_{i=2}^{k} \Pr[w_i \mid w_1 \dots w_{i-1}] \\ &= \Pr[w_1] \times \prod_{i=2}^{k} \Pr[w_i \mid w_{\max(1,i-n+1)} \dots w_{i-1}] \end{aligned} \tag{8.7}$$

[4] $\max(1, i-n+1)$ in the third line is to ensure that we access the correct indices for the first $n-1$ words, where there are less than $n-1$ previous words to look at.

and the following probability if $n = 1$:

$$\Pr[w_1 w_2 \dots w_k] = \prod_{i=1}^{k} \Pr[w_i] \tag{8.8}$$

where the n-gram probabilities are estimated from a training corpus as the following

$$\Pr[w_i] \approx \frac{\textit{Count}(w_i)}{\textit{total number of words}}$$

$$\Pr[w_j \mid w_i \dots w_{j-1}] \approx \frac{\textit{Count}(w_i \dots w_{j-1} w_j)}{\sum_w \textit{Count}(w_i \dots w_{j-1} w)}$$

This defines the "best" possible probabilistic model in terms of the Maximum Likelihood Principle from Subsection 4.2.1. [5] We now turn to the following example.

[5] We will prove this for $n = 1$ later.

Example 8.2.2. *We investigate a cowperson language which has two words in the dictionary: $\{Yee, Haw\}$. Suppose the training corpus is given as "Yee Haw Haw Yee Yee Yee Haw Yee." Then the unigram probabilities can be estimated as*

$$\Pr[\text{"}Yee\text{"}] = \frac{5}{8} \qquad \Pr[\text{"}Haw\text{"}] = \frac{3}{8}$$

We can also create the bigram frequency table as in Table 8.1 and we normalize the rows of the bigram frequency table to get the bigram probability table in Table 8.2.

previous \ next	"Yee"	"Haw"	Total
"Yee"	2	2	4
"Haw"	2	1	3

Table 8.1: Bigram frequency table of the cowperson language.

previous \ next	"Yee"	"Haw"	Total
"Yee"	2/4	2/4	1
"Haw"	2/3	1/3	1

Table 8.2: Bigram probabilty table of the cowperson language.

From Table 8.2, we get the following bigram probabilities:

$$\Pr[\text{"}Yee\text{"} \mid \text{"}Yee\text{"}] = \frac{2}{4} \qquad \Pr[\text{"}Haw\text{"} \mid \text{"}Yee\text{"}] = \frac{2}{4}$$
$$\Pr[\text{"}Yee\text{"} \mid \text{"}Haw\text{"}] = \frac{2}{3} \qquad \Pr[\text{"}Haw\text{"} \mid \text{"}Haw\text{"}] = \frac{1}{3}$$

Then by the bigram model, the probability that we see the sentence "Yee Haw Yee" out of all sentences of length 3 can be calculated as

$$\Pr[\text{"}Yee\text{"}] \times \Pr[\text{"}Haw\text{"} \mid \text{"}Yee\text{"}] \times \Pr[\text{"}Yee\text{"} \mid \text{"}Haw\text{"}] = \frac{5}{8} \times \frac{2}{4} \times \frac{2}{3} \simeq 0.21$$

8.2.2 *Maximum Likelihood Principle*

Recall the Maximum Likelihood Principle introduced in Subsection 4.2.1. It gave a way to measure the "goodness" of a model with probabilistic outputs.

Now we formally prove that the estimation methods given in Definition 8.2.1 satisfy the Maximum Likelihood Principle for the $n = 1$ case. A probabilistic model is "better" than another if it assigns more probability to the actual outcome. Here, the actual outcome is the training corpus, which also consists of words. So let us denote

the training corpus as a string of words $w_1 w_2 \dots w_T$. By definition, a unigram model will assign the probability

$$\Pr[w_1 w_2 \dots w_T] = \prod_{i=1}^{T} \Pr[w_i] \tag{8.9}$$

to this string. Remember that each of the w_i's are a member of a finite set of dictionary words. If we let V be the size of the dictionary, then the model is defined by the choice of V values, the probabilities we assign to each of the dictionary words. Let p_i be the probability that we assign to the i-th dictionary word, and let n_i be the number of times that the i-th dictionary word appears in the training corpus. Then (8.9) can be rewritten as

$$\Pr[w_1 w_2 \dots w_T] = \prod_{i=1}^{V} p_i^{n_i} \tag{8.10}$$

We want to maximize this value under the constraint $\sum_{i=1}^{V} p_i = 1$. A solution to this type of a problem can be found via the Lagrange multiplier method. We will illustrate with an example.

Example 8.2.3. *We revisit the cowperson language from Example 8.2.2. Here $V = 2$ and $T = 8$. Let $p_1 = \Pr[\text{"Yee"}]$ and $p_2 = \Pr[\text{"Haw"}]$. Then the probability assigned to the training corpus by the unigram model is*

$$\Pr[\text{"Yee Haw Haw Yee Yee Yee Haw Yee"}] = p_1^5 p_2^3$$

We want to maximize this value under the constaint $p_1 + p_2 = 1$. Then we want to find the point where the gradient of the following is zero.

$$f(p_1, p_2) = p_1^5 p_2^3 + \lambda(p_1 + p_2 - 1)$$

for some λ. The gradients are given as

$$\frac{\partial f}{\partial p_1} = 5p_1^4 p_2^3 + \lambda \qquad \frac{\partial f}{\partial p_2} = 3p_1^5 p_2^2 + \lambda$$

From $5p_1^4 p_2^3 + \lambda = 3p_1^5 p_2^2 + \lambda = 0$, we get $\frac{p_1}{p_2} = \frac{5}{3}$. Combined with the fact that $p_1 + p_2 = 1$, we get the optimal solution $p_1 = \frac{5}{8}$ and $p_2 = \frac{3}{8}$.

Problem 8.2.4. *Following the same Lagrange multiplier method as in Example 8.2.3, verify that the heuristic solution $p_i = \frac{n_i}{T}$ (the empirical frequency) is the optimal solution that maximizes (8.10) under the constraint $\sum_{i=1}^{V} p_i = 1$.*

8.3 Start and Stop Tokens

In this section, we present a convention that is often useful: start token $\langle s \rangle$ and stop token $\langle /s \rangle$. They signify the start and the end of

each sentence in the training corpus. They are a special type of vocabulary that will be augmented to the dictionary, so you will want to pay close attention to the way they contribute to the vocabulary size, number of words, and the n-gram probabilities. Also, by introducing these tokens, we are able to define a probability distribution over all sentences of finite length, not just a given length of k. For the sake of exposition, we will only consider the bigram model for the most parts of this section.

8.3.1 Re-estimating Bigram Probabilities

Consider the cowperson language again.

Example 8.3.1. *The training corpus "Yee Haw Haw Yee Yee Yee Haw Yee" actually consists of three different sentences: (1) "Yee Haw," (2) "Haw Yee Yee," and (3) "Yee Haw Yee." We can append the start and stop tokens to the corpus and transform it into*

$$\begin{aligned} &\langle s \rangle \ \textit{Yee Haw} \ \langle /s \rangle \\ &\langle s \rangle \ \textit{Haw Yee Yee} \ \langle /s \rangle \\ &\langle s \rangle \ \textit{Yee Haw Yee} \ \langle /s \rangle \end{aligned}$$

With these start and stop tokens in mind, we slightly relax the Assumption 2 of the n-gram model and investigate the probability of a word w being the first or the last word of a sentence, separately from other probabilities. We will denote these probabilities respectively as $\Pr[w \mid \langle s \rangle]$ and $\Pr[\langle /s \rangle \mid w]$. The former probability will be estimated as

$$\Pr[w \mid \langle s \rangle] \approx \frac{\text{Count}(\langle s \rangle \ w)}{\text{total number of sentences}} \tag{8.11}$$

which is the proportion of sentences that start with the word w in the corpus. The latter probability is estimated as

$$\Pr[\langle /s \rangle \mid w] \approx \frac{\text{Count}(w \ \langle /s \rangle)}{\text{Count}(w)} \tag{8.12}$$

which is the proportion of the occurrence of w that is at the end of a sentence in the corpus.

Also, notice that other bigram probabilities are also affected when introducing the stop tokens. In (8.6), the denominator originally did not include the occurrence of the substring at the end of the sentence because there was no word to follow that substring. However, if we consider $\langle /s \rangle$ as a vocabulary in the dictionary, the denominator can now include the case where the substring is at the end of the sentence. Therefore, the denominator is just equivalent to the Count of the substring in the corpus. Therefore, the bigram probabilities after introducing start, stop tokens can be estimated instead as [6]

[6] If we consider $\langle s \rangle$, $\langle /s \rangle$ as vocabularies of the dictionary, (8.13) can also include (8.11), (8.12).

$$\Pr[w_j \mid w_{j-1}] \approx \frac{\text{Count}(w_{j-1}w_j)}{\text{Count}(w_{j-1})} \tag{8.13}$$

Example 8.3.2. *We revisit Example 8.2.2. The bigram frequency table and the bigram probability table can be recalculated as in Table 8.3 and Table 8.4.* [7]

[7] Note that the values in the *Total* column now correspond to the unigram count of that word.

previous \ next	"Yee"	"Haw"	$\langle /s \rangle$	Total
$\langle s \rangle$	2	1	0	3
"Yee"	1	2	2	5
"Haw"	2	0	1	3

Table 8.3: Bigram frequency table of the cowperson language with start and stop tokens.

previous \ next	"Yee"	"Haw"	$\langle /s \rangle$	Total
$\langle s \rangle$	2/3	1/3	0/3	1
"Yee"	1/5	2/5	2/5	1
"Haw"	2/3	0/3	1/3	1

Table 8.4: Bigram probabilty table of the cowperson language with start and stop tokens.

Therefore, the bigram probabilities of the cowperson language, once we introduce the start and stop tokens, are given as

$$\Pr[\text{"Yee"} \mid \langle s \rangle] = \frac{2}{3} \qquad \Pr[\text{"Haw"} \mid \langle s \rangle] = \frac{1}{3}$$
$$\Pr[\text{"Yee"} \mid \text{"Yee"}] = \frac{1}{5} \quad \Pr[\text{"Haw"} \mid \text{"Yee"}] = \frac{2}{5} \quad \Pr[\langle /s \rangle \mid \text{"Yee"}] = \frac{2}{5}$$
$$\Pr[\text{"Yee"} \mid \text{"Haw"}] = \frac{2}{3} \quad \Pr[\text{"Haw"} \mid \text{"Haw"}] = \frac{0}{3} \quad \Pr[\langle /s \rangle \mid \text{"Haw"}] = \frac{1}{3}$$

8.3.2 *Redefining the Probability of a Sentence*

The biggest advantage of introducing stop tokens is that now we can assign a probability distribution over all sentences of finite length, not just a given length k. Say we want to assign a probability to the sentence $w_1 w_2 \dots w_k$ (without the start and stop tokens). By introducing start and stop tokens, we can interpret this as the probability of $w_0 w_1 \dots w_{k+1}$ where $w_0 = \langle s \rangle$ and $w_{k+1} = \langle /s \rangle$. Following the similar logic from (8.2), we can define this probability by the Chain Rule.

Definition 8.3.3 (Bigram Model with Start, Stop Tokens)**.** *A bigram model, once augmented with start, stop tokens, assigns the following probability to a sentence $w_1 w_2 \dots w_k$* [8]

[8] Notice that we do not have the term $\Pr[w_0]$ in the expansion. A sentence always starts with a start token, so the marginal probability that the first word is $\langle s \rangle$ can be understood to be 1.

$$\Pr[w_1 w_2 \dots w_k] = \prod_{i=1}^{k+1} \Pr[w_i \mid w_{i-1}] \tag{8.14}$$

where the bigram probabilities are estimated as in (8.13).

Example 8.3.4. *The probability that we see the sentence "Yee Haw Yee" in the cowperson language can be calculated as*

$$\begin{aligned}&\Pr[\text{"}Yee\text{"} \mid \langle s\rangle] \times \Pr[\text{"}Haw\text{"} \mid \text{"}Yee\text{"}] \times \Pr[\text{"}Yee\text{"} \mid \text{"}Haw\text{"}] \times \Pr[\langle /s\rangle \mid \text{"}Yee\text{"}] \\ &= \frac{2}{3} \times \frac{2}{5} \times \frac{2}{3} \times \frac{2}{5} \simeq 0.07\end{aligned}$$

Note that this probability is taken over all sentences of finite length.

Problem 8.3.5. *Verify that (8.14) defines a probability distribution over all sentences of finite length.*

8.3.3 Beyond Bigram Models

In general, if we have a n-gram model, then we may need to introduce more than 1 start or stop tokens. For example, in a trigram model, we will need to define the probability that the word is the first word of the sentence as $\Pr[w \mid \langle s\rangle \langle s\rangle]$. Based on the number of start and stop tokens introduced, the n-gram probabilities will need to be adjusted accordingly.

8.4 Testing a Language Model

So far, we discussed how to define a n-gram language model given a corpus. This is analogous to training a model given a training dataset. Naturally, the next step is to test the model on a newly seen held-out data to ensure that the model generalizes well. In this section, we discuss how to test a language model.

8.4.1 Shakespeare Text Production

First consider a bigram text generator — an application of the bigram model. The algorithm initiates with the start token $\langle s\rangle$. It then outputs a random word w_1 from the dictionary, according to the probability $\Pr[w_1 \mid \langle s\rangle]$. It then outputs the second random word w_2 from the dictionary, according to the probability $\Pr[w_2 \mid w_1]$. It repeats this process until the newly generated word is the stop token $\langle /s\rangle$. The final output of the algorithm will be the concatenated string of all outputted words.

It is possible to define a text generator for any n-gram model in general. Figure 8.4 shows the output of the unigram, bigram, trigram, quadrigram text generators when the models were trained on all Shakespeare texts.

Notice the sentence "I will go seek the traitor Gloucester." in the output of the quadrigram text generator. This exact line appears in *King Lear*, Act 3 Scene 7. This is not a coincidence. Figure 8.5 presents

```
<s> I
    I want
      want to
           to eat
              eat tasty
                  tasty food
                       food  </s>
I want to eat tasty food
```

Figure 8.3: An example run of the bigram text generator.

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

What means, sir. I confess she? then all sorts, he is trim, captain.
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram

King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

Figure 8.4: The outputs of unigram, bigram, trigram, quadrigram text generators trained on Shakespeare texts.

the snapshot of the bigram, trigram, and quadrigram text generators once they have outputted the phrase "go seek the." You can see that bigram models and trigram models assign very small probability to the word "traitor" because there are much more instances of phrases "the" or "seek the" in the corpus than "go seek the." On the other hand, the quadrigram model assigns a very large probability to the word "traitor" because there is only a limited number of times that the phrase "go seek the" appears in the corpus.

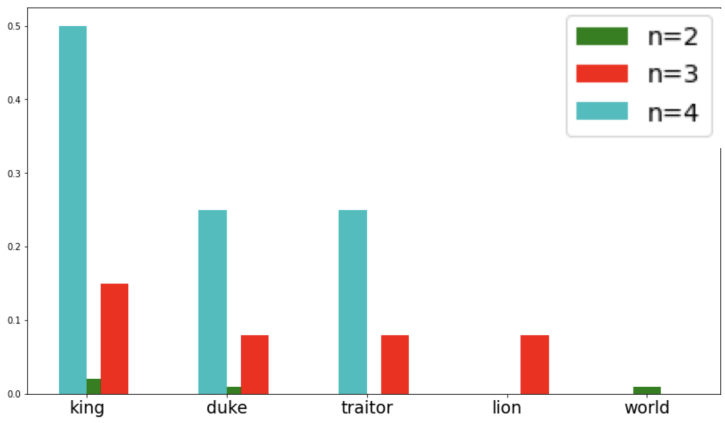

Figure 8.5: The probability of the next word given the previous three words are "go seek the."

Once the quadrigram model outputs the word "traitor" after the

phrase "go seek the," the problem is even worse. As can be see in Figure 8.6, the quadrigram model assigns probability of 1 to the word "Gloucester" meaning that the phrase "seek the traitor" only appears before the word "Gloucester." So the model has *memorized* one completion of the phrase from the training text. From this example, we can see that text production based on n-grams is sampling and remixing text fragments seen in the training corpus.

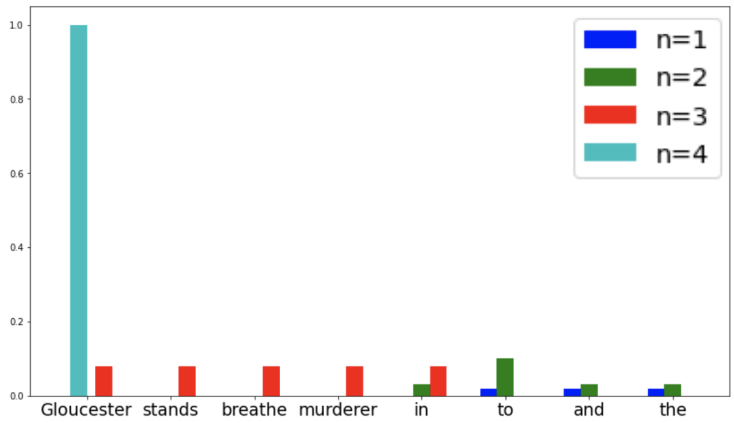

Figure 8.6: The probability of the next word given the previous three words are "seek the traitor."

The Shakespeare corpus consists of $N = 884,647$ words and $V = 29,066$ distinct vocabulary from the dictionary. There are about $V^2 \approx 844$ million possible combinations of bigrams, but Shakespeare only used around $300,000$ of them in his text. So 99.96% of the possible bigrams were never seen. The percentage is much higher for quadrigrams! Furthermore, for the quadrigrams that do appear in the corpus, most of do not even repeat. Thus what comes out of the quadrigram model looks like Shakespeare because it is a *memorized* fragment of Shakespeare.[9]

[9] Do this remind you of overfitting?

8.4.2 *Perplexity*

Having described a way to train a simple language model, we now turn our attention to a formal way of testing [10] a language model.

[10] This method is used even for testing state of the art models.

Just like any other model in ML, a language model will be given a corpus $w_1 w_2 \dots w_T$. Then we can assess the performance of the model by its *perplexity* on the corpus.

Definition 8.4.1 (Perplexity)**.** *The **perplexity** of a language model on the corpus $w_1 w_2 \dots w_T$ is defined as*

$$\Pr[w_1 w_2 \dots w_T]^{-\frac{1}{T}} = \sqrt[T]{\frac{1}{\Pr[w_1 w_2 \dots w_T]}} \tag{8.15}$$

Note that, perplexity is defined for any probabilistic language model: the Chain Rule of joint probability applies to every model,

and does not require the n-gram assumptions. That is, [11]

$$\Pr[w_1 w_2 \dots w_T] = \Pr[w_1] \times \prod_{i=2}^{T} \Pr[w_i \mid w_1 \dots w_{i-1}]$$

Then the perplexity of the model can be rewritten as

$$\sqrt[T]{\frac{1}{\Pr[w_1]} \times \prod_{i=2}^{T} \frac{1}{\Pr[w_i \mid w_1 \dots w_{i-1}]}} \tag{8.16}$$

Example 8.4.2. *Consider the uniform ("clueless") model which assumes that the probability of all words are equal in any given situation. That is, if V is the vocabulary size (i.e., size of the dictionary),*

$$\Pr[w_i] = \Pr[w_i \mid w_1 \dots w_{i-1}] = \frac{1}{V}$$

*for any given $w_1, \dots, w_i \in V$. This model assigns $\left(\frac{1}{V}\right)^T$ to **every** sequence of T words, including the corpus. Therefore, the perplexity of the model is*

$$\left(\left(\frac{1}{V}\right)^T\right)^{-\frac{1}{T}} = V$$

Now we try to understand perplexity at an intuitive level. (8.16) is the geometric mean [12] of the following T values:

$$\frac{1}{\Pr[w_1]}, \frac{1}{\Pr[w_2 \mid w_1]}, \dots, \frac{1}{\Pr[w_T \mid w_1 \dots w_{T-1}]}$$

Now note that a probabilistic model splits the total probability of 1 to fractions and distributes them to the potential options for the next word. So the inverse of an assigned probability for a word can be thought roughly as the *number of choices the model considered for the next word*. With this viewpoint, perplexity as written in (8.16) means: *how much has the model narrowed down the number of choices for the next word* **on average?** The clueless model had not narrowed down the possibilities at all and had the worst-possible perplexity equal to the number of vocabulary words.

Example 8.4.3. *Consider a well-trained language model. At any given place of text, it can identify a set of* 20 *words and assigns probability $\frac{1}{20}$ to each of them to be the next word. It happens that the next word is **always** one of the* 20 *words that the model identifies. The perplexity of the model is*

$$\left(\left(\frac{1}{20}\right)^T\right)^{-\frac{1}{T}} = 20$$

Interestingly enough, the true perplexity of English is believed to be between 15 and 20. That is, if at an "average" place in text, you ask humans to predict the next word, then they are able to narrow down the list of potential next words to around 15 to 20 words. [13]

[11] Assume for now that start and stop tokens do not exist in the corpus.

[12] The *geometric mean* of T numbers $a_1, a_2, \dots, a_T$ is defined as $(\prod_i a_i)^{1/T}$

[13] The perplexity of state of the art language models is under 20 as well.

8.4.3 Perplexity on Test Corpus

The *perplexity* of a language model is analogous to a *loss* of a ML model. [14] Similar to ML models we have been studying so far, it is possible to define a *train perplexity* and a *test perplexity*. The "goodness" of the model will be defined by how low the perplexity was on a previously unseen, held-out data.

[14] It is customary to use the logarithm of the perplexity, as we also did for logistic loss in Chapter 4.

For example, when n-gram models are trained on 38 million words and tested on 1.5 million words from Wall Street Journal articles, they show the following test perplexities in Table 8.5. [15] Note that the state-of-the-art deep learning models achieve a test perplexity of around 20 on the same corpus.

[15] To be more exact, the models were augmented with smoothing, which will be introduced shortly.

Unigram	Bigram	Trigram
962	170	109

Table 8.5: Test perplexities of n-gram models on WSJ corpus.

8.4.4 Perplexity With Start and Stop Tokens

When start and stop tokens are introduced to a corpus, we also need to redefine how to calculate the perplexity of the model. Again, we will only focus on a bigram model for the sake of exposition.

Say the corpus consists of t sentences:

$$\begin{aligned}
&\langle s \rangle\, w_{1,1} w_{1,2}, \ldots, w_{1,T_1}\, \langle /s \rangle \\
&\langle s \rangle\, w_{2,1} w_{2,2}, \ldots, w_{2,T_2}\, \langle /s \rangle \\
&\qquad\qquad \vdots \\
&\langle s \rangle\, w_{t,1} w_{t,2}, \ldots, w_{t,T_t}\, \langle /s \rangle
\end{aligned}$$

The probability of the corpus $w_{1,1} w_{1,2} \ldots w_{t,T_t}$ is redefined as the product of the probability of each of the sentences:

$$\begin{aligned}
\Pr[w_{1,1} w_{1,2} \ldots w_{t,T_t}] &= \prod_{i=1}^{t} \Pr[w_{i,1} w_{i,2} \ldots w_{i,T_i}] \\
&= \prod_{i=1}^{t} \prod_{j=1}^{T_i+1} \Pr[w_{i,j} \mid w_{i,j-1}] \qquad (8.17)
\end{aligned}$$

Now we apply the interpretation of the perplexity that it is the geometric mean of probabilities of each word. Notice that we multiplied $\sum_{i=1}^{t} (T_i + 1)$ probabilities to calculate the probability of the corpus. If we let $T = \sum_{i=1}^{t} T_i$ denote the total number of words (excluding start and stop tokens) of the corpus, the number of probabilities we multiplied can be written as $T^* = T + t$. [16]

[16] This can also be thought as adding the number of stop tokens to the number of words in the corpus.

Definition 8.4.4 (Perplexity with Start, Stop Tokens)**.** *The perplexity of a bigram model with start, stop tokens can be redefined as*

$$\sqrt[T^*]{\frac{1}{\prod_{i=1}^{t}\prod_{j=1}^{T_i+1}\Pr[w_{i,j} \mid w_{i,j-1}]}} \tag{8.18}$$

8.4.5 Smoothing

One big problem with our naive definition of the perplexity of a model is that it does not account for a zero denominator. That is, if the model assigns probability exactly 0 to the corpus, then the perplexity of the model will be ∞! [17]

[17] Mathematically, it is undefined, but here assume that the result is a positive infinity that is larger than any real number.

Example 8.4.5. *Suppose the phrase "green cream" never appeared in the training corpus, but the test corpus contains the sentence "You like green cream." Then a bigram model will have a perplexity of ∞ because it assigns probability 0 to the bigram "green cream."*

To address this issue, we generally apply *smoothing* techniques, which never allow the model to output a zero probability. By *reducing* the naive estimate of *seen* events and *increasing* the naive estimate of *unseen* events, we can always assign nonzero probability to previously unseen events.

The most commonly used smoothing technique is the 1-add smoothing (a.k.a, Laplace smoothing). We describe how the smoothing works for a bigram model. The main idea of the 1-add smoothing can be summarized as "add 1 to all bigram counts in the bigram frequency table." Then the bigram probability as defined in Definition 8.2.1 can be redefined as

$$\Pr[w_j \mid w_{j-1}] \approx \frac{\text{Count}(w_{j-1}w_j)+1}{\sum_w (\text{Count}(w_{j-1}w)+1)} = \frac{\text{Count}(w_{j-1}w_j)+1}{\sum_w (\text{Count}(w_{j-1}w)) + V} \tag{8.19}$$

where V is the size of the dictionary. If we had augmented the start and the stop tokens to the corpus, the denominator in (8.19) is just equal to $\text{Count}(w_{j-1}) + V^*$ [18] and so the bigram probability can be written as

$$\Pr[w_j \mid w_{j-1}] \approx \frac{\text{Count}(w_{j-1}w_j)+1}{\text{Count}(w_{j-1}) + V^*} \tag{8.20}$$

[18] $V^* = V + 1$ is the size of the dictionary after adding the start and the stop tokens. It is customary to add only one to the vocabulary. It may help to look at the number of rows and columns in the bigram frequency table 8.3.

Notice that the denominator is just V^*, the new vocabulary size, added to the unigram count of w_{j-1}.

Example 8.4.6. *Recall the cowperson language with the start and stop tokens from Example 8.3.2. Upon further research, it turns out the language actually consists of three words: $\{Yee, Haw, Moo\}$, but the training corpus*

"Yee Haw Haw Yee Yee Yee Haw Yee" left out one of the vocabularies in the dictionary. By applying add-1 smoothing to the bigram model, we can recalculate the bigram frequency and the bigram probability table as in Table 8.6 and Table 8.7

previous \ next	"Yee"	"Haw"	"Moo"	$\langle /s \rangle$	Total
$\langle s \rangle$	3	2	1	1	7
"Yee"	2	3	1	3	9
"Haw"	3	1	1	2	7
"Moo"	1	1	1	1	4

Table 8.6: Bigram frequency table of the cowperson language with start and stop tokens with smoothing.

previous \ next	"Yee"	"Haw"	"Moo"	$\langle /s \rangle$	Total
$\langle s \rangle$	3/7	2/7	1/7	1/7	1
"Yee"	2/9	3/9	1/9	3/9	1
"Haw"	3/7	1/7	1/7	2/7	1
"Moo"	1/4	1/4	1/4	1/4	1

Table 8.7: Bigram probability table of the cowperson language with start and stop tokens with smoothing.

The probability that we see the sentence "Moo Moo" in the cowperson language, which would have been 0 before smoothing, is now assigned a non-zero value:

$$\begin{aligned} &\Pr[\text{"}Moo\text{"} \mid \langle s \rangle] \times \Pr[\text{"}Moo\text{"} \mid \text{"}Moo\text{"}] \times \Pr[\langle /s \rangle \mid \text{"}Moo\text{"}] \\ &= \frac{1}{7} \times \frac{1}{4} \times \frac{1}{4} \simeq 0.01 \end{aligned}$$

Problem 8.4.7. *Verify that (8.19) defines a proper probability distribution over the conditioned event. That is, show that*

$$\sum_{w} \Pr[w \mid w'] = 1$$

for any w in the dictionary.

Another smoothing technique is called *backoff* smoothing. The intuition is that n-gram probabilities are less likely to be zero if n is smaller. So when we run into a n-gram probability that is zero, we replace it with a linear combination of n-gram probabilities of lower values of n.

Example 8.4.8. *Recall Example 8.4.5. The bigram probability of "green cream" can be approximated instead as*

$$\Pr[\text{"}cream\text{"} \mid \text{"}green\text{"}] \approx \Pr[\text{"}cream\text{"}]$$

Also, say we want to calculate the trigram probability of "like green cream," which is also zero in the naive trigram model. We can approximate it instead as

$$\Pr[\text{"}cream\text{"} \mid \text{"}like\ green\text{"}] \approx \alpha \Pr[\text{"}cream\text{"}] + (1-\alpha) \Pr[\text{"}cream\text{"} \mid \text{"}green\text{"}]$$

where α is a hyperparameter for the model.

There are other variants of the backoff smoothing, [19] with some theory for what the "best" choice is, but we will not cover it in these notes.

[19] For instance, Good-Turing and Kneser-Ney smoothing.