3. N 元语法语言模型

“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

预测是很困难的，尤其是预测未来，正如一句古老的俏皮话所说。但如何预测一些看起来更容易的事情，比如某人接下来要说的几句话是什么呢？例如，哪个单词可能会跟在后面：

Please turn your homework ...

希望你们中的大多数得出结论，一个很可能的单词是 in 或 over，但可能不是 refrigerator 或 the。在接下来的几节中，我们将通过引入模型，为每个可能的下一个单词指定一个概率，来形式化这种直觉。相同的模型还将用于为整个句子分配概率。例如，这种模型可以预测以下序列在文本中出现的概率更高：

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

（我突然注意到三个人站在人行道上）

这组相同单词有不同的排序:

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

您为什么要预测即将出现的单词，或为句子分配概率？在任何我们必须从嘈杂的、模糊的输入中识别单词的任务中，比如语音识别，概率都是必不可少的。为了使语音识别器意识到您说的是“I will be back soonish”而不是“I will be bassoon dish”，这有助于了解“back soonish”比“bassoon dish”更有可能出现。对于拼写纠错、语法纠错等写作工具，我们需要发现并纠正写作中的错误，比如“Their are two midterms”，其中“There”被误打为“Their”，或者“Everything has improve”，其中“improve”本应该是“improved”。短语“There are”比“Their are”更有可能出现，“has improved”比“has improve”更有可能出现，这使我们能够通过检测和纠正这些错误来帮助用户。

为单词序列分配概率在机器翻译中也很重要。假设我们正在翻译一个中文源句:

他　向　记者　介绍了　主要　内容

He　to　reporters　introduced　main　content

作为这个过程的一部分，我们可能已经建立了以下一套潜在的粗略英语翻译:

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

单词序列的概率模型可能认为“briefed reporters on”的概率大于“briefed to reporters”(一个尴尬的“to”跟在“briefed”之后)或“introduced reporters to”(它使用的动词在这个语境中不是很流利)，这让我们正确地选择了上面黑体的句子。

概率对于**增强性和替代性交流系统(AAC)**也很重要(Trnka 等人 2007，Kane 等人 2017)。如果人们在肢体上无法讲话或签名，但通常可以使用视线凝视或其他特定动作从系统要说的菜单中选择单词，则人们通常会使用此类 AAC 设备。单词预测可以用来为菜单推荐可能的单词。

将概率分配给单词序列的模型称为**语言模型(language model，LM)**。在本章中，我们介绍最简单的模型，**n 元语法(n-gram)**，该模型将概率分配给句子和单词序列。n-gram 是由 n 个单词组成的序列:2-gram(bigram)是由两个单词组成的序列，如“please turn”，“turn your”，或“your homework”；3-gram(trigram)是由三个单词组成的序列，如“please turn your”，或“turn your homework”。我们将看到如何使用 n-gram 模型，在给定前一个单词的情况下，来估计 n-gram 后一个单词的概率，并为整个序列分配概率。出于术语上的歧义，我们通常会放弃“模型”一词，而使用术语 n-gram(以及 bigram 等)来表示单词序列本身或为其指定概率的预测模型。我们将在后面介绍更复杂的语言模型，如第 9 章中的 RNN LMs。

3.1. N 元语法

让我们从计算 P(w | h)的任务开始，即：在给定历史 h 的条件下，计算单词 w 的概率。假设历史 h 是“its water is so transparent that”，我们想知道下一个单词是“the”的概率：

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

估计这个概率的一种方法是通过相对频率计数:取一个非常大的语料库，计算我们看到“its water is so transparent that”的次数，并计算它后面跟着“the”的次数。这就可以回答“根据我们看到历史 h 的次数，有多少次它后面跟着 w”，如下所示：

$$\begin{aligned} &P(\text{the} \mid \text{its water is so transparent that}) \\ &= \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})} \end{aligned} \tag{3.2}$$

有了足够大的语料库，如 web，我们可以计算这些计数，并根据公式 3.2 估计概率。现在，您应该暂停一下，到 web 上，自己计算一下这个估算值。

虽然这种直接从计数来估计概率的方法在许多情况下都很有效，但事实证明，在大多数情况下，即使是 web 也不足以给我们提供良好的估计。这是因为语言具有创造力； 我们会一直创建新的句子，而我们将永远无法统计全部句子。即使是例句的简单扩展，在 web 上也可能出现 0 的计数(例如 Walden Pond’s water is so transparent that the 在过去的计数为零)。

类似地，如果我们想知道“its water is so transparent”这类单词序列的联合概率，我们可以问“在所有可能的五个单词序列中，有多少个单词是 its water is so transparent?”我们必须得到“its water is so transparent”的计数，然后除以所有可能的五个单词序列的计数总和。这似乎是一个很大的估计!

出于这个原因，我们需要引入更聪明的方法来估计在给定历史 h 条件下的单词 w 的概率，或者整个单词序列 w 的概率。让我们从一些形式化的符号开始。为了表示特定随机变量 X_i 取值“the”或 $P(X_i =\text{“the”})$的概率，我们将使用化简后的 P(the)。我们将 N 个单词的序列表示为 $w_1 \dots w_n$ 或 $w_{1:n}$(因此表达式 $w_{1:n-1}$ 表示字符串 w_1，w_2，…，w_{n-1})。对于具有特定值 P(X = w_1，Y = w_2，Z = w_3，…，W = w_n)的序列中每个单词的联合概率，我们将使用 P(w_1，w_2，…， w_n)。

现在我们如何计算像 P(w_1,w_2，…，w_n)这样的整个序列的概率?我们可以做的一件事是使用概率的**链式规则(chain rule)**来分解这个概率:

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

运用链式规则，我们得到：

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

链式规则展示了计算一个序列的联合概率和计算在给定前一个单词的情况下的条件概率之间的联系。公式 3.4 表明，我们可以通过将多个条件概率相乘来估计整个单词序列的联合概率。但是，使用链式规则似乎并没有真正帮助我们！给定一长串前面的单词 $P(w_n \mid w_{1:n-1})$，我们没有任何方法可以计算出一个单词的确切概率。正如我们上面所说的，我们不能仅仅通过计算每个单词在每个长字符串后面出现的次数来估计，因为语言是有创造性的，任何特定的上下文可能以前从未出现过!

n 元语法模型的直觉是，与其在给定全部单词历史记录的情况下计算单词的概率，不如仅通过最后几个单词来估计历史记录。

例如，二元语法(bigram)模型仅通过使用一个先前单词 $P(w_n | w_{n-1})$的条件概率来近似给定所有先前单词 $P(w_n | w_{1:n-1})$的单词的概率。换句话说，不是计算概率：

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

针对公式 3.5，我们用下面的概率来作近似计算：

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

因此，当我们使用 bigram 模型来预测下一个单词的条件概率时，我们可以做出如下近似:

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

一个词的概率只取决于前一个词的假设被称为马尔科夫(Markov)假设。马尔可夫模型是概率模型的一类，它们假定我们可以预测某些未来单元的概率而不会过多地考虑过去。我们可以将二元语法(向过去看一个单词)推广到三元语法(向过去看两个单词)，然后推广到 n 元语法(n-gram)(向过去看 n-1 个单词)。

因此，这个 n 元近似的序列中下一个单词的条件概率的一般方程是：

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

在给定单个单词概率的二元语法假设的情况下，我们可以通过将公式 3.7 代入公式 3.4 计算出全部单词序列的概率:

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

我们如何估计这些二元或 n 元语法概率？ 估计概率的一种直观方法称为最大似然估计(MLE)。我们通过从语料库中获取计数，并对计数进行归一化(normalize)，使它们位于 0 到 1 之间，从而获得 n-gram 模型参数的 MLE 估计。

例如，在给定前一个单词 x 的条件下，要计算单词 y 的一个特定的二元语法概率，我们将计算二元语法 C(xy)的计数，并通过共享相同的第一个单词 x 的所有二元语法的总和，再进行归一化：

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

我们可以简化此等式，因为以给定单词 w_{n-1} 开头的所有二元语法计数的总和必须等于该单词 w_{n-1} 的一元语法计数(读者应该花点时间对此加以证明)：

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

让我们用一个三句话的小语料库来做一个例子。首先，我们需要在每个句子的开头添加一个特殊符号<s>，以便为我们提供第一个单词的二元语法上下文。我们还需要一个特殊的结束符号</ s >。

<s> I am Sam </s>

<s> Sam I am </s>

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

以下是对这个语料库中一些二元语法概率的计算：

P(I | <s>) = 2/3 =.67　　P(Sam | <s>) = 1/3 =.33　　P(am | I) = 2/3=.67

P(</s> | Sam) = 1/2 =.5　　P(Sam | am) = 1/2=.5　　P(do | I) = 1/3=.33

对于 MLE n-gram 参数估计的一般情况:

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

公式 3.12(类似于公式 3.11)通过将特定序列的观测频率除以前缀的观测频率来估计 n 元语法的概率。这个比率称为**相对频率(relative frequency)**。我们在上面说过，这种使用相对频率来估计概率的方法是最大似然估计(MLE)的一个例子。在 MLE 中，给定模型 M(即 P(T | M))，结果参数集将训练集 T 的可能性最大化。例如，假设“Chinese”这个词在一个有 100 万个单词的语料库(比如布朗语料库)中出现了 400 次。那么，从其他文本中(例如，100 万单词中)随机选出的一个单词，出现 Chinese 的概率是多少？其概率的最大似然估计值为 400/1000000 或 0.0004。现在，0.0004 并不是 Chinese 在所有情况下发生概率的最好估计；在其他语料库或语境中，Chinese 是一个不太可能出现的词。但在一个 100 万单词的语料库中，Chinese 出现 400 次的可能性很大。在第 3.4 节中，我们提出了稍微修改 MLE 估计的一些方法，以获得更好的概率估计。

让我们继续看一些例子，这些例子来自一个比上面 14 个单词的例子略大的语料库。我们将使用来自现在已经不存在的伯克利餐厅项目的数据，这是一个上世纪的对话系统，它回答了关于加州伯克利餐厅数据库的问题(Jurafsky 等人，1994)。这里有一些文本规范化用户查询的样本(9332 句的样本是在网站上):

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

图 3.1 显示了来自伯克利餐厅项目的一个 bigram 语法片段的 bigram 计数。请注意，大多数值都是零。事实上，我们选择的样本词是为了彼此连贯;从 7 个单词的随机集合中选择的矩阵将更加稀疏。

	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

图 3-1：二元语法计数

图注：伯克利餐厅项目语料库 9332 个句子中 8 个单词(V=1446)的 bigram 计数，

图 3.2 显示了归一化后的二元概率(将图 3.1 中的每个单元除以对应行的一元语法概率，其值取自下列集合的一元语法概率)：

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

	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

图 3-2：二元语法概率

图注：伯克利餐厅项目语料库 9332 个句子中 8 个单词的二元语法概率。

以下是其他一些有用的概率:

P(i|<s>) = 0.25 *P*(english|want) = 0.0011

P(food|english) = 0.5 *P*(</s>|food) = 0.68

现在，我们可以通过简单地将相应的二元语法概率相乘在一起，就可计算像“I want English food”或“I want Chinese food”这种句子的概率，如下所示：

$$
\begin{aligned}
&P(\text{<s> i want english food </s>}) \\
&\quad = P(\text{i}\,|\,\text{<s>})\,P(\text{want}\,|\,\text{i})\,P(\text{english}\,|\,\text{want}) \\
&\qquad\qquad P(\text{food}\,|\,\text{english})P(\text{</s>}\,|\,\text{food}) \\
&\quad = .25\times .33\times .0011\times .5\times .68 \\
&\quad = .000031
\end{aligned}
$$

我们把它留作练习 3.2 来计算"i want chinese food"的概率。

这些二元语法数据统计中捕获了哪些语言现象？上面的一些二元语法概率编码了一些我们认为本质上严格属于句法的事实，例如，“eat”之后出现的通常是名词或形容词，或者“to”之后出现的通常是动词。另一些可能是关于个人助理任务的事实，比如句子以单词"I"开头的可能性很高。有些甚至可能是文化因素而非语言因素，比如人们更倾向于寻找中国食物而不是英国食物。

一些实际问题：尽管出于教学目的，我们只描述了二元语法模型，但在实践中，更常见的是使用**三元语法(trigram)**模型，它以前两个单词为条件，而不是前一个单词，或者当有足够的训练数据时，使用 4-gram 甚至 5-gram 模型。请注意，对于这些较大的 n-grams，我们需要假定在句子末尾的左右两侧有额外的上下文。例如，为了计算句子开头的三元语法概率，我们对第一个三元语法使用两个伪词(即 P(I|<s><s>)。

我们通常使用对数概率的形式来表示和计算语言模型的概率。因为概率(根据定义)小于或等于 1，所以我们相乘的概率越多，乘积就越小。将足够多的 n-gram 相乘会导致数字下溢(underflow)。通过使用对数概率而不是原始概率，我们得到的数字并不那么小。

因为在对数空间中相加等同于在线性空间中相乘，因此我们通过将它们相加来组合**对数概率(log probabilies)**。在对数空间中进行所有计算和存储的结果是，我们只需要在最后需要报告它们的时候，才将它们转换回概率;然后我们可以取 logprob 的 exp：

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

3.2. 评估语言模型

评估语言模型性能的最佳方法是将其嵌入到应用程序中，并衡量应用程序的改进程度。这种端到端的评估称为**外部评估(extrinsic evaluation)**。外部评估是了解某个组件的特定改进是否真的会帮助完成手头任务的唯一方法。因此，对于语音识别，我们可以通过运行两次语音识别器来比较两种语言模型的性能，每一种语言模型运行一次，看看哪一种语言模型的转录更准确。

不幸的是，端到端的运行大型 NLP 系统通常非常昂贵。相反，如果有一个度量可以用来快速评估语言模型中潜在的改进，那就太好了。一个**内在评估(intrinsic evaluation)**度量是一个独立于任何应用程序的模型质量的测量。

对于语言模型的内在评估，我们需要一个测试集。与我们领域中的许多统计模型一样，n-gram 模型的概率来自于它所受训练的语料库，即**训练集(training set)**或训练语料库。然后，我们可以通过 n-gram 模型在一些不可见的数据(称为**测试集(test set)**或测试语料库)上的性能来衡量其质量。有时我们还会将不在训练集中的测试集和其他数据集称为**保留(held out)**语料库，因为我们将它们保留在训练数据之外。

因此，如果我们有一个文本语料库，并且想要比较两个不同的 n-gram 模型，我们将数据分成训练集和测试集，在训练集上训练两个模型的参数，然后比较两个训练过的模型对测试集的拟合程度有多好。

但是“适合测试集”是什么意思呢?答案很简单:给测试集赋值较高的概率(意味着它能更准确地预测测试集)的模型是更好的模型。给定两个概率模型，较好的模型与测试数据的拟合更紧密，或者能够更好地预测测试数据的细节，从而将较高的概率分配给测试数据。

因为我们的评估指标是基于测试集概率的，所以不要让测试句子进入训练集是很重要的。假设我们正在尝试计算特定“测试”句子的概率。如果我们的测试句子是训练语料库的一部分，那么，当它出现在测试集中时，我们会错误地将一个人为的高概率分配给它。我们把这种情况称为测试集上的训练。在测试集上进行训练会引入一种偏差，这种偏差会使所有概率看上去都过高，并在**困惑度**上导致巨大的不准确，我们下面介绍这种基于概率的度量方法。

有时，我们过于频繁地使用一个特定的测试集，以致于我们隐式地调整了它的特征。然后，我们需要一个新的测试集，一个真正看不见的测试集。在这种情况下，我们将初始测试集称为**开发测试集(devset)**。我们如何将数据划分为训练、开发和测试集?我们希望我们的测试集尽可能大，因为一个小的测试集可能意外地不具有代表性，但我们也希望尽可能多的训练数据。至少，我们希望选择最小的测试集，使我们有足够的统计能力来测量两个潜在模型之间在统计上的显著差异。在实践中，我们通常只是将我们的数据分为 80%的训练、10%的开发和 10%的测试。给定我们要划分为训练和测试的大型语料库，可以从语料库中的一些连续文本序列中获取测试数据，也可以从语料库的随机选择部分中删除较小的“条带”文本并将其合并进入测试集。

3.2.1. 困惑度

在实践中，我们不使用原始概率作为我们评估语言模型的度量标准，而是使用一种叫做**困惑度(perplexity)**的变量。测试集上的语言模型的困惑度(有时简称 PP)是测试集的逆反概率，由单词的数量归一化。对于一个测试集 $W = w_1w_2...w_N$，有:

$$\begin{aligned} PP(W) &= P(w_1w_2...w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1w_2...w_N)}} \end{aligned} \tag{3.14}$$

我们可以用链式规则来展开 W 的概率:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P(w_i \mid w_1w_2...w_{i-1})}} \tag{3.15}$$

因此，如果我们用一个二元语言模型来计算 W 的困惑度，我们得到:

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

注意，由于公式 3.15 中的倒数，如果单词序列的条件概率越高，则困惑度越低。因此，根据语言模型，最小化困惑度等价于最大化测试集概率。我们通常在公式 3.15 或 3.16 用的单词序列，是某个测试集中的整个单词序列。由于此序列将跨越许多句子边界，因此我们需要在概率计算中包括开始和结束句子标记<s>和</ s>。我们还需要在单词符记的总数 N 中包括句子结束标记</s>(但不包括句子开始标记<s>)。

还有另一种思考困惑度的方式:作为一种语言的加权平均的分支因子。语言的分支因子是任何单词后面可能出现的下一个单词的数量。考虑一下识别英文数字(0、1、2、……9)的任务。假设(在某些训练集中和某些测试集中)这 10 个数字中的每一个数字都以相等的概率 P = 1/10 出现，那么这个小型语言的困惑实际上是 10。为了验证这一点，想象一个长度为 N 的测试字符串，并假设在训练集中所有的数字发生的概率相等。由公式 3.15 可知，困惑度将是：

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

但是假设数字 0 的出现频率比其他数字高得多。假设 0 在训练集中出现 91 次，其他数字每个出现 1 次。现在我们看到下面的测试集:0 0 0 0 0 3 0 0 0 0 。我们应该预料到这个测试集的困惑度会更低，因为大多数时候下一个数字是零，这是很容易预测的，也就是说有很高的概率。因此，虽然分支因子仍为 10，但困惑度或加权分支因子较小。我们把这个精确的计算留给练习 12。

我们在 3.7 节中将看到，困惑度也与熵的信息理论概念密切相关。

最后，让我们看看如何使用困惑度来比较不同的 n-gram 模型。我们用《华尔街日报》的 3800 万个单词(包括句子开头的符记)训练了一元、二元和三元语法模型，使用了一个含有 19,979 个单词的词汇表。然后，我们用公式 3.16 在一个 150 万单词的测试集上计算每个模型的困惑度。下表显示了根据每种语法的 150 万个单词 WSJ 测试集的困惑度。

	Unigram	Bigram	Trigram
Perplexity	962	170	109

正如我们在上面看到的，n-gram 提供给我们有关单词序列的信息越多，困惑度就越低(因为如公式 3.15 所示，困惑度与根据模型的测试序列的可能性成反比)。

需要注意的是，在计算困惑度时，n-gram 模型 P 必须在不了解测试集的情况下构造，也不需要事先了解测试集的词汇量。测试集的任何知识都可能导致困惑度被人为地降低。两种语言模型的困惑只有在使用相同的词汇时才具有可比性。

困惑度的(内在的)改善，并不保证语言处理任务(如语音识别或机器翻译)性能的(外在的)改善。尽管如此，由于困惑往往与这些改进相关，它通常被用作对算法的快速检查。但是，在对模型进行评价之前，必须先对实际任务进行端到端的评价，以确定模型在困惑性方面的改进。

3.3.泛化与零

n-gram 模型，像许多统计模型一样，依赖于训练语料库。一个暗示是，概率通常会针对有关给定训练语料库的特定事实进行编码。另一个暗示是，随着我们增加 N 的值，n-grams 在对训练语料库进行建模方面做得越来越好。

我们可以通过借鉴 Shannon(1951)、Miller 和 Selfridge(1950)从不同的 n-gram 模型中生成随机句子的技术来将这两个事实可视化。最简单的可视化方法就是想象一元语法是如何工作的。想象英语中所有的单词都覆盖在 0 到 1 之间的概率空间中，每个词所覆盖的区间与它的频率成比例。我们在 0 到 1 之间选择一个随机值，然后打印其区间包含所选值的单词。我们继续选择随机数并生成单词，直到我们随机生成一个句子结尾的符记</s>为止。我们可以使用同样的技术来生成二元语法，首先生成一个以<s>开头的随机二元语法(根据它的二元语法概率)。假设该二元语法的第二个单词是 w。接下来，我们选择以 w 开头的随机二元语法(再次根据其二元语法概率绘制)，依此类推。

为了直观地了解高阶 n-gram 的强大功能，图 3.3 显示了根据莎士比亚作品中训练的 1-gram，2-gram，3-gram 和 4-gram 模型生成的随机句子。

我们训练模型的上下文越长，句子的连贯性就越强。在一元语法的句子中，单词与单词之间、或单词与句末标点之间没有连贯的关系。二元语法的句子具有局部的单词对单词的连贯性(尤其是如果我们认为标点符号算作一个词)。三元语法和四元语法的句子开始越来越像莎士比亚的作品了。确实，对这种四元语法

句子的仔细研究表明，它们看起来有点像莎士比亚。“It cannot be but so”这句话是直接来自国王约翰。这是因为，不要去指责莎士比亚，他的作品在语料库上并不是很大(N=884,647，V=29,066)，而我们的 n-gram 概率矩阵非常稀疏。可能的二元语法有 $V^2 = 844{,}000{,}000$ 个，可能的四元语法是 $V^4 = 7\times10^{17}$ 个。因此，一旦生成器选择了第一个四元语法(It cannot be but)，就只有五种可能的延续(that, I, he, thou, and so);事实上，对于许多四元语法，只有一个延续。

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

图 3-3：四个 n-gram 随机生成的八个句子

图注：根据莎氏作品进行计算，所有字符都被映射为小写，标点符号被视为单词。输出被手工更正为大写，以提高可读性。

为了了解语法对其训练集的依赖性，让我们看看在完全不同的语料库上训练的 n-gram 语法:《华尔街日报》(WSJ)报纸。莎士比亚和《华尔街日报》都是英语，所以我们可能会发现这两种体裁的 n-gram 符号有些重叠。图 3.4 显示了用《华尔街日报》的 4000 万个单词训练后的 unigram、bigram 和 trigram 语法生成的句子。

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N.B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

图 3-4：三个 n-gram 模型随机生成的三个句子

图注：三个 n-gram 模型从《华尔街日报》的 4000 万单词中随机生成的 3 个句子，将所有字符都小写，并将标点作为单词处理。输出随后被手工更正为大写，以提高可读性。

将这些例子与图 3.3 中的伪莎士比亚作比较。虽然它们都为“English-like sentences”建模，但在生成的句子中明显没有重叠，甚至在小短语中也几乎没有重叠。统计模型可能几乎毫无用处，因为训练集和测试集的预测器(predictors)与莎士比亚和《华尔街日报》不同。

当我们建立 n-gram 模型时，我们应该如何处理这个问题?第一步是确保使用与我们试图完成的任务相似体裁的训练语料库。为了建立法律文件翻译的语言模型，我们需要一个法律文件的训练语料库。为了建立问答系统的语言模型，需要建立问答训练语料库。

同样重要的是，获取适当方言或变体的训练数据，特别是在处理社交媒体帖子或口语文本时。例如，一些推特将使用非裔美国人语言(AAL)的特征——非洲裔美国人社区使用的许多语言变体的名称(King 2020)。这些特征包括像 finna 这样的词-它是一种助动词，用于标记立即将来时态-或者把“then”拼成“den”，就像这样的推特(Blodgett 和 O'Connor, 2017):

(3.18) Bored af den my phone finna die!!!

然而，来自尼日利亚英语等变种的推特，与来自美国英语的词汇和 n-gram 模式有明显不同(Jurgens 等人 2017)：

(3.19) @username R u a wizard or wat gan sef: in d mornin - u tweet, afternoon - u tweet, nyt gan u dey tweet. beta get ur IT placement wiv twitter

匹配的体裁和方言仍然是不够的。我们的模型可能仍然存在稀疏性问题。对于任何发生了足够次数的

n-gram，我们也许可以很好地估计它的概率。但是由于任何语料库都是有限的，因此一定会缺少一些完全可以接受的英语单词序列。也就是说，我们将有很多情况下假定的“零概率 n-grams”实际上应该具有一些非零概率。考虑一下《华尔街日报》 Treebank3 语料库中，在“denied the”之后的那些单词及其计数：

denied the allegations: 5
denied the speculation: 2
denied the rumors: 1
denied the report: 1

但是，假设我们的测试集具有以下短语：

denied the offer
denied the loan

我们的模型将错误地估计 P(offer|denied the)是 0!

这些零（0）—在训练集中不会出现，但在测试集中会出现—是个问题，有两个原因。

首先，它们的存在意味着我们低估了各种可能出现的单词的概率，这将损害我们希望在这些数据上运行的任何应用程序的性能。

其次，如果测试集之中任何一个单词的概率为 0，则测试集的整体概率为 0。根据定义，困惑度是基于测试集的逆概率的。因此，如果某些单词的概率为零，那么，因为我们不能除以 0，所以根本无法计算困惑度！

3.3.1. 未知单词

前节讨论了二元语法概率为零的那些单词的问题。但那些我们从未见过的单词概率问题怎么处理呢?

有时在我们的语言任务中，这种情况不会发生，因为我们知道所有可能发生的单词。在这样一个**封闭词汇(closed vocabulary)**的系统中，测试集只能包含本词汇库中的单词，不会有未知的单词。在某些领域(例如语音识别或机器翻译)中，这是一个合理的假设，在这些领域中，我们预先安装了发音词典或短语表，因此语言模型只能使用该词典或短语表中的单词。

在其他情况下，我们必须处理我们之前没有见过的单词，我们称之为未知单词，或**表外词(OOV)**单词。OOV 单词在测试集中出现的百分比称为 OOV 率。一个**开放词汇(open vocabulary)**的系统是这样一个系统，在这个系统中，首先，针对测试集之中潜在的未知词，我们添加一个名为<UNK>的伪词；然后，我们对那些未知词进行建模。

有两种常用的方法来训练未知词模型<UNK>的概率。

第一个方法是提前选择一个固定的词汇表，把这个问题变成一个封闭的词汇表:

1. 选择一个预先确定的词汇表(单词列表)。
2. 在文本规范化步骤中，将训练集中不属于该集合的任何单词(OOV)转换为未知单词符记<UNK>。
3. 根据<UNK>的计数估计其概率，就像训练集中的任何其他普通单词一样。

第二种方法，在我们事先没有预先的词汇表的情况下，是隐式地创建这样的词汇表，方法如下：基于单词的频率，用<UNK>替换训练数据中的单词。例如，我们可以用<UNK>替换所有在训练集中出现次数小于 n 次的单词，其中 n 是一个较小的数字，或者等价地提前选择单词大小 V(比如 50,000)，按频率选择前 V 个单词，其余的用 UNK 替换。在任何一种情况下，我们都将继续像以前一样训练语言模型，将<UNK>视为普通单词。

准确地选择<UNK>模型确实会对诸如困惑度等指标产生影响。通过选择小的词汇量和赋予高概率的未知词的方法，语言模型能够取得低困惑度。出于这个原因，困惑度应该只在具有相同词汇表的语言模型之间进行比较(Buck 等人，2014)。

3.4. 平滑

我们如何处理词汇表中的单词(它们不是未知的单词)，但它们出现在一个看不见的上下文的测试集之中(例如，它们出现在一个在训练中从未出现过的单词之后)?为了防止语言模型为这些不可见事件分配零概

率，我们将不得不从一些更频繁发生的事件中消除一些**概率质量(probability mass)**，并将其分配给我们从未见过的事件。这种修改被称为**平滑(smoothing)**或**折扣(discounting)**。在这一节和接下来的几节中，我们将介绍各种平滑的方法：拉普拉斯(加一)平滑、加 k 平滑、傻瓜回退和 Kneser-Ney 平滑。

3.4.1. 拉普拉斯平滑

做平滑处理的最简单方法是在我们将它们标准化为概率之前，将所有的 bigram 计数加 1。以前为 0 的所有计数现在都将为 1，1 的计数将为 2，以此类推。这种算法称为**拉普拉斯平滑(Laplace smoothing)**。拉普拉斯平滑在现代 n-gram 模型中表现得不够好，但它很有用地引入了许多我们在其他平滑算法中看到的概念，给出了一个有用的基线，也是用于其他任务(如文本分类，第 4 章)的实际平滑算法。

让我们从拉普拉斯平滑在一元语法概率中的应用开始。回想一下，单词 w_i 的一元语法概率的未平滑最大似然估计是其计数 c_i 的归一化(N 是单词符记总数)：

$$P(w_i) = \frac{c_i}{N}$$

拉普拉斯平滑仅对每个计数加一(因此其备用名称为**加一(add-one)**平滑)。由于词汇中有 V 个单词，并且每个单词都增加了，因此我们还需要调整分母以考虑额外的 V 个观察值。(如果不增加分母，那么 P 值会怎样？)

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V} \tag{3.20}$$

通过定义一个调整后的计数 c^*，可以方便地描述平滑算法如何影响分子，而不必同时更改分子和分母。这个调整后的计数更容易直接与 MLE 计数进行比较，并且可以通过 N 的归一化将其转换为类似于 MLE 计数的概率。为了定义此计数，因除了加 1 之外，我们仅更改了分子，故还需要乘以归一化因子 N/(N+V)：

$$c_i^* = (c_i + 1)\frac{N}{N + V} \tag{3.21}$$

现在我们可以通过将 N 做归一化把 c_i^*变成概率 P_i^*。

看待平滑的一种相关方法是，为了获得将被分配给零计数的概率质量，**打折(discounting)**(降低)一些非零计数。因此，我们不是引用折扣计数 c^*，而是用相对折扣 d_c 来描述平滑算法，即折扣计数与原始计数的比率：

$$d_c = \frac{c^*}{c}$$

现在我们对 unigram 情况有了直观的了解，让我们平滑伯克利餐厅项目的 bigrams。图 3.5 显示了图 3.1 中 bigrams 的加一平滑计数。

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

图 3-5：加一平滑 bigram 计数的 8 个单词

图注：(从 V = 1446 中)伯克利餐厅项目语料库有 9332 个句子。

图 3.6 显示了图 3.2 中 bigrams 的加一平滑概率。回想一下，通常的 bigrams 概率是通过用 unigram 计数归一化每一行计数来计算的：

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{3.22}$$

对于加一平滑的 biggram 计数，我们需要通过词汇表 V 中单词类型总数来增加 unigram 计数:

$$P^*_{\text{Laplace}}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w C(w_{n-1}w)+1} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \tag{3.23}$$

因此，上一节给出的每一个 unigram 计数都需要通过 V = 1446 来增加，其结果为图 3.6 中平滑的 bigram 概率。

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

图 3-6：加一平滑 bigram 概率的 8 个单词

图注：在 9332 个句子的 BeRP 语料库中，8 个单词(从 V = 1446 中)的加一平滑 bigram 概率。

重建计数矩阵通常很方便，这样我们就可以看到平滑算法对原始计数的改变程度。这些调整后的计数可以通过公式 3.24 计算出来。图 3.7 显示了重建计数。

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1]\times C(w_{n-1})}{C(w_{n-1})+V} \tag{3.24}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

图 3-7：对 8 个单词的加一重建计数

图注：在 9332 个句子的 BeRP 语料库中，对 8 个单词(从 V = 1446 中)的加一重建计数。

请注意，加一平滑对计数进行了很大的更改。C(want to)从 609 变为 238！我们也可以在概率空间中看到这一点：P(to | want)从不平滑情况下的 0.66 减少到平滑情况下的 0.26。查看折扣 d(新旧计数之间的比率)可以看出每个前缀字的计数减少得多么惊人。bigram 中 want to 的折扣是 0.39，而 Chinese food 的折扣是 0.10，是 10 的倍数！

发生计数和概率的急剧变化是因为太多的概率质量移到了全零。

3.4.2. 加 k 平滑

加一平滑的另一种选择，是从可见事件到不可见事件的概率质量稍微减少一点。我们不是给每个计数加 1，而是加一个分数计数 k(.5?. 05?. 01?)。因此，此算法被称为**加 k(add-k)**平滑。

$$P^*_{\text{Add-k}}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV} \tag{3.25}$$

加 k 平滑要求我们有选择 k 的方法;例如，这可以通过优化开发集(devset)来实现。尽管加 k 在一些任

务中很有用(包括文本分类)，但它仍然不能很好地用于语言建模，生成的计数具有很差的方差和经常不适当的折扣(Gale 和 Church, 1994)。

3.4.3. 回退和插值

我们一直在讨论的打折可以帮助解决零频率 n-gram 的问题。但我们还可以利用另外一个知识来源。如果我们试图计算 $P(w_n | w_{n-2}\, w_{n-1})$但是我们又没有一个特定的三元语法 $w_{n-2}\, w_{n-1}\, w_n$ 例子,那么,我们可以通过使用二元语法概率 $P(w_n | w_{n-1})$来估计其概率。同样的,如果我们没有计数来计算 $P(w_n | w_{n-1})$,那么我们可以查看一元语法的 $P(w_n)$。

换句话说，有时使用较少的上下文是一件好事，可以帮助生成更多模型还不太了解的上下文。有两种方法可以使用这个 n-gram“层次结构”。在**回退(bakkoff)**中，如果证据充分，则用三元语法，否则用二元语法，否则就用一元语法。换句话说，我们只有在没有证据证明高阶 n-gram 存在的情况下才会“回退”到低阶 n-gram。相比之下，在**插值(interpolation)**中，通过权衡和组合 trigram、bigram、和 unigram 计数的方式，我们总是混合来自所有 n 元语法估计器的概率估计。

在简单线性插值中，我们通过对所有模型进行线性插值来组合不同阶的 n-gram。因此，通过混合 unigram、bigram 和 trigram 概率及其对应的加权 λ 系数，我们估计 trigram 概率 $P(w_n|w_{n-2}\, w_{n-1})$:

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) = {} & \lambda_1 P(w_n | w_{n-2} w_{n-1}) \\ & + \lambda_2 P(w_n | w_{n-1}) \\ & + \lambda_3 P(w_n)\end{aligned} \tag{3.26}$$

其中，λ_s 总和为 1：

$$\sum_i \lambda_i = 1 \tag{3.27}$$

在一个稍微复杂的线性插值版本中，每个λ权重是通过上下文条件计算的。这样，如果我们对一个特定的二元语法具有特别准确的计数，那么我们就认为基于该二元语法的三元语法的计数将更加可信，因此我们可以使这些三元语法的 λ_s 更高，从而在插值中使该三元语法的权重更大。公式 3.28 显示了具有上下文条件权重的插值公式：

$$\begin{aligned}\hat{P}(w_n | w_{n-2} w_{n-1}) = {} & \lambda_1(w_{n-2:n-1})\ P(w_n | w_{n-2} w_{n-1}) \\ & + \lambda_2(w_{n-2:n-1})\ P(w_n | w_{n-1}) \\ & + \lambda_3(w_{n-2:n-1})\ P(w_n)\end{aligned} \tag{3.28}$$

这些λ值如何设置？简单插值和条件插值λ_s 都是从**保留(held-out)**语料库中学习的。保留语料库是一种额外的训练语料库，我们可以通过选择最大化保留语料库似然度的λ值来设置这些λ值之类的超参数。也就是说，我们固定 n 元语法概率，然后搜索λ值(当插入公式 3.26 时)给保留集最大的概率。有很多方法可以找到λ_s 的最优集合。一种方法是使用 EM 算法，这是一种收敛于局部最优λ_s 的迭代学习算法(Jelinek 和 Mercer, 1980)。

在回退的 n-gram 模型中，如果我们需要的 n-gram 计数为零，那么我们通过回退到(N-1)-gram 来近似它。我们将继续退缩，直到达到具有重要历史意义的计数。

为了让回退模型给出一个正确的概率分布，我们必须对高阶 n-gram 进行打折，为低阶 n-gram 保留一些概率质量。就像加法平滑一样，如果不对高阶 n-gram 进行打折，而我们只使用了未打折的 MLE 概率，则只要我们将具有零概率的 n-gram 替换为低阶 n-gram，我们就可以添加概率质量，通过语言模型对所有可能的字符串的概率进行分配，总概率将大于 1！除了这个显式的折扣因子，我们还需要一个函数 α 来将这个概率质量分配到低阶 n-gram。

这种打折的回退也被称为**卡茨回退(Katz backoff)**。在 Katz 回退中，如果我们之前看过这个 n-gram(即，如果我们有非零计数)，则我们依赖于打折概率 P^*；否则，我们递归地返回较短历史(N-1)-gram 的卡茨概率。因此，计算回退 n-gram 的 P_{BO} 概率如下：

$$P_{BO}(w_n \mid w_{n-N+1:n-1}) = \begin{cases} P^*(w_n \mid w_{n-N+1:n-1}), & \text{if } C(w_{n-N+1:n}) > 0 \\ \alpha(w_{n-N+1:n-1}) P_{BO}(w_n \mid w_{n-N+2:n-1}), & \text{otherwise.} \end{cases} \tag{3.29}$$

卡茨回退通常与一种称为“好图灵”(Good-Turing)的平滑方法结合在一起。组合的 Good-Turing 回退算法涉及到相当详细的计算，这种计算是为了估计 Good-Turing 平滑、P^*和 α 值。

3.5. Kneser-Ney 平滑

最常用和性能最好的 n-gram 平滑方法之一是插值的 **Kneser-Ney 算法**(Kneser 和 Ney 1995, Chen 和 Goodman 1998)。

Kneser-Ney 起源于一种称为绝对打折的方法。回想一下，频繁 n-gram 的计数打折对于节省一些概率量是必要的，以便使平滑算法分配给看不见的 n-gram。

为了看到这一点，我们可以使用 Church 和 Gale(1991)的一个聪明的想法。考虑一个计数为 4 的 n-gram。我们需要对这个计数作一些打折。但是我们应该打折多少呢?Church 和 Gale 的聪明主意是查看一个保留的语料库，然后看看在训练集中计数为 4 的所有 bigrams 的计数是多少。他们从美联社的 2200 万个单词中计算出了一个 bigram 语法，然后在另外 2200 万个单词中检查了每个 bigram 的计数。平均而言，在前 2200 万个单词中出现 4 次的 bigram 在后 2200 万个单词中出现 3.23 次。图 3.8(Church 和 Gale,1991)显示了 c 介于 0 到 9 之间的 bigrams 的计数。

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

图 3-8：训练集和保留集的 bigrams 计数

图注：训练集包括美联社新闻热线的 2200 万单词，其 bigram 计数分别为 0,1,2，…9；对应地，保留集语料库也包含 2200 万单词，其 bigram 计数分别为 0.0000270，0.448，1.25，...8.26.

在图 3.8 中注意到，除了 0 和 1 的保留计数之外，只需从训练集中的计数中减去 0.75，就可以很好地估算出保留集中的所有其他二元语法计数！**绝对打折(absolut discunting)**是通过从每次计数中减去一个固定的(绝对的)折扣 d 来形式化这种直觉。直觉是，由于我们已经对非常高的计数有了很好的估计，一个小的折扣 d 不会对它们有太大的影响。它将主要修改较小的计数，对于这些计数我们不一定要相信其估值。图 3.8 表明，在实践中，这种折扣实际上对计数从 2 到 9 的 bigrams 是一个很好的折扣。适用于 bigrams 绝对折扣的插值公式：

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}\, v)} + \lambda(w_{i-1}) P(w_i) \tag{3.30}$$

第一项是折扣的 bigram，第二项是带插值权重λ的 unigram。我们可以将所有的 d 值都设置为 0.75，也可以为计数是 1 的 bigrams 单独保持 0.5 的折扣值。Kneser-Ney 折扣(Kneser 和 Ney，1995)以一种更复杂的方式来处理较低阶的 unigram 分布，从而增强了绝对折扣。假设我们正在对一个 bigram 和一个 unigram 模型进行插值，考虑一下预测下面这个句子中的下一个单词的工作。

I can’t see without my reading_____.

在这里，“glasses(眼镜)”这个词似乎比“Kong(孔)”这个词更容易出现，所以我们希望我们的 unigram

模型更喜欢“glasses”这个词。但事实上，更常见的是“Kong”，因为“Hong Kong(香港)”是一个非常高频的词。一个标准的 unigram 模型将赋予“Kong”比“glasses”更高的概率。我们想要捕捉的直觉是，虽然“Kong”是高频的，但主要只是高频在“Hong Kong”，即“Hong”一词之后。glass 这个词的分布范围要广得多。

换句话说，我们想创建一个称为 $P_{CONTINUATION}$ 的 unigram 模型，而不是创建 P(w)。$P_{CONTINUATION}$ 回答了“How likely is w to appear as a novel continuation?”问题，而 P(w)回答了"How likely is w?"问题。在一个新的看不见的上下文之中，我们如何估计将 w 这个词视为新延续(novel continuation)的概率? Kneser-Ney 的直觉是，将我们对 $P_{CONTINUATION}$ 的估计，建立在“number of different contexts word w has appeared in”上，即它完成的 bigram 类型的数量。每一种字母类型在第一次被看到时都是一种新延续。我们假设，过去出现在更多上下文中的单词，也更有可能出现在新的上下文中。单词 w 作为新延续出现的次数可以表示为：

$$P_{CONTINUATION}(w) \propto |\{v : C(vw) > 0\}| \tag{3.31}$$

为了将这个计数转化为一个概率，我们使用 bigram 类型的总数进行归一化。总而言之:

$$P_{CONTINUATION}(w) = \frac{|\{v : C(vw) > 0\}|}{|\{(u', w') : C(u', w') > 0\}|} \tag{3.32}$$

基于不同隐喻的等效表述是使用在 w 之前出现的单词类型数量(重复公式 3.31)：

$$P_{CONTINUATION}(w) \propto |\{v : C(vw) > 0\}| \tag{3.33}$$

按所有单词前面的单词数进行归一化，如下所示：

$$P_{CONTINUATION}(w) = \frac{|\{v : C(vw) > 0\}|}{\sum_{w'} |\{v : C(v\,w') > 0\}|} \tag{3.34}$$

一个频词(Kong)出现在仅一个上下文中(Hong)的将具有较低的延续概率。

于是，针对 bigrams 的**插值的 Kneser-Ney** 平滑的最终方程为：

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i) \tag{3.35}$$

λ是一个归一化常数，用于分配我们已经打折的概率质量：

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)} |\{w : C(w_{i-1}w) > 0\}| \tag{3.36}$$

第一项 $d / \Sigma_v C(w_{i-1}\, v)$是归一化折扣，第二项$|\{w：C(w_{i-1}\, w)> 0\}|$是可以跟随 w_{i-1} 的单词类型的数量，或者等效地，我们已经打折的单词类型的数量；换句话说，就是我们使用归一化折扣次数的数量。

一般递推公式如下:

$$P_{KN}(w_i \mid w_{i-n+1:i-1}) = \frac{\max(c_{KN}(w_{i-n+1:i}) - d, 0)}{\sum_v c_{KN}(w_{i-n+1:i-1}\, v)} + \lambda(w_{i-n+1:i-1}) P_{KN}(w_i \mid w_{i-n+2:i-1}) \tag{3.37}$$

其中，计数 C_{KN} 的定义取决于我们是计算被插值的最高阶 n-gram(例如，如果我们正在插值的是 trigram、bigram 和 unigram，则是 trigram)，还是一个低阶的 n-gram(如果我们正在插值的是 trigram、bigram 和 unigram，则是 bigram 或 unigram):

$$c_{KN}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{, for the highest order} \\ \text{continuationcount}(\cdot) & \text{, for lower orders} \end{cases} \tag{3.38}$$

延续计数是(·)的唯一单个单词上下文的数量。

在递归结束时，用均匀分布对 unigrams 进行插值，其中参数ε是空字符串：

$$P_{KN}(w) = \frac{\max(c_{KN}(w) - d, 0)}{\sum_{w'} c_{KN}(w')} + \lambda(\varepsilon)\frac{1}{V} \quad (3.39)$$

如果我们想包含一个未知词<UNK>，那么它只是作为一个计数为零的常规词汇条目，因此它的概率将是λ加权均匀分布λ(e)/V。

Kneser-Ney 平滑法的最佳性能版本被称为**修正的 Kneser-Ney** 平滑法，这是由 Chen 和 Goodman(1998)提出的。改进的 Kneser-Ney 并不使用单个固定折扣 d，而是对 n-gram 使用三种不同的折扣 d1、d2 和 d3 +，其计数分别对应为 1、2 和 3 或更多。详见 Chen 和 Goodman(1998，p.19)或 Heafield 等人(2013)。

3.6. 巨大的语言模型和傻瓜回退

通过使用来自网络或其他大量的集合的文本，可以建立非常大的语言模型。Google 发行的 Web 1 万亿 5-gram 语料库包括各种大的 n-gram 集合，包括从 1-gram 到 5-gram 的所有五词(five-word)序列单词，这些单词出现在至少 40 本不同的书中，这些书来自含有 1,024,908,267,229(约 1.024 万亿，译者注)个单词的文本，这些文本来自公开发行的可访问的英文网页(Franz 和 Brants，2006 年)。谷歌还发布了 Google Books Ngrams 语料库，其中包含从其图书收藏中提取的 n-gram，包括来自中文、英语、法语、德语、希伯来语、意大利语、俄语和西班牙语的另外 8000 亿个 n-gram 符记(Lin 等人，2012a)。更小但更精心策划的英语 n-gram 语料库包括来自 COCA(当代美国英语语料库)的 10 亿单词的美国英语语料库(Davies, 2020)中最常用的 100 万个 n-gram。COCA 是一个平衡的语料库，这意味着它有大致相同数量的来自不同体裁的词:1990-2019 年期间的网络、报纸、口语对话记录、小说等，并有每个 n-gram 的上下文，以及体裁和出处的标签)。

Google Web 语料库的一些 4-grams 示例：

4-gram	Count
serve as the incoming	92
serve as the incubator	99
serve as the independent	794
serve as the index	223
serve as the indication	72
serve as the indicator	120
serve as the indicators	45

在构建使用如此大的 n-gram 集合的语言模型时，效率考虑非常重要。不是将每个单词存储为字符串，而是在内存中表示为 64 位哈希数，单词本身存储在磁盘上。概率通常仅使用 4-8 位(而不是 8 字节浮点数)进行量化，n-grams 被存储在倒序前缀树(reverse tries)中。

N-gram 也可以通过修剪来缩小，例如仅存储计数大于某个阈值(例如，用于 Google n-gram 发布的计数阈值为 40)的 n-gram，或使用熵来修剪次要的 n-gram(Stolcke，1998)。另一个选择是使用**布隆过滤器(Bloom filters)**之类的技术来构建近似语言模型(Talbot 和 Osborne 2007，Church 等人 2007)。最后，高效的语言模型工具包，如 KenLM (Heafield 2011, Heafield 等人 2013)，使用排序数组，有效地将概率和回退合并到单个值中，并使用合并排序，以最少的次数通过大型语料库，有效地构建概率表。

尽管使用这些工具包可以使用完整的 Kneser-Ney 平滑法构建 web 级的语言模型，但 Brants 等人(2007)表明，对于非常大的语言模型，一个非常简单的算法可能就足够了。这个算法被称为**傻瓜回退(stupid backoff)**。傻瓜回退放弃了试图使语言模型成为真实概率分布的想法。高阶概率没有折扣。如果一个高阶 n-gram 的计数为零，我们就简单地回退到一个低阶 n-gram，并用固定的权重(与上下文无关)加权。该算法不会产生概率分布，因此我们将遵循 Brants 等人(2007)的观点 ，将其称为 S：

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{count(w_{i-k+1}^{i})}{count(w_{i-k+1}^{i-1})}, if \ \ count(w_{i-k+1}^{i}) > 0 \\ \lambda S(w_i \mid w_{i-k+2}^{i-1}), otherwise \end{cases} \tag{3.40}$$

回退以 unigram 结束，其概率 S(w) = count(w)/N。Brants 等人(2007)发现，λ的值取 0.4 为好。

3.7. 高级专题：困惑度与熵的关系

我们在 3.2.1 节中引入了困惑度，作为评估测试集上的 n-gram 模型的一种方法。更好的 n-gram 模型是为测试数据分配更高概率的模型，而困惑度是测试集概率的归一化版本。困惑度度量实际上源于交叉熵的信息理论概念，它解释了困惑度的神秘性质(例如，为什么有逆概率？)及其与熵的关系。**熵(entropy)**是信息的度量。给定一个随机变量 X，其范围是我们预测的范围(单词、字母、词类、被称为 X 的集合)，并具有特定的概率函数，则将其称为 p(X)，即随机变量 X 的熵是：

$$H(X) = \sum_{x \in \chi} p(x) \log_2 p(x) \tag{3.41}$$

原则上，对数(log)可以在任何底数(base)下计算。如果我们使用以 2 为底数的对数，则得到的熵值将以比特来度量。一种直观的理解熵的方式是：在最优编码方案中，熵是编码某个决策或信息所需要的比特数的下界。

考虑一个来自标准信息理论教科书 Cover 和 Thomas(1991)的例子。想象一下，我们想对一场赛马下注，但是距离 Yonkers 赛马场太远了，所以我们想向博彩者发送一条短信，告诉他要投注的八匹马中的哪一匹。编码此消息的一种方法是仅使用马匹数字的二进制表示形式作为代码；因此，马 1 的编码是 001，马 2 是 010，马 3 是 011，依此类推，马 8 是 000。如果我们整天投注，而且每匹马都用 3 位编码，那么平均而言，我们每场比赛将发送 3 位。

我们可以做得更好吗？假设价差(spread)是所下注的实际分布，我们将其表示为每匹马的先验概率，如下所示：

Horse 1	1/2	Horse 5	1/64
Horse 2	1/4	Horse 6	1/64
Horse 3	1/8	Horse 7	1/64
Horse 4	1/16	Horse 8	1/64

遍及马的随机变量 X 的熵为我们提供了位数的下限，为：

$$\begin{aligned} H(X) &= -\sum_{i=1}^{i=8} p(i) \log p(i) \\ &= -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - \frac{1}{8}\log\frac{1}{8} - \frac{1}{16}\log\frac{1}{16} - 4(\frac{1}{64}\log\frac{1}{64}) \\ &= 2 \text{ bits} \end{aligned} \tag{3.42}$$

一个平均每场比赛 2 位(bits)的代码可以用更短的编码来表示更多的马，用更长的编码来表示更少的马。例如，我们可以将最有可能的马编码为 0，其余的马编码为 10，然后是 110、1110、111100、111101、111110 和 111111。

如果马的概率是相等的呢?我们在上面看到，如果我们用等长二进制编码来表示马的数目，那么每匹马需要 3 个比特来编码，所以平均值是 3。熵是一样的吗?在这种情况下，每匹马的概率是 1/8。这样选择马匹的熵为：

$$H(X) = -\sum_{i=1}^{i=8} \frac{1}{8}\log\frac{1}{8} = -\log\frac{1}{8} = 3 \text{ bits} \tag{3.43}$$

到目前为止，我们一直在计算单个变量的熵。但是我们用熵来做的大部分事情都涉及到序列。例如，

对于语法，我们将计算单词 W = {w_0,w_1,w_2，…，w_n}序列的熵。一种实现方法是使变量在单词序列范围内变化。例如，我们可以计算某种语言 L 中长度为 n 的所有有限单词序列的随机变量的熵，如下所示：

$$H(w_1,w_2,...,w_n)=\sum_{W_1^n\in L}p(W_1^n)\log p(W_1^n) \tag{3.44}$$

我们可以将**熵率(entropy rate)**(也可以将其视为每个单词的熵)定义为该序列的熵除以单词数：

$$\frac{1}{n}H(W_1^n)=-\frac{1}{n}\sum_{W_1^n\in L}p(W_1^n)\log p(W_1^n) \tag{3.45}$$

但是为了测量一种语言的真实熵，我们需要考虑无限长度的序列。如果我们把语言看作一个随机过程 L，它产生一个单词序列，并允许 W 代表单词 w_1,w_2，…，w_n 的序列，那么 L 的熵率 H(L)定义为：

$$\begin{aligned}H(L)&=\lim_{n\to\infty}\frac{1}{n}H(w_1,w_2,...,w_n)\\&=-\lim_{n\to\infty}\frac{1}{n}\sum_{W\in L}p(w_1,w_2,...,w_n)\log p(w_1,w_2,...,w_n)\end{aligned} \tag{3.46}$$

Shannon-McMillan-Breiman 定理(Algoet 和 Cover 1988，Cover 和 Thomas 1991)指出，如果语言在某些方面是规则的(确切地说，既是固定的又是遍历的)，那么：

$$H(L)=\lim_{n\to\infty}-\frac{1}{n}\log p(w_1,w_2,...,w_n) \tag{3.47}$$

也就是说，我们可以采用足够长的单个序列，而不是对所有可能的序列求和。Shannon-McMillan-Breiman 定理的直觉是，足够长的单词序列将在其中包含许多其他较短的序列，并且这些较短的序列中的每一个将根据其概率在较长的序列中再次出现。

如果随机过程分配给序列的概率相对于时间索引的变化是不变的，则称其为**平稳(stationary)**过程。换句话说，单词在时间 t 的概率分布与在时间 t + 1 的概率分布相同。马尔可夫模型，进而 n-grams，是平稳的。例如，在 bigram 中，P_i 仅取决于 P_{i-1}。因此，如果我们将时间索引移动 x，则 P_{i+x} 仍然取决于 P_{i+x-1}。但是自然语言并不是平稳的，因为正如我们在第 12 章中所展示的那样，即将出现的单词的概率可能取决于任意距离和时间相关的事件。因此，我们的统计模型仅给出自然语言的正确分布和熵的近似值。

总之，通过做一些不正确但方便的简化假设，我们可以计算随机过程的熵(通过取一个很长的输出样本并计算其平均对数概率的方式)。

现在，我们准备介绍**交叉熵(cross-entropy)**。当我们不知道产生某些数据的实际概率分布 p 时，交叉熵很有用。它允许我们使用一些 m，它是 p 的模型(即 p 的近似值)。m 在 p 上的交叉熵定义为：

$$H(p,m)=\lim_{n\to\infty}-\frac{1}{n}\sum_{W\in L}p(w_1,w_2,...,w_n)\log m(w_1,w_2,...,w_n) \tag{3.48}$$

也就是说，我们根据概率分布 p 来描述序列，但根据 m 来对它们的概率的对数进行求和。

再次，按照 Shannon-McMillan-Breiman 定理，进行平稳的遍历过程：

$$H(p,m)=\lim_{n\to\infty}-\frac{1}{n}\log m(w_1,w_2,...,w_n) \tag{3.49}$$

这意味着，对于熵，我们可以用一个足够长的单个序列来估计模型 m 在某个分布 p 上的交叉熵，而不是对所有可能的序列求和。交叉熵之所以有用是因为交叉熵 H(p,m)是熵 H(p)的上界。对于任何模型 m:

$$H(p)\le H(p,m) \tag{3.50}$$

这意味着我们可以使用一些简化的模型 m 来帮助估计(根据概率 p 来描述的符号序列的)真实熵。m 越

精确，交叉熵 H(p,m)就越接近真实熵 H(p)。因此，H(p,m)和 H(p)之间的差异是对模型精确程度的衡量。在两个模型 m_1 和 m_2 中，交叉熵越小的模型越准确。(交叉熵永远不会低于真实熵，因此模型不会因低估真实熵而犯错。)

我们终于可以看到困惑度和交叉熵之间的关系，如我们在公式 3.49 中看到的那样。当观察到的单词序列的长度达到无穷大时，交叉熵被定义为极限。我们将需要一个近似的交叉熵，依赖于一个固定长度的(足够长的)序列。对单词序列 W 的模型 $M = P(w_i \mid w_{i-N+1} \ldots w_{i-1})$，这种近似的交叉熵为：

$$H(W) = -\frac{1}{N}\log P(w_1, w_2, \ldots, w_N) \tag{3.51}$$

一个关于单词 W 序列的模型 P 的困惑度，现在正式定义为这个交叉熵的表达式:

$$\begin{aligned} \text{Perplexity}(W) &= 2^{H(W)} \\ &= P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1, \ldots, w_{i-1})}} \end{aligned} \tag{3.52}$$

3.8. 总结

本章介绍了语言建模和语言处理中使用最广泛的工具之一 n-gram。

•语言模型提供了一种方法来为一个句子或其他单词序列分配概率，并从前面的单词中预测一个单词。

•n-gram 是马尔科夫模型，从一个固定窗口的前单词来估计单词。n-gram 概率可以通过在语料库中计数和归一化(**最大似然估计**)来估计。

•n-gram **语言模型**在某些任务中被外部评估，或者在内部使用**困惑度**。

•根据语言模型的测试集的困惑度是由模型计算出的反测试集概率的几何平均值。

•**平滑**算法提供了一种更复杂的方法来估计 n-grams 的概率。常用的 n-grams 平滑算法依赖于通过**回退或插值**的低阶 n-grams 的计数。

•回退和插值都需要**打折**来创建概率分布。

•**Kneser-Ney** 平滑处理利用了单词是新延续的可能性。插值的 Kneser-Ney 平滑算法将打折概率与低阶连续概率混合在一起。

3.9. 文献和历史说明

n-gram 的基本数学原理最早是由马尔科夫(1913)提出的，他使用现在称为马尔科夫链(bigrams 和 trigrams)的方法来预测普希金的《尤金·奥涅金》(Eugene Onegin)中即将出现的字母是元音还是辅音。马尔科夫将 20,000 个字母分类为 V 或 C，并计算出给定字母为前一个或两个字母后的元音的 bigram 和 trigram 概率。Shannon(1948)应用 n-gram 来计算英语单词序列的近似值。基于 Shannon 的工作，到 1950 年代，马尔可夫模型已广泛用于工程学，语言学和心理工作中，用于对单词序列进行建模。从 Chomsky (1956)开始，包括 Chomsky(1957)、Miller 与 Chomsky(1963)在内的一系列极具影响力的论文中，Noam Chomsky 认为，“有限状态马尔可夫过程”虽然可能是有用的工程启发式，但不能成为人类语法知识的完整认知模型。这些争论导致许多语言学家和计算语言学家几十年来忽略了统计建模方面的工作。

n-gram 模型的复兴来自于 Jelinek 和他在 IBM 托马斯·j·沃森研究中心的同事，他们受到了香农和卡内基梅隆大学贝克的影响，而 Baker 又受到了 Baum 和他同事的影响。这两个实验室分别成功地在他们的语音识别系统中使用了 n-gram (Baker 1975b, Jelinek 1976, Baker 1975a, Bah 等人 1983, Jelinek 1990)。

加一平滑法源自拉普拉斯(Laplace)1812 年的继承法则，是 Jeffreys(1948 年)根据 Johnson(1932 年)较早提出的加 k 建议首次将其作为工程解决方案应用于零频率问题的。Gale 和 Church(1994)总结了加一算法的问题。

在 80 年代和 90 年代提出了各种不同的语言建模和平滑技术，其中包括 Good-Turing 折扣--由 Katz 首次应用于 IBM 在 n-gram 上的平滑处理(Nadas 1984, Church 和 Gale 1991) --Witten-Bell 折扣(Witten and Bell,1991)、以及使用词类信息的各种**基于分类的 n 元语法(class-based n-gram)**模型。

从 1990 年代后期开始，Chen 和 Goodman 进行了许多精心控制的实验，比较了不同的折扣算法、缓存模型、基于分类的模型和其他语言模型参数(Chen 和 Goodman 1999，Goodman 等人 2006)。他们展示了改进的 Kneser-Ney 插值的优点，它成为 n-gram 语言建模的标准基线，特别是因为他们表明缓存和基于分类的模型仅提供了较小的额外改进。这些文章推荐给任何对 n-gram 语言建模有进一步兴趣的读者。SRILM (Stolcke, 2002)和 KenLM (Heafield 2011, Heafield 等人 2013)是用于构建 n-gram 语言模型的公开可用工具包。

现代语言建模通常使用神经网络语言模型来完成，它解决了 n-gram 的主要问题：随着 n-gram 阶数的增加，参数数量呈指数增长，n-gram 无法生成从训练到测试的数据集。相反，神经语言模型将单词投射到一个连续的空间中，其中具有相似上下文的单词具有相似的表示形式。我们将在第 7 章中介绍前馈语言模型(Bengio 等人 2006，Schwenk 2007)，并在第 9 章中介绍递归语言模型(Mikolov 2012)。

3.10. 练习

3.1 Write out the equation for trigram probability estimation (modifying Eq. 3.11).

Now write out all the non-zero trigram probabilities for the I am Sam corpus on page 32.

3.2 Calculate the probability of the sentence i want chinese food. Give two probabilities, one using Fig. 3.2 and the ‘useful probabilities’ just below it on page 34, and another using the add-1 smoothed table in Fig. 3.6. Assume the additional add-1 smoothed probabilities *P*(i|<s>) = 0.19 and *P*(</s>|food) =0.40.

3.3 Which of the two probabilities you computed in the previous exercise is higher, unsmoothed or smoothed? Explain why.

3.4 We are given the following corpus, modified from the one in the chapter:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I am Sam </s>

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

Using a bigram language model with add-one smoothing, what is P(Sam | am)? Include <s> and </s> in your counts just like any other token.

3.5 Suppose we didn’t use the end-symbol </s>. Train an unsmoothed bigram grammar on the following training corpus without using the end-symbol </s>:

<s> a b

<s> b b

<s> b a

<s> a a

Demonstrate that your bigram model does not assign a single probability distribution across all sentence lengths by showing that the sum of the probability of the four possible 2 word sentences over the alphabet *{*a,b*}* is 1.0, and the sum of the probability of all possible 3 word sentences over the alphabet *{*a,b*}* is also 1.0.

3.6 Suppose we train a trigram language model with add-one smoothing on a given corpus. The corpus contains V word types. Express a formula for estimating P(w3|w1,w2), where w3 is a word which follows the bigram (w1,w2), in terms of various N-gram counts and V. Use the notation c(w1,w2,w3) to denote the number of times that trigram (w1,w2,w3) occurs in the corpus, and so on for bigrams and unigrams.

3.7 We are given the following corpus, modified from the one in the chapter:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I am Sam </s>

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

If we use linear interpolation smoothing between a maximum-likelihood bigram model and a maximum-likelihood unigram model with $\lambda_1 = 1/2$ and $\lambda_2 = 1/2$, what is P(Sam|am)? Include <s> and </s> in your counts just like any other token.

3.8 Write a program to compute unsmoothed unigrams and bigrams.

3.9 Run your n-gram program on two different small corpora of your choice (you might use email text or newsgroups). Now compare the statistics of the two corpora. What are the differences in the most common unigrams between the two? How about interesting differences in bigrams?

3.10 Add an option to your program to generate random sentences.

3.11 Add an option to your program to compute the perplexity of a test set.

3.12 You are given a training set of 100 numbers that consists of 91 zeros and 1 each of the other digits 1-9. Now we see the following test set: 0 0 0 0 0 3 0 0 0 0. What is the unigram perplexity?