3 N 元语言模型

> “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.
>
> ——使用《简-奥斯汀》训练的的三元模型随机生成的句子

正如那句老笑话所说，预测尤其是关于未来的预测是很困难的。但预测一些看起来容易得多的事情，比如某人下一句话会说什么，会如何呢？例如，接下来跟在这句话后面的是什么词呢？

The water of Walden Pond is so beautifully ...

你可能会得出结论，接下来的词很可能是 blue 、 green或 clear，但很可能不是refrigerator 或 this。在本章中，我们通过引入**语言模型**（**Language Models**，简称**LMs**）将这种直觉形式化，语言模型为每一个可能的下一个词分配一个**概率**。语言模型还可以为整个句子分配一个概率，比如下面的句子在文本中出现的概率要高得多：

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

突然之间，我注意到有三个人站在人行道上

而不是这组词以不同的顺序出现的另一个句子：

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

“在人行道上，突然三个人站着，我注意到全部”

我们为什么要预测即将出现的单词，或者知道一个句子的概率呢？其中一个原因是为了实现文本生成：选择上下文中更合适的词。例如，我们可以纠正语法或拼写错误，比如 *Their are two midterms*（其中 *There* 被错误拼写为 *Their*）或者 *Everything has improve*（其中 *improve* 应该是 *improved*）。因为 *There are* 的概率比 *Their are* 更高， *has improved* 比 *has improve* 的概率更高，所以语言模型可以帮助用户选择更符合语法的变体。或者对于语音系统来说，它能够识别你说的是 *I will be back soonish* ，而不是 *I will be bassoon dish* ，因为它知道 *back soonish* 是一个更可能的词序。语言模型还可以帮助**增强和替代性沟通（augmentative and alternative communication）**（Trnka等人，2007；Kane等人，2017）。如果人们因为身体原因无法说话或使用手语，他们可以使用**AAC**系统，通过眼睛注视或其他动作从菜单中选择单词，此时词预测就可以用于为菜单建议更可能的单词。

自然语言处理（NLP）中词预测重要性还因为：**大语言模型（large language models）**的构建仅仅是通过训练它们预测单词来实现的！！正如我们将在第7-9章中看到的那样，大语言模型仅仅通过从邻近词中训练预测即将出现的词，就能学到大量的语言知识。

在本章中，我们将介绍最简单的语言模型：n元语法（n-gram）模型。n元语法即n个单词序列：一个2-gram（我们称之为 bigram ）是像 *The water* 或 *water of* 这样的两个单词序列，3-gram（ trigram ）是像 *The water of* 或 *water of Walden* 这样的三个单词序列。但我们也在术语上有些含混模

糊地使用 n-gram 一词来指一种可以根据前n-1个单词估计一个单词概率的概率模型，从而为整个序列分配概率。

在后面的章节中，我们将介绍基于变换器（transformer）架构的更强大的神经大型语言模型（large language models）。但由于n元语法有着非常简单且清晰的形式化表达，因此我们用它们来介绍一些大型语言建模中的重要概念，包括训练集和测试集、困惑度、采样和插值。

3.1 N-Grams

我们从计算 P(w|h) 开始，P(w|h) 表示在某个历史 h 发生后出现单词 w 的概率。假设历史 h 是 *The water of Walden Pond is so beautifully*，我们想知道下一个词是 *blue* 的概率：

P(blue|The water of Walden Pond is so beautifully)

估计这个概率的一种方法是直接从相对频率计数中得出：在一个非常大的语料库中，统计我们看到 *The water of Walden Pond is so beautifully* 的次数，以及它后面跟随 *blue* 的次数。这实际上是在回答这个问题：“在我们看到历史 h 的时候，有多少次是跟随了单词 w”，具体公式如下：

P(blue|The water of Walden Pond is so beautifully) =

C(The water of Walden Pond is so beautifully blue) / C(The water of Walden Pond is so beautifully)

如果我们有足够大的语料库，我们可以得出这两个计数，并根据公式 3.2 估计这个概率。但是，即使整个Web的语料库也不足以给我们提供完整句子计数的准确估计。这是因为语言具有**创造性**，新的句子不断被创造出来，我们无法指望从“所有句子”这样的大对象得到准确的计数。因此，我们需要更聪明的方法来估计给定历史 h 后出现单词 w 的概率，或者整个单词序列 W 的概率。

为了方便后续表达，我们先约定一些符号进行表示。在本章中我们将继续以“单词”（words）为单位，尽管在实际操作中，我们通常会在采用像 BPE tokens（字节对编码）的标记上计算语言模型。要表示某个随机变量 Xi 取值为 *the* 的概率即 P(Xi = “the”)，简写为 P(the)。我们将一个 包含 n 个单词的序列表示为 w1...wn 或 w1:n。因此，表达式 w1:n－1 表示字符串 w1, w2, ..., wn－1，等效符号表示为 $w_{<n}$。对于一个序列中每个单词取特定值时的联合概率 P(X1 = w1, X2 = w2, X3 = w3, ..., Xn = wn)，我们将使用 P(w1, w2, ..., wn)来简化表示。

那么，我们如何计算像 P(w1, w2, ..., wn) 这样序列的概率呢？我们可以使用概率的**链式法则（chain rule of probability）**来分解这个概率：

$$P(X1...Xn) = P(X1)P(X2|X1)P(X3|X1:2) ...$$

$$P(Xn|X1:n-1) = \prod_{i=1}^{n} P(Xk|X1:k-1)$$

将链式法则应用于单词，我们得到：

$$P(w1:n) = P(w1)P(w2|w1)P(w3|w1:2) ... P(wn|w1:n-1)$$

$$= \prod_{i=1}^{n} P(wk|w1:k-1) \quad (3.4)$$

链式法则展示了计算一个序列的联合概率与计算给定前面单词条件下某个单词的条件概率之间的联系。方程 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，使用链式法则似乎并没有真正帮助我们！我们仍然不知道如何精确计算给定一长串前面单词后一个单词的概率 P(wn|w1:n－1)。正如我们上面所说的，我们无法仅通过统计语料库中每个长字符串后出现的单词次数来估计，因为语言是创造性的，任何特定的上下文可能从未出现过！

3.1.1 马尔可夫假设

n-gram 模型的直觉是我们可以仅通过最后几个单词来近似历史，而不是计算一个单词在其整个历史下的概率。

例如，二元语法模型通过使用前一个单词的条件概率 P(w_n|w_{n-1}) 来近似给定所有前面单词的概率 P(w_n|w_{1:n-1})。换句话说，我们不是计算概率：

P(blue|The water of Walden Pond is so beautifully) (3.5)

而是用以下概率来近似：

P(blue|beautifully) (3.6)

当我们使用二元语法模型来预测下一个单词的条件概率时，我们实际上是在做如下近似：

P(w_n|w_{1:n-1}) ≈ P(w_n|w_{n-1}) (3.7)

假设一个单词的概率仅依赖于前一个单词被称为马尔可夫假设。马尔可夫模型是一类概率模型，其假设我们可以在不回顾太久的过去的情况下可以预测某个未来单位的概率。我们可以将二元语法（查看过去一个单词）推广到三元语法（查看过去两个单词），进而推广到 n 元语法（查看过去 n －1 个单词）。

让我们看一下 n 元语法在序列中下一个单词条件概率的近似一般公式。我们在这里使用 N 来表示 n 元语法的大小，因此 N = 2 表示二元语法，N = 3 表示三元语法。然后，我们将一个单词在其整个上下文中的概率近似为：

P(w_n|w_{1:n-1}) ≈ P(w_n|w_{n-N+1:n-1})(3.8)

根据对单个单词概率的二元假设，我们可以通过将方程 (3.7) 代入方程 (3.4) 来计算完整单词序列的概率：

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

3.1.2 如何估计二元语法或 n 元语法概率？

估计概率的一种常用方法称为**最大似然估计**（**Maximum Likelihood Estimation**，简称 **MLE**）。我们通过从语料库中获取计数来得到 n 元语法模型参数的 MLE 估计，并对这些计数进行**归一化**，使其落在 0 和 1 之间。对于概率模型，归一化意味着通过某个总计数进行除法，使得结果概率在 0 和 1 之间，并且总和为 1。

例如，为了计算给定前一个单词 w_{n-1} 的特定二元语法概率 $P(w_n|w_{n-1})$，我们计算二元语法的计数 $C(w_{n-1}w_n)$，并通过所有第一个单词 w_{n-1} 的 2-gram 总和进行归一化：

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

我们可以简化这个方程，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} 的单元语法计数（读者可以花一点时间去理解这一点）：

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

让我们用一个包含三句话的小语料库来进行示例。我们首先需要在每句话的开头添加一个特殊符号 `<s>`，以提供第一个单词的二元上下文。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>

以下是从该语料库中计算的一些二元概率：

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

对于 MLE n 元语法参数估计的一般情况：

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

方程 3.12（类似于方程 3.11）通过将特定序列的观察频率除以前缀的观察频率来估计 n 元语法概率。这个比率被称为**相对频率**。我们上面提到过，将相对频率用作估计概率的方法是最大似然估计（MLE）的一个例子。在 MLE 中，得到的参数集将最大化了给定模型 M 的出现或重现训练集 T 的可能性（即 $P(T|M)$）。例如，假设单词 *Chinese* 在一个百万字的语料库中出现了 400 次。那么，从另一篇如百万字的文本中随机选择一个单词是*Chinese*的概率是多少？它的 MLE 概率为 $\frac{400}{1000000}$ 或 0.0004。现在，0.0004 并不是在所有情况下*Chinese*出现概率的最佳估计；在某些其他语料库或上下文中，可能*Chinese*是一个非常不可能出现的词。但这是使*Chinese*这个词在百万字语料库中出现 400 次的最有可能的概率。我们将在第 3.6 节中介绍一些方法，稍微修改 MLE 估计，以获得更好的概率估计。

接下来，我们将使用来自现已停止维护的伯克利餐厅项目（Berkeley Restaurant Project）的小型但真实的语料库中的一些示例。该对话系统来自上世纪，回答关于加利福尼亚州伯克利市餐厅数据库的问题（Jurafsky 等，1994）。以下是一些用户查询的示例（文本标准化，转为小写并去掉标点符号）：

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

图 3.1 显示了从伯克利餐厅项目语句的一部分的二元计数。可以发现，大多数值为零。实际上，我们选择的样本词是相互关联的，在随机选择的八个单词上的矩阵将更加稀疏。

	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 在 Berkeley Restaurant Project 语料库的 9332 个句子中，Bigram 占了其中的 8 个单词（V = 1446 个）。零计数呈灰色。每个单元格显示行标签字后面的列标签字的计数。因此，行 i 和列 Want 中的单元格表示 Want 在语料库中跟随了 i 827 次。

图 3.2 显示了归一化后的二元概率（将图 3.1 中每个单元格除以其行对应的单元语法）：

	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 Berkeley Restaurant Project 语料库中 9332 个句子中 8 个单词的二元组概率。零概率呈灰色。

以下是一些其他有用的概率：

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

现在我们可以通过简单地将相应的二元概率相乘来计算句子的概率，比如 *I want English food* 或 *I want Chinese food*，如下所示：

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

这些二元统计中捕捉到了哪些语言现象？上面的某些二元概率编码了一些我们认为是严格**句法**性质的事实，例如，紧跟在*eat*之后的通常是名词或形容词的事实，或紧跟在*to*之后的通常是动词的事

实。其他的可能是关于个人助理任务的事实，例如以“我”开头的句子概率很高。而有些甚至可能是文化性的，而非语言性的，例如人们寻找中国食物的概率高于寻找英文食物的概率。

3.1.3 在大型 n 元语法模型中处理规模问题

在实际应用中，语言模型可能非常庞大，从而出现一些实际问题：

对数概率

语言模型的概率通常以对数空间存储和计算，称为对数概率。这是因为概率（根据定义）小于或等于 1，因此我们相乘的概率越多，乘积就会变得越小。如果将足够多的 n 元语法相乘，可能会导致数值下溢。在对数空间中相加等同于在线性空间中相乘，因此我们通过相加对数概率来组合它们。通过相加对数概率而不是直接相乘概率，我们得到的结果不会那么小。我们在对数空间中进行所有计算和存储，如果需要在最后报告概率时，只需通过对对数概率取指数（exp）来转换回概率：

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

在本书的实际应用中，当未指定基数时，我们将 log 视为自然对数 ln。

更长的上下文

虽然为了教学目的我们只描述了二元语法模型，但当训练数据足够时，我们会使用三元语法模型，即预测词条件依赖于前两个单词，或 4 元语法、5 元语法模型。对于这些更大的 n 元语法，我们需要假设句子开头左右有额外的上下文。例如，在句子开头计算三元概率时，我们为第一个三元语法使用两个伪单词（即 $P(I|<s><s>)$ ）。

现有研究已经创建了一些大型 n 元语法数据集，例如从当代美国英语语料库（Corpus of Contemporary American English，COCA）中提取的百万个最常见的 n 元语法，这个语料库包含 10 亿个单词（Davies, 2020）；谷歌的 5 元语法语料库来自 1 万亿字的英语网页文本（Franz 和 Brants, 2006）；谷歌图书 n 元语法语料库（包含来自中文、英文、法文、德文、希伯来文、意大利文、俄文和西班牙文的 8000 亿个标记）(Lin 等, 2012)。

我们甚至可以使用极长范围的 n 元语法上下文。infini-gram（∞-gram）项目（Liu 等, 2024）允许任意长度的 n 元语法。他们的想法是避免预先计算巨大的 n 元语法计数表（在空间和时间上的代价昂贵），而使用一种称为后缀数组的高效表示法在推理时快速计算任意 n 的 n 元语法概率。这使得在包含 5 万亿个标记的庞大语料库中计算每种长度的 n 元语法成为可能。

在构建大型 n 元语法语言模型时，效率考虑非常重要。通常标准是使用 4-8 位量化概率（而不是 8 字节浮点数），将单词字符串存储在磁盘上，仅在内存中表示为 64 位哈希，并使用特殊数据结构（如“反向字典树”）表示 n 元语法。通常还会对 n 元语法语言模型进行修剪，例如只保留计数大于某个阈值的 n 元语法，或使用熵来修剪不太重要的 n 元语法（Stolcke, 1998）。高效的语言模型工具包，如 KenLM（Heafield 2011，Heafield 等 2013），使用排序数组，并使用归并排序高效地在通过大语料库的最小遍数中构建概率表。

3.2 评估语言模型：训练集和测试集

评估语言模型性能的最佳方法是将其嵌入到一个应用中，并测量该应用表现性能。这种端到端的评估称为**外部评估**。外部评估是判断语言模型（或任何组件）特定改进是否真的有助于当前任务的唯

一方法。因此，对于作为语音识别或机器翻译某个任务组件的 n 元语法语言模型，我们可以通过运行语音识别器或机器翻译器两次（每次使用一个语言模型），比较两个候选语言模型的性能，看看哪个能提供更准确的转录。

不幸的是，运行大型自然语言处理系统的端到端评估通常非常昂贵。因此，拥有一个可以快速评估语言模型潜在改进的指标是有帮助的。**内部评估**指标是指在不依赖于任何应用的情况下衡量模型质量的指标。在下一节中，我们将介绍**困惑度（perplexity）**，这是衡量语言模型性能的标准内部指标，适用于简单的 n 元语法语言模型和第 9 章中更复杂的神经大型语言模型。

为了评估任何机器学习模型，我们需要至少三个不同的数据集：训练集、开发集和测试集。

训练集是我们用来学习模型参数的数据；对于简单的 n 元语法语言模型，它是我们获取计数并将其归一化为 n 元语法语言模型概率的语料库。

测试集是一个不同的、被保留的数据集，与训练集不重叠，我们用它来评估模型。我们需要一个单独的测试集，以便提供一个无偏的估计，了解我们训练的模型在应用到一些新的未知数据集时的泛化能力。一个完美捕捉训练数据但在任何其他数据上表现糟糕的机器学习模型，在应用到任何新数据或问题时就毫无用处！因此，我们通过模型在这个未见的测试集或测试语料库上的表现来衡量 n 元语法模型的质量。

我们应该如何选择训练集和测试集？测试集应该反映我们希望模型用于的语言。如果我们要将语言模型用于化学讲座的语音识别，测试集应该是化学讲座的文本。如果我们要将其作为将酒店预订请求从中文翻译成英文的系统的一部分，测试集应为酒店预订请求的文本。如果我们希望我们的语言模型是通用的，那么测试集应从各种文本中抽取。在这种情况下，我们可能会从不同来源收集大量文本，然后将其分为训练集和测试集。仔细划分是很重要的；如果我们正在构建一个通用模型，我们不希望测试集仅由来自一篇文档或一个作者的文本组成，因为这不会很好地衡量一般性能。

因此，如果我们获得一组文本语料库，并希望比较两个不同的 n 元语法模型的性能，我们将数据划分为训练集和测试集，并在训练集上训练两个模型的参数。然后我们可以比较这两个训练模型在测试集上的适应性。

但“适应测试集”意味着什么呢？标准的答案很简单：哪个语言模型给测试集分配的概率更高——这意味着它更准确地预测测试集——哪个模型就更好。给定两个概率模型，表现更好的模型是那个更好地预测测试数据细节的模型，因此将给测试数据分配更高的概率。

由于我们的评估指标基于测试集概率，因此重要的是不要让测试句子混入训练集。假设我们试图计算特定“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，当它出现在测试集时，我们会错误地给它分配一个人造的高概率。我们称这种情况为在测试集上训练。对测试集的训练引入了一种偏差，使得概率看起来都太高，并导致困惑度这一基于概率的指标出现巨大的不准确性。

即使我们不在测试集上训练，如果我们在进行不同更改后多次在测试集上测试我们的语言模型，我们可能会隐式地调整其特征，通过观察哪些更改似乎使模型更好。因此，我们只希望在确定模型准备好后，在测试集上运行模型一次或非常少的几次。

出于这个原因，我们通常会有一个称为**开发测试集（development test set or devset）** 的第三数据集。我们将在这个数据集上进行所有测试，直到最后，然后在测试集上测试一次以查看模型的表现。

我们如何将数据划分为训练集、开发集和测试集？我们希望测试集尽可能大，因为小的测试集可能偶然而不具代表性，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，以便提供足够的统计能力，以衡量两个潜在模型之间的统计显著性差异。重要的是，devset 应从与测试集相同类型的文本中抽取，因为它的目标是衡量我们在测试集上的表现。

3.3 评估语言模型：困惑度

我们之前提到，我们评估语言模型的标准是哪个模型对测试集分配的概率更高。一个更好的模型在预测即将出现的单词时表现更好，因此在测试集中每个单词出现时，它会对其分配更高的概率，表现得不那么惊讶。实际上，一个完美的语言模型会正确地猜测语料库中的每个下一个单词，为其分配概率 1，而为所有其他单词分配概率 0。因此，给定一个测试语料库，一个更好的语言模型将为其分配比较差的语言模型更高的概率。

然而，实际上我们并不使用原始概率作为评估语言模型的指标。原因在于，测试集（或任何序列）的概率依赖于其中的单词或标记数量；测试集的概率随着文本长度的增加而变小。我们更希望有一个基于每个单词的、并对长度归一化的指标，这样我们可以在不同长度的文本之间进行比较。我们使用的指标是**困惑度（perplexity）**，它是一个基于概率的函数，是自然语言处理（NLP）中最重要的指标之一，适用于评估大型语言模型以及 n 元语法模型。

语言模型在测试集上的困惑度（有时缩写为 PP 或 PPL）是测试集的逆概率（测试集概率的倒数），并按单词（或标记）数量进行归一化。因此，有时它被称为每个单词或每个标记的困惑度。我们通过取 N 次根来按单词数量 N 进行归一化。对于测试集 $W = w_1 w_2 \dots w_N$：

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

或者我们可以使用链式法则扩展 W 的概率：

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

注意，由于方程 (3.15) 中的逆关系，**单词序列的概率越高，困惑度就越低**。因此，模型在数据上的困惑度越低，模型越好。最小化困惑度等同于最大化根据语言模型的测试集概率。为什么困惑度使用逆概率？这实际上源自信息论中困惑度的原始定义与交叉熵率；对此感兴趣的人可以在第 3.7 节的高级部分找到解释。同时，我们只需记住，困惑度与概率之间存在反向关系。

计算测试集 W 的困惑度的细节取决于我们使用的语言模型。使用一元法法模型计算的 W 的困惑度：

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i)}}$$

使用二元语法模型计算的W 的困惑度仍然是几何平均，但现在是二元概率逆数的几何平均。

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

我们通常在方程 (3.15) 或 (3.17) 中使用的单词序列是某个测试集中的整个单词序列。由于该序列将跨越多个句子边界，如果我们的词汇表包含句子间标记 `<EOS>` 或分别作为开始和结束标记的 `<s>` 和 `</s>`，那么我们可以将它们包含在概率计算中。如果这样做，我们也会在单词标记的总计数 N 中包括每个句子的**一个**标记 `</s>`。

我们上面提到过，困惑度是文本和语言模型的函数：给定文本 W，不同的语言模型将具有不同的困惑度。因此，困惑度可以用来比较不同的语言模型。例如，我们在《华尔街日报》（Wall Street Journal，简称WJS）报纸的 3800 万字数据上训练了单元、二元和三元语法。然后，我们使用方程 (3.16) 计算了这些模型在 WSJ 测试集上的困惑度，一元语法模型使用方程 (3.16)，二元语法模型使用方程 (3.17)，三元使用相应的方程。下表显示了根据每个语言模型计算的 150 万字测试集的困惑度。

	Unigram	Bigram	Trigram
Perplexity	962	170	109

正如我们所看到的，n 元语法对单词序列提供的信息越多，n 元语法为字符串分配的概率就越高。三元模型比单元模型更不惊讶，因为它更清楚接下来可能出现什么单词，因此会为它们分配更高的概率。而且，概率越高，困惑度越低（因为方程 (3.15) 显示，困惑度与模型根据测试序列的概率成反比）。因此，较低的困惑度表明语言模型对测试集的预测能力更强。

注意，在计算困惑度时，语言模型必须在不知道测试集的情况下构建，否则困惑度将人为地偏低。而且，**只有当两个语言模型使用相同的词汇表时，它们的困惑度才可比较**。

基于困惑度的（内部）改进并不保证语言处理任务（如语音识别或机器翻译）的（外部）性能改进。尽管如此，由于困惑度通常与任务改进相关联，因此它常被用作一种方便的评估指标。尽管如此，当有可能时，模型在困惑度上的改进应通过在实际任务上的端到端评估进行确认。

3.3.1 作为加权平均分支因子的困惑度

困惑度实际上也可以被视为语言的加权平均分支因子（**weighted average branching factor**）。语言的分支因子是指在任何单词之后可以跟随的可能下一个单词的数量。例如，考虑一个小型的人工语言，该语言是确定性的（没有概率），任何单词可以跟随任何单词，且其词汇仅由三种颜色组成：

$$L = \{\text{red}, \text{blue}, \text{green}\} \quad (3.18)$$

那么这个语言的分支因子是 3。

现在让我们创建一个相同语言模型的概率版本，称之为 A，每个单词之间以相等的概率 $\frac{1}{3}$ 连接（可以视为它是在三个颜色计数相等的训练集上训练的），测试集为 T = “red red red red blue”。

首先，我们来确认如果我们计算这个人工数字语言在该测试集（或任何类似测试集）上的困惑度，我们确实得到了 3。根据方程 (3.15)，A 在 T 上的困惑度为：

$$\begin{aligned}\text{perplexity}_A(T) &= P_A(\texttt{red red red red blue})^{-\frac{1}{5}} \\ &= \left(\left(\frac{1}{3}\right)^5\right)^{-\frac{1}{5}} \\ &= \left(\frac{1}{3}\right)^{-1} = 3\end{aligned}$$

但现在假设在训练集中红色的概率非常高，语言模型 B 的概率如下：

$$P(\text{red}) = 0.8 \quad P(\text{green}) = 0.1 \quad P(\text{blue}) = 0.1 \quad (3.20)$$

我们应该期望语言模型 B 在相同测试集 *red red red red blue* 上的困惑度较低，因为大多数情况下下一个颜色将是红色，这是非常可预测的，即具有很高的概率。因此，测试集的概率将更高，而困惑度与概率成反比，因此困惑度会更低。因此，尽管分支因子仍然是 3，但困惑度或加权分支因子更小：

$$\begin{aligned}\text{perplexity}_B(T) &= P_B(\texttt{red red red red blue})^{-1/5} \\ &= 0.04096^{-\frac{1}{5}} \\ &= 0.527^{-1} = 1.89\end{aligned}$$

3.4 从语言模型中采样句子

可视化语言模型所蕴含的知识的一个重要方法是从语言模型中进行**采样**。从分布中采样意味着根据其可能性选择随机点。因此，从语言模型中采样——它代表了句子的分布——意味着生成一些句子，每个句子的生成都是根据模型定义的概率选择的。因此，我们更有可能生成模型认为具有高概率的句子，而不太可能生成模型认为具有低概率的句子。

这种通过采样可视化语言模型的技术最早是由香农（Shannon, 1948）和米勒与塞尔弗里奇（Miller and Selfridge, 1950）提出的。对于unigram的情况，这种工作方式最容易可视化。想象一下，英语语言中的所有单词覆盖在 0 到 1 之间的数轴上，每个单词覆盖一个与其频率成比例的区间。图 3.3 显示了一个可视化，使用从本书文本计算出的unigram语言模型。我们选择一个介于 0 和 1 之间的随机值，找到概率线上该点，并打印出包含此选定值的单词。我们继续选择随机数并生成单词，直到随机生成句子结束标记 </s>。

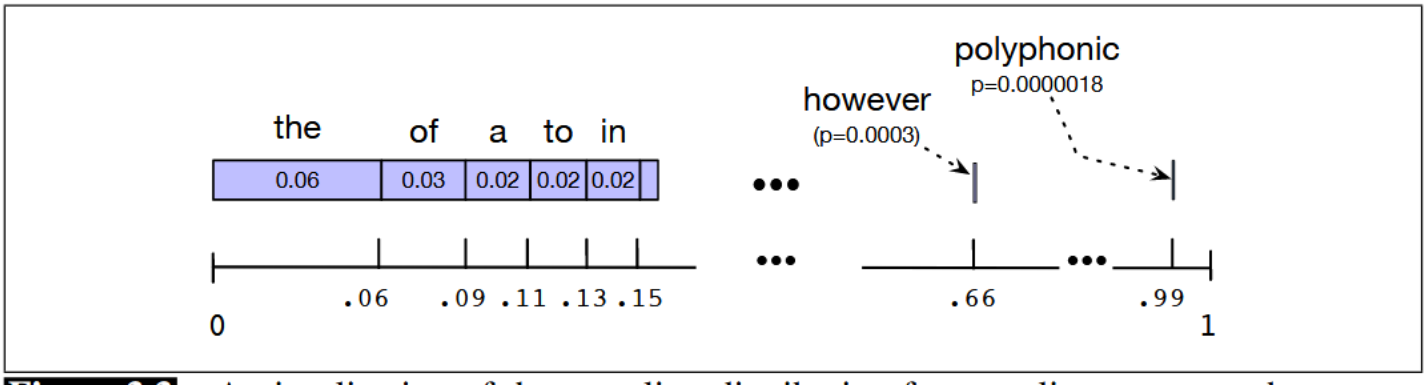

图 3.3 通过重复采样一元词来对句子采样的采样分布的可视化。蓝色条代表每个单词的相对频率（我们将它们从最频繁到最不频繁排序，但顺序的选择是任意的）。数轴显示累积概率。如果我们选择一个0到1之间的随机数，它就会落在某个单词对应的区间内。对随机数落在频繁单词之一（the、of、a）的较大间隔中的期望远高于罕见单词之一（polyphonic）的较小间隔中的期望。

我们可以使用相同的技术生成二元语法，首先生成一个以 <s> 开头的随机二元语法（根据其二元概率）。假设该二元语法的第二个单词是 *w*。接下来，我们选择一个以 *w* 开头的随机二元语法（同样根据其二元概率），依此类推。

3.5 泛化与训练集过拟合

n 元语法模型与许多统计模型一样，依赖于训练语料库。这意味着概率中往往编码了特定训练语料库中的事实。另一个含义是，随着 N 值的增加，n 元语法在建模训练语料库方面的表现会越来越好。

我们可以使用前一节中的采样方法来可视化这两个结论！为了直观了解高阶 n 元语法的增强能力，图 3.4 展示了从基于莎士比亚作品训练的单元、二元、三元和 4 元语法模型生成的随机句子。

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 3.4 根据莎士比亚作品计算出的 4 个 n 元模型随机生成 8 个句子。所有字符都映射为小写，标点符号被视为单词。输出的大小写经过手工更正，以提高可读性。

可以看到，上下文越长，句子就越连贯。单元模型句子之间没有连贯关系，也没有句尾标点；二元模型句子在单词之间有一些局部的连贯性（尤其是将标点视为单词时）；三元模型句子开始看起来更像莎士比亚的风格；而4 元句子看起来有点过于像莎士比亚。其中的词 *It cannot be but so* 直接来自《约翰王》。这并不是说莎士比亚的作品不好，他的作品在语料库中并不算大(N = 884,647, V = 29,066 ），我们的 n 元语法概率矩阵稀疏得可笑。仅二元语法就有 $V^2 = 844,000,000$ 种可能，而可能的 4 元语法数量是 $V^4 = 7 \times 10^{17}$ 。因此，一旦生成器选择了前 3 元（ *It cannot be* ），对于第 4 个元素，只有七个可能的下一个词（ *but* ， *I* ， *that* ， *thus* ， *this* ，和句号）。

为了了解对训练集的依赖性，让我们来看一下基于完全不同语料库——华尔街日报（WSJ）训练的语言模型。莎士比亚和华尔街日报都是英语，因此我们可能会期待这两个类型的 n 元语法之间存在某种重叠。图 3.5 显示了从 4000 万字的华尔街日报文本中训练的单元、二元和三元语法生成的句子。

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 3.5 根据《华尔街日报》的 4000 万个单词计算出的三个 n-gram 模型随机生成三个句子，所有字符均小写，并将标点符号视为单词。然后对输出的大小写进行手工更正，以提高可读性。

将这些例子与图 3.4 中的伪莎士比亚进行比较。虽然它们都建模“类似英语的句子”，但生成的句子之间没有重叠，甚至在小短语中也几乎没有重叠。如果训练集和测试集之间的差异像莎士比亚与华尔街日报那么大，统计模型作为预测工具几乎没有用处。

在构建 n 元语法模型时，我们应该如何解决这个问题？一个步骤是确保使用的训练语料库与我们试图完成的任务具有相似的类型。为了构建一个用于翻译法律文件的语言模型，我们需要一个法律文件的训练语料库。为了构建一个用于问答系统的语言模型，我们需要一个问答的训练语料库。

获取适当方言或变体的训练数据同样重要，尤其是在处理社交媒体帖子或口语转录时。例如，一些推文会使用非裔美国英语（AAE）的特征——这是在非裔美国人社区中使用的多种语言变体的名称（King, 2020）。这样的特征可能包括 finna 这个词——一个标志着即将发生的未来时态的助动词——它在其他变体中并不存在，或拼写如 den 代替 then ，例如这条推文（Blodgett 和 O’Connor, 2017）：

(3.22) *Bored af den my phone finna die!!!*

而来自像尼日利亚克里奥尔语这样的英语基础语言的推文在词汇和 n 元语法模式上与美式英语有明显不同（Jurgens 等, 2017）：

(3.23) *@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*

另外，测试集是否可能包含一个我们从未见过的单词？如果单词 *Jurafsky* 在我们的训练集中从未出现，但在测试集中出现，会发生什么？答案是，虽然某些单词可能是未见的，但实际上我们的自然语言处理算法是针对**子词标记**（**subword tokens**）运行的。通过子词标记化（如第 2 章中的 BPE 算法），任何单词都可以建模为已知小的子词序列，必要时甚至可以是单个字母的序列。因此，尽管在本章中为了方便我们称之为单词，但语言模型的词汇实际上是标记的集合，而测试集不可能包含未见的标记。

3.6 平滑（Smoothing）、插值（Interpolation）与回退（Backoff）

使用最大似然估计来计算概率存在一个问题：任何有限的训练语料库都可能缺少一些完全可接受（可能出现）的英语词序列。也就是说，某些特定的 n 元语法在训练数据中从未出现，但在测试集中出现。例如，我们的训练语料库中可能有 *ruby* 和 *slippers* 这两个词，但恰好没有短语 *ruby slippers* 。

这些未见的序列或零——在训练集中不存在但在测试集中出现的序列——对我们来说是一个问题，原因有两个。首先，它们的存在意味着我们低估了可能出现的词序列的概率，这会影响我们希望在这些数据上运行的任何应用的性能。其次，如果测试集中任何单词的概率为 0，那么整个测试集的概率也是 0。困惑度是基于测试集的逆概率定义的。因此，如果某些上下文中的单词具有零概率，我们根本无法计算困惑度，因为我们无法除以 0！

处理这些假定的“零概率 n 元语法”的标准方法称为**平滑**（**smoothing**）或**折扣**（**discounting**）。平滑算法会从一些更频繁的事件中削减一些概率质量，并将其分配给未见事件。在这里，我们将介绍一些简单的平滑算法：**拉普拉斯加一平滑**（**Laplace (add-one) smoothing**）、**愚蠢回退**（**stupid backoff**）和 n 元语法**插值**（**interpolation**）。

3.6.1 拉普拉斯平滑

最简单的平滑方法是在所有 n 元语法计数上加一，然后将它们归一化为概率。以前为零的计数现在将变为 1，计数为 1 的将变为 2，以此类推。这个算法称为**拉普拉斯平滑**（**Laplace smoothing**）。尽管拉普拉斯平滑在现代 n 元语法模型中表现不够理想，但它有效地引入了我们在其他平滑算法中看到的许多概念，提供了有用的基线，并且对于其他任务（如文本分类，第 4 章）也是一种实用的平滑算法。

首先让我们讨论拉普拉斯平滑在单元语法概率中的应用。回想一下，未平滑的最大似然 估计给定单词 w_i 的单元语法概率是其计数 c_i 除以总的单词标记数量 N 。

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

拉普拉斯平滑只是对每个计数加一（因此它的另一个名称是**加一**(**add-one**)平滑）。由于词汇表中有 V 个单词，并且每个单词频数增加，因此我们还需要调整分母，以考虑额外的 V 次观察。（想想如果不增加分母，我们的 P) 值会发生什么？）

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

为了方便描述平滑算法对分子的影响，而不是同时改变分子和分母，我们定义一个调整后的计数 c^* 。这个调整后的计数更容易直接与 MLE 计数进行比较，并可以通过除以 N 归一化为概率。由于我们通过加一改变了分子，那么就需要乘以一个归一化因子 $\frac{N}{N+V}$ 。

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

现在我们可以通过除以 N 将 c_i^* 进行归一化 转换为概率 P_i^* 。

另一种看待平滑的方法是将一些非零计数进行折扣（降低），以便将概率质量分配给零计数。因此，我们可以用相对折扣 d_i 来描述平滑算法，即折扣计数与原始计数的比率，而不是直接使用折扣计数 c^* 。

$$d_i = \frac{c_i^*}{c_i}$$

现在我们对单元语法的情况有了直观理解，让我们对伯克利餐厅项目的二元语法进行平滑。图 3.6 显示了图 3.1 中二元语法的加一平滑计数。

	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 3.6 加一平滑二元组可计算 Berkeley Restaurant Project 语料库中 9332 个句子中的 8 个单词（V = 1446）。以前的零计数呈灰色。

图 3.7 显示了图 3.2 中二元语法的加一平滑概率。请记住，正常的二元概率是通过归一化每一行的单元的计数计算的：

$$P_{\text{MLE}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

对于加一平滑的二元计数，我们需要在单元计数的基础上加上词汇表中总的单词类型数量 V ：

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

因此，上一节中给出的每个单元计数都需要增加 $V = 1446$ ，最终的平滑二元概率如图 3.7 。

	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 3.7 对包含 9332 个句子的 BeRP 语料库中的 8 个单词（V = 1446）进行加一平滑二元组概率。以前的零概率呈灰色。

	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 3.8 在包含 9332 个句子的 BeRP 语料库中，Add-1 重构了 8 个单词（V = 1446）的计数。之前的零计数呈灰色。

为了看到平滑算法对原始计数的改变程度，我们可以重建计数矩阵。这些调整后的计数可以通过下面的方程计算。图 3.8 显示了重建的计数。

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

注意，加一平滑对计数进行了非常大的改变。将图 3.8 与图 3.1 中的原始计数进行比较，我们可以看到 $C(\text{want to})$ 从 608 变为 238！在概率空间中，我们也可以看到这一点：在未平滑情况下 $P(to|\text{want})$ 从 0.66 降至平滑情况下的 0.26。查看折扣 d （新旧计数之间的比率）显示了每个前缀词的计数减少得h很明显；二元语法 *want to* 的折扣为 0.39，而 *Chinese food* 的折扣为 0.10，差异达 10 倍！这一剧烈变化发生的原因是太多的概率质量被转移到了所有的零上。

3.6.2 加 k 平滑

加一平滑的一个可能的改进是转移较少的概率质量。与其对每个计数加 1，不如加一个小于 1的分数计数 k （例如 0.5？0.01？）。因此，这个算法被称为加 k 平滑。

加 k 平滑需要我们有一种选择 k 的方法；这可以通过在开发集（devset）上进行优化来实现。尽管加 k 平滑对于某些任务（包括文本分类）很有用，但事实证明，它在语言建模中表现仍不理想，生成的计数方差较差，折扣也常常不合适（Gale 和 Church，1994）。

3.6.3 语言模型插值

我们可以利用另一种知识来源来解决零频率 n 元语法的问题。如果我们想计算 $P(w_n|w_{n-2}w_{n-1})$ ，但没有特定三元语法 $w_{n-2}w_{n-1}w_n$ 的样本，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，如果我们没有足够的数据来计算 $P(w_n|w_{n-1})$ ，我们可以参考单元语法概率 $P(w_n)$ 。换句话说，有时使用**较少的上下文**可以帮助我们对模型未学到的上下文进行更好的泛化。

使用这种 n 元语法层次结构的最常见方法称为 **插值**：通过插值（加权后相加）三元语法、二元语法和单元语法概率来计算一个新的概率。在简单的线性插值中，我们通过线性插值来组合不同阶的 n 元语法。因此，我们通过混合单元语法、二元语法和三元语法概率，每个概率乘以一个 λ 权重，来估计三元语法概率 $P(w_n|w_{n-2}w_{n-1})$：

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

这些 λ 的和必须为 1，使得方程 3.30 等同于加权平均。在稍微复杂一些的线性插值版本中，每个 λ 权重是根据上下文进行动态计算的。这样，如果我们对于某个特定的二元语法有非常准确的计数，我们可以假设基于该二元语法的三元语法计数更加可信，因此我们可以使该三元语法的 λ 更高，从而在插值中赋予该三元语法更大的权重。方程 3.31 显示了基于上下文条件权重的插值公式，其中每个 λ 的参数是前两个单词的上下文：

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

这些 λ 值是如何确定的呢？无论是简单插值还是条件插值，λ 值都是从一个**保留**(**held-out**)语料库中学习的。所谓的保留语料库是指我们从训练数据中保留的一部分数据，用来设置这些 λ 值。我们通过选择能够最大化保留语料库似然的 λ 值来实现这一点。也就是说，我们固定 n 元语法的概率，然后寻找能够在方程 3.30 中插入并使保留集概率最高的 λ 值。有多种方法可以找到这组最优的 λ 值。一种方法是使用 EM 算法，这是一种迭代学习算法，可以收敛到局部最优的 λ 值（Jelinek 和 Mercer, 1980）。

3.6.4 愚蠢回退

回退（backoff）是一种替代插值的方法。在回退模型中，如果我们需要的 n 元语法计数为零，我们会通过回退到 (n-1)元语法来近似该 n 元语法。如果仍然为0，我们会继续回退，直到找到有计数的历史。为了使回退模型提供正确的概率分布，我们必须对高阶 n 元语法进行折扣，以为低阶 n 元语法留出一些概率质量。然而，在实践中，常用的不是折扣，而是一种更简单的非折扣回退算法，称为**愚蠢回退**（**stupid backoff**，Brants 等人，2007）。

愚蠢回退放弃了让语言模型成为真正概率分布的想法。高阶 n 元语法的概率没有折扣。如果某个高阶 n 元语法的计数为零，我们仅仅回退到低阶 n 元语法，并乘以一个固定的（与上下文无关的）权重。这个算法不会产生一个概率分布，因此我们按照 Brants 等人（2007）的做法，将其称为 S：

$$S(w_i|w_{i-N+1:i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})} & \text{if count}(w_{i-N+1:i}) > 0 \\ \lambda S(w_i|w_{i-N+2:i-1}) & \text{otherwise} \end{cases}$$

回退最终停在单元语法，其得分为 $S(w) = \dfrac{\text{count}(w)}{N}$。Brants 等人（2007）发现一个值 0.4 的 λ 表现良好。

3.7 困惑度与熵的关系

我们在第 3.3 节中介绍了困惑度，其是评估 n 元语法模型在测试集上表现的一种方法。一个更好的 n 元语法模型会给测试数据分配更高的概率，困惑度是测试集概率的归一化版本。事实上，困惑度来源于信息论中的 **交叉熵**（**cross-entropy**） 概念，这也解释了困惑度的一些看似神秘的特性（例如，为什么使用逆概率？）以及困惑度与熵的关系。**熵（Entropy）**是衡量信息量的一种方式。给定一个随机变量 X (它的取值范围覆盖了我们预测的内容（单词、字母、词性等）)，我们称这些取值的集合为 χ ,并使用一个特定的概率函数 $p(x)$ 表示，随机变量 X 的熵定义如下：

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

对数 $\log$ 可以在任何底数下计算。如果我们使用底数 2，则熵的值将以 **比特(bits)** 为单位。

一种直观理解熵的方式是:在最优编码方案下，编码某个决策或信息所需的最少比特数。考虑一个来自经典信息论教材 Cover 和 Thomas（1991）的例子。假设我们想为一场赛马下注，但去 Yonkers 赛马场路途遥远，因此我们希望向博彩公司发送一个简短的信息，告诉他要押哪匹马。我们可以使用马匹编号的二进制表示作为编码，例如，马 1 编码为 001，马 2 编码为 010，马 3 编码为 011，依此类推，马 8 编码为 000。如果我们整天都在下注，每匹马都用 3 个比特编码，平均下来，我们每场比赛会发送 3 个比特。

我们还有更好的选择吗？比如能否采用变长的编码数？

假设投注的分布反映了每匹马的先验概率，如下：

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

随机变量 X 覆盖了马匹的取值范围，它的熵给出了编码每场比赛所需比特数的下限（即表示一匹马最少需要的）：

$$\begin{aligned} H(X) &= -\sum_{i=1}^{i=8} p(i) \log_2 p(i) \\ &= -\tfrac{1}{2}\log_2\tfrac{1}{2} - \tfrac{1}{4}\log_2\tfrac{1}{4} - \tfrac{1}{8}\log_2\tfrac{1}{8} - \tfrac{1}{16}\log_2\tfrac{1}{16} - 4(\tfrac{1}{64}\log_2\tfrac{1}{64}) \\ &= 2 \text{ bits} \end{aligned}$$

我们可以通过为更有可能的马匹使用较短的编码，为不太可能的马匹使用较长的编码，来设计一个平均每场比赛只需 2 个比特的编码方案。例如，我们可以用代码 0 编码最有可能的马匹，其他马匹依次编码为 10、110、1110、111100、111101、111110 和 111111。

如果马匹的概率是均等的呢？我们之前看到，若为每匹马使用等长的二进制编码，则每匹马需要 3 个比特，因此平均为 3。这时熵是相同的吗？在这种情况下，每匹马的概率为 $\frac{1}{8}$ 。那么马匹选择的熵

为：

$$H(X) = -\sum_{i=1}^{i=8} \frac{1}{8} \log_2 \frac{1}{8} = -\log_2 \frac{1}{8} = 3 \text{ bits}$$

到目前为止，我们一直在计算单个变量的熵。但大多数情况下，我们需要使用熵来处理序列。例如，对于一个语法，我们会计算某个单词序列 $W = \{w_1, w_2, \ldots, w_n\}$ 的熵。实现此目的的一种方法是使用一个范围涵盖单词序列的变量。例如，我们可以计算某个语言 L 中长度为 n 的所有单词序列的随机变量的熵，如下所示：

$$H(w_1, w_2, \ldots, w_n) = -\sum_{w_{1:n} \in L} p(w_{1:n}) \log p(w_{1:n})$$

我们可以定义熵率（或每个单词的熵）为该序列的熵除以单词数量。

$$\frac{1}{n} H(w_{1:n}) = -\frac{1}{n} \sum_{w_{1:n} \in L} p(w_{1:n}) \log p(w_{1:n})$$

然而，要衡量语言的真正熵，我们需要考虑无限长度的序列。如果我们将语言视为一个生成单词序列的 随机过程 L，并用 W 表示单词序列 $w_1, \ldots, w_n$，那么 L 的熵率 H(L) 定义如下：

$$\begin{aligned} H(L) &= \lim_{n \to \infty} \frac{1}{n} H(w_{1:n}) \\ &= -\lim_{n \to \infty} \frac{1}{n} \sum_{W \in L} p(w_{1:n}) \log p(w_{1:n}) \end{aligned}$$

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律的（确切地说，如果它是平稳且遍历的），

$$H(L) = \lim_{n \to \infty} -\frac{1}{n} \log p(w_{1:n})$$

那么我们可以取一个足够长的序列，而不需要对所有可能的序列求和。Shannon-McMillan-Breiman 定理的直觉认为，一个足够长的单词序列将包含许多较短的序列，并且这些较短的序列将根据它们的概率在较长的序列中重复出现。

一个 **平稳** （**stationary**）的随机过程是指其分配给序列的概率相对于时间索引的变化而不变。换句话说，时间 t 时单词的概率分布与时间 t + 1 时的概率分布相同。马尔可夫模型和 n 元语法都是平稳的。例如，在二元语法中， P_i 仅依赖于 P_{i-1}。因此，如果我们将时间索引平移 x ， P_{i+x} 仍然依赖于 P_{i+x-1}。但实际中，自然语言不是平稳的，因为即将出现的单词的概率可能依赖于任意远的、与时间有关的事件（见附录 D）。因此，我们的统计模型只能近似自然语言的正确分布和熵。

总结来说，通过做出一些不正确但方便的简化假设，我们可以通过获取很长的输出样本并计算其平均对数概率来计算某些随机过程的熵。

现在我们准备引入 交叉熵 。当我们不知道生成某些数据的实际概率分布 p 时，交叉熵是有用的。它允许我们使用某个模型 m 作为 p 的模型（即 p 的近似）。模型 m 对 p 的交叉熵定义如下：

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

也就是说，我们根据分布 p 抽取序列，但根据 m 的概率来求和它们的对数。

同样根据 Shannon-McMillan-Breiman 定理，对于一个平稳遍历过程，我们可以通过取一个足够长的序列来估计模型 m 在分布 p 上的交叉熵，而不需要对所有可能的序列求和。

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

交叉熵之所以有用，是因为它为熵 H(p)提供了一个上界。对于任意一个模型 m：

$$H(p) \le H(p,m)$$

这意味着我们可以使用某个简化模型m 来帮助估计根据分布p 抽取符号序列的真实熵。模型 m越准确，交叉熵 H(p, m) 就越接近真实熵 H(p) 。因此， H(p, m)和 H(p)之间的差异是模型准确性的衡量标准。在两个模型 m_1 和 m_2 间，更准确的模型将是交叉熵较低的那个（交叉熵永远不会低于真实熵，因此模型不能通过低估真实熵而出错）。

最后，我们来看看困惑度与交叉熵之间的关系，如我们在方程 3.41 中看到的。交叉熵定义为观察到的单词序列长度趋于无穷时的极限。我们通过依赖一个（足够长的）固定长度的序列来近似这个交叉熵。模型 $M = P(w_i|w_{i-N+1:i-1})$ 在单词序列 W 上的交叉熵近似如下：

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

模型P在单词序列W上的困惑现在被正式定义为2的交叉熵的次方：

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

3.8 总结

本章通过 n 元语法模型介绍了语言建模，这是一种经典模型，帮助我们引入了许多语言建模中的基本概念。

- 语言模型提供了一种方法，可以为一个句子或其他词序列、标记序列分配一个概率，并根据前面的词或标记预测下一个词或标记。
- **n 元语法**可能是最简单的语言模型。它们是马尔可夫模型，通过固定窗口的前几个词来估计当前词。n 元语法模型可以通过在训练语料库中进行计数并归一化这些计数（最大似然估计）进行训练。
- 可以使用**困惑度**在测试集上评估 n 元语法语言模型的性能。
- 根据语言模型，测试集的困惑度是测试集概率的函数：它是模型给出的测试集逆概率，按长度归一化。
- 从语言模型中**采样**意味着生成一些句子，并根据模型定义的概率选择每个句子。
- **平滑算法**提供了一种用来估计训练中未见事件的概率的方法。常用的 n 元语法平滑算法包括加一平滑，或通过插值依赖于低阶 n 元语法的计数。