A Bit of Progress in Language Modeling Extended Version

**1引言**

**1.1概述**

语言建模是一门确定一系列单词的概率的艺术。这在许多领域都很有用，包括语音识别、光学字符识别、手写识别、机器翻译和拼写更正（Church，1988；Brown等人，1990；Hull，1992；Kernighan等人，1990；Srihari和Baltus，1992）。最常用的语言模型非常简单（例如Katz平滑三元模型）。不过，在这个简单模型上有很多改进，包括缓存、聚类、高阶n-grams、跳过模型和句子混合模型，我们将在下面描述所有这些。不幸的是，这些更复杂的技术很少被结合起来研究。完全有可能，两种分开工作的技术不能很好地协同工作，而且，正如我们将要展示的那样，甚至有可能某些技术协同工作会比任何一种单独工作都好。在本文中，我们将首先分别检查上述每一种技术，看看技术上的变化或其限制。然后，我们将研究各种组合的技术，并与没有计数截止的Katz平滑三元图进行比较。在一个100000字的小训练数据集上，我们可以得到高达50%的复杂度减少，这是一个熵。在更大的数据集上，改进率下降，在我们最大的数据集284000000字上下降到41%。在没有标点符号的类似大集合上，降幅为38%。在这个数据集上，我们实现了8.9%的字错误率降低。相对于一个公平的基线，这些可能是语言模型最大的困惑减少。

论文的结构如下。首先，在本节中，我们将描述我们的术语，简要介绍我们研究的各种技术，并描述我们的评估方法。在下面的章节中，我们将更详细地描述每种技术，并给出关于该技术的各种变化的实验结果，确定每种最佳变化或其极限。特别是对于缓存，我们展示了trigram缓存的潜力几乎是unigram缓存的两倍。对于聚类，我们发现了比传统聚类稍好的变体，并检查了其局限性。对于n-gram模型，我们检查了多达20grams，但显示即使是最大的模型，性能也稳定了5到7grams。对于跳跃模型，我们给出了不同跳跃技术的第一个详细比较，以及我们在5克水平上所知道的第一个。对于句子混合模型，我们显示多达64个句子类型的混合可以导致改进。然后，我们进行实验，比较所有技术，并以各种方式组合所有技术。我们所有的实验都是在三到四个数据大小上进行的，显示了哪些技术随着更多的数据而改进，哪些变得更糟。在结论部分，我们讨论我们的结果。最后，在附录中，我们给出了有助于证明Kneser-Ney平滑的合理性的证明，并描述了处理大数据量、优化参数、聚类和平滑的实现技巧和细节。

1.2技术介绍

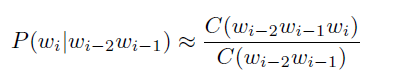
语言模型的目标是确定单词序列w1…wn，P（w1…wn）的概率。这种概率通常分为其组成概率：



由于很难计算大i的形式P（wi | w1…wi−1）的概率，我们通常假设单词的概率仅取决于前两个单词，即trigram假设：

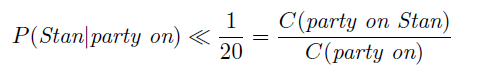


实践证明效果良好。然后根据训练语料库中的数来估计三叉概率。我们让C（wi-2wi-1wi）表示wi-2wi-1wi在我们的训练语料库中的出现次数，同样，对于C（wi-2wi-1）。然后，我们可以估计：



不幸的是，通常这种近似会非常嘈杂，因为有许多三个字的序列从来没有出现过。例如，考虑一下“party on Tuesday”的顺序。什么是星期二的聚会？我们的训练语料库可能不包含任何短语的实例，因此C（party on Tuesday）将为0，而“party on”可能仍有20个实例。因此，我们可以预测P（星期二| party on）=0，这显然是低估。在语言模型的许多应用中，这种0概率是非常有问题的。例如，在语音识别器中，无论声学多么清晰，都无法识别概率为0的单词。

平滑技术会从某些事件中消除一些可能性。想象一下，在我们的培训数据中有一个“陈斯坦生日派对”的例子。通常，当某件事只发生一次时，它被高估了很多。特别地，



通过从“Stan”等词中去掉一些概率，并将其重新分配到“Tuesday”等词中，可以避免零概率。在平滑的三元模型中，额外的概率通常根据平滑的bigram模型等分布。虽然最常用的平滑技术Katz平滑（Katz，1987）和Jelinek Mercer平滑（Jelinek和Mercer，1980）（有时称为删除插值）工作良好，甚至存在更好的平滑技术。特别是，我们之前已经展示过（Chen和Goodman，1999）Kneser-Ney smoothing的版本（Ney等人，1994）优于所有其他平滑技术。在附录中，我们给出了部分解释这种最优性的证明。在Kneser-Ney平滑中，退避分布被修改：使用特殊分布而不是正态bigram分布。使用Kneser-Ney平滑代替更传统的技术是我们使用的第一个改进。

对trigram模型最明显的扩展是简单地移动到更高阶的n-grams，比如4-grams和5-grams。我们将证明，事实上，移动到5-grams可以得到显著的改进。此外，在过去，我们已经证明了平滑和n-gram阶之间存在着显著的相互作用（Chen和Goodman，1999）：高阶n-grams在Kneser-Ney平滑中的效果比在其他一些方法中更好，特别是Katz平滑。我们还将研究从高阶n-grams可以得到多少改进，最多检查20-grams。

n-gram模型的另一个简单扩展是跳过模型（Rosenfeld，1994；Huang et al.，1993；Ney et al.，1994），在这个模型中，我们的条件与前两个词不同。例如，我们可以不计算P（wi | wi-2wi-1），而是计算P（wi | wi-3wi-2）。后一种模型可能不太好，但可以与标准模型结合，以产生一些改进。

聚类（也称为分类）模型试图利用词之间的相似性。例如，如果我们看到“星期一聚会”和“星期三聚会”这样的短语出现，那么我们可以想象“星期二”这个词与“星期一”和“星期三”都相似，也很可能会遵循“party on”这一短语。之前关于单词聚类的大多数研究都集中在如何获得最佳聚类上。我们集中研究了使用集群的最佳方法，并将报告一些新技术的结果，这些新技术比以前的方法工作得更好。我们还展示了聚类和平滑之间的重要交互作用。

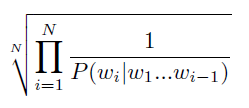
缓存模型（Kuhn，1988；Kuhn and De Mori，1990；Kuhn and De Mori，1992）利用这样的观察：如果你使用一个词，你很可能会再次使用它。它们往往易于实现，并导致相对较大的困惑改善，但相对较小的字错误率改善。我们展示了通过使用trigram缓存，我们可以得到几乎两倍的改进，从一个unigram缓存。

句子混合模型（Iyer and Ostendorf，1999；Iyer et al.，1994）利用观察结果，即有许多不同的句子类型，并且为每种类型的句子建立模型可能比使用一个全局模型要好。传统上，只使用4到8种类型的句子，但我们表明，通过使用64种或更多的混合语句可以得到改进。

**1.3评估**

在本节中，我们首先描述并证明我们使用困惑或熵作为评估技术的合理性。然后，我们将在下面的章节中描述实验中使用的数据和实验技术。

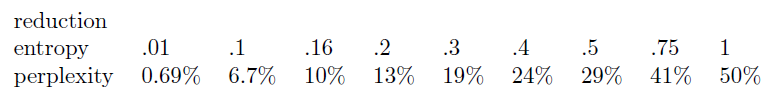
衡量语言模型性能最常用的方法是perplexity。如果一个语言模型将100个单词的概率分配为相等的话，那么它将有100个perplexity。一般来说，语言模型的复杂度等于测试数据中测得的单词反概率的几何平均值：



困惑有许多特性，使其作为衡量语言模型性能的一个指标具有吸引力；除其他外，任何数据源的“真实”模型对该源的困惑程度都是最低的。因此，我们模型的困惑程度越低，在某种意义上就越接近真实模型。虽然有几个替代困惑的方法被证明与语音识别性能有更好的关联，但它们通常有一些自由变量需要针对特定的语音识别器进行优化；其他方法在我们的框架中计算起来要困难得多。

另一种衡量困惑的方法是熵，它只是困惑的log2。熵有一个很好的特性，即它是使用最佳编码器对测试数据进行编码所需的每个字的平均比特数。对于那些熟悉信息论的人来说，我们实际测量的是给定模型的测试数据的交叉熵。由于这是迄今为止最常见的熵类型，我们滥用这个术语，只说熵，当我们真正的意思是一个特定的交叉熵。

本文将同时使用熵和困惑度量：熵约化有一些很好的性质，包括可加性和良好的图形化，但困惑约化在文献中更为常见。下表可能会有所帮助。注意，熵和复杂度减少之间的关系大致是线性上升的，大约0.2位。



我们所有的实验都是在NAB（北美商业新闻）语料库（Stern，1996）上进行的。我们在4种不同的训练数据大小下进行了大部分实验：100000字，1000000字，10000000字，整个语料库（除了1994年的华尔街日报数据）大约284000000字。在所有情况下，我们对一组单独的heldout数据进行参数优化，然后对一组测试数据进行测试。三个数据集都没有重叠。heldout和testset总是每50句话从两个不重叠的1000000个单词集合中抽取，取自1994部分。在附录中，我们描述了我们使用的实现技巧；这些技巧使得在大量的训练数据上训练非常复杂的模型成为可能，但这使得在大型测试集上测试变得困难。出于这个原因，我们总共只用了20000字来测试或帮助数据。另一方面，我们不想简单地使用一个20000字的连续测试或heldout集，因为这只会构成一些文章，因此可能会因为太多的同质性而出现问题；因此，我们选择使用不重叠的1000000字集中的每50个句子。我们所有的实验都是用相同的58546个单词词汇完成的。句子结尾、段落结尾和文章结尾符号都包含在复杂度计算中，但词汇外的词则不包含在内。

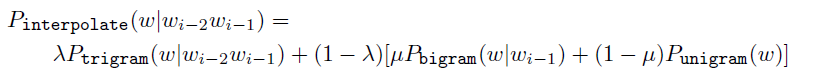
在其他语料库以及其他数据大小上尝试我们的实验会很有趣。在我们之前的工作中（Chen和Goodman，1999），我们比较了语料库和数据大小。我们发现不同的语料库在质量上是相似的，而且最重要的差异是训练数据的大小。因此，我们决定将实验集中在不同的训练数据大小上，而不是不同的语料库上。

我们的工具包与众不同之处在于它允许所有参数进行联合优化。特别是，在结合多种技术时，有许多插值和平滑参数需要优化。我们对heldout数据使用Powell算法（Press等人，1988）来联合优化所有这些参数。

**2.smoothing**

有许多不同的平滑技术可以使用，而且主题是一个令人惊讶的微妙和复杂的。那些对平滑感兴趣的人应该参考我们之前的工作（Chen和Goodman，1999），在那里对几乎所有常用的平滑算法进行了详细的描述和详细的比较。我们将在这里的讨论局限于四种主要技术：简单插值、Katz平滑、退避Kneser-Ney平滑和插值Kneser-Ney平滑。在本节中，我们将描述这四种技术，并回顾先前的结果，包括插值Kneser-Ney平滑（或其上的微小变化）优于所有其他平滑技术的重要结果。

在语言建模中结合技术的最简单方法是简单地将它们插在一起。例如，如果有三元模型、双元模型和单元模型，则可以使用

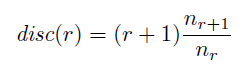


式中，λ和μ是常数，使得0≤λ，μ≤1。在实际应用中，我们还使用统一分布的进行插值；这确保没有单词被分配概率为0。此外，我们还需要处理这样的情况，例如，从未见过trigram上下文wi-2wi-1，C（wi-2wi-1）=0。在这种情况下，我们使用插值bigram模型等。鉴于其简单性，简单插值的效果出奇地好，但其他技术，如Katz平滑，工作得更好。

Katz平滑（Katz，1987）基于Good-Turing公式（Good，1953）。请注意，如果一个特定的单词序列（即“party on Stan”）只出现一次（可能是十亿个单词中的一次），那么它可能被大大高估了——它可能只是偶然出现，其真实概率远小于十亿分之一。结果发现，同样的事情在较小程度上适用于发生两次的序列，以此类推。让表示出现r次的n-grams的数目，即

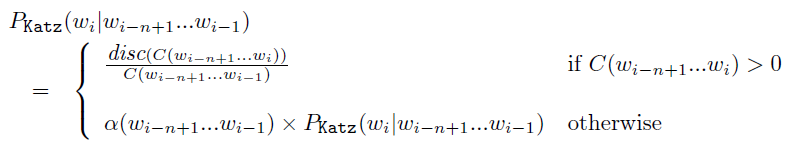


很好地证明了，在一些非常弱的假设下，对于任何出现r次的n-gram，我们应该对它进行折扣，假设它出现次，



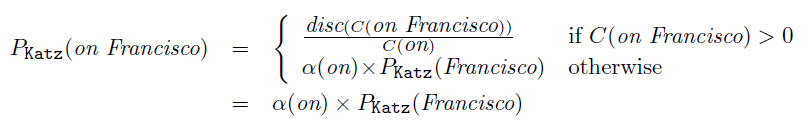
（通常写为r∗）。在语言建模中，估计几乎总是小于r。这将留下一定数量的概率“剩余”。实际上，让N代表训练集的总大小，这个剩余概率将等于；这表示为从未见过的事件分配的概率数量。这确实是一个非常令人惊奇且普遍有用的事实，我们可以通过观察曾经发生过的事情的比例来预测我们期望发生从未发生过的事情的频率。

对于给定的上下文，Katz平滑使用两个公式之一。如果之前看到过单词序列wi-n+1…wi，则Katz平滑使用序列的折扣计数除以上下文wi-n+1…wi-1的计数。另一方面，如果这个序列以前从未出现过，那么我们回到下一个较低的分布wi-n+2…wi-1。基本上，我们使用以下公式：

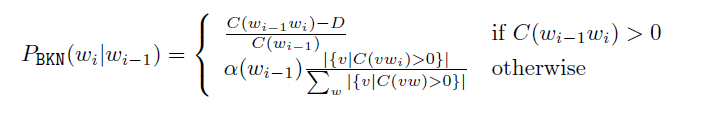


其中α（wi-n+1…wi-1）是选择的归一化常数，以便概率总和为1。

Katz平滑是最常用的平滑技术之一，但事实证明其他技术的效果更好。Chen和Goodman（1999）对许多平滑技术进行了详细的比较，发现Kneser-Ney平滑的修正插值形式（Ney等人，1994）始终优于所有其他平滑技术。Kneser-Ney平滑背后的基本洞察力如下。考虑一个短语的传统bigram模型，比如。由于San Francisco这个短语相当常见，因此传统的unigram概率（如Katz平滑或删除插值等技术所使用的）也将相当高。这意味着，例如，使用Katz平滑等模型，其概率为：

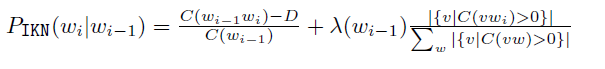


也会相当高。但是，Francisco这个词出现在极少数的上下文中，它出现在一个新的上下文中的概率非常低。Kneser-Ney平滑使用基于每个单词出现的上下文数量而不是单词出现的数量的修正退避分布。因此，像PKN（Francisco | on）这样的概率是相当低的，而对于在许多上下文中出现的单词Tuesday，PKN（Tuesday | on）则相对较高，即使在训练数据中没有出现Tuesday这个短语。KneserNey平滑也使用比Katz平滑更简单的折扣方案：与其使用良好的图灵计算折扣，不如使用单个折扣D（在保留数据上优化）。特别是，Backoff Kneser-Ney平滑使用以下公式（此处为bigram给出），其中{v | C（v wi）>0}|是wi在上下文中可以出现的单词v的数目。



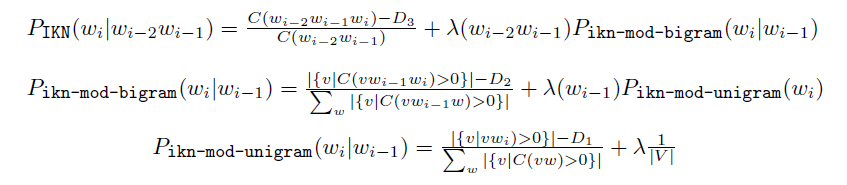
同样，α是一个归一化常数，使得概率之和为1。该公式一般可方便地推广到高阶n-grams。例如，对于trigrams，修改了unigram和bigram分布

Chen和Goodman（1999）表明，Katz平滑和Backoff-Kneser-Ney平滑等仅在缺少高阶计数时才回退到低阶分布的方法在低计数（如一计数和二计数）上效果不佳。这是因为这些低计数的估计相当差，并且估计忽略了低阶分布中的有用信息。插值模型总是将高阶分布和低阶分布结合起来，通常效果更好。特别是，插值Kneser-Ney平滑的基本公式是

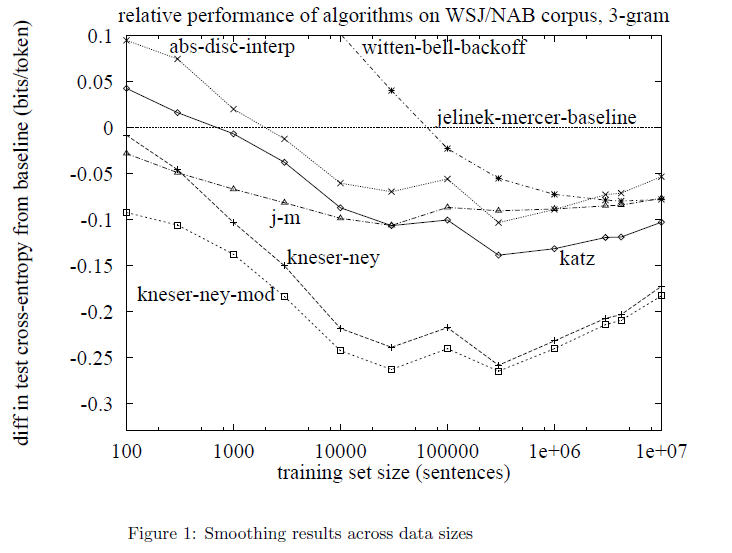


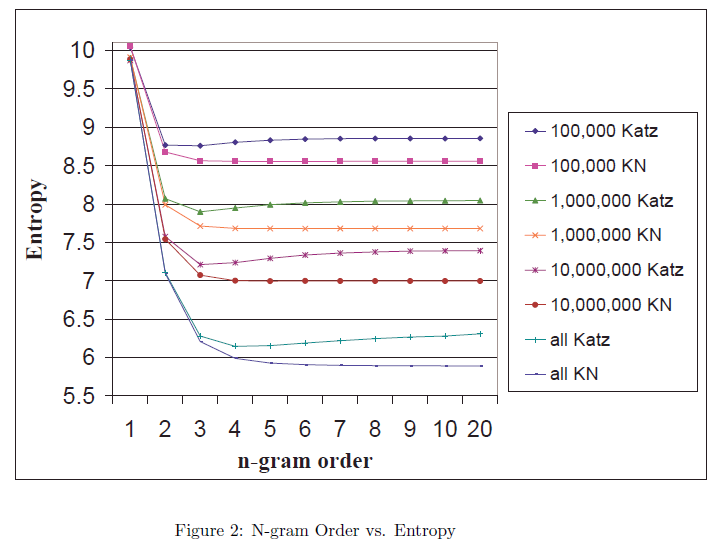
其中，λ（wi-1）是一个归一化常数，使得概率总和为1。Chen和Goodman（1999）对KneserNey平滑法提出了一个额外的修改，即使用多重折扣，一对一，二对二，三或三对多。这种公式，即修正的Kneser-Ney平滑，通常比插值的Kneser-Ney效果稍好。然而，在我们关于组合技术的实验中，它将使我们的系统需要搜索的参数的数量增加近两倍，在一项初步研究中，当许多技术组合时，它的工作效果并不比内插Kneser-Ney好。因此，在本文的其余部分中，我们使用插值的Kneser-Ney而不是修改的Kneser-Ney。在附录中，我们给出了一些关于我们的平滑技术实现的更多细节，包括用于Katz平滑的标准改进。文中还给出了Kneser Ney平滑的证明参数和实例代码，说明了插值kneserney平滑易于实现。

为了完备性，我们给出了插值的KneserNey光滑三元图的精确公式。在实际中，为了避免零概率，我们总是用均匀分布来平滑unigram分布，但是为了简单起见，我们从其他公式中加入了unigram平滑；这里包括它是为了完整性。让V代表词汇的大小。



在图1中，我们重复Chen和Goodman（1999）的结果。这些是本文中的唯一结果，与论文的其余部分不完全相同，但我们希望它们非常具有可比性。用于这些实验的基线是一个简单版本的Jelinek-Mercer平滑，使用一个bucket；该版本与我们描述的第一个平滑技术相同，即简单插值。Kneser-Ney平滑是本文中使用的Kneser-Ney平滑的插值版本，Kneser-Ney mod是有三个折扣而不是一个折扣的版本。Katz平滑与本文中的版本基本相同。j-m是Jelinek Mercer smoothing的缩写，有时在其他地方称为deleted interpolation；abs disc interp是绝对折扣的插值版本。训练集的大小是用句子来衡量的，而不是用单词，每个句子大约有20个单词。注意Jelinek-Mercer平滑和Katz平滑交叉，一个在较低的数据大小更好，另一个在较高的大小。这是我们在多个数据大小上运行本文中所有实验的动机之一。另一方面，在那些对多个语料库进行的实验中，我们没有发现任何一种技术在一个语料库中效果更好，而在另一个语料库中效果更好。因此，我们对不在多个语料库上运行的决定相当有信心。Chen和Goodman（1999）对这些技术进行了更全面的比较，并进行了更深入的分析。Chen和Goodman（1998）给出了一个超集，也作为教程介绍。





**3高阶n-grams**

虽然trigram的假设在实践中已经证明是合理的，但在许多情况下，甚至更长的上下文也会有帮助。因此，很自然地可以放松三元组假设，而不是计算P（wi | wi-2wi-1），而是使用更长的上下文，例如P（wi | wi-4wi-3wi-2wi-1），这是一个五-grams模型。在许多情况下，在训练数据中看不到形式为wi-4wi-3wi-2wi-1的序列，系统将需要退回到四个图、三元图、双元图，甚至是单元图，或使用四个图、三元图、双元图或甚至单元图进行插值，但是在看到如此长的序列的情况下，它可能是wi的一个很好的预测器。

一些较早的实验显示，在较长的背景下，它们没有什么好处。这部分是由于平滑。如Chen和Goodman（1999）所示，一些平滑方法在处理高阶n-grams时明显优于其他方法。特别是，插值Kneser-Ney平滑的优点在高阶n-grams下要比低阶n-grams下大得多。

我们对n-gram顺序与困惑之间的关系进行了多种实验。特别是，我们尝试了Katz平滑和插值Kneser-Ney平滑在n-gram阶从1到10，以及20，并且超过了我们的标准数据大小。结果如图2所示。

正如可以看到的，并且之前已经观察到的（Chen和Goodman，1999），Katz平滑的行为与Kneser-Ney平滑的行为非常不同。Chen和Goodman认为造成这种差异的主要原因是退避平滑技术，如Katz平滑，甚至是Kneser-Ney平滑的退避版本（我们在这项工作中只使用插值Kneser-Ney平滑），在低计数，特别是一个计数上效果不佳，并且随着n-gram阶次的增加，计数次数增加。特别是，Katz平滑在trigram级别上有最好的性能，当超过这个级别时实际上会变得更糟。另一方面，Kneser-Ney平滑基本上是单调的，即使经过20-gram。

Kneser-Ney平滑的平台点取决于可用的训练数据量。对于少量的100000字，平台点是在三元水平上，而当使用完整的训练数据2.8亿字时，即使在6克（0.02比特比5克好）和7克（0.01比特比6克好）中也会出现小的改进。这种大小的差异很有趣，但并不具有实际意义。4-gram和5-gram之间的差异，.06位可能很重要，因此，在我们的其余实验中，我们经常使用基于5-gram数据的模型，这似乎在计算资源和性能之间给出了一个很好的折衷。

注意在实践中，超越三联图通常是不切实际的。内存和性能之间的折衷通常需要对4gram和5-gram进行大量修剪，从而减少它们带来的潜在改进。在本文中，我们忽略了内存性能的权衡，因为这会使已经很难进行的比较过于复杂。相反，我们寻求建立一个可能的最佳系统，忽略了内存问题，把在给定内存大小下寻找最佳系统的更实际、更有趣、更复杂的问题留给未来的研究（以及一些过去的研究（Goodman和Gao，2000））。请注意，本节中所做的许多实验，如果没有本文结尾处简要介绍的专用工具，以及附录中更详细的介绍，是完全无法完成的。

**4 Skipping**

当一个人移动到越来越大的n-grams时，以前看到确切上下文的可能性越来越小；但是看到类似上下文（其中包含大多数单词的上下文）的可能性增加。跳跃模型（Rosenfeld，1994；Huang等人，1993；Ney等人，1994；Martin等人，1999；Siu和Ostendorf，2000）利用了这一观察结果。这项技术也有变化，例如使用格的技术（索尔和佩雷拉，1997年；杜邦和罗森菲尔德，1997年），或者结合类和词的模型（布拉西格，1999年）。

在考虑5克上下文时，我们可以考虑five gram的许多子集，例如P（wi | wi−4wi−3wi−1）或P（wi | wi−4wi−2wi−1）。也许我们从来没有见过“ShowJohn a good time”b，但是我们已经看到了“Show Stan a good time”这个短语，一个正常的5克预测P（time | Show John a good）将返回到P（time | John a good），然后从P（time | a good）返回到P（time | a good），这个概率相对较低。另一方面，P形式的跳跃模型（wi | wi-4wi-2wi-1）将高概率分配给P（Show \_\_\_\_\_\_ a good）。

这些跳过的5克然后用一个正常的5克插值，形成如下模型



式中，与通常一样，0≤λ≤1和0≤μ≤1和0≤（1-λ–）1。

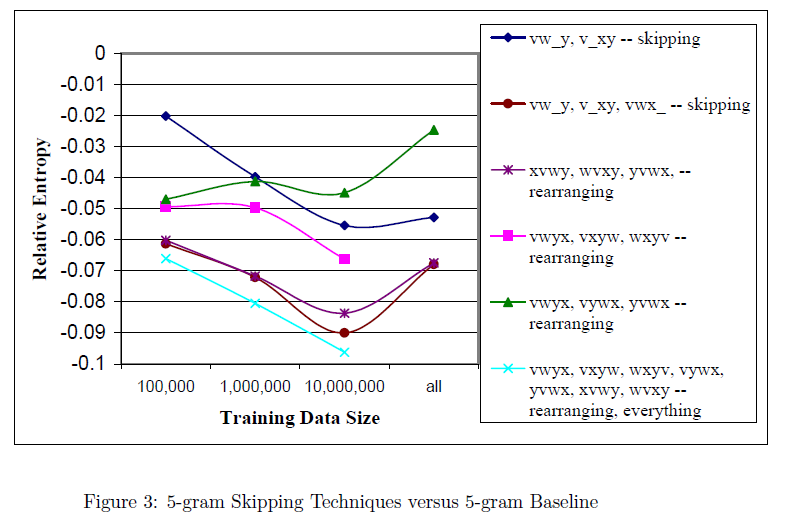
Skipping 的另一种（也是更传统的）用法是作为一种穷人的高阶n-克。例如，可以创建表单的模型



在这种形式的模型中，没有组件概率依赖于两个以上的单词，如三元组，但总体概率是4-gram-like，因为它依赖于wi-3、wi-2和wi-1。我们可以进一步扩展这一概念，以5-gram-like、6-gram-like甚至7-gramlike的方式组合所有上下文对，每个成分的概率永远不依赖于前两个词。

我们做了两组实验，一组是5克，一组是三联。对于5-gram跳过实验，所有上下文最多依赖于前面的四个单词wi-4、wi-3、wi-2、wi-1，但使用这四个单词的方式多种多样。我们尝试了六个模型，所有模型都是用一个基线5gram模型插值的。为了可读性和简洁性，我们定义了一个新的符号，让v=wi-4，w=wi-3，x=wi-2和y=wi-1，这样我们就可以避免后面的大量下标。结果如图3所示。

第一个模型插值了vw y和v xy的依赖关系。这个简单的模型在最小的训练数据量上不能很好地工作，但是对于较大的训练数据量是有竞争力的。接下来，我们在这个模型上尝试了一个简单的变化，它也在vwx中插值。通过简单的添加，可以在所有级别上都有很大的改进，大约比简单的跳过模型多0.02到0.04位。我们的下一个变体是类似的，但又增加了对缺失单词的依赖性。特别是，我们将xvwy、wvxy和yvwx插值在一起；也就是说，所有的模型都依赖于相同的变量，但是插值顺序被修改了。例如，在xvwy中，我们指的是P（z | vwxy）形式的模型，P（z | vw y）内插P（z | vwy），P（z | vwy）内插P（z | y），P（z）内插P（z）y。所有这些实验都是用内插Kneser-Ney平滑法进行的，所以除了第一个概率外，所有的实验都使用修正的退避分布。该模型与前一个模型相同，但对于每个分量，都以完整的5克开始插值。我们曾希望，在训练数据中出现完整的5克的情况下，这会使跳跃模型更加精确，但这一点都没有帮助。

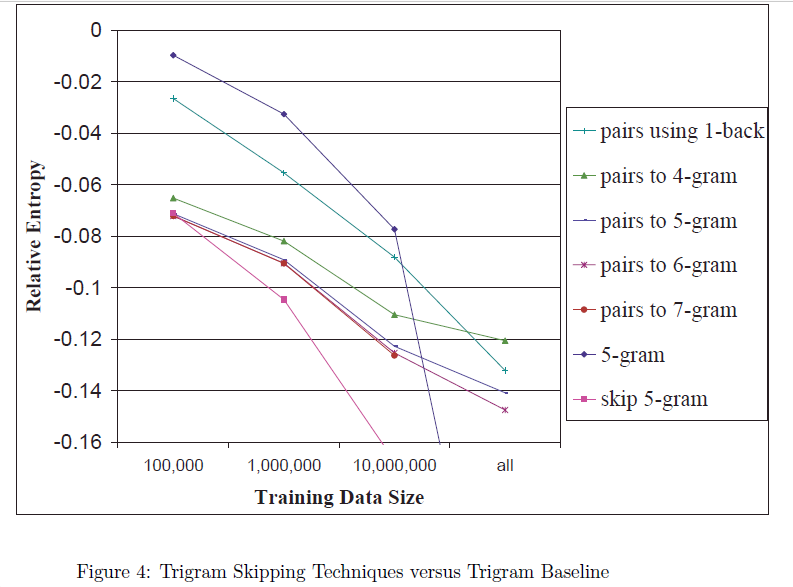


我们还想尝试更激进的方法。例如，我们尝试将vwyx与vxyw和wxyv（以及基线vwxy）一起插值。此模型将前面四个单词中的每一个放在一个组件的最后（最重要）位置。这个模型的工作效果不如前两个模型，这导致我们得出结论，到目前为止y字是最重要的。我们还尝试了一个vwyx，vywx，yvwx的模型，它将y字放在退避模型中的每个可能位置。这是最糟糕的模型，再次证实了y字是关键的直觉。然而，正如我们通过将vw x添加到vw y和v xy中所看到的，具有最终x位置的组件也很重要。这也适用于三联图。

最后，我们想得到一个关于5-gram模型工作情况的上限。为此，我们将vwyx、vxyw、wxyv、vywx、yvwx、xvwy和wvxy相加。这个模型被选为一个包含尽可能多的成对和三倍单词组合的模型。其结果是，与之前最好的模型相比，边际增益小于0.01位。

我们认为这些结果并不特别令人鼓舞。特别是，与稍后将呈现的句子混合结果相比，跳过模型获得的可能性似乎更小。此外，虽然句子混合模型似乎会导致更大的增益-使用更多的数据，跳过模型似乎会获得最大的增益约10000000字。据推测，在最大的数据量下，5-gram模型正在得到很好的训练，而跳转模型有用但5-gram不可用的实例更少。

我们还研究了类似三联图的模型。这些结果如图4所示。比较的基线是三联图模型。为了比较，我们还展示了5-gram模型相对于trigram的相对改进，以及vw y、v xy和vwx对跳过5-gram的相对改进。对于trigram跳过模型，每个组件从不依赖于前面的两个词。我们试了5次这种形式的实验。首先，基于使用1-back word（y）的对最有用的直觉，我们对xy、wy、vy、uy和ty模型进行了插值。这不是特别好的工作，除了在最大的规模。据推测，在这样的大小，一些适当的例子1背字一直看到。下一步，我们尝试通过4-gram级别使用所有单词对：xy、wy和wx。考虑到它的简单性，这非常有效。我们使用所有5-gram对、所有6-gram对和所有7-gram对尝试了类似的模型；最后一个模型包含15个不同的对。然而，相对于4-gram对的改进仍然是微不足道的，特别是考虑到大量增加的参数。

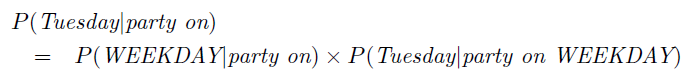


相对于基线，三元跳转的结果要比五元跳转的结果好得多。当更多的数据被使用时，它们似乎并没有稳定下来，而且就它们得到的改进而言，它们与句子混合模型更具可比性。而且，它们会导致更多当用于少量数据时（当然，最好的5-gram跳转模型总是比最好的三元跳转模型好）比单独使用5-gram的改进使它们成为一种合理的技术，可用于少量和中等数量的训练数据，特别是在不能使用5-gram的情况下。

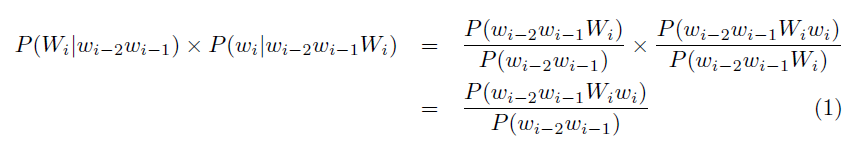
**5群集**

**5.1使用群集**

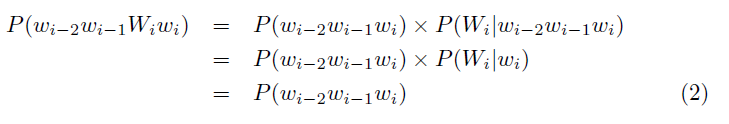
接下来，我们将描述我们的聚类技术，这些技术与传统的聚类技术（Brown et al.，1992；Ney et al.，1994）稍有不同（并且，正如我们将要展示的，稍微更有效）。考虑一个概率，例如P（星期二|party on）。也许训练数据中没有“周二聚会”的例子，尽管“周三聚会”和“周五聚会”等短语确实出现了。我们可以把单词放在类上，例如“星期二”这个单词放在工作日的类上。现在，我们可以考虑“星期二”这个词出现的概率，给出“派对开始”这个词，也可以考虑下一个词是工作日。我们将用来表示这种可能性。然后我们可以分解概率



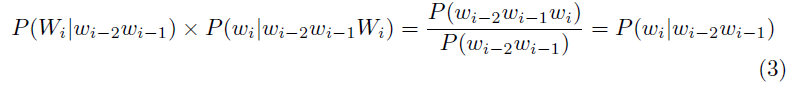
当每个单词只属于一个类（称为硬聚类）时，这种分解是一个严格的等式，这是一个可以简单证明的事实。让Wi表示单词Wi的簇。那么，



现在，由于每个单词都属于一个集群，P（Wi | wi）=1，因此



把方程2代入方程1，我们得到



现在，虽然方程3是一个严格的等式，但当考虑平滑时，使用聚集概率比非聚集概率更精确。例如，即使我们从未见过“星期二聚会”的例子，也许我们见过其他短语的例子，例如“星期三聚会”，因此，概率P（平日聚会）将相对较高。虽然我们可能从来没有见过“星期二聚会”的例子，但在我们用一个低阶模型回退或插值之后，我们可能能够准确地估计P（星期二|在星期二）。因此，我们的平滑聚类估计可能是一个好的估计。我们称之为这种特殊的聚类预测聚类。（另一方面，我们将证明，如果集群很差，预测性集群也会导致性能下降。）

注意，预测聚类还有其他用途以及改善困惑。预测聚类可以显著地加速最大熵训练（Goodman，2001），提高35倍，还可以压缩语言模型（Goodman和Gao，2000）。

我们可以做的另一种聚类是在上下文中对单词进行聚类。例如，如果“party”在类事件中，“on”在类介词中，那么我们可以写



或者更通常



把方程4和方程3结合起来，我们得到

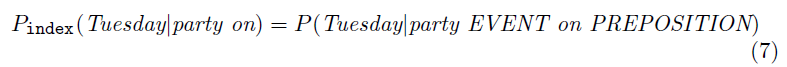


由于等式5没有考虑到前面单词的精确值，所以我们总是（在这项工作中）用一个普通的三元模型对其进行插值。我们将方程5的插值称为三元fullibm聚类。我们称之为fullibm是因为它是IBM发明的一种技术的推广（Brown et al.，1992），它使用近似P（w | Wi−2Wi−1W）≈P（w | w）来得到



当用一个普通的三元组插值时，我们称之为ibm集群。考虑到fullibm集群使用的信息比常规ibm集群更多，我们假设它会带来改进。如图所示，它的工作原理大致相同，至少在使用普通的三元模型进行插值时是这样。

或者，我们可以简单地更改退避顺序（称为索引群集），而不是总是丢弃信息：



在这里，我们稍微滥用符号来使用|右边单词的顺序来表示回退/插值顺序。因此，方程式7意味着我们将从P（星期二介词上的party EVENT）到P（星期二介词上的EVENT）到P（星期二介词上的EVENT）再到P（星期二介词上的EVENT）。注意，因为每个单词都属于一个集群，所以其中一些变量是多余的。例如，在我们的符号中



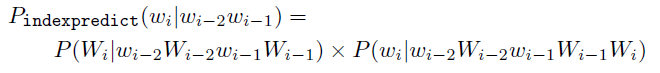


我们通常将指数聚类模型写成。

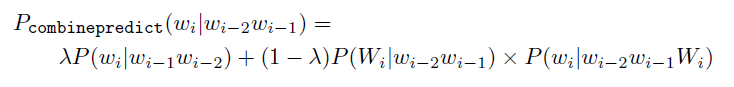
有一个特别值得注意的技术，fullibmpredit。这是我们发现的性能最好的技术（除了组合技术）。这项技术利用了预测聚类背后的直觉，将问题分解为对聚类的预测，然后对给定聚类的单词进行预测。此外，在每一级，它通过组合基于词和基于簇的估计来平滑该预测。它不是用普通的三元模型插值的。它的形式



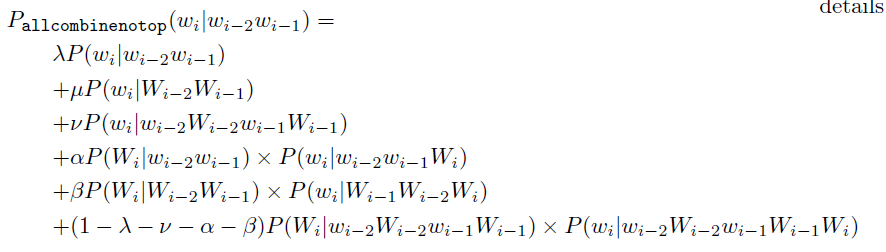
关于这些主题有许多不同的说法。碰巧，没有一个比ibm集群工作得更好，所以我们只简单地描述它们。一种是指数预测，将指数和预测聚类相结合：

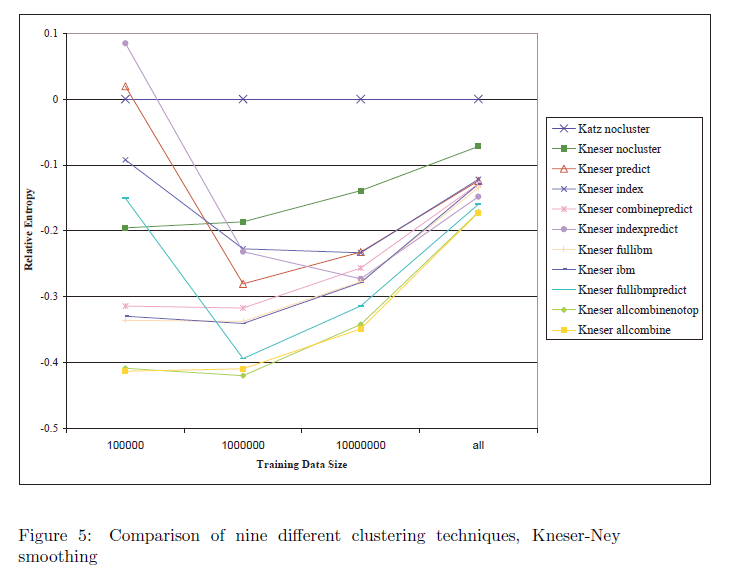


另一种方法是组合预测法，将一个正常的三元图与一个预测聚类三元图进行插值：

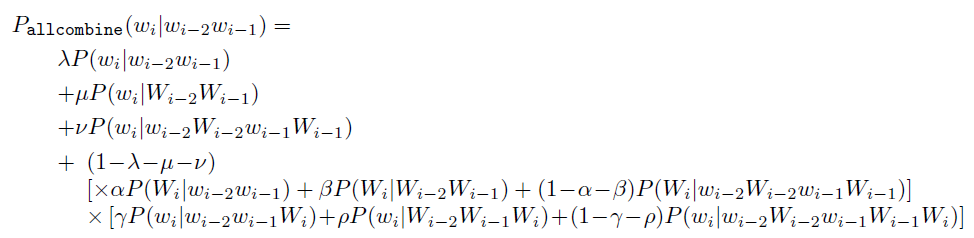


最后，我们想得到一些关于通过聚类可以获得多少的上限，所以我们尝试将所有这些聚类技术结合在一起，得到我们称之为allcombinenotop的东西，它是一个普通的三元组、一个类似fullibm的模型、一个索引模型、一个预测模型、一个真正的fullibm模型和一个indexpredit模型的插值。

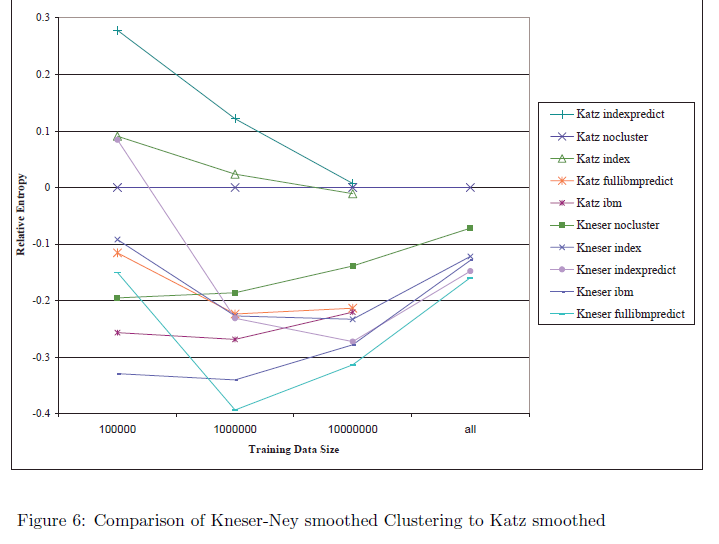




或者，对于另一种组合方式，allcombine，它首先在集群级别插入predicttype模型，然后再与word级别的模型插入。



在图5中，我们展示了九种不同聚类技术的比较，所有细节都使用Kneser-Ney平滑。每个训练规模的集群是分别构建的。为了与下一个图表保持可比性，我们使用Katz平滑作为相对熵比较的基线。注意这个值



聚类的数量随着训练数据的增加而减少；在较小的数据量下，最佳聚类模型的数据量约为0.2位；在最大的数据量下，数据量约为0.1位。由于集群是一种处理数据稀疏性的技术，这并不奇怪。接下来，请注意ibm集群始终工作得非常好。在我们尝试的所有其他技术中，只有4种其他技术工作良好或更好：fullibm集群，这是一个简单的变体；allcombine和allcombinenotop，在fullibm中插入；fullibmpredit。fullibmpredit工作得非常好，比ibm集群高出0.05位。然而，在最小的训练规模下它有一个问题，在这种情况下它更糟。我们认为，最小训练规模的集群非常差，当这种情况发生时，预测式集群会遇到麻烦，因为它会平滑处理可能不相关的单词，而ibm集群则在正常的三元模型中进行插值，使其对差的集群更加健壮。所有使用预测聚类和不插值未聚类三元图的模型在最小训练规模下实际上都比基线差。

请注意，我们使用固定词汇表的特定实验是对最小级别的聚类的严格测试。在5.8万个单词的词汇中，很多单词根本不会出现在10万个单词的训练数据中。我们试图通过在每个单词中添加一个“伪计数”来部分地处理这个问题，即与一个伪单词同时出现。这将具有使所有看不见的单词和计数非常低的单词相似的属性，希望将它们放在同一个集群中。10万字的训练数据应该更多地解释为一个系统如何处理坏的集群，而不是一个实际的系统如何工作，其中的词汇将匹配训练数据。

在图6中，我们展示了使用Katz平滑的几种技术和使用Kneser-Ney平滑的相同技术的比较。结果是相似的，但有一些有趣的例外：特别是，indexpredit对Kneser-Ney平滑模型的效果很好，但对Katz平滑模型的效果很差。这表明平滑可以对其他技术产生显著影响，例如聚类。另一个结果是，在所有9种聚类技术中，无论大小，Kneser-Ney版本总是优于Katz平滑版本。事实上，Kneser-Ney平滑版在各种尺寸的每种技术的内插式和回退式绝对折扣版上也都表现出色。

还有另外两种执行集群的方法，我们将不在这里探讨。首先，我们可以将一组词（完整的上下文）而不是单个词进行分类。也就是说，暂时改变符号，而不是计算





例如，在三元组模型中，可以将“NewY ork”和“Los Angeles”等上下文归类为“CITY”，将“on threed”和“late tomorrow”归类为“TIME”。对于这种聚类，有许多难题需要解决。另一种可以做的条件聚类是根据经验确定，对于给定的上下文，聚类和单词的最佳组合，即变差图方法（Blasig，1999）。

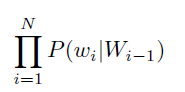
**5.2 Finding Clusters**

以前的大量研究都集中在如何最好地找到集群（Brown等人，1992年；Kneser和Ney，1993年；Yamamoto和Sagisaka，1999年；Ueberla，1995年；Pereira等人，1993年；Bellegarda等人，1996年）。大多数以前的研究发现，不同的聚类方法之间只有很小的差异。然而，一个结果是，自动导出的聚类优于部分语音标签（Niesler等人，1998），至少在有足够训练数据的情况下（Ney等人，1994）。我们并没有探索不同的发现集群的技术，只是简单地选择了一个我们认为是好的，基于先前的研究。

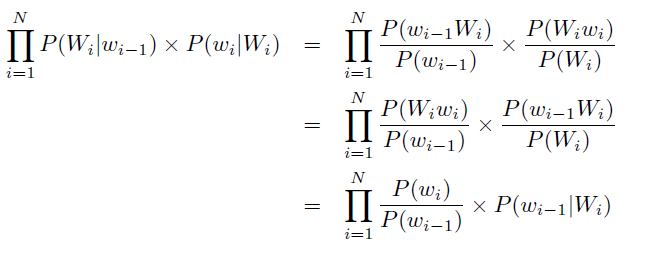
不同位置使用的簇不需要相同。特别是，对于ibm集群这样的模型，使用，我们将wi集群称为预测集群，wi−1和wi−2条件集群称为集群。预测聚类和条件聚类可以是不同的（Yamamoto和Sagisaka，1999）。例如，考虑一对单词，比如a和an。一般来说，a和an可以跟在同一个单词后面，因此，对于预测聚类，属于同一个聚类。但是，很少有单词可以同时跟在a和an后面，所以对于条件聚类，它们属于不同的聚类。实验还发现，用于预测聚类和条件聚类的最优聚类数目是不同的，本文总是分别优化条件聚类和预测聚类的数目，并在每种训练数据大小下对每种技术进行再优化。

这是一个特别耗时的实验，因为每次更改集群数量时，都必须从头开始重建模型。我们总是尝试2次幂的簇数，例如1、2、4等，因为这允许我们尝试大量的簇数，而与最佳数的距离永远不会超过2倍。检查heldout数据的性能图表，这似乎产生了足够接近最优的集群数量。

集群是自动发现使用一个工具，试图尽量减少困惑。特别是，对于条件聚类，我们试图最小化P（wi | wi−1）形式的bigram的训练数据的复杂性，这相当于最大化



对于预测聚类，我们试图最小化P（Wi | Wi−1）×P（Wi | Wi）训练数据的困惑。（我们不最小化P（Wi | Wi–-1）×P（Wi | Wi–-1Wi），因为我们正在对非光滑的训练数据进行最小化，对于任何聚类，后一个公式的细节将等于P（Wi | Wi–-1）。如果我们使用剔除一个的方法（Kneser和Ney，1993），那么我们可以使用后一个公式，但这种方法更困难。），

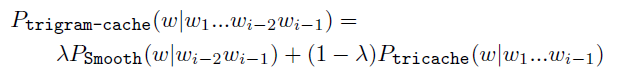


现在，P（wi）P（wi-1）将独立于所使用的集群；因此，尝试最大化  
 N i=1p（wi-1 | wi）就足够了。这非常方便，因为它与对条件集群所做的正好相反。这意味着我们可以对两者使用相同的聚类工具，只需切换程序用于获取聚类的原始计数的顺序。我们给出了更多关于第9节中使用的聚类算法的细节。

**6 Caching**

如果一个演讲者使用一个词，他很可能在不久的将来再次使用同一个词。这种观察是缓存的基础（Kuhn，1988；Kuhn and De Mori，1990；Kuhn and De Mori，1992；Kupiec，1989；Jelinek et al.，1991）。特别是，在一个unigram缓存中，我们从最近说的单词（如果文章标记可用，则为同一篇文章中的所有单词，如果不可用，则为固定数量的先前单词）形成一个unigram模型。然后，可以使用传统的n-gram对这个unigram缓存进行线性插值。

另一种类型的缓存模型取决于上下文。例如，我们可以从前面的单词中形成一个平滑的bigram或trigram，并用标准的trigram进行插值。特别是，我们使用



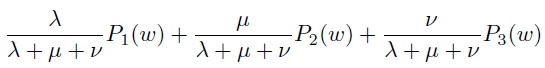
其中Ptricache（w | w1…wi-1）是一个简单的插值三元模型，使用同一文档中前面单词的计数。

另一种技术是使用条件缓存。在这种技术中，我们根据之前是否看到过上下文来对trigram缓存进行不同的加权。

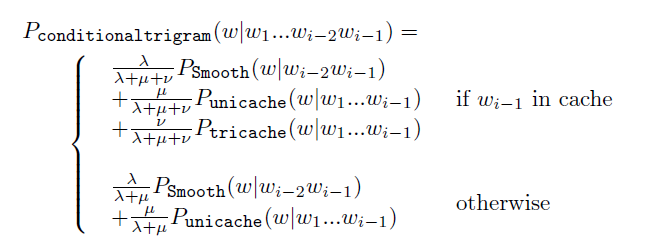
我们在这里离题一会儿，来谈谈我们使用的一个技巧。当内部细节极化三个概率P1（w）、P2（w）和P3（w），而不是使用



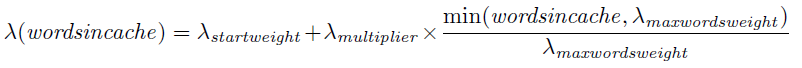
我们实际使用



这使我们能够简化搜索的约束，我们还相信通过添加一个有用的维度来搜索，有助于我们的参数搜索例程。当我们有时不使用这三个组件中的一个时，它特别有用。特别是，对于条件缓存，我们使用以下公式：

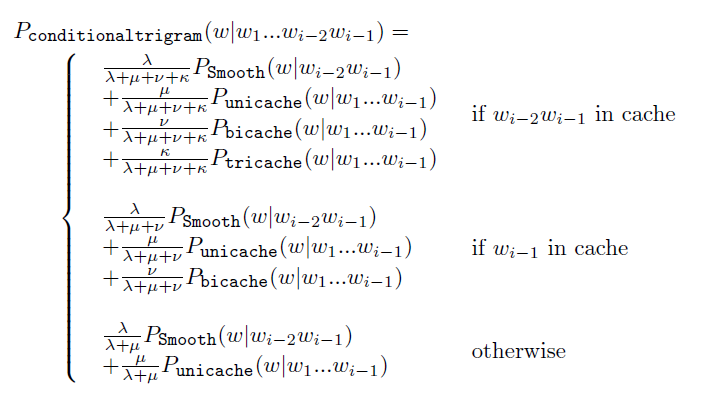


我们尝试了另一个改进。我们假设我们拥有的数据越多，每个缓存就越有用。因此，我们将λ、μ和ν设为缓存中数据量（当前文档中的字数）的线性函数



其中，与通常一样，λstartweight、λmultiplier和λmaxwordsweight是根据heldout数据估计的参数。然而，我们的参数搜索引擎几乎总是将λmaxwordsweight设置为我们允许它具有的最大值1000000，同时将λ乘数分配给一个小值（通常为100或更小），这意味着变量权重基本上被忽略。

最后，我们可以尝试有条件地组合unigram、bigram和trigram缓存。



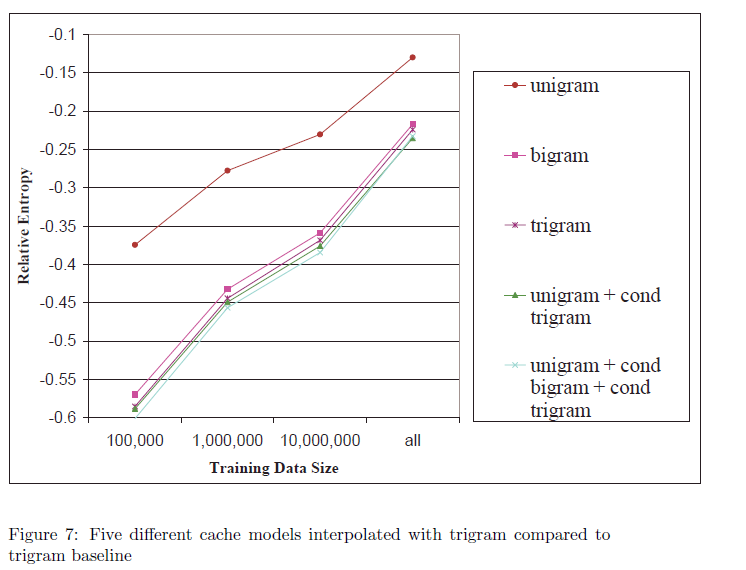


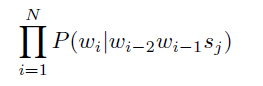
图7给出了运行这五个缓存模型的结果。所有的细节都是用Kneser-Ney平滑的三联图插值的。出于技术原因，每个n-gram缓存模型都使用简单的插值进行平滑处理。可以看到，缓存可能是我们可以应用的最强大的技术之一，它可以提高小数据上高达0.6位的性能。即使是在大数据上，这种改进仍然是相当可观的，高达0.23位。在所有数据大小上，n-gram缓存的性能都比unigram缓存好得多，但是使用哪个版本的n-gram缓存似乎只会产生一个很小的差异。

值得注意的是，所有这些结果都假设前面的单词是完全已知的。然而，在语音识别系统中，许多产品场景不包括用户校正。这样，缓存就有可能“锁定”错误。例如，如果用户说“识别语音”，系统听到“破坏一个漂亮的海滩”，那么，稍后，当用户说“语音识别”时，系统可能会听到“海滩破坏点火”，因为“海滩”的概率将显著提高。因此，在真正的产品中从缓存中获得改进可能是一个更困难的问题。

**7 Sentence Mixture Models**

Iyer和Ostendorf（1999；Iyer等人。（1994）观察到，在一个语料库中，可能有几种不同的句子类型；这些句子类型可以按主题、风格或某些其他标准进行分组。无论它们如何分组，通过分别对每个句子类型建模，都可以提高性能。例如，在《华尔街日报》的数据中，我们可以假设有三种不同的句子类型：金融市场句子（包含大量数字和股票名称）、商业句子（升职、降职、合并）和一般新闻故事。我们可以计算一个句子每种类型的概率，然后取不同句子类型的概率的加权和。因为一个句子中的长距离相关性（大量的数字，或者大量的提升）被这样一个模型捕获，所以整体模型更好。当然，一般情况下，我们在听到句子之前不知道句子的类型。因此，我们将句子类型视为一个隐藏变量。

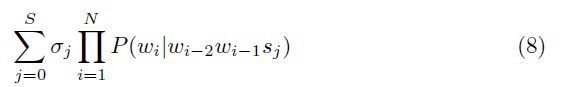
让sj表示所考虑的句子是j类型的句子的条件。那么，假设该句子是j类型的，那么该句子的概率可以写成



有时，全局模型（跨所有句子类型）比任何单个句子类型都要好。让s0成为一个始终正确的特殊上下文：

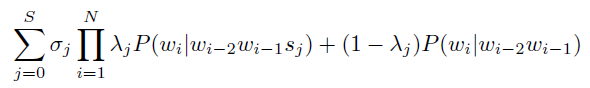


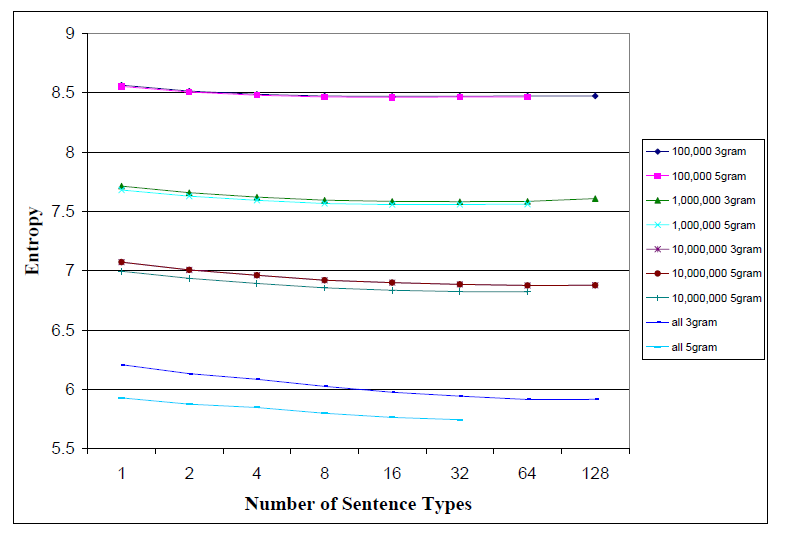
设S个不同的句型（4≤S≤8为典型）；设σ0…σS为在S j=0σj=1的约束下对保留数据优化的句型插值参数。句子w1…wn的总概率等于



等式8可以理解为存在一个隐藏变量，即句子类型；每个句子类型的先验概率为σj。我们计算每种句子类型的测试句子的概率，然后根据该句子类型的先验概率对这些概率求和。

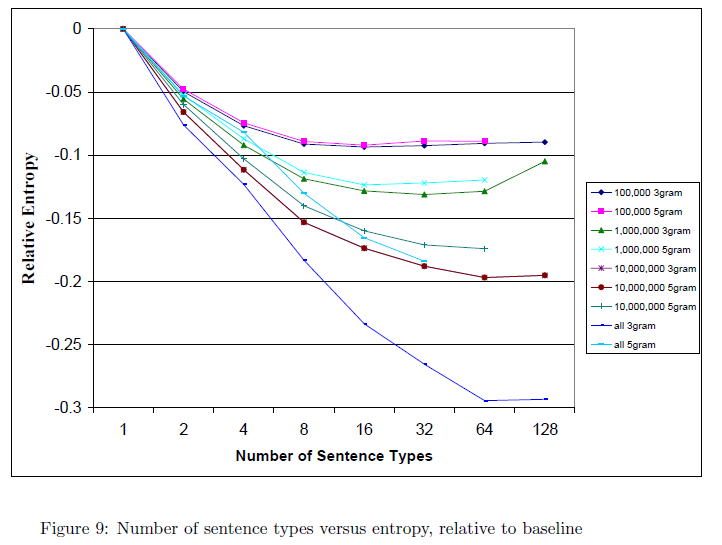
概率P（wi | wi–-2wi–-1sj）可能会受到数据稀疏性的影响，因此它们是用全局模型P（wi | wi–-2wi–-1）进行线性插值的，使用对保留数据优化的插值权重。





使用与单词聚类相同的聚类程序，找到训练数据的句子类型；在这种情况下，我们试图最小化句子聚类的单语码困惑。也就是说，让s（i）表示指定给单词i所属句子的句子类型。（一个句子中的所有单词都被分配到同一个句子类型）我们试图将句子以最大化 的方式分组。这是一种比Iyer和Ostendorf（1999）使用的简单得多的技术。它们使用两个阶段的过程，第一阶段是一个单图相似聚集聚类方法；第二阶段是一个基于EM的n-gram重建方法。此外，他们的技术使用了软聚类，其中每个句子可以属于多个聚类。我们假设他们的技术得到的模型比我们的好。

我们在句子混合模型上做了大量的实验。我们试图研究训练数据大小、n-gram顺序和句子类型数之间的关系。因此，我们在我们的标准数据大小下，使用三元组和5-grams进行了一些实验，将句子类型的数量从1（没有句子混合的正常模型）改变为128。



所有实验均采用Kneser-Ney平滑法。结果如图8所示。我们再次给出了相同的结果，但是相对于它们各自的n-gram模型基线进行了图示，如图9所示。请注意，我们不信任128种混合物的结果；对于这些实验，我们使用了与其他实验相同的20000字heldout数据。128个句子类型共有773个参数，系统可能没有足够的heldout数据来准确估计参数。尤其是，高原显示为128:10000000，所有的训练数据都没有显示在直升机数据中。理想情况下，我们可以用一个更大的heldout set来运行这个实验，但它已经需要5.5天，20000字，所以这是不切实际的。

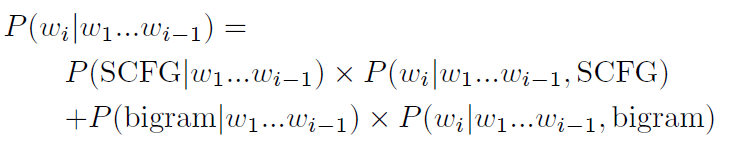
结果非常有趣，原因有很多。首先，我们怀疑句子混合模型在更大的训练数据量上会更有用，事实上是这样；对于100000个单词，句子混合模型的最大改进只有大约.1位，而对于284000000个单词，几乎是.3位。这预示着句子混合模型的未来：随着计算机的速度越来越快，训练数据的大小也应该增加。其次，我们怀疑，由于5-grams和句子混合模型都试图建立长距离依赖关系模型，因此它们的组合所带来的改进将小于单个改进的总和。如图8所示，对于100000和1000000字的训练数据，三元图和5克图之间的差别很小，所以这个问题不是很重要。对于10000000个单词和所有的训练数据，存在一些负交互作用。例如，在所有训练数据上有4个句子类型，对三元组的改进是0.12位，对五元组的改进是0.08位。类似地，32种混合物的三元图和5克图的改进分别为.27和.18。因此，大约三分之一的改善似乎是相关的。

Iyer和Ostendorf（1999）报告了5种混合成分和8种成分的实验，使用3800万字的训练数据，没有发现显著差异。然而，我们更彻底的调查显示，通过使用更多的混合语句，尤其是使用更多的训练数据，确实有很大的改进空间，而且这种潜力至少可以扩展到我们最大大小的64个句子类型。这是一个重要的结果，导致仅使用少量组件的潜在改进几乎是原来的两倍。

我们认为这个新结果是我们研究中最有趣的结果之一。特别是，我们在这里使用的技术相对简单，对这些技术的许多扩展可能会导致更大的改进。例如，可以创建句子类型和超类型，然后将句子类型与其超类型和全局模型（全部合并）平滑在一起，而不是简单地使用全局模型平滑句子类型。这将缓解在最大数量的混合物中看到的数据稀疏效应。

我们的句子混合模型结果令人鼓舞，但与之前的结果相比，细节令人失望。虽然Iyer和Ostendorf（1999）在5种混合语中减少了大约19%的困惑，减少了大约3%的单词错误率，但在类似的数据中，我们在4种混合语中只减少了大约9%和1.3%。我们知道他们之间最大的不同系统和我们的不同之处在于集群技术：我们使用了一个相当简单的技术，而他们使用了一个相当复杂的技术。其他可能性包括他们使用的不同的平滑（Witten Bell平滑）与我们的Katz或插值Kneser-Ney平滑；以及他们使用五个簇而我们使用四个簇的事实。然而，Katz和内插Kneser-Ney是非常不同的技术，但是，正如我们稍后将要报告的那样，句子混合模型与两者产生的改进大致相同，因此我们不认为平滑的差异解释了不同的结果。另外，Iyer和Ostendorf（1999）发现当使用8个聚类而不是5个聚类时没有显著差异，并且我们发现在相似数据上使用8个而不是4个聚类时只有4%的差异。值得注意的是，Iyer和Ostendorf的基线困惑为211，而我们的基线困惑为95，训练集相同，但词汇和测试集不同。也许，无论是什么未知因素导致了这种基线困惑的差异，都为句子混合模型提供了更大的改进空间。值得注意的是，我们在缓存方面的改进也远低于Iyer和Ostendorf：分别为7.8%和16.7%。我们的缓存实现非常相似，主要区别在于它们排除了停止字。这增加了对“不同基线/测试条件”解释的支持。如果复杂度降低的差异是由于混合模型实现的某些差异而不是测试条件的差异，则可以实现额外10%的复杂度降低，这是值得进一步探索的量。

当组合非常不同的细节语言模型类型时，句子混合模型也很有用。例如，Jurafsky等人。（1995）使用句子混合模型将随机上下文无关语法（SCFG）模型与bigram模型相结合，结果略好于单独使用的任何一个模型。Jurafsky等人的模型。实际上是这样的

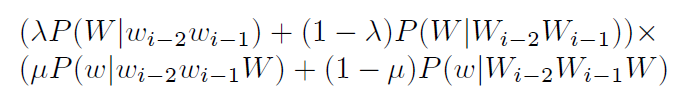


这就相当于一个方程8形式的模型。这种公式的优点是，当用于堆栈解码器时，它允许以相对较少的开销使用句子混合模型，而第10.5节中讨论的公式8 Charniak（2001）使用句子级混合模型将语言模型与三元模型结合起来，显著减少困惑。

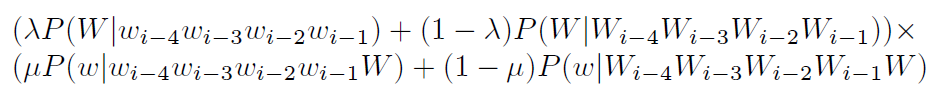
**8 Combining techniques**

在本节中，我们将介绍有关组合技术的其他结果。虽然我们所介绍的每一种技术各自都能很好地工作，但我们将展示其中一些技术协同工作，并且其中一些技术是部分冗余的。例如，我们已经证明，从KneserNey模型和5-gram模型的共同改进比从单独改进更大。类似地，正如我们已经展示的，句子混合模型与5-grams结合时的改进仅仅是句子混合模型自身改进的2/3个左右，因为这两种技术都增加了数据的稀疏性。在这一部分中，我们系统地研究了三个问题：平滑对每种技术有什么影响；每种技术与所有其他技术结合时有多大帮助；以及每种技术分别和一起如何影响字的错误率。

有许多不同的方法来结合技术。组合技术的最明显方法是简单地对它们进行线性插值，但这不太可能导致最大的可能改进。相反，我们试图结合概念。举一个简单的例子，回想一下fullibmpredit集群三元组的形式：



我们可以简单地用一个正常的5-gram插入这个簇状三元图，但是当然，将5-gram的概念与fullibmpredit的概念结合起来，使用一个簇状的5-gram会更有意义：

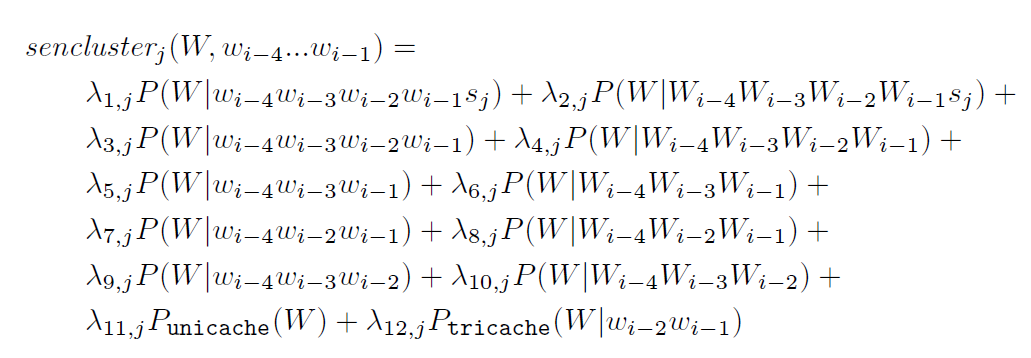


我们将遵循这种结合概念的思想，而不是简单地在本节中插入。这往往会导致良好的性能，但复杂的模型。

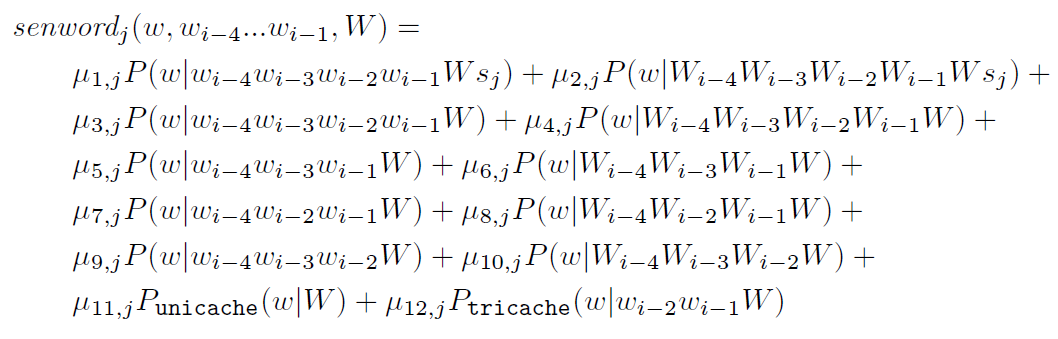
当我们将句子混合模型与缓存相结合时，我们需要回答更多的细节问题。Iyer和Ostendorf（1999）为每种句子类型使用了单独的缓存，将每个单词放入最可能的句子类型的缓存中。这将需要在我们的系统中进行大量的额外工作；此外，我们还进行了试验性实验，将句子混合模型和缓存与单个全局缓存相结合，发现组合带来的改进几乎等于单个改进的总和。由于Iyer和Ostendorf也得到了几乎等于和的改进，我们认为不使用单独的缓存是一种合理的组合技术。

我们的整体组合技术有点复杂。在最高的细节层次上，我们使用一个句子混合模型，在该模型中，我们对每种句子类型的特定句子模型进行求和。在特定的句子混合模型中，我们将不同的技术与预测聚类相结合。也就是说，我们首先结合特定句子、全局、缓存和全局跳过模型来预测下一个单词的簇，然后再根据给定的簇来预测单词本身。

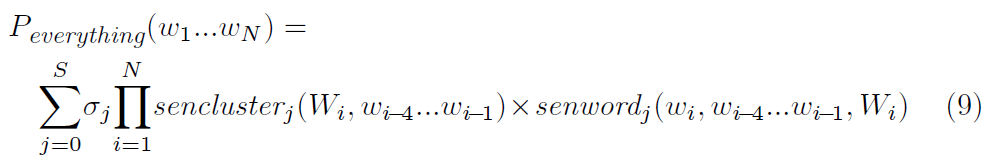
对于每个句子类型，我们希望用全局5-gram模型、三个跳过模型和两个缓存模型线性地插入特定于句子的5-gram模型。由于我们使用的是fullibmpredit集群，因此我们希望基于单词和集群来实现这一点。设λ1，j…λ12，j，μ1，j…μ12，j为插值参数。然后，我们定义了以下两个非常相似的函数。



接下来，我们定义了用于预测给定聚类的单词的类似函数：

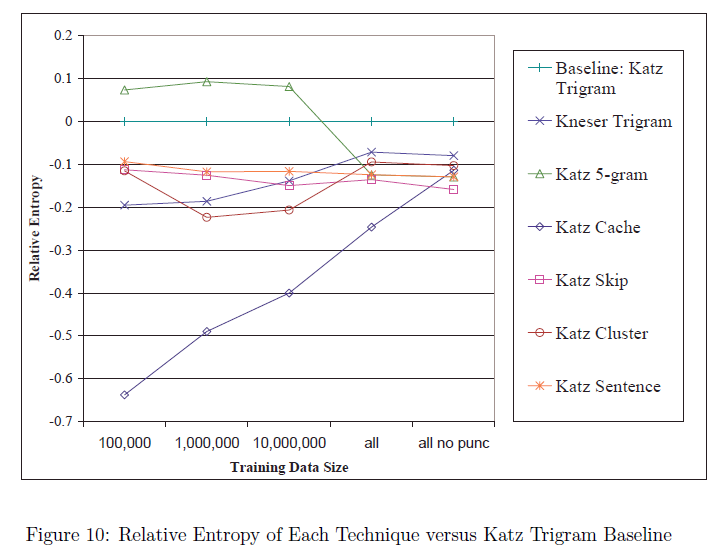


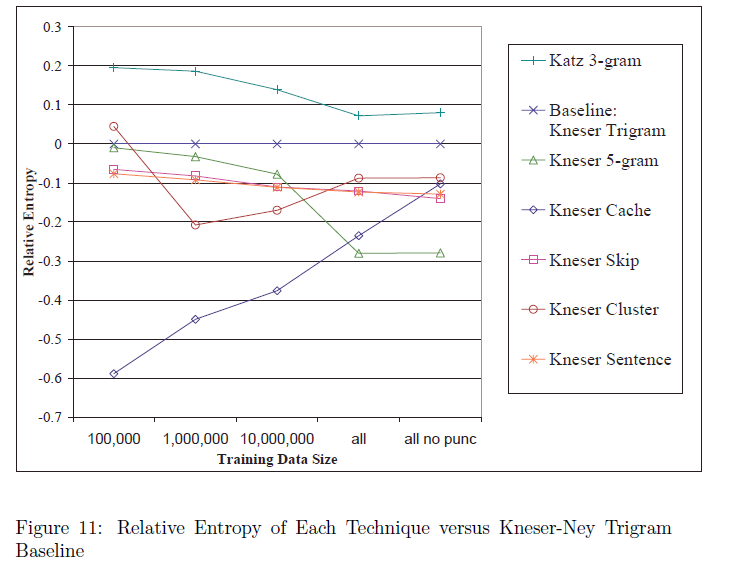
现在，我们可以写出我们的概率模型：

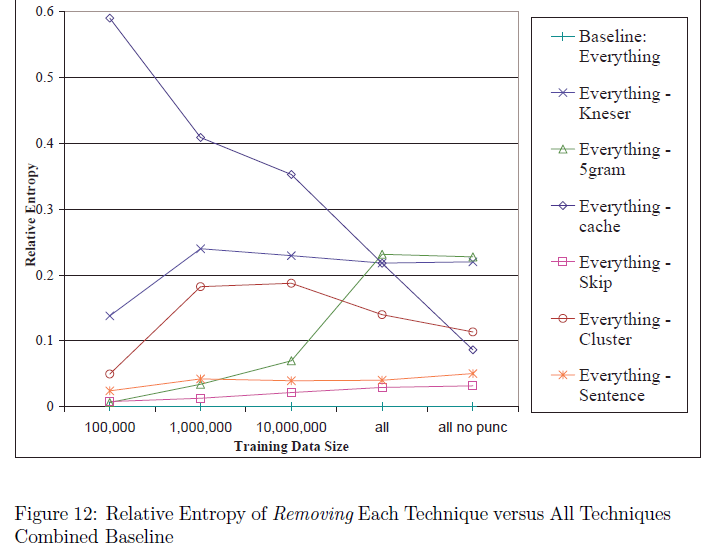


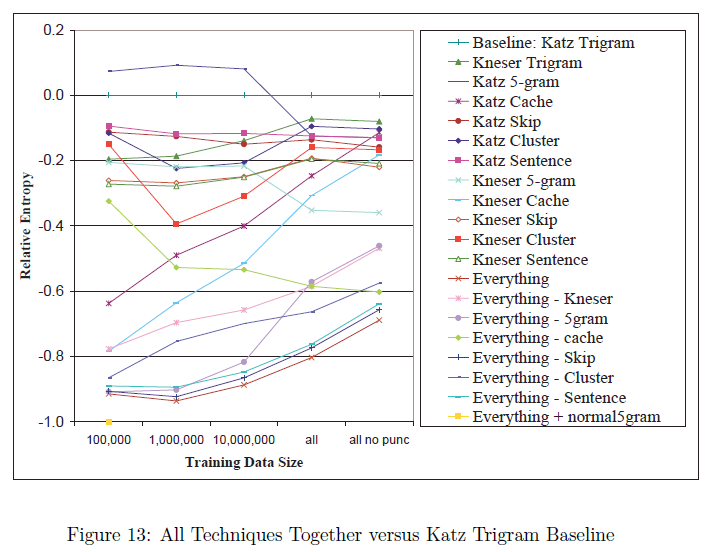
显然，将所有这些技术组合在一起并不容易，但是正如我们将要展示的，组合的效果是非常粗略的相加，而且这项工作是值得的。

我们做了几组实验。在这些实验中，当我们执行缓存时，它是一个单程序缓存和条件三元缓存；当我们使用句子混合模型时，我们使用4个混合；当我们使用三元跳转时，它是w y和wx；当我们使用5元跳转时，它是vwy与v xy和vwx插值。我们的单词错误率实验是在没有标点符号的情况下进行的，因此，为了便于比较，我们在本节中对“all no punc”进行了额外的熵实验，它与“all”集合相同，但没有标点符号。







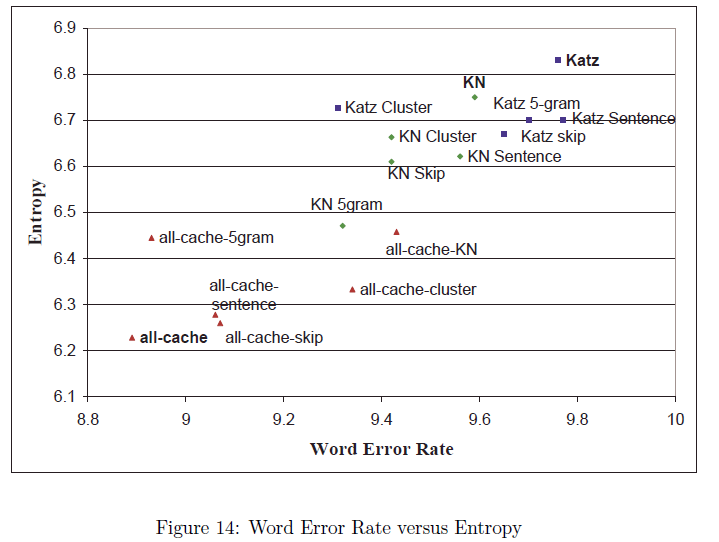


在第一组实验中，我们分别使用了每种技术和Katz平滑。结果如图10所示。接下来，我们使用相同的技术进行实验，但是使用Kneser-Ney平滑；结果如图11所示。除了5-grams，Kneser-Ney平滑显然是一个很大的增益外，所有与平滑无关的技术的结果都是相似的；事实上，如果没有Kneser-Ney平滑，5-grams在中小数据量下实际上是有害的。这是一个很好的协同作用的例子，两种技术结合在一起比单独使用两种技术帮助更多。缓存是中小型数据的最大增益，而与Kneser-Ney平滑结合使用时，5克是大型数据的最大增益。缓存在大多数数据大小上仍然是关键，但是当将Kneser-Ney平滑和聚类与其他技术结合时，它们的优势更加明显。

在下一组实验中，如图12所示，我们尝试从所有技术的组合中移除每种技术（方程式9）。基线是所有技术的组合-“一切”，然后我们显示，例如，除Kneser-Ney以外的所有技术的性能，除5克模型外的所有技术的性能，等等。在图12中，我们将所有技术一起显示，而不是Katz平滑三元图。我们在这个图上再加一个点。对于100000字，我们的Everything模型比普通的Katz模型低0.91位，这是一个很好的结果，但是我们知道100000字的模型受到了以最小数据大小进行fullibmpredit聚类的性能不佳的影响。因此，我们在一个正常的5-gram中插入一个单词级的值，这是一种表示为“Everything+normal5gram”的技术，这导致熵减少了1.0061-1位。这种增益显然没有实际价值——中小型的熵增益大部分来自缓存，缓存不会导致严重的字错误率降低。不过，这确实给了这篇论文一个很好的标题。在较大的尺寸下插值正常的5克基本上没有改善。

我们还进行了单词错误率实验，重新筛选了100个WSJ94开发和评估的最佳列表，大约600个语句。100个最佳列表中的1个最佳错误率为10.1%（我们的识别器模型甚至比用于重新排序的基线稍差），100个最佳错误率（重新排序的最小可能）为5.2%。我们无法通过使用缓存（当缓存由识别器的输出组成时）来提高字的错误率，并且当插值参数是在正确的历史记录上而不是在已识别的历史记录上估计时，实际上由于使用缓存而受到损害。

图14显示了每种技术的字错误率改进，可以使用Katz平滑、Kneser-Ney平滑，也可以从除缓存之外的所有技术中删除。对于单词错误率来说，最重要的一个因素是Kneser-Ney平滑法的使用，它本身带来了一个很小的增益，但同时也使跳转更加有效，而且5克的效果更好。集群也带来了显著的收益。在除聚类外的所有情况下，Kneser-Ney平滑模型都比相应的Katz平滑模型具有更低的误码率。奇怪的聚类结果（Katz熵更高）可能是由于噪声，也可能是因为我们分别为两个系统优化了聚类数量，优化了困惑，可能导致了一些聚类



这并不是降低单词错误率的最佳方法。总的来说，在Katz平滑基线模型上，我们得到了8.9%的字错误率降低。这是非常好的，虽然不如人们所期望的从我们的困惑减少。这可能是因为我们重新调整了n个最佳列表，而不是将语言模型直接集成到搜索中，或者重新调整了大型格。

**9 Implementation Notes**

实际上实现这里描述的模型并不简单。我们在这里给出一些关于最重要的实现技巧的注释，其中一些技巧相当新颖，在附录中给出了更多细节。首先，我们描述了我们的参数搜索技术。接下来，我们将讨论用于处理所构造的非常大的模型的技术。然后，我们考虑使研究成为可能的架构策略。最后，我们给出了一些实现我们的聚类方法的提示。

这项研究所需的模型规模非常大。特别是，许多技术对数据大小具有大致的乘法效果：从三联图移动到5克至少会导致2倍的增长；完全预测聚类几乎会导致4倍的增长；句子混合模型和跳过的组合会导致大约4倍的增长。那么，整个模型的大小大约是标准三元模型的32倍。建立和使用这样一个复杂的模型是不切实际的。

相反，我们使用一个简单的技巧。我们首先对测试数据（文本或n个最佳列表）和heldout数据（用于参数优化）进行遍历，并确定实验所需的完整值集。

然后，我们查看培训数据，只记录这些值。这大大减少了运行我们的实验所需的内存量，大约将其减少到可管理的1.5G字节。我们使用的另一个技巧是将测试数据分成几块-测试数据越少，我们需要存储的值就越少。附录描述了我们验证这种“作弊”导致的结果与非作弊模型得到的结果相同的一些方法。

系统的仔细设计也是必要的。特别是，我们使用了“model”的概念，一个带有一组参数的抽象对象，它可以返回给定历史记录的单词或类的概率。我们创建了可以组成其他模型的模型，方法是在单词级、类级或句子级对它们进行插值，甚至像预测聚类那样将它们相乘。这使我们能够以多种方式组合实现缓存、各种平滑技术等的基本模型。

我们的平滑技术和插值都需要优化自由参数。在某些情况下，这些自由参数可以从训练数据中通过省略一个out技术来估计，但是通过使用参数的Powell搜索，优化延迟数据的复杂性，可以获得更好的结果（Chen和Goodman，1999），这就是这里使用的技术。允许我们共同优化所有参数，而不是像通常所做的那样，先优化一个模型，然后优化另一个模型，然后优化它们的插值参数。这也使得在本文中的每一个实验中，相对容易地找到该模型的最佳参数设置，而不是使用次优猜测或相关模型的结果。

尽管所有的平滑算法都是为这项研究而重新实现的（只重用少量代码），但其细节却紧跟着Chen和Goodman（1999）。这包括我们对Katz平滑和Kneser-Ney平滑都使用了单格分布的加性平滑。也就是说，我们发现了一个常数b，它被添加到所有unigram计数中；这导致在小的训练数据情况下更好的性能，并且允许我们比较不同训练大小的困惑，因为没有unigram接收到0个计数，这意味着0个概率从未返回。

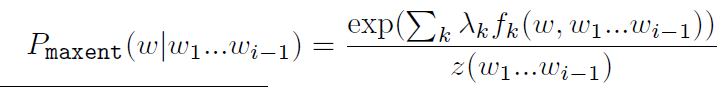
生成集群的技术并不缺乏，而且似乎很少有证据表明，优化同一标准的不同技术导致集群的质量显著不同。然而，我们注意到，不同的算法可能需要显著不同的运行时间。尤其是，聚集聚类算法可能需要比自顶向下的分裂算法多得多的时间。在自顶向下的分裂算法中，可以使用额外的技巧，包括Buckshot技术（Cutting et al.，1992）。我们还使用了改编自Brown等人的计算技巧。（1992年）。关于所使用的集群技术的更多细节，请参见附录B.4。

**10 Other techniques**

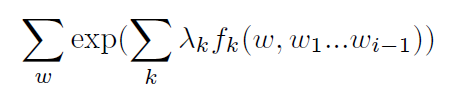
在本节中，我们将简要讨论最近对语言建模感兴趣的其他一些技术；我们已经用其中的一些技术做了一些实验。这些技术包括最大熵模型、整句最大熵模型、潜在语义分析、基于句法分析的模型和基于神经网络的模型。Rosenfeld（2000）给出了一个更广泛、不同的领域视角，以及本文所讨论的技术的额外参考。

**10.1 Maximum Entropy Models**

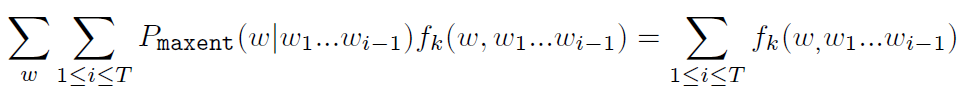
最大熵模型（Darroch和Ratcliff，1972）自Rosenfeld（1994）引入语言建模以来，受到了广泛的关注。最大熵模型是



其中z是一个规范化函数，简单地设置为



λk是实数，通过广义迭代标度（GIS）等学习算法学习（Darroch和Ratcliff，1972）。fk是其输入的任意函数，通常返回0或1。它们可用于捕获n个程序、缓存和集群，并跳过模型。它们还可以用于更复杂的技术，如下面描述的触发器。fk的一般性和强大性是最大熵模型的一个重要特点。最大熵模型还有其他吸引人的地方。给定训练数据w1…wT，可以求出λs，使得对于每个k，



换言之，在给定模型的情况下，我们期望fk发生的次数是我们在训练数据中观察它的次数。也就是说，我们可以在单词w和历史w1…wi-1上定义任意谓词，并构建一个模型，使所有这些谓词都像我们观察到的那样经常是真的。此外，该模型在满足这些约束条件的同时，使熵最大化，即尽可能平滑（接近均匀分布）。这是一种相当诱人的能力。

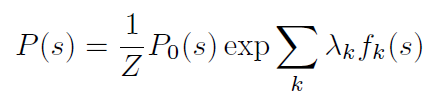
此外，最近有两项研究使我们对最大熵模型的使用特别乐观。第一种是Chen和Rosenfeld（1999b）的平滑技术，这是第一种与KneserNey平滑同样有效（或优于KneserNey平滑）的最大熵平滑技术。第二个是我们自己最近的加速技术（Goodman，2001），这导致了比传统的最大熵训练技术（可能非常慢）高出35倍或更多的加速。

对于最大熵模型，有乐观和悲观的理由。一方面，最大熵模型导致了一些最大的困惑减少。Rosenfeld（1994）报告说，当使用最大熵模型，结合缓存和传统的三元组时，困惑减少了39%。另一方面，当我们与可比较的内插n-gram模型进行比较时，我们自己的最大熵试点实验是否定的：我们不知道任何研究中，最大熵模型比可比较的n-gram模型有显著的改进。此外，即使使用我们的加速技术，最大熵模型的训练也非常耗时，而且在测试情况下使用也很慢。总的来说，最大熵技术对于现实世界来说可能太不实际了。

最大熵模型值得一提的一个特殊方面是单词触发器（Rosenfeld，1994）。在最大熵模型中，词触发器是最重要的增益来源。在触发器模型中，单词“school”增加了自身的概率，以及类似单词（例如 “teacher”）的概率。Rosenfeld通过使用单词触发器获得了大约25%的复杂度降低，尽管当与已经包含缓存的模型结合时，增益可能会降低到7%-15%。Tillmann和Ney（1996）结合一个已经有缓存的模型实现了大约7%的困惑减少，Zhang等人。（2000）报告减少了11%。

**10.2 Whole Sentence Maximum Entropy Models**

最大熵方法的最新变化是整句最大熵方法（Rosenfeld等人，2001）。在这种变化中，预测的是整个句子的概率，而不是单个单词的概率。这允许使用整个句子的特征，例如连贯性（Cai et al.，2000）或可分解性，而不是词级特征。如果s代表整个句子，那么整个句子的最大熵模型就是



其中P0（s）是一些起始模型，例如n-gram模型。注意，在本例中，标准化因子Z是一个常量，无需为每个上下文分别计算标准化因子。事实上，在大多数情况下，根本不需要计算它。他们的支持者认为，这是这些模型的主要优点之一，尽管其他技术（Goodman，2001）也减轻了标准化的负担。

整句法有几个问题。首先，整句最大熵模型的训练特别复杂（Chen和Rosenfeld，1999a），在实际中需要蒙特卡罗-马尔可夫链等抽样方法。第二，当一个完整的句子模型被分成所有的单词时，它的好处可能很小。考虑一个特征，比如可分解性：根据某种语法，句子是可分解的吗。假设这个特性为每个句子提供了完整的信息。（注意，在实践中，语法的广度足以解析几乎所有语法句子，而广度足以解析许多不符合语法的句子，从而减少了它提供的信息。）现在，在一个平均20个单词的句子中，1个比特的信息每字只减少0.05比特的熵，对于一个非常复杂的特征来说，相对较小的增益。我们在这些模型中看到的另一个问题是，它们的大部分特性可以通过其他方式捕获。例如，全句最大熵模型的一个普遍提倡的特征是“连贯性”，即一个句子中的词都应该有相似的主题。但其他技术，如缓存、触发和句子混合模型，都是提高句子连贯性的方法，而不需要花费整个句子的方法。因此，我们对这种方法的长期潜力持悲观态度

**10.3 Latent Semantic Analysis**

Bellegarda（2000）指出基于潜在语义分析（LSA）的技术是非常有前途的。LSA类似于主成分分析（PCA）和其他降维技术，似乎是一种很好的方法来降低困扰语言建模的数据稀疏性。与四千二百万字的卡茨三部曲相比，这项技术显著减少了困惑（约20%）和字错误率（约9%相对）。正式地将这些结果与利用类似长期信息的传统缓存结果进行比较会很有趣。Bellegarda通过使用基于LSA的聚类技术，在这些结果的基础上得到了额外的改进；复杂度的减少与传统IBM风格的聚类技术的复杂度的减少类似。

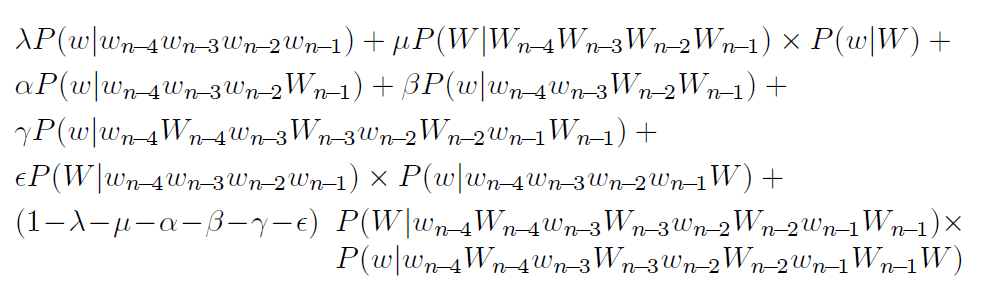
**10.4 Neural Networks**

Bengio等人最近在使用神经网络进行语言建模方面做了一些有趣的工作。（2000年）。为了解决数据稀疏性问题，首先将每个词映射成30个连续特征向量，然后根据这些连续特征学习不同输出的概率。通过反向传播学习映射的方法与网络中其他权值的学习方法相同。最好的结果是大约25%的困惑减少了一个基线删除插值风格的三元图。

我们想看看神经网络模型与标准的clus-details-tering模型的比较。我们在相同的语料库、词汇表上进行了一些实验，并将实验分为训练、测试和heldout数据。我们很感激他们为我们提供了相关的数据。首先，我们验证了它们的基线是正确的；使用简单的插值，我们得到了350.6的困惑，而对于删除的插值样式基线，我们得到了348的困惑，这看起来非常合理。另一方面，内伊的一个揉三角肌平滑有一个316的困惑。剩下的实验是用Kneser-Ney平滑法。

接下来我们尝试了一个ibm集群4-gram模型，它在某些方面类似于具有三个上下文单词的神经网络模型。聚类模型的困惑度为271，而神经网络的困惑度为291。然而，291数字不是用三元函数插值的，而我们的集群模型是这样的。我们还运行了一个6-gram的ibm集群模型。该模型的困惑度为271，比最好的神经网络模型评分258差5%左右；258模型是用三角形插值的，所以这是一个公平的比较。最后，我们尝试构建一个“全力以赴”的模型。这并不是我们能建立的最好的模型——例如，它不包括缓存——但它是我们能建立的最好的模型，使用与前5个单词中的最佳神经网络相同的输入。实际上，我们只使用了前4个单词，因为我们没有从第5个单词中得到太多好处。这个模型的形式





困惑度为254.6，比最好的神经网络好2%。最佳的神经网络模型依赖于三元模型的插值，并且只对低频事件使用三元模型。由于他们的三元模型有一个相对较高的困惑348，相比Kneser-Ney-smooted的316，我们假设一个显着的困惑减少可以得到简单地使用Kneser-Ney-smooted的三元。

这意味着神经网络模型的结果确实非常好。看看神经网络模型和传统的聚类模型有多大的共同点是非常有趣的。一个简单的实验是用一个聚类模型来插值一个神经网络模型，看看有多少增益是可加的。考虑到所用神经网络的相对简单性和非常好的结果，这似乎是一个非常有前途的研究领域。

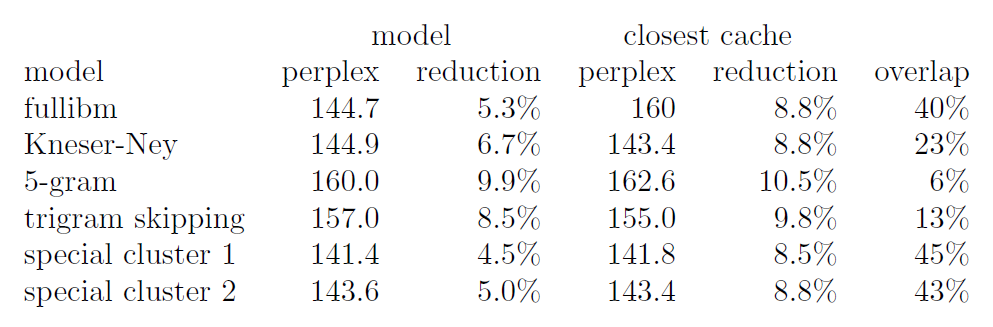
**10.5 Structured Language Models**

结构化语言模型（SLMs）是语言建模研究中最有趣、最令人兴奋的新领域之一。第一个成功的结构化语言模型是Chelba和Jelinek（1998）的工作，而其他最近的模型则更为成功（Charniak，2001）。结构化语言模型背后的基本思想是，一个措辞恰当的统计解析器可以被看作是语言的生成模型。此外，统计分析器通常可以考虑较长距离的依赖关系，例如主题与其直接或间接对象之间的依赖关系。这些依赖关系可能比前两个词更有用，如三元模型所捕获的那样。Chelba能够比基准三叉模型减少11%的困惑，而Charniak则达到了令人印象深刻的24%的减少。

我们假设结构化语言模型细节的许多好处可能与其他技术（如跳过、聚类或5grams）是多余的。一个模型拥有多少信息，而另一个模型已经捕获了多少信息，这是一个困难的问题。虽然在两种不同的模型下测量单词条件熵的公式是众所周知的，但它们依赖于计算联合概率。如果这两个模型已经被稀疏地估计了，比如传统的三元组和结构化语言模型，那么计算这些联合概率是没有希望的。所以，我们决定采用更近似的方法。一种简单实用的方法是尝试对两个模型进行插值。乍一看，如果插值导致没有增益，那么模型必须捕获相同的信息，并且如果增益在公共基线上是相加的，那么信息是独立的。不幸的是，至少第一个假设是错误的。特别是，当将结构化语言模型与使用trigram缓存插值的Kneser-Ney平滑三元图进行比较时，我们假设信息的重叠最小，与单独的缓存模型/Kneser-Ney三元图相比，插值几乎没有任何增益。这仅仅是因为插值是一种结合信息的粗糙方法（尽管我们不知道还有更好的方法）。cache模型比结构化语言模型好得多，插值权重都归cache模型，因此结构化语言模型无能为力；除了cache模型具有较低的复杂度外，我们什么也学不到，这是我们已经知道的。

我们当时的策略有点复杂，也有点难以解释。首先，我们使用两种系统的版本，使用简单的插值进行平滑。这是SLM程序和我们的工具包中唯一已经实现的平滑技术。这消除了平滑作为一个因素。接下来，我们尝试将SLM与具有不同缓存大小的trigram进行比较（当然，我们从未缓存过超出文档边界的内容），并将其与SLM进行插值。我们假设SLM和缓存，无论大小，都捕获了大致正交的信息。这让我们计算出，如果这两个模型不相关的话，我们期望的困惑减少量。例如，基线SLM，没有用三元组插值，有167.5的困惑。fullibm集群模型的复杂度为144.7；类似地，一个包含160个单词上下文的trigram缓存的trigram的复杂度为143.4，与集群模型差不多。SLM与160字上下文缓存的结合复杂度为130.7，比单独缓存降低了8.8%。当与fullibm集群模型结合使用时，困惑度为137.0，比fullibm集群模型降低了5.3%。所以，如果这些改进是不相关的，我们假设减少8.8%，而我们只看到5.3%。这是我们预期的降幅的60%。因此，我们说，与聚类模型的重叠大约是40%。

下表显示了我们各种实验的结果。“model”列描述了我们用SLM插值的模型。第一个“复杂”列显示模型的复杂度，第一个“简化”列显示使用SLM对模型进行插值后得到的简化度。第二个“困惑”列显示了最相似的缓存模型的困惑，第二个“减少”列显示了使用SLM插值此缓存模型所带来的困惑减少。最后一列“重叠”显示了第一次减少和第二次减少之间的比率：模型和SLM之间的假定重叠。



三元跳跃模型是一个所有对的模型，通过5克的水平。特殊的cluster 1和特殊的cluster 2模型是集群跳过模型，旨在捕获与我们假设的结构化语言模型所做的相似的上下文。

我们本想研究一些组合，例如Kneser-Ney平滑与fullibm集群，但是我们测试的最佳缓存模型的困惑度为141.8，而Kneser-Ney集群模型的困惑度要低得多。

从这些结果中很难得出任何过于有力的结论。一个奇怪的结果是与Kneser-Ney平滑法有很大的重叠-23%。我们怀疑，在结构化LM中，将语言模型分解成单个组件会产生平滑效果。或者，也许我们的整个评估是有缺陷的。

我们还研究了单个单词概率的相关性。我们检查了每个单词的每个模型与基线三元模型的概率的差异。然后我们测量了这些差异的相关性。

这些结果与其他结果相似。

用其他结构化语言模型进行同样的实验会很有趣。不幸的是，目前最好的，Charniak's，并不能预测从左到右的单词。因此，Charniak只能在句子层面上进行插入。句子级的插入会使这些实验更难解释。

总体而言，我们有理由相信，SLM带来的增长中有很大一部分（大约40%或更多）来自于类似于集群的信息。这并不奇怪，因为SLM对部分语音标签和非终端对象有明确的模型，这些对象的行为类似于词簇。我们用Ciprian Chelba做了这些实验，得出了相反的结论：我们得出的结论是杯子是半空的，而Chelba得出的结论是杯子是半满的。

**11结论**

**11.1以前的组合**

以前很少有研究试图将两种以上的技术结合起来，甚至之前的大多数研究都没有特别系统地将两种技术结合起来。此外，这两种技术的典型组合之一是基于缓存的语言模型。由于缓存模型只是用另一个模型进行线性插值，因此没有太大的交互空间。

以前的几篇论文确实值得一提。最近一次是马丁等人的。（1999年）。他们将插值的Kneser-Ney平滑、类、单词短语和跳过结合起来。不幸的是，它们与我们使用的基线不同，而是与它们所称的内插线性贴现（一种较差的基线）相比。然而，他们比内插的Kneser-Ney也得到了改善；他们在这一基线上获得了大约14%的困惑减少，而我们在同一基线上获得了34%的困惑减少。它们在聚类方面的改进与我们的类似，在跳过模型方面的改进也与我们的类似；它们在单词短语方面的改进（我们不使用这些短语）很少（约3%）；因此，结果上的差异主要是由于我们实现了附加技术：缓存、5-grams和句子混合模型。它们在插值Kneser-Ney上的字错误率减少了6%，而我们的是7.3%。我们假设错误率的降低与困惑度的降低不成正比的原因是两方面的。首先，我们减少困惑的4%来自缓存，我们没有在字错误率结果中使用缓存。其次，他们能够直接将简单的模型集成到识别器中，而我们需要重新存储n个最佳列表，从而减少我们可以纠正的错误数。

另一件值得一提的作品是罗森菲尔德（1994）的作品。在这项工作中，结合了大量的技术，使用最大熵框架和插值。许多技术是在多种训练数据大小下测试的。最佳系统使用缓存和最大熵系统插值Katz平滑三元图。最大熵系统包含简单的跳跃技术和触发。最好的系统在与我们相似的数据上有32%到39%的复杂度降低。罗森菲尔德仅从单词触发中就可以得到大约25%的减少（第45页），这是我们不使用的技术。总的来说，Rosenfeld的结果是非常好的，如果使用更现代的技术，比如Kneser-Ney平滑三元模型（内插），使用更快的机器和更新的训练技术实现更小的截断，或者使用更新的技术平滑最大熵模型，则其结果很可能超过我们的结果。Rosenfeld用一些简单的基于类的技术进行了实验，但没有成功；我们假设也可以使用更现代的分类技术。罗森菲尔德在使用无监督自适应（与我们所采用的方法相同）时，字的错误率降低了10%。他达到了13%的假设超视距-用户立即纠正错误。

令人惊讶的是，几乎没有其他工作结合了两种以上的技术。我们所知道的唯一值得注意的研究是ofWeng等人的研究。1997年，他进行了多个语料库、四元图和类似于句子混合模型的基于类的方法相结合的实验。将所有这些技术结合起来，可以从仅限Hub4的语言模型中减少18%的困惑。这个模型是在不同于我们的模型的文本类型上进行训练和测试的，因此无法与我们的工作进行比较。

**11.2所有进入这里的人，都要放弃希望**

在这一节中，我们认为有意义的、实际的降低单词错误率是没有希望的。我们指出，三联图仍然是事实上的标准，并不是因为我们不知道如何打败它们，而是因为没有任何改进可以证明成本是合理的。我们几乎没有证据表明熵比困惑更能衡量进步，而且熵的改善也很小。我们得出的结论是，大多数语言建模研究，包括我们的研究，通过与一个草人基线进行比较，忽略实现的成本，呈现出没有实质内容的进步幻觉。我们继续描述，如果有的话，语言建模研究是值得的。

**11.2.1实际的字错误率降低是没有希望的**

与trigram基线相比，大多数语言建模改进需要的空间要多得多，而且通常需要的时间也要多得多。大多数语言建模论文都包含一个段落，比如“Trigram模型是最先进的”。三元模型显然是脑损伤。我们做了一些稍微不那么伤脑筋的事。我们的模型比三元组的复杂度要低得多，字的错误率也要低得多。”然而，论文没有提到，但几乎普遍适用的是，得到的模型比简单的三元组模型需要更多的空间和/或时间。三元模型是最先进的，因为它们被用于大多数商业语音识别器中，并且至少是大多数研究识别器的第一步。这是因为以下两个原因：它们相当节省空间，并且可以使用各种技巧将trigram存储与搜索集成在一起。例如，在我们微软的识别器中，我们使用一种存储方法，其中三元组被存储为语音树（Alleva等人，1996）。许多更复杂的语言建模技术，特别是聚类技术，与这种类型的存储不兼容，因此在实际的、优化的识别器中效率要低得多。

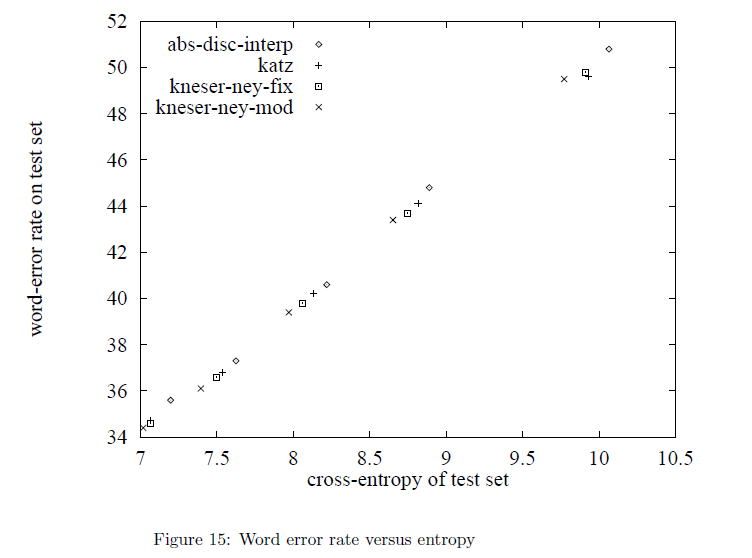
在实践中考虑我们实现的每一种技术。句子混合模型需要比相应的全局模型更多的存储空间，因为每个混合都必须存储不同的模型，并且仍然需要存储一个全局模型，以便使用特定于混合的模型进行插值。此外，如果集成到识别器的第一个过程中，则必须为每个混合类型探索不同的路径，从而显著减慢识别器的速度。缓存有一些我们已经讨论过的问题：虽然它不会显著增加识别器的存储空间，但它要求用户在每句话之后更正所有错误；否则，错误将被“锁定”。我们在这里描述的这种群集模型显著增加了存储空间，并且与用于搜索的前缀树的种类。5克需要更多空间比三联图还慢，搜索速度也慢。Kneser-Ney平滑在理论上有改进，但在实践中，大多数语言模型都是用高计数截止值来构建的，这样可以节省空间，加快搜索速度；而用高计数截止值来构建的，平滑并不重要。跳过模型需要更复杂的搜索和更大的空间，并导致边际改进。简而言之，我们所研究的所有改进在可预见的未来都不可能对实际系统产生影响，除了缓存，缓存通常是可用的，但通常在默认情况下关闭，因为用户不喜欢或不能接受一致地更正识别器错误的培训。

我们的结果调查并重新实现了自Katz（1987）以来引入的大多数有前途的语言建模技术。困惑的改善是有史以来最大的报道之一，但字错误率的改善相对较小。这在一定程度上反映了我们对实验中的困惑而不是错误率的强调，以及实验中的缺陷。也许重新划分格而不是n个最佳列表会带来更大的改进。也许将语言建模集成到搜索中会有所帮助。也许尝试优化参数和技术来减少单词错误率，而不是减少困惑会更好。但是，尽管有这些缺陷，我们认为我们的结果是相当有代表性的。另一些人尽最大努力改进语言建模，结果好坏参半。例如，Martin等人。（1999）从一个公平的基线（更容易的基线）得到12%的相对错误率，比我们的稍好。但大多数改进的成本都很高——语言模型的大小是原来的四倍，CPU时间是原来的32倍。简单地增加混合成分的数量或放宽声学模型中的搜索阈值，就可以更容易、更便宜地获得更大的改进。事实上，只要使用更多的语言模型训练数据，就可能有类似的改进。

**11.2.2 Perplexity is the wrong measure**

在这个亚类中，我们几乎没有证据表明单词错误率与熵的相关性比与困惑的相关性好。由于相对熵的减少比相对困惑的减少要小得多，因此很难得到有用的字错误率的减少。

我们考虑做以下实验：将部分测试数据增量地添加到我们的训练集中，并使用它来计算困惑/熵和单词错误率。然而，我们注意到实际进行实验是不必要的，因为（在一个稍微理想化的世界中）我们可以预测结果。如果我们加上x%的测试数据，那么我们就有了这个x%的完美信息。有了一个足够强大的n-gram模型（比如1000克），并且没有搜索错误，我们的语音识别器将得到准确的部分，从而字错误率相对减少了x%。类似地，它将需要0位来预测我们已经看到的训练数据部分，从而导致熵减少x%。这一理论分析使我们假设了熵与字误率之间的线性关系。



当然，人们可以进行更真实的实验，看看困惑和熵是否与单词错误率有更好的对应关系。我们最近的实验并没有揭示这个问题——在我们测试的相对狭窄的困惑范围内，单词错误率和熵之间的相关性，以及单词错误率和困惑之间的相关性是相同的。然而，考虑陈和古德曼（1998）以前的工作。在图15中，我们展示了关于单词错误率和熵之间的相关性的实验。很明显，这种关系很强，在很大范围内呈线性，包括大约0的截距，正如我们的分析预测的那样。

还要考虑其他领域，如文本压缩。这里，简单地说，熵和目标函数之间的关系，所需的比特数来表示数据，又是线性的。

这些都不是有力的论据，特别是考虑到许多研究者已经得到了熵和单词错误率不相关的结果。尽管如此，它们还是令人担忧。如果是真的，它们意味着我们必须大大减少困惑，才能减少有意义的单词错误率。例如，从100到90，10%的复杂度减少，对应于从6.64位到6.49位的减少，这仅仅是2%的熵减少。看似有意义的困惑减少是一个平凡的熵减少。我们担心这意味着不太可能在商业上减少困惑。

**11.2.3 Progress is Illusory**

在语言建模研究中，一个常见且相关的陷阱是草人基线。大多数语言建模论文都指出了trigram模型的一个问题，通常是它们无法捕获长距离的依赖关系，然后继续给出一些方法来建模更长的上下文。这些论文几乎从未与平滑的5-gram进行过比较，5-gram是一种简单、明显的方法，可以使trigram捕获更长的上下文，更不用说与具有缓存的trigram（一种10多年前就知道的技术）或更新的模型（如句子混合模型）进行比较了。如果我们不在乎速度和空间，结果是另一种打败三联图的方法。这些新技术是否比以前的技术更好，或者可以与它们结合进行更大的改进，或者提供比其他技术更好的速度或空间权衡，只是很少有人探索。当使用缓存比较时，它通常是在将缓存与其自身模型相结合的上下文中，而不是与基线三元图相结合的上下文中。困惑导致了介绍、结论和摘要，它们通常会与使用缓存的技术进行比较，而不使用缓存的三元组进行比较。有时缓存看起来不像缓存。例如，如果熵达到最大，它可能是一个触发模型。但由于最有用的触发器只是自触发器，基本上是缓存特性，所以它毕竟只是一个缓存。

糟糕的实验技术和误导性的论文绝非语言建模所独有。但是，三联图基线在实践中的普及，让研究人员直面称之为“最新技术”，使得人们特别容易产生进步的错觉

**11.2.4 So, what now?**

考虑到前面的章节，有人应该做任何语言建模研究吗？也许 吧。有四个方面我看到了一些希望：语言模型压缩、语言模型扩展、语言建模的新应用和基础研究。

尽管Kneser（1996）、Seymore和Rosenfeld（1996）、Stolcke（1998）、Siu和Ostendorf（2000）以及我们（Goodman和Gao，2000）都做了很好的工作，但语言模型压缩或修剪是一个研究相对较少的领域。这里需要照顾。例如，Seymore等人的技术。Stolcke几乎可以用于任何识别器，但是我们自己的技术使用聚类，并且在一些语音识别器中与搜索的交互很差。不过，这似乎是一个可以取得进展的领域。类似地，还没有全面的工作显示我们在本文中研究的所有技术的空间/复杂度权衡，也没有任何关于修剪插值模型的工作，例如跳过模型或句子混合模型。

语言建模研究的另一个领域是语言模型自适应。一个非常常见的产品场景只涉及非常少量的域内培训数据和大量的域外数据。在这个领域已经有了相当数量的研究。许多研究并没有简单地将域内和域外语言模型插值在一起，这在我们的实验中非常有效。这方面最好的研究可能是Iyer和Ostendorf（1997）。我们怀疑可以找到更好的技术，但我们自己的尝试失败了。

语言模型工作得很好，而且理解得很好，可以应用到许多领域（Church，1988；Brown等人，1990；Hull，1992；Kernighan等人，1990；Srihari和Baltus，1992）。与其试图改进它们，不如尝试以新的方式使用它们可能会有成效。几乎所有的机器学习问题都有可能使用语言建模技术作为解决方案，而识别语言模型工作良好的新领域可能与尝试进行基础研究一样有用。

语言建模基础研究还有许多未完成的领域。这些都不会产生巨大的实际影响，但它们将有助于推动这一领域的发展。其中一个区域是Kneser-Ney平滑的连续版本。插值的KneserNey平滑非常好。无论我们使用的是哪种模型，使用Kneser-Ney平滑比使用任何其他我们尝试过的平滑技术都更有效。但Kneser-Ney平滑仅限于离散分布；它不能处理小数。分数计数在期望最大化（EM）算法中非常常见。这意味着我们目前还不知道如何对通过EM学习到的分布进行平滑，例如大多数隐马尔可夫模型或概率上下文无关文法的实例。连续版的Kneser-Ney可用于所有这些领域。一个需要更多研究的相关领域是软聚类与硬聚类。硬聚类将每个单词分配给一个集群，而软聚类则允许将同一个单词放在不同的集群中。基本上还没有将硬聚类与软聚类进行比较的工作，但是有几种软风格的技术，包括Bengio等人的工作。（2000）和Bellegarda（2000）都取得了很好的成功，暗示这些技术可能更有效。我们没有尝试软聚类的一个原因是，我们不知道如何对其进行适当的平滑：软聚类模型具有分数，最好与连续版本的Kneser-Ney平滑一起使用。

**11.2.5 Some hope is left**

总之，语言建模是一个非常困难的领域，但并不是一个完全没有希望的领域。基础研究还是有可能的，而且不断有新的，如果不实用的话，那么肯定会有有趣的语言建模结果。在一些领域，有用的语言建模研究似乎也很有希望。但是语言建模，就像大多数的研究一样，也许更是如此，并不是一个容易让人心灰意冷或抑郁的领域。

**11.3 Discussion**

我们相信我们的结果——在非常小的数据集上减少了50%的困惑，在大的数据集上减少了41%（在没有标点符号的数据上减少了38%——是语言建模方面有史以来最好的报告，通过从公平基线的改进来衡量，一个没有计数截止的Katz平滑三元模型。我们还系统地研究了平滑、高阶n-grams、跳过、句子混合模型、缓存和聚类。

我们最重要的结果可能是插值KneserNey平滑在我们所研究的每种情况下的优越性。我们之前的研究（Chen和Goodman，1998）表明，Kneser-Ney平滑在训练数据大小、语料库类型和n-gram顺序方面始终是最好的技术。我们现在已经证明了它也是跨集群技术中最好的，并且它是构建高性能语言模型的最重要因素之一，特别是使用5-grams的模型。

我们仔细检查了高阶n-grams，发现性能改进在大约5-gram级别上处于平稳状态，并且我们在20-gram级别上给出了第一个结果，表明在超过7-gram级别上没有任何改进。

我们系统地研究了跳绳技术。我们研究了类似三叉神经的模型，发现使用成对到5克的水平几乎可以获得所有的好处。我们还对5-gram跳过模型进行了实验，找到了3个上下文的组合，获得了大部分好处。

我们仔细研究了句子混合模型，结果显示，通过增加混合的数量，可以比之前预期的有更多的改进。在我们的实验中，将句子类型的数量增加到64，可以比少量类型提高近一倍。

我们的缓存结果表明，在中小型训练数据中，缓存是目前最有用的减少困惑的技术。它们还表明，一个三元缓存可以导致一个unigram缓存的熵减少几乎两倍。

接下来，我们系统地研究了集群，尝试了9种不同的技术，找到了一种新的集群技术fullibmpredit，它比标准的ibm集群要好一点，并检查了集群改进的局限性。

我们还表明，在某些情况下，聚类性能可能依赖于平滑。

我们的字错误率减少了8.9%，从结合所有技术，除了缓存也是非常好的。

最后，我们把所有的技术放在一起，根据训练数据的大小，可以减少38%-50%的困惑。这些结果与最近报告的其他组合结果（Martin等人，1999年）相比较有利，其中，基本上使用这些技术的一个子集，从可比基线（绝对折扣）来看，困惑减少了一半。结果表明，平滑是语言建模中最重要的因素，它与其他技术的交互作用不容忽视。

在某些方面，我们的结果有点令人沮丧。我们建立的整体模型是如此复杂、缓慢和庞大，对于一个产品系统来说是完全不实际的。尽管有这么大的规模和复杂性，我们的字错误率的改进是适度的。对我们来说，这意味着语言模型研究对语音识别器的实用价值是有限的。另一方面，语言建模对于语音识别以外的许多领域都是有用的，并且是机器学习技术的一个有趣的测试平台。

此外，我们的部分成果令人鼓舞。首先，它们表明语言建模的进展仍在继续。例如，在我们的系统中，一个重要的技术，句子混合模型，只有几年的历史，而且，正如我们所展示的，它的潜力只被部分挖掘。

同样，这么多技术的结合也是新颖的。此外，我们的结果表明，这些不同技术的改进大体上是相加的：根据简单地将图11的结果相加，可以预期最大训练量的改进为.9位，而总数大约为.8位，非常相似。这意味着进一步的增量改进也可能导致最佳模型的改进，而不是简单的重叠或冗余。

正如我们在第10节中所指出的，目前还有许多其他有希望的语言建模技术，如最大熵模型、神经网络、潜在语义分析和结构化语言模型。弄清楚如何将这些技术与我们已经实现的技术结合起来应该会带来更大的收益，但也会带来更复杂的模型。