中图分类号：TP391
论 文 编 号：10006SY1506330

北京航空航天大學

硕士学位论文

基于神经网络的高性能语言建模技术研究

作 者 姓 名　姜楠
学 科 专 业　计算机应用技术
指 导 教 师　荣文戈　副教授
培 养 院 系　计算机学院

Nerual Network Based Highly Efficient Language Modeling

A Dissertation Submitted for the Degree of Master

Candidate: Jiang Nan

Supervisor: Associate Prof. Rong Wenge

School of Computer Science & Engineering

Beihang University, Beijing, China

中图分类号：**TP391**

论 文 编 号：**10006SY1506330**

硕 士 学 位 论 文

基于神经网络的高性能语言建模技术研究

作 者 姓 名 姜楠　　申请学位级别 工学硕士

指导教师姓名 荣文戈　　职　　称 副教授

学 科 专 业 计算机应用技术　　研 究 方 向 自然语言处理

学 习 时 间 自 2015 年 09 月 01 日　起　　至 2018 年 月 日止

论文提交日期 2018 年 月 日　　论文答辩日期 2018 年 月 日

学位授予单位 北京航空航天大学　　学位授予日期 2018 年 月 日

关于学位论文的独创性声明

本人郑重声明：所呈交的论文是本人在指导教师指导下独立进行研究工作所取得的成果，论文中有关资料和数据是实事求是的。尽我所知，除文中已经加以标注和致谢外，本论文不包含其他人已经发表或撰写的研究成果，也不包含本人或他人为获得北京航空航天大学或其它教育机构的学位或学历证书而使用过的材料。与我一同工作的同志对研究所做的任何贡献均已在论文中作出了明确的说明。

若有不实之处，本人愿意承担相关法律责任。

学位论文作者签名：_____ 日期：　　年　　月　　日

学位论文使用授权书

本人完全同意北京航空航天大学有权使用本学位论文（包括但不限于其印刷版和电子版），使用方式包括但不限于：保留学位论文，按规定向国家有关部门（机构）送交学位论文，以学术交流为目的赠送和交换学位论文，允许学位论文被查阅、借阅和复印，将学位论文的全部或部分内容编入有关数据库进行检索，采用影印、缩印或其他复制手段保存学位论文。

保密学位论文在解密后的使用授权同上。

学位论文作者签名：_____ 日期：　　年　　月　　日

指导教师签名：_____ 日期：　　年　　月　　日

摘　要

随着深度学习技术的发展，出现了丰富多样的基于神经网络的自然语言模型，这些神经网络模型一般都是通过学习大规模语料库来调整自身的参数分布，以提高模型在相关任务上的性能和精度。然而，对于目前的一些较大规模应用而言，语言模型的训练过程还显得较为缓慢，效率需要进一步提高。导致计算缓慢的原因是这些模型在训练和测试的时候，通常需要预测整个词表的单词从而选择出最佳的候选单词，该步骤占用了绝大部分计算资源和时间，一般称为大词表问题。为了应对这个挑战，研究人员提出了各种不同类型的方案，如：单词拆分算法、基于抽样的近似算法和层次概率模型等，其中层次概率模型只关注局部概率，可以很大程度降低训练和测试过程中所占用的计算资源，因此获得了广泛的关注。

针对大词表问题，为了减少模型的计算时间并提高模型效率，本论文探讨了基于二叉树和基于类别的两种层次概率模型，并设计了相应的改进策略，其主要改进要点包括：1）提出了一种改进的层次模型的编码策略，引入基于树和基于类的并行度更高的损失函数；2）针对层次化模型对于聚类算法的敏感性问题，对基于 N-gram、句法信息和语义信息的不同聚类算法对各种层次概率模型产生的影响进行了分析；3）对多种不同的结构化预测算法进行了分析，同时分析了语言模型在排序和打分两个任务中的选择策略。

在实验验证环节，本文在语言建模任务上进行了评测，采用 WikiText-2, WikiText-103 和 One Billion Words 数据集作为实验的基准数据。在模型的评价指标方面，本文不仅考虑了传统的困惑度误差，还将文本比较中经常用到的编辑距离（单词错误率）作为一个重要的比较指标。除此之外，还比较了不同模型之间的训练和测试阶段内存占用量，以及相同数据量情况下模型的计算速度和模型的收敛速度等衡量指标。实验结果表明，与其他概率归一化方法相比，加速比提高，得到更高效的树聚类，与其他基于抽样的优化相比，性能相对较好。

关键词：层次概率，层次聚类，神经语言模型，递归神经网络，自然语言处理

Abstract

Recent decades has witnessed great progress and achievements, in the filed of natural language processing. Variants of neural networks based architecture have been proposed and successfully applied to the neural language models, neural acoustic models, neural translation model and etc. These neural models can leverage knowledge in texts by learning parameters from massive online corpora, and abundant cases are presented over various text tasks, like sentence classification and word vector learning. While they are extremely slow for the real-world challenge, as they try to predict candidates from a large vocabulary, in the process of training and inference. As an alternative to vocabulary truncation and sampling-based approximation methods, we explore the historical proposed tree-based and class-based hierarchical softmax methods.

In this research, aiming at reducing neural model's computational time as well as making them compatible with general purpose modern graphics processing units, we introduce a series of efficient and effective approaches and categorise our contributions as: a) Firstly, we reform their structural composition and introduce a compact tree-based loss function for the tree-based hierarchical softmax methods and class-based loss function for the corresponding class-based hierarchical softmax; b) Secondly, we discuss the impact of several ngram-based, syntactic and semantic clustering algorithms for the vanilla hierarchical softmax as these structural models are sensitive to the hierarchical clustering methods; c) Thirdly, we discuss possible inference algorithms for the hierarchical softmax variants, assuring the model can fetch presumable predictions under different circumstances.

Finally, Our experiments were carried out on language modelling tasks with standard benchmarks datasets, i.e., WikiText-2, WikiText-103 and One Billion Word datasets. Except for the traditional perplexity metric, we also extended our comparison over the word error rate and memory footprint and etc. Consist improvement with several intrinsic evaluation criterions: word error rate and perplexity, were also achieved over other conventional optimisation methods.

Key words: Hierarchical Softmax, Hierarchical Clustering, Neural Language Model, Recurrent Neural Network, Natural Language Processing.

目　　录

图　　目

表　目

第一章　绪论

1.1　课题来源与意义

近几年来，随着互联网（Internet）的兴起和发展，互联网上积累的数据呈现急剧膨胀的态势。根据国际数据信息公司[1]（International Data Corporation, IDC）的统计和预测，2018 年全球网络数据量已经达到 1.8 ZB，预计到 2025 年全球网站数据累积总量还将增长约 50 倍。随着这类无标注的文本和图像数据（Unlabeled Data）的大量涌现，如何利用现有的机器学习算法（Machine Learning），从这类无标注数据中学习内在规律并提取出有用的信息已经变成了一个重要的挑战[1]。自从 2006 年以 Geoffrey Hinton 为代表的学者们提出深度学习（Deep Learning, DL）理念 [2] 以来，这一新的思路为解决如何利用爆炸的数据量，以提取有效知识带来了新的前景。在这之后的发展中，基于神经网络（Neural Network，NN）的表示学习技术（Representation Learning）开始快速发展并拓展到相关研究领域中。尤其在图像分类（Image Classification）、语音自动识别（Automatic Speech Recognition，ASR）[3] 和神经机器翻译（Neural Machine Translation，NMT）[4] 等多个任务上，基于深度模型的方法在精确度和效率上远远超过了基于特征提取（Feature Extraction）的传统方法。

随着应用领域的拓展，深度学习技术逐渐在自然语言处理中（Natural Language Processing, NLP）得到广泛应用。例如，蒙特利尔大学教授 Yousha Bengio 提出用神经网络来训练语言模型（Language Model，LM），并对模型中的各个结构细节进行了相关探索[5]。因为模型的输入维度是固定的 N 个单词的词向量而不是动态长度的序列，所以该方法不能有效处理单词的长距离单词依赖问题（Long Term Dependency）。在后续的工作中，其学生 Tomas Mikolov 针对这个问题提出了采用循环神经网络（Recurrent Neural Network，RNN）[6] 对上下文信息作为建模策略的方法，并逐步将该理论进行了拓展和简化[7]。循环神经网络主要特点是能够记忆该单词之前的所有单词的信息，即所谓的全局上下文信息（Global Context），用来预测下一个单词出现的对数概率分布。因为 RNN 模型在训练过程中学习到了已经出现的单词信息，句子的长距离依赖关系可以被学习到，所以该方法在建模理论上克服了最初的神经网络语言模型的无法利用语句长距离

[1] https://www.idc.com/

上下文依赖的缺点。另外，在模型训练的过程中，语义相似的单词将被映射到的同一低维子空间中，满足了单词语义相似（Semantic Similarity）的要求。相比统计语言模型（Statistical Language Model）领域中的 N-gram 模型，他不需要平滑技术（Smoothing Technology）来对文本中出现次数少的单词做处理。到目前为止，RNN 模型已经演变出各种结构，应用在非常多的文本任务上面，并且都能取得了较为满意的结果。

由于 RNN 网络的计算时间与句子的平均长度正相关，所以基于 RNN 建模的算法计算时间都很大，需要更长的计算时间收敛。同时，我们需要看到最简单的 RNN 模型所使用的参数数量是神经网络模型的两倍多，这也意味着 RNN 模型需要占用更大的内存空间，消耗了更多的计算资源，导致其无法广泛应用到现实场景中去。在文献 [8] 中，Tomas Mikolov 提出了多种优化策略来消除该模型的各种缺点，例如：缩短模型求导步数、对词表（Vocabulary）做分解 [9] 等策略，这些计算策略能保证模型的计算精度，同时极大提高了 RNN 网络的运算效率。

1.2　国内外研究现状

语言模型可以用来估算一段序列组合的可能性，该模型在机器翻译（Machine Translation）、语音识别（Speech Recognition）等任务上都有着极为重要的作用。考虑到语言模型的漫长的发展历史，我们可以划分为两个主要阶段：统计语言模型（Statistical Language Model）和神经网络语言模型（Neural Language Model）。其中，统计模型指代的是 N-gram 语言模型，而随着深度学习的爆发，各种神经网络语言模型变体逐渐发展并占据了主导地位。我们接下来会对这两个演化阶段所涉及的算法一一介绍。

1.2.1　N-gram 语言模型

首先介绍传统 N-gram 语言模型，该算法属于典型的基于稀疏表示（Sparse Representation）的语言模型，因为一个单词表示方式是独热表示（One-Hot Representation）。该传统模型的意义不仅是提供了一种文本建模策略，而且定义了如何评价语言模型的好坏，并且定义了语言模型所涉及的相关拓展方向。

接下来给出 N-gram 统计语言模型的形式化定义。假设给定一个长度为 m 的单词字符串，w_1 到 w_m 依次表示这段文本中的各个单词，我们需要求解一个概率分布 $p(w_1, \cdots, w_m)$，以表示该字符串存在或者出现的可能性。在求解过程中，我们通常使用

链式法则（Chain Rules），将求解整个序列概率问题分解成计算单个单词的条件概率问题，如下形式：

$$p(w_1,\cdots,w_m)=p(w_1)\prod_{t=1}^{m}p(w_t|w_1,\cdots,w_{t-1}) \tag{1.1}$$

在实际求解过程中，如果文本的长度较长，公式 (1.1) 中右部 $p(w_t|w_1,w_2,\cdots,w_{t-1})$ 的估计误差会很大。因此，N 元模型（N-gram Model）被提出，用于估算条件概率并简化实际应用中的计算复杂度。需要注意的是，单词距离大于 n 单词会被直接忽略，该模型可以写成如下形式：

$$p(w_t|w_1,w_2,\cdots,w_{t-1})\approx p(w_t|w_{t-(n-1)},\cdots,w_{t-1}) \tag{1.2}$$

若 $n=1$，此时称其为一元模型（Uni-gram Model），公式 (1.2) 将会退化成 $p(w_i),i\in[1,n-1]$. 此时，整个句子的概率计算公式为：$p(w_1,\cdots,w_m)=p(w_1)\cdots p(w_m)$。从这个退化的公式中可以得知：在一元模型中，整个字符串的概率是出现的各个单词词频概率的乘积，它直接丢失了文本词组之间的顺序信息。当 $n=2$ 时，称为二元模型（Bi-gram Model），公式 (1.2) 将会退化为 $p(w_t|w_{t-1})$。除此之外，还有 $n=3$ 时的三元模型（Tri-gram model），使用 $p(w_t|w_{t-2},w_{t-1})$ 作为近似概率分布。当 $n>1$ 时的模型均可以保留一定的词序信息。

传统方法采用单词或者 n 元组的词频来作为 n 元组的概率计算方法，该方法简单有效且能满足线上负载的计算需求。但是随着 N 的增大，模型的参数呈现指数爆炸式增长、概率计算复杂度也相应上升。目前在线存储的 9 元模型（Google N-gram Viewer）[2]已经达到了计算机存储和数据读取的极限。

神经网络语言模型的建模思路源于 N-gram 模型，它主要解决的问题是如何对文本的上下文信息进行建模。历史上提出的模型可以主要划分为：传统前向传递神经网络（Feed Forward Neural Network, FFNN）、循环神经网络（Recurrent Neural Network, RNN）以及双线性模型（Bilinear Model）等。以下我们将对这三种主要的建模方案进行探讨。

1.2.2 前馈神经网络语言模型

由于神经网络对参数进行高度共享，因此对低频单词具有天然的平滑能力。这里所指的神经网络语言模型（Neural Probabilistic Language Model, NPLM），最早由 Bengio

[2]https://books.google.com/ngrams

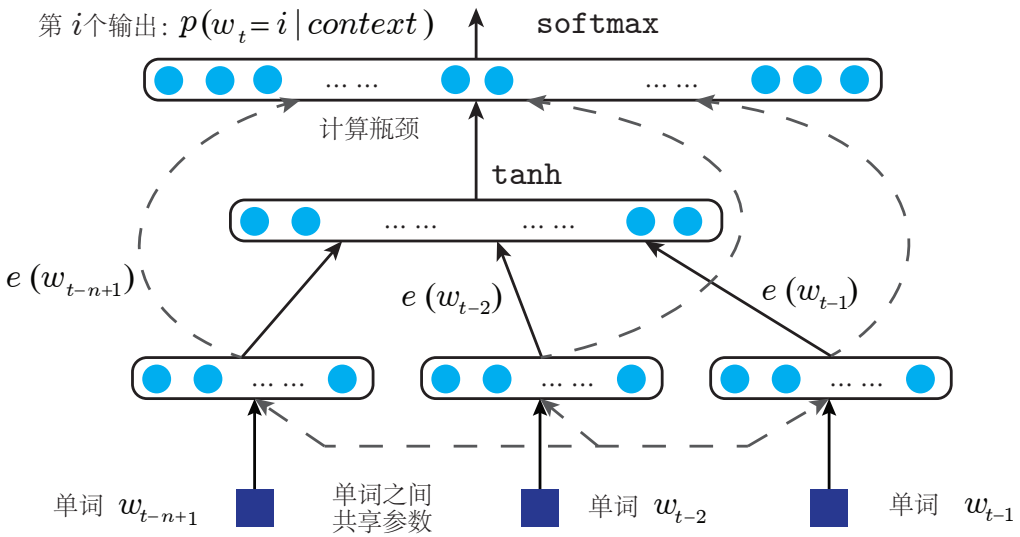

图 **1**　前馈神经网络语言模型

等人在 2001 年提出[5]，近年来一些学者开始展开这方面的研究，并取得一系列成果，如 [10–13] 。具体而言，NPLM 通过一个多层感知网络（Multi-Layer Perceptron, MLP）来计算公式 (1.2) 中概率。如图 1所示，输入层（Input Layer）用于表征前 n 个单词的高维分布；隐藏层（Hidden Layer）表征单词的上下文信息；最后一层是输出预测层（Output Layer）预测下一时刻的可能出现的单词概率分布。为了解决 N 过大所带来的数据稀疏问题，Bengio 等人提出拼接（Concatenation）前 N 词的词向量作为输入，如下所示：

$$x = [e(w_{t-(n-1)}), \cdots, e(w_{t-2}), e(w_{t-1})] \tag{1.3}$$

模型的隐藏层 h 和输出层 y 可以依照下面的公式计算获得：

$$\begin{aligned} h &= \tanh(Hx + b) \\ y &= Wx + Uh + b' \end{aligned} \tag{1.4}$$

其中参数矩阵 H, W, U 指代层与层之间的连接的权重[14]，参数向量 b, b' 均为模型中的偏置项（Bias）。如果存在 W，那么模型能直接学习一个线性模型，需要训练的时间将会减半；如果不存在 W，模型学习到非线性的网络模型从而具有更好的泛化性（Generalization）。因此在后续工作中，很少有使用输入层到输出层直连边的工作（即不存在 W 矩阵），下文我们也忽略这一种直连的操作。假设不考虑 W 矩阵，整个模型计算量最大的运算，就是从隐藏层到输出层的矩阵运算 Uh，后续的模型均有对这一矩阵乘法计算做优化。

1.2.3 对数双线性语言模型

2007 年，Mnih 和 Hinton 在神经网络语言模型（NPLM）的基础上提出了对数双线性语言模型（Log-Bilinear Language Model, LBL）[15]。LBL 模型与 NPLM 模型的区别正如它们的名字所示，其中 LBL 的模型结构是一个对数双线性结构； NPLM 的结构不包含双线性结构，仅仅是简单的前馈网络。具体来讲，LBL 模型的代价函数为：

$$\begin{aligned} c &= \sum_{i=1}^{n-1} U_i e(w_i), \\ p(w_t = w | w_{1:t-1}) &= \frac{\exp(c^\top e(w))}{\sum_j \exp(c^\top e(w_j))} \end{aligned} \tag{1.5}$$

其中 c 代表了语言模型的上下文信息，U_i 指代的是对应单词的权重向量。然后基于上下文信息表示 c 和下一个单词的目标词汇表中所有单词 $e(w), w \in \mathcal{V}$ 的表示之间的相似度来计算下一个单词的可能的概率分布。

公式 (1.5) 所描述 LBL 模型的代价函数与公式 (1.4) 所描述 NPLM 模型的代价函数的主要区别有：1) LBL 模型不包含非线性的激活函数 tanh，而由于 NPLM 是非线性的神经网络结构，激活函数必不可少；2) LBL 模型中只有一份词向量 e，也就是说，无论一个词是作为上下文，还是作为目标词，使用的是同一份词向量。其中第二点（只有一份词向量），只在最初提出的 LBL 模型中存在，后续的改进工作均不包含这一特点。

后来，Mnih 等人以 LBL 模型为基础，并对其所存在缺点做了一系列改进工作。其中最重要的两个模型为：逆向量语言模型（inverse vector LBL，ivLBL）[16] 和层次对数双线性语言模型（Hierarchical LBL，HLBL）[17]。

1.2.4 循环神经网络语言模型

对于循环神经网络来说，它能直接对公式 (1.1) 中的序列概率 $p(w_t|w_1, w_2, \cdots, w_{t-1})$ 进行建模，而不需要使用公式 (1.2) 对其进行简化 [7, 18] 。RNNLM 模型结构如图 2 所示，它的核心方法在于其隐藏层的计算公式:

$$h_t \leftarrow \phi(e(w_t) + U h_{t-1} + b), \tag{1.6}$$

其中 ϕ 为非线性激活函数。在上述公式中，h_t 表示文本中第 t 个词 w_t 所对应的隐藏层，由词向量 $e(w_t)$ 以及上一个步的隐藏层输出 h_{t-1} 计算得到。隐藏层的初始状态为 h_0，随着模型逐个读入单词：$w_1, w_2, \cdots$，隐藏层根据公式 (1.6) 被逐步计算出来并被输出：$h_1, h_2, \cdots$ 。通过这种自身迭代更新方式，囊括了此单词前所有上文的信息，相比 NPLM

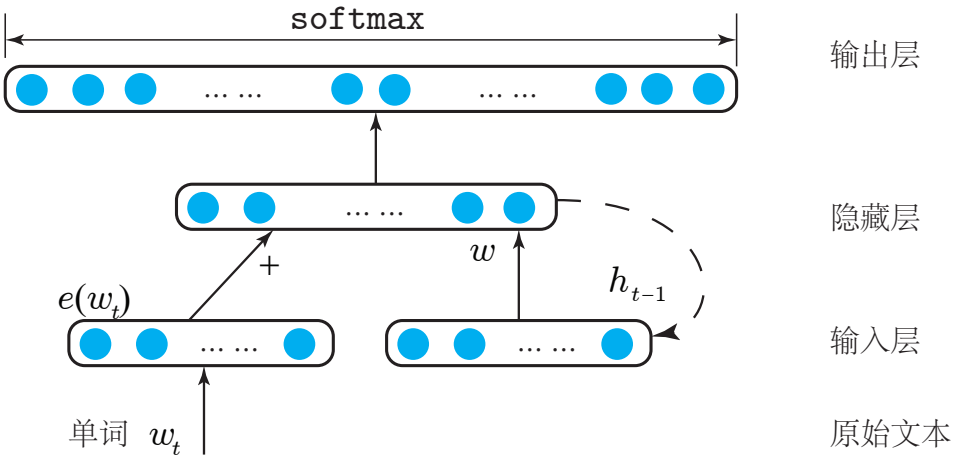

图 **2**　循环神经网络结构图

模型，RNNLM 模型理论上能学习到更丰富、更长距离的知识，也有更大的潜力达到更好的效果。最后，RNNLM 模型的输出层与 NPLM 模型的输出层都采用 softmax 算法，在这一部分上两者是一致的。

除了介绍的两种经典的建模方法，最近研究者提出可以使用带有门限机制（Gating）的 RNN 来防止模型的长距离依赖问题，例如长短记忆网络（Long Short-Term Memory, LSTM），门限记忆节点（Gated Recurrent Unit, GRU）和其他网络。

1.3　论文研究内容

在历史文献当中，源词和目标词分别被称为模型的输入和输出。源词通常可以用分布式表示（Distributed Representation）来表示，称为输入词嵌入（Word Embedding），可以使用基于外部语料库的连续词袋模型（CBoW）或跳跃单词模型（Skipgram）来训练。而这两个模型来源于语言模型任务，并且为了在特征空间中产生可能的词向量分布而被大大简化。另外，输出字通常表示为字索引（Indexing）或 1 – K 编码，并且可以与 softmax 概率函数直接关联。

语言模型的大词表问题是目前理论应用到实际过程中必须要克服的问题，我们当然可以通过配置高性能服务器来缓解该问题的后果。一旦应用到较大规模的数据集上，即使是目前最好的中央处理器（CPU）或者通用计算图形处理器（GPGPU），仍然需要数周时间才能训练完善。因此，在保证原有模型的准确率和精度的前提下，如何提高模型的训练速度是本文主要讨论和研究的内容。为此我们考察了两个主要的研究目标：上下文信息建模效率和精度对比和大词表问题的优化和研究。

针对上下文信息建模手段，目前主要采用的方案有以下几种：一种是采用子词（Subword-level）或者字符级别的词（Character-level）来直接缩小词表大小；一种是通过采样技术（Sampling-based Approximation）来减少必要的训练时间；另一种是通过基于分类的多元分类（class-based Hierarchical Softmax, cHSM）来加速模型和采用基于树模型的多层二元分类模型（tree-based Hierarchical Softmax, tHSM）。同时，我们还需要针对 CPU 和 GPGPU 设备分别进行探讨。因为传统的线性运算模型在流行的 GPGPU 并行运算方案中并不适用，所有需要结合不同的运算设备分别讨论可行的方案。

在本节中，我们将这个目标词表示扩展到一个分层的形式，使它们适配基于类和基于树的分层概率计算。首先，我们提出了一个在分层结构上建模参数的字编码方案。因此，考虑到 GPGPU 上的并行吞吐性能，我们推导出紧凑的代价函数及其梯度。同时，类或树上的单词分布对其性能有很大的影响，应该在训练阶段之前定义，这些动态交换算法在训练过程中改变了单词群或子树结构在这个研究中。我们采用了几个分层聚类和词汇分割策略，用统计，句法和语义知识来初始化其结构，以达到一个稳定和可以预期的性能。而且，在推理过程中，不同于传统的 softmax 情况，得到最好的候选者自然是可行的，层次推理不能直接用 softmax 方法来实现。我们讨论基于树和基于类的搜索策略的两种不同的推理情况：1）打分：输出给定序列的概率；2）排序：在给定的上下文中获取得分最高的一个候选单词。

1.4 论文的组织结构

第一章：“绪论”，主要介绍了本论文的研究背景和意义，另外简要说明了语言模型的发展历史以及本文的主要工作，并对本文的组织架构进行了说明。

第二章：“相关技术介绍”，对历史上的各个学术分支在语言模型的任务上的相关工作进行了介绍。

第三章：“树状层次概率模型”，介绍了基于二叉树的层次概率模型，并与传统树状模型做了理论层面的比较。同时还研究了在推理测试阶段，二叉树层次概率模型应用的贪心策略，以保证实际测试结果性能和效率。

第四章：“类别层次概率模型”，介绍基于分类的层次概率模型，并分析了词表非均匀划分所产生的后果，进而探讨了类别不均匀问题所带来的影响以及相关解决策略。最后探讨了在测试阶段，语言模型的任务需求和分类层次概率模型相应的解决算法。

第五章：“语言建模实验及结果分析”，实证研究了本论文提出的层次概率模型的实际效果，并和其他算法在各个指标维度上进行了比较和分析。

最后结论部分，总结了全论文的贡献和工作，并提出了未来的工作方向，同时撰写了结束语。

第二章　相关技术介绍

在自然语言处理研究领域中，基于神经网络的分布式表示（Distributional Representation）通常被称作为词向量（Word Vector）、词嵌入（Word Embedding）或者分布式表示（Distributed Representation）[19]。基于神经网络的词向量表示技术是通过对语言模型的两个组成部分进行建模，即：1）单词的语境，也称为上下文（Context）；2）以及上下文与下一步的目标单词之间的关系（Next-Utterance Prediction）。神经网络结构和组合方式相对灵活，可以表示更加复杂的上下文语义结构（Complex Context）。历史上的词向量表示方法主要是基于单词共现矩阵（Co-occurrence Matrix），仅仅是利用了单词的顺序信息。单词的语义向量是通过将单词共现矩阵进行矩阵分解（Matrix Decomposition），例如奇异值分解算法（Singular Value Decomposition，SVD）、潜在语义分析模型（Latent Semantic Analysis，LSA）或者潜在语义索引（Latent Semantic Indexing，LSI) 等分解算法。其中对于 SVD 分解算法，第一个分解获得的矩阵就是我们需要的单词的词向量矩阵。N-gram 算法也是使用单词的顺序信息作为上下文信息建模策略，当 N 线性增加时，N-gram 模型所需要存储的单词序列组合总数会呈指数爆炸增加，即维数灾难问题（Curse of Dimensionality）[1]，它深深限制了 N-gram 模型的实际应用。假若使用神经网络替代 N-gram 模型来表示单词概率分布，当 N 线性增加时神经网络语言模型所需要的参数数量仅以线性速度增长。这样我们就可以利用神经网络模型可以对更复杂的文本结构进行建模，在从训练后的模型获得的词向量中，将包含更多的语义信息、结构知识。

到目前为止，在语言模型的领域中，研究者提出的神经网络模型主要包括：传统前向传递神经网络（Feed Forward Neural Network, FFNN）、循环神经网络（Recurrent Neural Network, RNN）三种基本的建模方案。当然基于这些基本组件，我们还可以构造更复杂的模型结构，但是我们无法保证复杂模型一定在效率和精确度上优于简洁的模型结构。复杂的模型可以表征更复杂的本文语义信息，例如句子的依存关系（Dependency）结构或者句子的句法分析树（Syntactic Structure）结构，但是这样的效果不一定能保证模型的泛化性能（Generalization）会很好，因为日常聊天文本的语序是存在较多混乱的数据，很难找到良好定义的句子关系，所以也无法让模型从混乱的文本分布中学习到有效的句

[1]https://en.wikipedia.org/wiki/Curse_of_dimensionality

子结构，以提升模型的精度。显然是假设越弱模型越好，因为针对复杂的网络文本数据，我们模型训练所需要的要求越弱，模型越容易收敛并获得我们想要的结果。模型假设越强，模型所需要的数据质量越高，实际稳定性就会降低，产生得不偿失的后果。

除此以外，针对在本论文所讨论的语言模型的大规模应用过程中所遇到的大词表问题，历史文献中提出的解决方法主要可以划分为：单词拆分算法（Vocabulary Truncation）、采样估计模型（Sampling-based Approximation）和词表层次分解（Vocabulary Factorisation），我们会在接下来的章节陆续介绍。其中本论文重点是研究基于类别的多元分类模型（class-based hierarchical softmax，cHSM）和基于二叉树的二元分类模型（class-based hierarchical softmax，tHSM），这两种算法我们分别在下一章更详细讨论和介绍。

2.1　语言模型

在这一节中，我们将首先从语言模型提出的实际背景的形式化定义开始，以阐述语言模型的训练目标和实际应用场景 [20]。一个好的语言模型应当考虑对文本的两种特征进行建模：语法特征（Syntactics）和语义特征（Semantics）。为了保证句子的语法正确，我们需要考虑的是所生成的单词的前面的上下文（Previous Context），这也就意味着语法特性往往只对局部特征（Local Representation）进行建模。而语义特征的一致性则复杂了许多，它意味着我们需要考虑更大、更完善的上下文信息乃至整个文档语料，来获取全局语义（Global Context）而不仅仅是局部语义信息（Local Context）。

具体地来说，给定一个包含 T 个单词的序列：$w_1, w_2, \cdots, w_T$，这个序列的概率或者对数概率（Log-probability），即描述该序列作为一个合理且合法的句子的流畅性（Fluency）和出现该单词组合的句子的可能性（Likelihood）。由于直接求解整个序列的概率比较复杂，所以我们将采用马尔可夫链规则（Markov Chain）[2]，分解成一系列的条件概率（Conditional Probability）的联合分布（Joint Distribution）：

$$\log p(w_1, \cdots, w_T) = \sum_{t=1}^{T} \log p(w_t | w_{1:t-1}), \tag{2.1}$$

其中前 $t-1$ 个单词在本论文中一律记作 $w_{1:t-1}$。进而，条件概率 $p(w_t|w_{1:t-1})$ 表示给定其前面上下文 $w_{1:t-1}$ 作为预测的下一个单词的条件概率分布（Conditional Probability）。最

[2]https://en.wikipedia.org/wiki/Markov_chain

早的传统方法可以由前馈网络来建模这个上下文信息，同时用 softmax 函数来表示下一个单词的概率分布计算公式 [21]。在训练的过程中，整个模型以最小化每个单词预测的交叉熵损失（Cross Entropy Loss）为目标：

$$\ell = \sum_{t=1}^{T} \ell_t = \sum_{t=1}^{T} \log p(w_t | w_{1:t-1}) \tag{2.2}$$

其中文本或者句子的交叉熵代价函数 ℓ 的实际意义为：是当编码方案不一定完美时，平均编码长度是多少（最优的编码长度就是 1）。这里的编码长度的概念可以理解为：如何用 01 字节对所有的单词进行最短的编码，保证单词之间两两不互相冲突。交叉熵代价函数是由 KL 散度（Kullback–Leibler Divergence），也叫做相对熵（Relative Entropy），推导而获得的。这里我们需要注意的是，KL 散度的计算公式是：

$$\mathrm{KL}(p \| q) = -\sum_{t=1}^{T} p(x) \log q(x) - (\sum_{t=1}^{T} p(x) \log p(x)) \tag{2.3}$$

其中 $p(x)$ 指的是文本数据的真实概率分布，$q(x)$ 是由模型从数据中计算得到的预测概率分布。由于 $p(x)$ 和 $q(x)$ 在公式中的地位不是相等的，所以 $\mathrm{KL}(p \parallel q) \not\equiv \mathrm{KL}(q \parallel p)$。对于给定的文本语料训练数据集，第二项的计算值是常数，由此可知其导数是 0，它不会对模型中的参数更新产生任何影响。所以在通常情况下，我们采用交叉熵函数 ℓ 作为模型的代价函数而不是相对熵代价函数 KL，但其实我们希望模型优化的目标是 KL 使得模型的分布拟合实际的分布。

2.1.1 语言模型的应用

语言模型的实际用途包括垃圾邮件分类、机器翻译或者自动写诗歌等，当我们对这些纷繁复杂的应用作一定抽象之后，可以将语言模型的主要作用分为以下三种情况：1）为句子打分（Sentence Scoring）。给定一个未知的单词序列，我们需要计算其作为一个句子的流畅性和出现的可能性；2）挑选最佳的候选词（Next-utterance Prediction）。给定上下文，选择概率最高的单词。这里所指的上下文的形式是广义的，它可能是：前面的句子给定，选择下一个最佳单词；前面和后面的句子给出，选择中间最适配的单词；后面的句子给定，选择上一个最佳单词（逆序文本建模也是有其独特应用地位的）；3）训练词向量（Pretrained Word Embedding）。语言模型的输入，在经过大规模数据集训练之后，可以作为词向量输出，从而为其他文本任务提供有效的单词预训练特征。

2.1.2　长距离依赖

不同于前馈神经网络语言模型，循环神经网络语言模型使用隐藏层状态（Hidden State）来记录单词的顺序性信息（Sequence Order），它能够捕捉句子中存在的长距离依赖关系（Long-term Dependency）。这里的长距离依赖主要指语义对应关系（Semantic Correspondence）和选择限制（Selectional Preferences）[22]。

一方面，两个单词距离较远但是在文本中会存在很强的相关性关系。这里的所指的相关性，包括两个单词具有语义对应关系，或者他们需要满足句子的语法结构（Grammatical Structure）。举例来说，下面两个句子节选自网络文本：

He doesn't have very much confidence in himself
She doesn't have very much confidence in herself

尽管句子中间的其他词可能会发生变化，"（He，himself）"与"（She，herself）"这两个词对之间的对应关系却是固定的，但是该词对的对应关系中间相隔很长的距离（6 个单词）。这种单词的长距离依赖关系也不仅仅只出现在英语语言中，在汉语、法语等语言中，也都存在有此类型的单词之间的组合对应关系。如果采用传统的 N-gram 建模策略，我们必须要建模 6-gram 及以上的单词概率分布，才能保证模型能学习到这种对应关系。

另一方面，文本中存在的依赖关系将会限制我们选择接下来的单词候选集，称其为选择限制关系。譬如："我要用叉子吃沙拉" 和 "我要和我的朋友一起吃沙拉" 这两句句子中，"叉子" 所指代的是一种工具，但是 "我的朋友" 则表明是伴侣的意思。如果有人说："我想要用 '双肩背包' 来吃 '沙拉"' 就会觉得非常奇怪，因为这里的 "双肩背包" 既不是一种工具更不是伴侣的含义。假若我们人为地破坏了这种选择限制，那就会生成大量无意义的文本。

如图 3 所示，模型在预测过程中，在每一步都需要对词表中所有单词做预测，我们需要将每一步都做选择限制，减少模型的预测分支，这样才能保证模型生成的结果尽可能的语义一致且连贯。每次开放式的对所有单词做预测所带来的后果可能是，那些高频单词会不断出现在预测序列里面，因为训练语聊里面这些单词出现频率就非常高，所以模型将这些高频词预测概率大也是合乎情理的。但问题出现在，一旦我们模型不断预测高频词，我们生成的句子很难传达有效的语义信息。

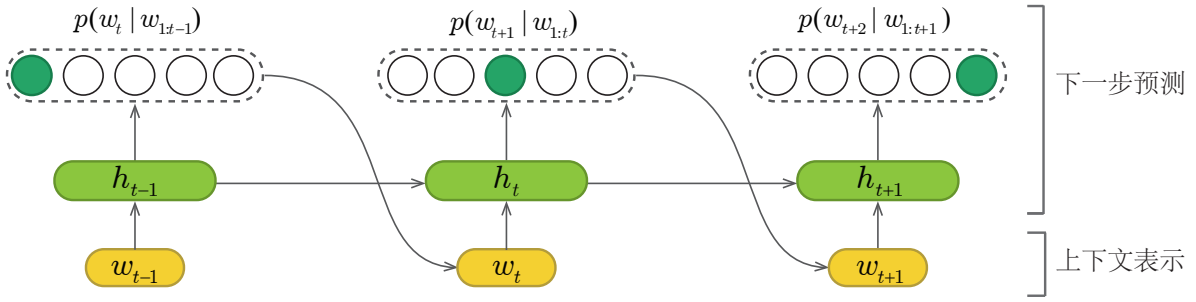

图 **3** 循环神经网络语言模型图例

2.1.3 循环神经网络

考虑到上面描述长距离依赖问题，在本实验中，我们将采用的基准模型是循环神经网络语言模型。近年来，基于时间序列建模的循环神经网络已经非常流行。如图 3 所示，由于在当前时间步 t 的输出 w_{t+1} 正好是下一个时间步的输入 w_{t+1}，所以将上一步的输出接到下一步的输入。当我们需要做文本生成时，才需要利用这样的自循环链接（Auto-associative Connection）。每个句子都需要用开始词（即，$\langle s \rangle$）和结束词（即 $\langle /s \rangle$）标记进行封装。在预测下一个单词 $\hat{w}_{t+1}$ 之前，需要用到前一时刻的隐藏状态 h_{t-1} 和当前读入的单词 w_t 来计算网络单元的输入 h_t。

$$\hat{w}_{t+1} = \arg\max_{w \in \mathcal{V}} \mathrm{softmax}(Vh_t + d) \tag{2.4}$$

从形式上讲，循环神经网络是一个参数化的非线性函数 $\phi(w_t, h_{t-1})$，循环地将一系列向量映射到一系列隐藏状态。将 $\phi(w_t, h_{t-1})$ 函数应用于任何这样的序列，在每个时间步骤 t 产生隐藏状态 h_t，如下所示：

$$h_t \leftarrow \phi(W\theta_t^w + Uh_{t-1} + c), \tag{2.5}$$

其中 W, U, V 是模型参数，向量 θ_t^w 对应于源词 w_t 的词向量。参数 U 是随时间共享的，即：当我们需要计算 h_t，我们需要用到上一步的隐藏层的输出 h_{t-1}。

自 Elman 提出基本结构 [6] 以来，许多扩展模型能学习长距离依赖关系，解决梯度消失（Gradient Vanishing）和梯度爆炸（Gradient Exploding）等问题，如长短期记忆网络（LSTM）[23]、门限记忆单元（GRU）[24] 和近似递归神经网络（Quasi-RNNs）[25]。

长短记忆网络（LSTM）是根据 RNN 所改进的最著名的模型之一，LSTM 模型的计算公式定义如下[26]：1）输入门 i_t 控制当前输入 x_t 和前一步输出 h_{t-1} 进入新的 Cell 的信息量；2）遗忘门 f_t 决定是否清除或者保持单一部分的状态；3）变换输出和前一状态到

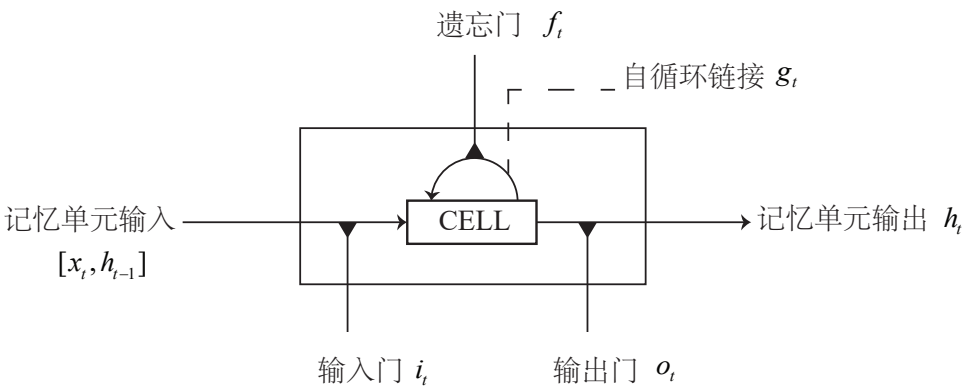

图 **4** **LSTM** 模型示意图

最新状态 g_t；4）输出门 o_t 计算 Cell 的输出；5）cell 状态更新 s_t。其计算步骤是，计算下一个时间戳的状态使用经过门处理的前一状态和输入；6）最终 LSTM 的输出 h_t 使用一个对当前状态进行 tanh 变换。其计算公式如下：

$$\begin{aligned} i_t &= \sigma(W^i x_t + U^i h_{t-1} + b^i) \\ f_t &= \sigma(W^f x_t + U^f h_{t-1} + b^f) \\ g_t &= \phi(W^g x_t + U^g h_{t-1} + b^g) \\ o_t &= \sigma(W^o x_t + U^o h^{t-1} + b^o) \\ s_t &= g_t \odot i_t + s_{t-1} \odot f_t, \quad h_t = s_t \odot \phi(o_t) \end{aligned} \tag{2.6}$$

其中 $\odot$ 代表对应元素相乘（Element-wise Matrix Multiplication），函数 $\phi(x), \sigma(x)$ 的定义：

$$\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

在具体算法实现过程中，LSTM 模型由于参数之间相似性和独立性，我们可以并行计算实现该模型。具体来说，模型的输入门、遗忘门、输出门和状态更新门所涉及的矩阵乘法操作我们可以同时计算，从而提高 LSTM 模型的计算效率[27]。因为 LSTM 的计算模型更容易并行，所以他与 GRU 的计算时间相差无几，尽管 LSTM 的模型需要的参数量是 GRU 模型的 1.5 倍。

$$[i^t, f^t, g^t, o^t] = [\sigma, \sigma, \phi, \sigma] \times W \times [x_t, h_{t-1}]^\top \tag{2.8}$$

其中 W 指的是 8 个小矩阵互相层叠起来，即 $[[W^i, W^f, W^g, W^o], [U^i, U^f, U^g, U^o]]$。

接下来我们介绍基于 LSTM 简化的模型，门限记忆单元（GRU）把 LSTM 模型中的遗忘门 f_t 和输入门 o_t 用更新门 z_t 替代。同时，它把 cell 的状态值和隐状态 h_t 进行了合并。图 5 是 GRU 更新 h_t 的过程[28], 主要包括：更新门 z_t、重置门 r_t 和节点内部更新值

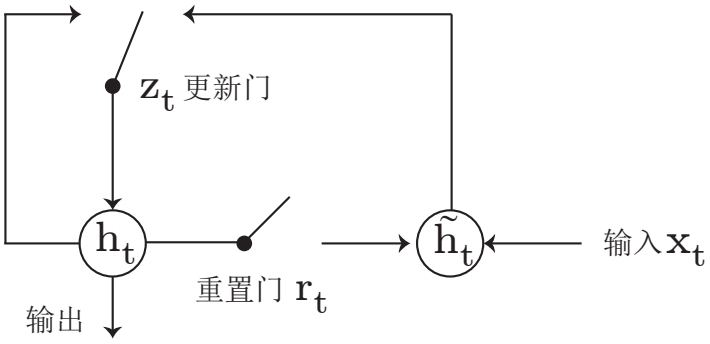

图 **5 GRU** 模型示意图

$\tilde{h}_t$ 和隐藏层输出值 h_t，这里与 LSTM 的区别是 GRU 中没有输出门，其计算公式如下：

$$\begin{aligned} z_t &= \sigma(W^z x_t + U^z h_{t-1}) \\ r_t &= \sigma(W^r x_t + U^r h_{t-1}) \\ \tilde{h}_t &= \tanh(W^h x_t + U^h(h_{t-1} \odot r_t)) \\ h_t &= (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1} \end{aligned} \tag{2.9}$$

如果重置门 $r_t \to 0$，那么计算 h_t 时将会丢弃之前的所有信息，即：$\tilde{h}_t = \tanh W^h x_t$；若 $z_t \to 1$，直接把前一刻隐藏层输出复制到当前时刻 $h_t = h_{t-1}$。一般来说那些具有短距离依赖的单元重置门比较活跃（如果 $r_t = 1, z_t = 0$，在这种情况下，GRU 模型变成了一个最传统的 RNN 模型，它只能处理短距离依赖问题），具有长距离依赖的单元更新门比较活跃。

2.2 大词表问题

softmax 函数和 log-softmax 函数通常用于多标签分类（Multi-class Classification）问题中的标准概率归一化（Probability Normalization）。然而当我们需要分类的目标变多时，即对于语言模型的大词表问题中，softmax 函数会占用绝大部分计算时间，需要我们对这个计算瓶颈做更详细分析。这里我们首先给出 softmax，log-softmax 函数和相应的梯度（Gradient）的定义：

$$\begin{aligned} p(w|h) &= \frac{\exp(h^\top v_w)}{\sum_{w_j \in \mathcal{V}} \exp(h^\top v_{w_j})} \\ \frac{\partial p(w_i|h)}{\partial v_{w_j}} &= p(w_j|h)(\delta_{ij} - p(w_i|h))h^\top \end{aligned} \tag{2.10}$$

$$\begin{aligned} \log p(w|h) &= \theta^w h - \log \sum_{u \in \mathcal{V}} \exp(\theta^u h) \\ \nabla_{\theta^u} \log p(w|h) &= (\delta_{uw} - p(w|h))h \end{aligned} \tag{2.11}$$

其中 h 指代的是隐藏层输出的向量（即上下文表示），θ^w 表示单词 w 的目标单词向量[29]。同样对于公式中出现的克罗内克 δ 函数 δ_{uv}（Kronecker Function），它的定义为：如果 u, v 指代的是相同的单词结果是 1，如果 u, v 指代的不是相同的单词则是 0。第一行计算公式指代的是 softmax 函数，第二行计算公式指代的是 softmax 函数的导数，第三行计算公式指代的是 log-softmax 函数，第四行计算公式指代的是 log-softmax 的导数。

如公式 (2.10) 和 (2.11) 所示，前向概率传播函数 $\log p(w|h)$（即第一行方程）和后向梯度优化函数 $\partial p(w_i|h)/\partial v_{w_j}$（即第二行的方程）都需要计算目标词汇表中的所有单词，因此在这个部分上花费的时间会随着词表中包含的单词数量的增加而线性地增长[30]。举个例子来说，假设词表中包含 $|\mathcal{V}|$ 数量的单词来说，总的时间复杂度（Time Complexity）是 $O(|\mathcal{H}||\mathcal{V}|)$，其中 $|\mathcal{H}|$ 表示的是隐藏层输出的维度。对于基于 CPU 设备的线性计算设备来说，该部分计算占用总计算时间也是足够大的。若我们将该部分计算迁移到 GPGPU 设备上，他将求解过程分解为：计算矩阵乘法（Multiply），计算求和函数（Summary），计算概率归一化的除法函数（Divide）。第一步求解操作占用了主要的计算消耗，即使对于采用现代通用计算体系结构的 GPGPU 来说，尽管其非常适合于具有高计算并行的矩阵乘法，这种计算负担也是相当高的。因此，采用现代硬件体系结构的训练和推理中，输出词向量矩阵的超大尺寸仍然是一个计算瓶颈。而且，目前的 GPGPU 设备的显存（Graphical Memory）是有限的，通常 Nvidia 公司提供的设备为 12GB 显存，目前最大的计算设备 Nvidia-K80 显存为 24GB，然而这个设备造价相当昂贵[3]。因为我们不能任意增大显存大小，所以对词表大小有一个天然的上限。

需要注意的是，softmax 概率计算瓶颈是求和函数还是矩阵乘法函数？softmax 算法的瓶颈不能归因于循环函数（即方程 2.10 ）中的 $\sum_{u \in \mathcal{V}}$，尽管它随着词汇量的增长而线性增长，但是矩阵张量的乘法计算的规模更大。因为矩阵计算需要更多的时间来获得结果，但线性求和可以更快地计算完成。所以，相比于 “for” 循环函数而言，主要的计算时间用于矩阵乘法计算。我们在这里说明的目的是：只有那些避免大矩阵乘法运算，即避免计算其他冗余字的计算概率才能达到实际最大的边际加速比，而那些仍然涉及全局

[3]https://www.nvidia.com/zh-cn/data-center/tesla/

概率归一化的方法，实际上并不能真正改善这个部分的计算占用的时间，那些试图优化求和函数的计算效率所获得的收益更是微乎其微。

因此，在保证模型计算准确度不下降的情况下，我们期望能缓解概率归一化函数的计算瓶颈进而提高模型的训练速度。目前能有效缓解大词表问题的算法主要分为以下三类：单词拆分算法（Vocabulary Truncation）、采样估计模型（Sampling-based Approximation）和词表层次分解（Vocabulary Factorisation）。下面将对三种不同思路的算法进行详述。

2.2.1 单词拆分算法

针对大词表问题的解决方法，最直接、最简单的策略就是我们放弃使用大词表，转而保留一个较小的词表来保证训练和测试的时候模型的内存（Memory Footprint）占用小和计算效率高。当我们使用较小的词表来保留高频词之后，就会剩下很多其他不在词表里面的单词被映射成 *unk*。这相当于我们直接丢弃和模型的输入信息，可能使模型的性能变差。那么针对剩余的过多的词表外的单词（Out-Of-Vocabulary，OOV），Schwenk 等人提出可以使用传统的 N-gram 语言模型来估算其可能的概率分布 [31]。这样做一方面保证神经网络模型可以在有限时间内训练完，同时保证模型的最后的测试结果不会很差。

然而我们需要注意到，当我们的词表继续减少并且我们选取的训练样本的词表包含的所有单词数量不断增加的时候，我们会发现训练样本中存在非常多的 ⟨unk⟩ 字符，这直接导致我们转换后的句子变成了无法读或者理解的句子。当我们肉眼可见的句子是无法理解的时候，已经导致输入模型的信息噪声非常大，这样使得神经网络的模型训练非常困难，进而导致模型效果变得非常地差。所以这种方案只是一定程度上缓解了大词表问题，但是不失为一种简单且有效的尝试方案。

除了上述直接的建模方案外，目前采用的方案是将单词按照字符级别来划分。英文单词是由多个字符（Character）组成的，可以将一个单词按照字符统计规律划分成任意多个子词（Sub-word）。例如，Tucker 等人提出利用二元对编码（Byte Pair Encoding，BPE）能将一个单词划分成两部分：前缀（Prefix）和后缀（Suffix）[32–34]，如图 6 所示。从该图中我们可以看出，单词被划分成前缀和后缀两部分，其中 + 代表是单词的前缀，同时 ⟨/*w*⟩ 是单词的后缀。由于划分规则是从训练数据中学习到的，我们可以指定需要缩减的词表大小，所以该算法的动态适应性很强。最初 Kucker 等人提出该算法目的是问了方便文本信息压缩（Zip 软件），后来 Sennrichr 等人将该算法应用在目前主流的机

器翻译模型中，获得很好的精度和性能收益 [35]。后续的提出的算法中，直接将单词拆分成字符序列，模型训练得到的将不再是词向量而是子词向量（Sub-word Embedding）或者字符向量（Character Embedding）。在测试的时候，我们将通过组合子词或者字符向量来获得这个单词的实际词向量。尽管如此，我们仍然需要看到它这样将单词解构成前缀和后缀的操作依然带来了一定的损失，因为句子的长度加倍了，对循环神经网络的长距离关系学习能力提出了更高的挑战 [36]。

```
cheekbones      ->  cheek+   bones</w>
entertains      ->  enter+   tains</w>
development     ->  develop+ ment</w>
international   ->  inter+   national</w>
```

图 **6** 词到子词划分样例

2.2.2 采样估计模型

第二种策略也是通过减少单词数量从而减少计算量。与上面暴力的将单词拆分成子词不同的是：这种采样算法是在训练的时候，通过采样算法选择少量单词，用来估计 softmax 函数的可能的概率分布。因为实际只有一个单词是需要我们预测的，其他的单词都是可以认为是一个噪声单词。

目前流行的采样算法（Sampling Algorithm）主要是针对公式 (2.10) 中对所有单词概率规约那一项进行概率估计，主要有：重要性采样（Importance Sampling） [37]，噪声差分估计（Noise Contrastive Estimation, NCE） [17] 和 Blackout 采样算法 [38]。第一种算法在 Bengio 等人的论文实验中被证明模型无法收敛 [37]，因此目前主流模型都使用后面的两种采样算法。接下来，我们将会对这两种方法进行更加详细的描述。

对于 NCE 噪声差分估计算法来说，模型需要学习将正确的单词 w_0 与随机生成的单词 $\{w_1 \cdots w_k\}$ 做一个二元分类。其中 w_0 指代的是训练样本中真正的下一个单词，$\{w_1 \cdots w_k\}$ 是采用先验分布 $q(w)$ 产生的随机噪声单词。实际实验中，我们常用的采样的概率分布模型 $q(w)$ 是单词词频分布（Word Frequency Distribution）。正例归一化后的概率 $\tilde{p}(y=1|h)$ 和所有负例联合概率 $\tilde{p}(y=0|h)$ 的公式可以写成：

$$\begin{aligned} \tilde{p}(y=1|h) &= \frac{\exp(\theta_0^w h)}{\exp(\theta_0^w h) + k * q(w_0)} \\ \tilde{p}(y=0|h) &= \prod_{i=1}^{k} \frac{k * q(w_i)}{\exp(\theta_i^w h) + k * q(w_i)} \end{aligned} \tag{2.12}$$

需要注意的是噪声概率 $\tilde{p}(y=0|h)$ 需要对 k 个噪声样本做加和运算而不是对整个词表单词进行的，因此该算法的计算复杂度就是 $O(k+1)$，与整个词表的大小无关，所以该算法与简单的 softmax 算法的加速比是 $O(|\mathcal{V}|/(k+1))$。

除了上面这种最经典的算法之外，后续又衍生出了其他的从采样算法。最近才提出的 Blackout 采样算法针对噪声概率归一化的时候与当前上下文的相关，对上述的 NCE 算法进行了进一步优化和修正 [38]，同时也对模型引入了更多的计算量。其模型的代价函数计算公式定义为：

$$\begin{aligned} \ell &= -\log(p(w_0|h)) - \sum_{w_i \sim q(w)} \log(1-p(w_i)) \\ p(w_i) &= \frac{\exp(\theta_i^w h)}{\sum_{w_i \sim q(w)} \exp(\theta_i^w h)}. \end{aligned} \tag{2.13}$$

简短介绍完了两种采样算法，我们来做一个总结。总的来说，这些采样近似算法可以显着加快训练速度，但仍然需要时间来利用单词分布 $q(w)$ [39] 来采样大量的噪音单词，采样单词足够多的情况占用的时间仍然很客观，需要进一步的手段去优化。另一方面，Softmax 函数在推理测试的时候必须要计算，意味着在测试推理的时候该算法失效了。因为候选词是在整个词汇表中预测的，而不仅仅对当前的采样的早噪声单词做预测，而且我们更无法得知正确的单词是什么[40]。

这里我们需要注意的是，由于我们在做采样估计的时候引入了采样函数 $q(w)$，模型将会花费额外时间在采样计算上面。从离散分布采样的初始程序通常需要词汇数量具有线性时间复杂度。因此我们需要寻找一个性能优异的带权重的随机数生成器（Weight Random Number Generator）或者是离散采样函数（Discrete Sampling Function）。

2.2.3 词表层次分解

介绍完单词拆分、采样估计算法之后，我们再从模型角度分解优化这个大词表问题。层次分解方法（Factorization）可以大大降低学习和推理过程中表示概率分布的所占用的计算内存，因为它只计算局部概率（Local Probability）并且选择每一层的分数最高的候选路径而不是保存全局计算结果。目前主要的分解策略可以分为：基于类别的多元分类模型（Class-based Hierarchical Softmax, cHSM）和基于二叉树的二元分类模型（Tree-based Hierarchical Softmax, tHSM）。接下来我们将介绍这两种算法的计算过程。

首先我们考虑基于类别的分解策略。假设我们将文本所对应的词表中的每一个词划

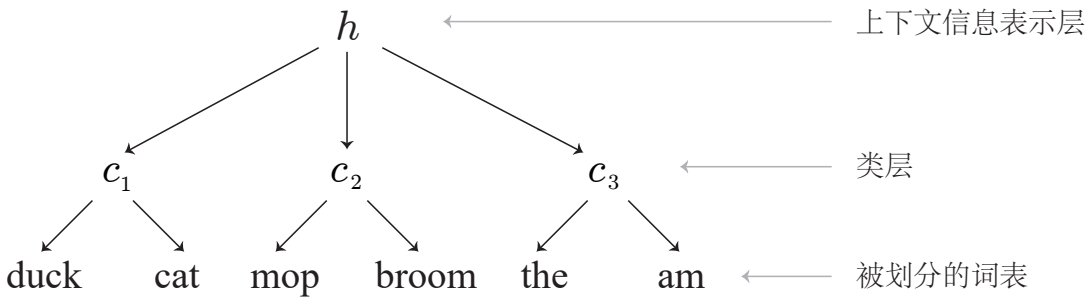

图 7 **cHSM** 算法可视化模型

分成各个类簇，每个单词属于且只属于一个类（Class 或者 Group）。在此假设之上，我们进而来计算单词在在语料中的对数分布时，可以转化为：先计算类的概率分布，然后乘以在所属类上该词的概率分布[41]，于是可将公式 (2.10) 转化为如下形式：

$$\begin{aligned} p(w|h) &= p^c(C(w)|h) \cdot p^w(w|C(w),h), \\ w \in C(w), \mathcal{V} &= \bigcup_{i=1}^{C} c_i, \quad c_i \bigcap c_j = \phi, \text{若} \quad i \neq j, \end{aligned} \tag{2.14}$$

其中第一项概率 $p^c(C(w)|h)$ 表示每个类别的概率，第二项概率 $p^w(w|C(w),h)$ 表示在类别 $C(w)$ 中所有单词的局部概率。

该模型计算一个词的概率分布所需要的计算复杂度正比于: $O = |\mathcal{H}||C|$。在该计算公式中，C 表示的是该语料中所有词的分类数目，我们可以按照不同策略划分词表，最简单的一种策略就是根据语料中词的词频进行均匀划分。当 C 取 1 或取词典大小 $|V|$ 时，此结构等同于标准的 Softmax 结构，此时词表还是原来的结构，并没有任何优化效果。因此，对于绝大多数情况，我们取 $C \ll \mathcal{V}$ 并且 $|C| \ll 1$，这样的结构才能保证有效降低 softmax 的计算复杂度。图 7 展示了类别划分和分步概率计算的示例。从图 7 中我们可以得知，词表 [duck,cat,mop,broom,the,am] 被划分成三个类别：c_1 →[duck,cat]，c_2 →[mop,broom] 并且 c_3 →[the,am]。

接下来我们来分析一下基于类别的分层模型计算速度提升效果。假设每个类包含 $\sqrt{|\mathcal{V}|}$ 单词，表示词汇被划分为相同大小的组，这样的话级联概率（Cascaded Probability）只涉及 $2\sqrt{|\mathcal{V}|}$ 单词 softmax 计算，所以最优时间复杂度可以减少到 $O(|\mathcal{H}|\sqrt{|\mathcal{V}|})$ [42]。虽然在更常见的情况下，词表划分算法会产生不同大小的分组，这需要外部的其他结构来建立高效的数据结构，并且这些稍微复杂的算法尚未被完全探索和分析。此外，我们还可以通过不同的类大小和分区算法来调整精度和效率之间的权衡关系。

另一方面，我们再介绍基于二叉树分解词表的建模策略。tHSM 方法将单步骤多类

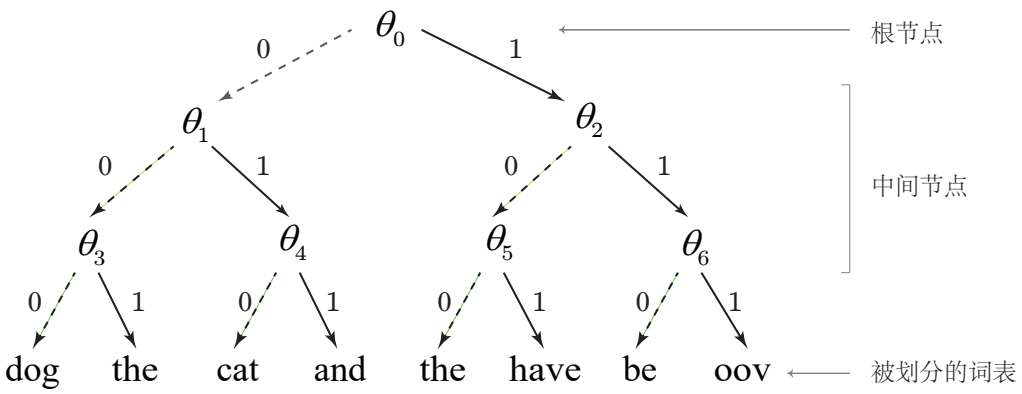

图 **8 tHSM** 算法可视化模型

分类分解为多步骤二元分类步骤。正因为如此，词汇组织为二叉树，其中单词分布在二叉树的叶子节点上，二叉树的所有的中间节点的都是内部参数。在该二叉树是平衡树（Balanced Binary Tree）的结构情况下，每个词将会被赋予相等长度的 01 前缀，最优时间复杂度可以减少到 $O(|\mathcal{H}|\log|\mathcal{V}|)$。图 8 展示了一颗平衡二叉树词表分解和访问路径的示例。关于如何构建这颗二叉树，最早的工作中，单词在二叉树上的分布可以由 WordNet 与人类专家 [43] 或单词 Unigram 分布 [19] 的霍夫曼编码构建。然而，在现实世界的大规模文本应用中，专家知识的构建是相当昂贵的。所以，一般人们也不会采用这样的方案来构建单词的二叉树分布。霍夫曼编码方案只考虑单数据统计，而单词的语境，句法或语义信息尚未被考虑和得到彻底的讨论。

Mikolov 等人当年曾提出使用基于二叉树的层级 softmax 模型来加速的训练方案，加速比能达到理论的最大速度。但是需要注意的是，当时提出的背景是基于 CPU 计算方案构建的，如今随着应用领域的推广越来越多的算法应用到实际问题中，我们需要在并行吞吐量更高的 GPGPU 上进行计算，因此基于 GPGPU 进行建模的 tHSM 算法和历史上研究众多的 cHSM 算法尚未被研究提及，因此我们将会需要在本研究中详细研讨。

2.3 离散采样算法

前面我们提到在使用基于采样的模型时，我们不可避免地需要使用到离散采样函数，我们在这一个部分来讨论离散采样函数的构建策略。我们首先给出离散采样问题的定义：设一共有 $|\mathcal{V}|$ 个单词，第 i 个单词出现的概率是 $p(w_i)$，如何高效地产生这样的随机变量序列 $w_1, w_2, \cdots$？

这个问题在数学、工程领域中，称之为模拟离散取样（Simulated Discrete Sampling），

```
df = cumsum(phi); // 计算累积分布函数
amp_k = sum(cdf < rand()*cdf(end)) + 1;
```

图 **9 Matlab** 实现的离散采样函数

其含义指的是根据有限类别的非均匀概率分布，来模拟有放回的采样，从而生成我们需要的采样序列。因为我们的计算机只能快速产生伪随机数（Psudo Random Number Generator），即对均匀分布（Uniform Distribution）做采样。其他的复杂采样函数均建立在均匀分布的变换之上。所以该问题的关键就是如何通过将均匀分布变换到我们实际变量的分布，生成指定概率分布的随机噪声。

2.3.1　随机采样函数

当我们使用概率模型实现推理和学习算法时，通常需要从离散分布进行抽样。也就是说，从参数 $\pi \in \mathbb{R}^K$ 中的多项式分布，使得 $\pi_k \geq 0$ 和 $\sum_{\mathcal{V}} \pi_k = 1$。更常见的情况是, 我们只知道：$\phi \in \mathbb{R}^{\mathcal{V}}$，其中 $\phi_k \geq 0$，但我们不知道规范化常量 $\sum_{\mathcal{V}} \phi_k$。也就是说，$\phi$ 只与多项式参数 π 成正比。然而给定 π 分布，我们希望根据 π 快速生成变量序列。我们可以使用 Matlab 代码，很容易地完成这个计算操作，如图 9 所示。

其中 CDF 指代的是累积分布函数（Cumulative Distribution Function，CDF）。该计算算法很简单，最容易实现。但是我们注意到该算法初始化需要消耗 $O(\mathcal{V})$ 时间复杂度。计算每个样本时候的时间复杂度也是 $O(\mathcal{V})$。假如我们能找到阈值的更好的数据结构来完成比较操作 <，每个样本的复杂度将可能被降低到 $O(\log \mathcal{V})$。

实际上除了二叉搜索算法外，我们发现还有更高效的算法，可以将每个样本产生的复杂度降低到 $O(1)$，并且保证总的采样复杂度是 $O(K)$（假设我们需要采样 K 个单词）。

2.3.2　基于线性搜索的逆变换方法

累积分布函数的取值范围是 $[0, 1]$，并且是一个单调递增的（Monotonic Increasing）函数。与累积分布函数息息相关的一个算法叫做，逆变换取样算法（Inverse Transform Sampling），其定义如下：给定一个累积分布函数，我们计算其逆函数（Inverse Function），就可以把均匀随机变量转换成该累积分布函数。逆函数定义如下：$X = F_X^{-1}(Y)$。

如图 10 所示，在 Y 轴上随机取样获得 y_i，接下来向右和函数曲线取交点，求解交点的 X 轴位置 x_i，X 就是满足条件的概率分布。假若 X 的取值不是连续的而是离散的，那么我们只需快速地搜索出超过均匀取样的元素的那个样本。

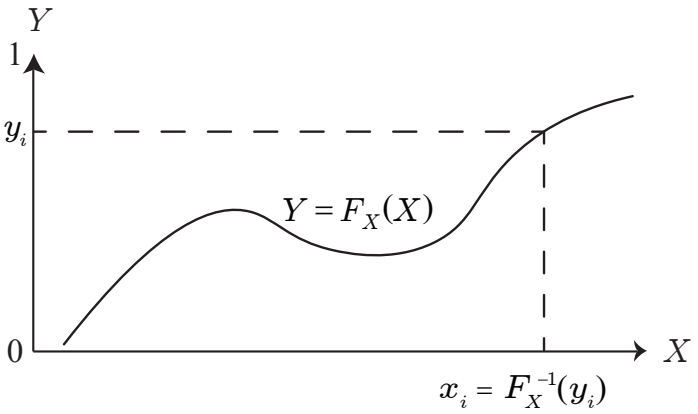

图 **10** 累积分布函数求解拟函数示意图

在离散概率分布的情况下，其时间复杂度是 $O(|\mathcal{V}|)$。在图 9中，所展示的代码使用了线性搜索算法 (Linear Search)。如果目标概率分布的数组是有序的，那我们就可以使用数据结构中的查找算法更快地找到结果。例如，我们可以用二分法 (Binary Search) 搜索合适的样本结果，其对应的时间复杂度是 $O(\log|\mathcal{V}|)$。在 $\mathcal{V}$ 很小的情况下，使用线性查找算法或者二分搜寻算法也足够得快；当 $\mathcal{V}$ 足够的大，即所谓的大词表问题的时候，我们可以采用别名方法 (Alias Method)、近似方法等 [44]。本实验实际用到了别名算法，我们接下来稍作介绍。

2.3.3 别名方法

设 Z 为离散的随机变量，它有 $\mathcal{V}$ 个可能的结果 $z_0, z_1, \ldots, z_{\mathcal{V}-1}$。为了简化下面的讨论，我们研究另一个变量 Y，其中 $P\{Y=i\} = P\{Z=z_i\}$。当 Y 取值 i 时，让 Z 为 z_i。所以 Z 可以从 Y 生成。随机变量 X 被均匀分布在 $(0,\mathcal{V})$ 中，这个概率密度函数是

$$f(x) = 1/\mathcal{V}, 0 < x < \mathcal{V} \tag{2.15}$$

那么我们现在生成的 Y' 是：

$$Y' = \begin{cases} \lfloor x \rfloor & \text{若}(x - \lfloor x \rfloor) < F(\lfloor x \rfloor) \\ A(\lfloor x \rfloor) & \text{否则} \end{cases} \tag{2.16}$$

其中 $A(i)$ 是别名函数。当 x 落入范围 $[i, i+1)$（i 是一个整数）时，y 的概率 $F(i)$ 为 i，概率 $1-F(i)$ 是 $A(i)$。由于 x 是均匀分布的，

$$\begin{aligned} P\{x \in [i, i+F(i))\} &= \int_i^{i+F(i)} \frac{1}{\mathcal{V}} dx = (i + F(i) - i) \times 1/\mathcal{V} = F(i)/\mathcal{V}, \\ P\{x \in [i+F(i), i+1)\} &= \int_{i+F(i)}^{i+1} \frac{1}{\mathcal{V}} dx = (i + 1 - (i+F(i))) \times 1/\mathcal{V} = (1-F(i))/\mathcal{V} \end{aligned} \tag{2.17}$$

我们将满足 $A(j) = i$ 的值集合 j 表示为 $A^{-1}(i)$。生成的变量 Y' 具有以下概率质量函数：

$$P\{Y' = i\} = F(i)/n + \sum_{j \in A^{-1}(i)} \frac{1 - F(j)}{\mathcal{V}} \tag{2.18}$$

别名方法是构造 A 和 F 的算法，所以 $P\{Y' = i\}$ 等于 $P\{Y = i\}$。由于 A 和 F 的域都是整数 $0, 1, \ldots, \mathcal{V} - 1$，所以它们可以存储在数组中，并且可以在 $O(1)$ 中查找值，其中空间效率是 $O(\mathcal{V})$。虽然别名方法可用于生成丰富的噪声字，但是它需要线性建立时间和恒定的采样时间 [45, 46]。

2.4　本章小结

本章对国内外学者在语言模型的任务上相关工作进行了介绍。首先，我们介绍了语言模型的建模策略和对应的应用方案。其次详细讨论了循环神经网络的语言模型的建模方式，并讨论了其结构的优缺点。同时为了解决使用循环网络所带来的梯度爆炸问题，我们进而引入了长距离短期神经网络和门限记忆节点两种基于门限机制的循环网络；另一方面，我们还考虑了语言模型大规模应用所带来的大词表问题，并作了详细分析原因。具体来说，我们还讨论了历史上所提出的解决方案，并做了分类讨论。主要涉及三种思路：缩减词表大小，用采样算法近似估计和模型层次分解。针对各种模型提出的结构做了介绍，并分析了各种算法的优劣处。最后我们分析了实验中需要用到的高效采样函数的构造算法。

第三章　树状层次概率计算模型

在本章中，我们考虑将预测端词表概率进行分步求解，来解决语言模型中遇到的大词表问题。我们将这个预测端的词表用二叉树的分层的形式表示，通过多次的逻辑分类算法判断二叉树的访问路径从而避免了对所有单词做归一化运算。为了使单词基于二叉树的路径的编码适配基于树的分层概率计算，我们首先提出了一个在二叉树分层结构上对单词进行极性编码的方案。与此同时，因为考虑到 GPGPU 设备上允许算法能并行计算，传统的基于 CPU 设备只能线性计算，这为我们算法的效率进一步提高提供了可能。为了保证算法的并行计算的高吞吐量，我们进而推导出二叉树概率模型的紧凑的代价函数以及其模型参数的梯度计算公式，使得该分层模型在 GPGPU 设备上的能并行地计算模型的每一层内部概率函数，而不是从根节点到叶子节点线性计地算模型内部各层的概率函数，避免浪费 GPU 设备的高并行性能的特性。

同时，我们需要注意到，由于所有单词都分布在这个二叉树的叶子节点上，单词的内部节点对于目前的模型无法包含单词只能作为单词的路径参数，所以需要我们在模型训练之前给一个初始化定义。然而不同算法所生成的基于二叉树上的单词分布，可能生成一颗平衡二叉树，也可能生成一颗霍夫曼二叉树，又或者是高度偏斜的二叉树，不同的树分布对其最终任务性能有很大的影响，所以应该在训练阶段之前就给出一个较为合理的单词分布。在本实验中，我们并不打算考虑那些在模型训练过程中动态交换二叉子树的算法进而调整和改变了单词在二叉子树上的分布结构，而是采用了几种分层聚类策略，利用单词或者文本的统计信息，句法结构和语义知识和已有的层级聚类算法来初始化单词的二叉树分布结构，以达到一个稳定和可以预期的模型性能。另外，我们不考虑单词的一词多义的现象，即一个单词不会出现在两条不同的路径上，不同的的二叉树访问路径所对应的单词一定不相同。

另外，在语言模型的测试推理过程中，主要的两个任务可以分解为：1）给句子打分（Sentence Scoring）。输出给定单词序列的概率值；2）给定上下文，对所有的单词进行排序（Word Ranking）。在给定的上下文中选择获取得分最高的一个或多个候选单词。不同于传统的 softmax 算法情况，得到最好的候选者单词是天然可行的并且能直接被计算出来，层次推理模型不能直接应用 softmax 方法来实现。尽管我们可以计算出所有单

词的概率，然后再依照 softmax 算法的情况类推，这样的计算方式是十分冗余的，直接将层次模型的计算优点给忽略了。因此，我们需要讨论并研究基于二叉树的搜索策略，以满足上述两种不同的推理情形。

3.1　基于二叉树的单词极性编码

首先，我们需要知道所有的单词分布在二叉树的叶节点上，因此我们可以通过访问从根到叶节点的所有内部节点来定位每个特定的词，所以不同的访问路径代表不同的单词。举例来说，单词 w 的路径表示所有内部节点 θ_i^w 和它访问的边 d_i^w。

为了方便理解，首先直观上说明一下传统 softmax 和树状层次概率模型的差异点。可以认为传统的 softmax 将单词均匀放置在一个特征空间（Feature Space）里面。基于二叉树的算法则是首先将这个所有单词所在的特征空间划分成两个子空间（Subspace），这里需要注意的是，我们没有规定空间划分必须均匀（Equal Partition），所以这两个子空间不一定相同。接下来，对每一个子空间，接着划分成两个更小的子空间，直到每个子空间里面包含的单词有且仅有一个为止。在模型计算概率的时候，更多的时候是计算特征划分到这个子空间的概率。因此每次都需要计算二叉树访问路径的概率，这个概率值就是将模拟特征划分到这个子空间的可能性。通过这个路径概率的求和我们可以避免对所有单词的概率计算，因为每次都做逻辑分类算法（Logistic Classification），只要计算一个分支的概率，由于两个分支的概率值之和一定等于 1，另一个分支的概率可以直接推导出来，从而不需要再计算一遍了。

接下来，来说明传统二叉树编码和我们提出的编码方案之间的差异点。不同于传统的基于二叉树的单词的 0,1 编码方案，在模型中需要引入单词的极性编码方案（Polarity Encoding Scheme），即单词的路径是由 $-1,+1$ 组成。其含义同原来的 01 编码方案类似，但他能帮助推导出更加简洁有效的模型代价函数。在上一章中，图 8 给展示了一个路径编码的例子。

由于实际层次聚类算法（Hierarchical Agglomerative Clustering）产生的二叉树分布通常是不平衡的，所以不同单词的路径长度不一致，然而基于 python 语言的 Numpy 和 Theano 库目前都不支持动态长度的设置（Dynamic Type），所以需要介绍一下实际实现过程中用到的辅助计算矩阵，其结果和使用 C++ 等动态语言所计算输出的结果是一致的，仅仅是因为实现语言框架的限制，导致引入这一操作。需要注意到，对于平衡二叉

表 **1** 并行树状概率计算模型的符号助记表

符号	含义	取值范围
l^w	单词 w 所对应的叶子节点和中间节点的路径	Int32
d_j	表示路径 l^w 中第 j 个节点对应的编码（根节点对应编码 0）	$\{-1,+1\}$
$d^w=[d_0^w,d_1^w,\cdots,d_{l^w-1}^w]$	单词 w 的极性编码，他由 l^w-1 位编码构成，	$\{-1,+1\}^{l_w}$
$\theta_j^w\in\mathbb{R}^m$	表示路径 l^w 中第 j 个非叶子节点对应的向量	Float32
$\theta^w=[\theta_1^w,\theta_2^w,\theta_3^w,\cdots,\theta_{l^w}^w]$	表示路径 l^w 所对应的参数矩阵	Float32
Γ	路径查找表，给定路径序列，可以获得对应的单词	—

树，所有单词的访问路径长度是一样的，所以不需要使用掩码矩阵。对于非平衡二叉树，由于单词的访问路径长度都不是很相似，因此我们可以使用掩码矩阵来进行计算，在这里掩码矩阵标记为 bitmask。当使用掩码矩阵时，相当于将非平衡二叉树填补形成一颗完全二叉树，如图 11 所示。但是那些填补单词的子树都代表该单词，即：存在多条路径指代同一个单词，而在实际求解实验过程中，需要保证一条路径只对应一个单词，因为多余的计算路径会被掩码覆盖掉，无法反映在实际模型的代价函数里面。

接下来详细说明模型的编码方式和所涉及的参数，如表 1 所示。其中，θ_i^w 表示到达叶子单词节点 w 的路径上第 i^{th} 层上的非叶节点，并且 θ_i^w 是一个 m 维度的向量，即：$\theta_i^w\in\mathbb{R}^m$ 其中 $i\in[0,l^w-1]$。同样的，d_i^w 表示连接第 $(i-1)^{th}$ 层和第 i^{th} 层节点的边。对于每个非叶子节点来说，向下移动到左边的分支标记为 -1，选择右边的分支标记为 $+1$。因此：$i\in[0,l^w-1],d_i^w\in\{-1,+1\}$。除此之外，$l^w$ 表示从根节点到叶子单词节点的路径长度。如果该二叉树是平衡树，即单词都均匀分布在同一层叶子节点上，那么二叉树的深度（Tree Depth）是 $l^w\approx\log|\mathcal{V}|$ 。通过这个编码方案，可以使用极性路径来定位每个单词，即：将单词索引（Indexing）或稀疏向量表示（Sparse Representation）改变为单词极性编码元组 (d^w,θ^w)。

在实际运行环境中，需要通过维护一个路径查找表 Γ（Path Looking-up Table），用来记住每个单词 w 从根到叶的所有访问内部节点的索引。这样，就可以从 $\Gamma(w)$ 中选择对应行的所有节点，从参数矩阵 Θ 中检索 θ^w。其中 Θ 的第一维的维度是：

$$\sum_{i=0}^{\log|\mathcal{V}|}2^i=2^0+2^1+\cdots+2^{\log|\mathcal{V}|}=|\mathcal{V}|-1. \tag{3.1}$$

因此，p-tHSM 模型没有增加模型额外参数，除了预先给定的路径查找表 Γ 和预先定义的路径掩码矩阵 bitmask。

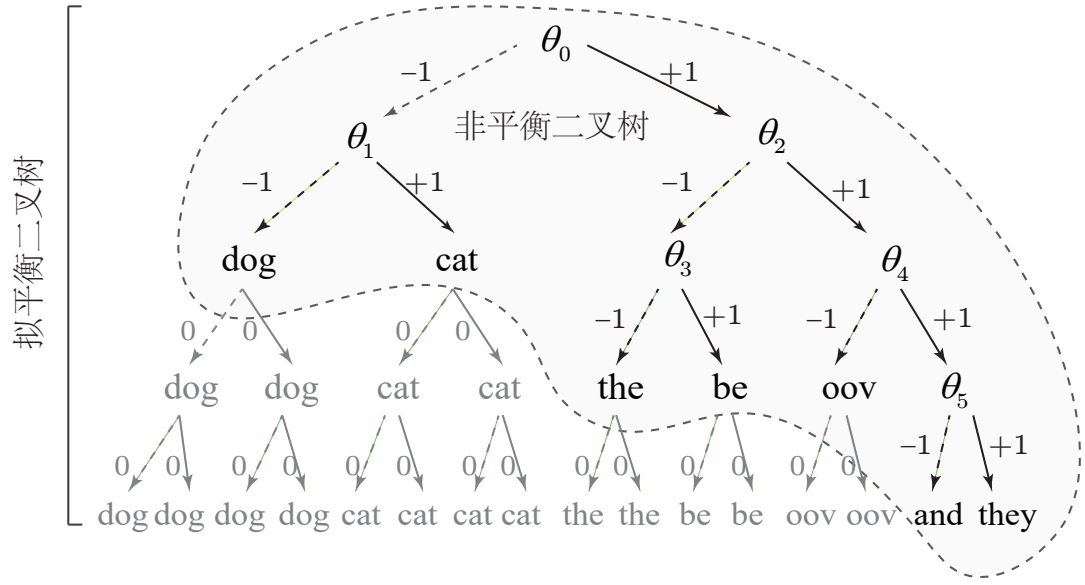

图 **11**　非平衡二叉树调整成平衡二叉树

除此以外，需要从矩阵 $\mathcal{D}$ 中获得第 w^{th} 行向量来检索 d^w，这里所指的 w 是词汇表中的单词索引而不是字符单词，即：$w \in [0, |\mathcal{V}| - 1]$。此外，我们需要区分模型的不同参数是在不同阶段确定的：$\{\Gamma, \mathcal{D}\}$ 是由层次聚类算法预先给定的，Θ 是通过训练数据集上的代价函数的梯度下降来优化的。模型实际调用的各个参数取值如表 2 所示。

表 **2**　**p-tHSM** 算法的编码和参数样例结果

词表	路径查找表 Γ	单词路径参数
dog	-1,-1,0,0,0	θ_0, θ_1
cat	-1,+1,0,0,0	θ_0, θ_1
the	-1,-1,0,0,0	$\theta_0, \theta_2, \theta_3$
be	-1,+1,0,0,0	$\theta_0, \theta_2, \theta_3$
oov	-1,+1,0,0,0	$\theta_0, \theta_2, \theta_4$
and	+1,+1,+1,-1	$\theta_0, \theta_2, \theta_5$
they	+1,+1,+1,+1	$\theta_0, \theta_1, \theta_5$

3.2　基于二叉树的代价函数和导数

接下来给出模型的实际构建过程，如图 12 所示，模型内部参数向量记为 θ_i^w、边记为 d_i^w，单词在树的叶子节点上。其中，浅绿色的路径（从根节点到叶子节点 w）被定义为参数对 (d^w, θ^w)。这里的 d^w 是一个向量，θ^w 是一个参数矩阵。例如, 如图中的参数实际结果是 $d^w = [+1, -1, +1]$，$d^v = [+1, -1, -1]$。

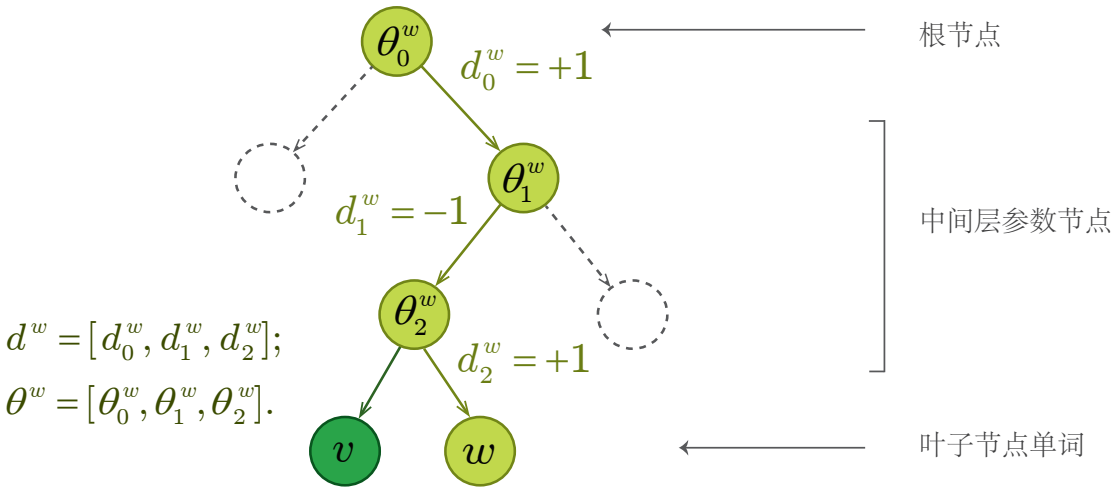

图 **12** 树状层次概率模型

在目标词树的每个步骤中，需要对每个非叶节点是左子树还是右子树进行逻辑预测。所以当给定 i^{th} 节点和隐藏层的输出向量 h 的时候，可以计算标签 $d_i^w \in \{-1, 1\}$ 的概率为：

$$p(d_i^w|\theta_i^w, h) = \sigma(\theta_i^w h)^{d_i^w} \times (1 - \sigma(\theta_i^w h))^{1-d_i^w}, d_i^w \in [0, 1] \tag{3.2}$$

其中 $\sigma(z) = 1/(1 + \exp(-z))$ 表示 σ 函数。根据 σ 函数的对称性规则：$\sigma(z) + \sigma(-z) = 1$，可以进一步改造和优化该公式。以下是该公式的具体证明过程如下：

$$\begin{aligned} \sigma(z) + \sigma(-z) &= \frac{1}{1 + \exp(-z)} + \frac{1}{1 + \exp(z)} \\ &= \frac{1 + \exp(z)}{1 + \exp(z)} = 1 \end{aligned} \tag{3.3}$$

利用 σ 函数的对称性计算规则将公式 (3.2) [47]，缩写成：

$$p(d_i^w|\theta_i^w, h) = \sigma(\theta_i^w h)^{d_i^w}, d_i^w \in [-1, +1] \tag{3.4}$$

由于公式 (3.4) 中的其中某一项 d_i^w 是在指数项上面，所以我们尽管做了这个基于 σ 函数的变换，我们的计算公式仍然不是最优化的计算形式，仍然还是有空间来提升公式的紧凑性，以保证实际计算过程中的高并行度。更进一步的讲，我们可以将上面的公式缩写成以下形式：

$$p(d_i^w = \pm 1|\theta_i^w, h) = \sigma(d_i^w \theta_i^w h) \tag{3.5}$$

我们将指数项移到 σ 函数计算内部，上述计算公式就是我们所需要用到的计算形态。经过这样的操作之后，我们就完成了单节点概率计算的紧凑表示。针对计算一个单词的概率，我们需要考虑从根节点到该单词叶子节点的所涉及的所有层的概率公式。然而因为

在训练的时候，我们实际上预先知道每个单词所对应的路径，所以我们实际上可以同时的计算各个节点的概率值，然后将所有节点相乘起来。这样做的好处将在后续章节进行详细说明。通过以上的操作，虽然在一定程度上增加了计算的复杂度，但是这使得后续在解决计算公式和实际运算流程之间实现一个高效的并行计算成为可能。

因此，对于我们 softmax 函数计算的单词的条件概率，在这里指的是：单词 w 在二叉树上的路径搜寻的概率，即从根到相应叶节点的路径概率 (d^w, θ^w) 的联合乘积。具体计算公式如下所示：

$$\begin{aligned}\log p(w|h) &= \log \prod_{i=0}^{l^w-1} p(d_i^w|\theta_i^w, h) \\ &= \sum_{i=0}^{l^w-1} \log \sigma(d_i^w \theta_i^w h) \\ &= \log \sigma(d^{w\top} \theta^w h) \\ &= \zeta(-d^{w\top} \theta^w h)\end{aligned} \tag{3.6}$$

其中 $\zeta(z)$ 代表的是 softplus 函数: $\zeta(z) = \log(1 + \exp(z))$。该函数的导数是 σ 函数 [48], 其导数计算公式是:

$$\frac{\mathrm{d}\zeta(z)}{\mathrm{d}z} = \frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)} = \sigma(z) \tag{3.7}$$

如图 13 所示，softplus 函数通常被视为修正线性单元（Rectified Linear Unit，ReLU）的替代函数，因为他们两个函数除了零点附近分布不一样以外，其他地方分布相同。ReLU 函数的计算公式为：$\mathrm{ReLU}(x) = \max(0, x)$。但是 softplus 函数在零点附近可导，ReLU 函数在零点附近不可导。ReLU 被使用的更多，因为它能获得近似线性的参数表示，这样能防止模型学习出复杂的表示，缓解模型过拟合问题。

前面我们已经介绍完模型的单词概率计算公式，接下来我们给出模型相应的损失函数 $\ell(\theta|h, w)$，它是定义在二叉树上面的负对数似然函数（Negative Log-Likelihood，NLL），也可以看作是所有单词概率对数误差的平均值：

$$\begin{aligned}\ell(\theta|h, w) &= -\log \prod_{i=0}^{l^w-1} \sigma(d_i^w \theta_i^w h) \\ &= -\log \sigma(d^{w\top} \theta^w h) \\ &= \log(1 + \exp(-d^{w\top} \theta^w h)) \\ &= \zeta(-d^{w\top} \theta^w h)\end{aligned} \tag{3.8}$$

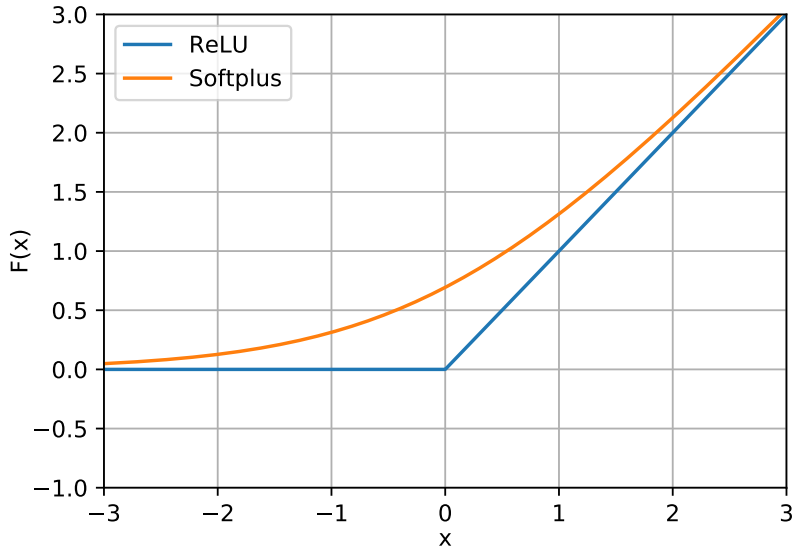

图 **13** **Softplus** 和 **ReLU** 函数的示意图

从此推导公式中，我们可以发现：最小化基于二叉树的负对数似然值，意味着直接最大化 softplus 损失和下一个单词的分布。

接下来，我们比较一下传统 tHSM 算法和我们推导获得的 p-tHSM 算法的异同点。在传统的 tHSM 算法中，该模型逐层计算每个节点的对数概率。因此，这个词的整体联合对数概率通过各层线性相加。从而，tHSM 算法的时间复杂度为 $O(|\mathcal{H}|\log|\mathcal{V}|)$，其计算公式为：

$$\ell(\theta|h,w)=\sum_{i=0}^{l^w-1}\{(1-d_i'^w)\log(\sigma(\theta_i^{w\prime}h))+d_i'^w\log(1-\sigma(\theta_i^{w\prime}h))\} \tag{3.9}$$

其中 $d_i'^w\in\{0,1\}$。我们可以更加形象化的理解这个函数的计算过程：这两个部分对数概率误差从二叉树的叶子底部到根节点，分别对每一层的内部节点的左右子树的联合概率进行合并，其实就是在模仿层次聚类函数从底到顶的合并节点过程。

从公式 (3.9) 所描述的 tHSM 模型和公式 (3.8) 所描述的 p-tHSM 模型二者计算公式不同，我们可以分析出这两个算法的的主要区别在于：1）tHSM 算法涉及许多微小的矩阵乘法操作，而在 p-tHSM 算法中我们直接将所有参数 (d^w,θ^w) 2D 矩阵是以运行时内存消耗为代价的来提高模型单位时间的计算量，我们转而考虑这个向量和相对较大的矩阵的乘法，如图 12 所示；2）除了我们的 p-tHSM 模型的损失函数更加紧凑之外，这条路径上所有节点的对数概率而不是逐层计算而是并行的计算，为 p-tHSM 模型提供更好的时间效率。

当我们计算完模型的代价函数后，我们就需要计算模型的参数的导数。接下来，模

型的所有参数 $\{\theta^w, h\}$ 针对模型的代价函数的导数是：

$$\begin{aligned}\frac{\partial \ell}{\partial \theta^w} =&(\sigma(d^{w\top}\theta^w h) - 1)d^{w\top}h \\ \frac{\partial \ell}{\partial h} =&(\sigma(d^{w\top}\theta^w h) - 1)d^{w\top}\theta\end{aligned} \tag{3.10}$$

在这个模型中，我们采用二叉树分解的方法，主要是考虑到它一个优点是：它避免了在整个词汇表中计算概率的归一化，因为二叉树中所有的单词的总概率自然等于 1。

$$\sum_{w\in\mathcal{V}} p(w|h) = \sum_{w\in\mathcal{V}}\sum_{i=0}^{l^w-1}\sigma(d_i^w\theta_i^w h) = 1. \tag{3.11}$$

3.3　基于二叉树的推理算法

接下来我们介绍，在推理阶段层次概率模型所需要解决的主要问题。

3.3.1　语句打分任务

首先，考虑到本章开始提出的第一个问题，即给定序列 $[w_1, \cdots, w_T]$，求该序列出现的概率 $\log p(w_1, \cdots, w_T)$？直观地说，当给定相应的上下文 h 时，我们可以通过获取一个特定单词 w 的概率或对数概率来分解问题：

$$\begin{aligned}p(w|h) =&\sigma(d^{w\top}\theta^w h) \\ \log p(w|h) =&-\zeta(-d^{w\top}\theta^w h)\end{aligned} \tag{3.12}$$

其中概率 $p(w|h)$ 和对数概率 $\log p(w|h)$ 可以直接通过公式 (3.6) 和 (3.8) 逐步计算出来。

因此，我们可以推导出这个序列的概率为：

$$\log p(w_1, \cdots, w_T) = \sum_{t=1}^{T}\log p(w_t|h_t) = -\zeta(-d^{w_t\top}\theta^{w_t}h_t) \tag{3.13}$$

显然，这种类型的操作比传统的 softmax 方法有效得多，因为它只需要 $O(|\mathcal{H}|\log|\mathcal{V}|)$ 计算复杂度，而传统的 softmax 算法却需要 $O(|\mathcal{H}||\mathcal{V}|)$。

3.3.2　单词排序任务

关于第二种情况（也被称为求解 $\arg\max$ 问题），即当给定一个上下文信息时，在整个词表中搜索最有可能的下一个词（概率最大的单词）。这里的上下文可以指前文、后文或者前后文的窗口。在本实验中，我们所采用的语言模型的情况下，所指代的是前文。

在选择最佳的候选单词之前，词表中所有单词的概率可以被首先计算出，然后将他们按照概率数值大小做排序，获得所有单词的顺序结果。虽然这个过程求解的是全局最优解，但是仍然是相对缓慢的，其计算时间复杂度是 $O(|\mathcal{V}|)$，它涉及到遍历整个词表的

二叉分层树。如在算法 1 中所描述的，为了避免两个小概率的精确度损失问题，我们选择计算每个内部节点的对数概率。

如果我们不采用全局最优搜索算法，而是用局部贪心算法搜索次优结果，那么计算效率能有效提升。具体来说，对于 i-th 层中的节点，当 $p(d_i^w|\theta_i^w,h) \geq 0.5$ 时选择左边的子树，相反的情况下选择右子树，如算法 2 所示。因此，该策略的时间复杂度是 $O(\log|\mathcal{V}|)$。

输入: 隐藏层输出 h;
路径列表 route=[] ;
// 全局概率计算函数;
while $k \leq \log \mathcal{V}$ **do**
  if $p(d_k|\theta_k,h) \geq 0.5$ **then**
    // 左分支;
    $k = k * 2$;
  else
    // 右分支;
    $k = k * 2 + 1$
  end
  // 将 k 添加到路径列表;
  route.append(k)
end
通过查找 Γ 路径表，找出对应的单词;
$w = \Gamma(\text{route})$;
输出: 预测的单词 w.

算法 1: 全局单词最优算法

输入: 隐藏层输出 h;
路径列表 route=[] ;
// 逐层贪心路径搜索;
while $k \leq \log \mathcal{V}$ **do**
  if $p(d_k|\theta_k,h) \geq 0.5$ **then**
    // 访问左分支;
    $k = k * 2$;
  else
    // 访问右分支;
    $k = k * 2 + 1$;
  end
  // 将 k 添加到路径列表;
  route.add(k);
end
// 通过查找 Γ 路径表，找出对应的单词;
$w = \Gamma(\text{route})$;
输出: 预测的单词 w.

算法 2: 逐层贪心搜索算法

3.4　基于二叉树的聚类算法

词汇表中每个单词的极性编码 (d^w, θ^w) 与二叉树的结构密切相关，因此对于提出的 p-tHSM 算法，我们采用了多种树聚类算法来初始化单词在二叉树上的分布，以提高模型的精确性和稳定性。总的来说，这些聚类算法为每个单词生成二进制前缀字符串，表示在树中单词的路径位置，并将用于初始化 p-tHSM 模型。

1）一元单词聚类算法。它根据文本中单词的词频对单词进行排序，并将词频当作权重从底端到顶端进行合并，该算法也称为霍夫曼编码（Huffman Encoding）[19]，算图 14 展示了构造霍夫曼树的过程[49]。如图 15 所示，我们算法中用到输入信息 freq 指的是单词的词频分布，XY 基坐标都是对数坐标。

```
def build_huffman(frequenties):
    Q=Queue.PriorityQueue() # 初始化优先队列 Q;
    for v in frequenties:  #((freq,word),index)
        Q.put(v)
    idx=0
    while Q.qsize()>1:# 由底至顶合并节点
        l,r=Q.get(),Q.get()
        node=Node(l,r,idx)
        idx+=1
        freq=l[0]+r[0]
        Q.put((freq,node))
    return Q.get()[1] # 霍夫曼树
```

图 **14**　基于单词频率的霍夫曼建树策略

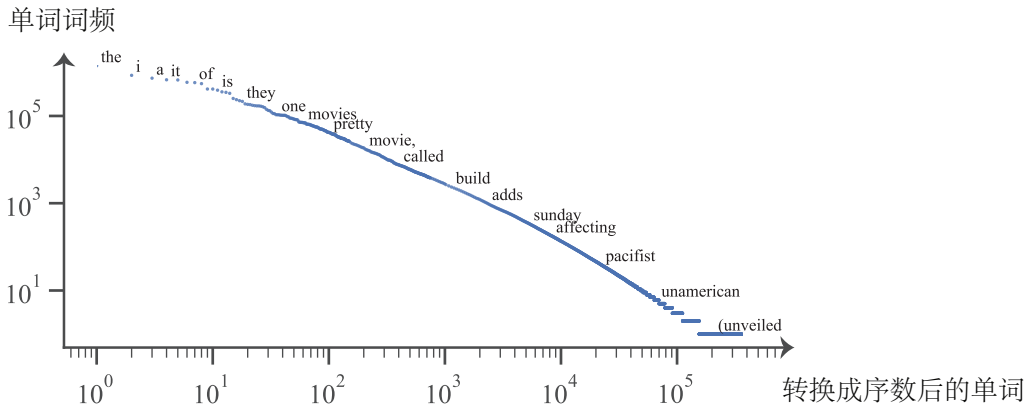

图 **15**　单词词频对数线性分布示意图

当我们构造完霍夫曼二叉树后，我们需要遍历这棵树来获取所有单词的极性编码。1）首先我们计算待压缩数据中所有的字符及其出现次数，根据次数的不同对每个单词分配不同的权值（一般用出现频率/总字符数）；2）其次对所有单词按其权值构造一颗

霍夫曼树; 3）对这棵树所有子树的左支用 −1 编码，右支用 +1 编码。每个位于叶子节点的字符的编码为从根节点到该叶子节点路径上的 −1, +1 编码值, 这个在树上重新得到的编码称为霍夫曼编码。具体遍历过程可以参考图 16，算法的输出单词路径查找表 Γ 就是我们需要提前准备的单词的极性编码。

```
def pre(self,path=None,param=None,words=None):
  # 初始化
  if words is None: words=[]
  if path is None:  path=[]
  if param is None: param=[]

  if self.left: # 左子树存在
    if isinstance(self.left[1],Node):
      self.left[1].pre(path+[-1],param+[self.index],words)
    else: # 访问到左叶子节点
      word=(self.left[1],param+[self.index],path+[-1])
      words.append(word)
  if self.right: # 右子树存在
    if isinstance(self.right[1],Node):
      self.right[1].pre(path+[1],param+[self.index],words)
    else:  # 访问到右叶子节点
      word=(self.right[1],param+[self.index],path+[1])
      words.append(word)
  return words # 单词路径查找表 Γ
```

图 **16** 前序遍历函数生成单词路径查找表

```
class Node(object):
 def __init__(self,left=None,right=None,index=None):
  self.left=left
  self.right=right
  self.index=index

 def __repr__(self):
  string=str(self.index)
  if self.left:
   string+=', -1:'+str(self.left.index)
  if self.right:
   string+=', +1:'+str(self.right.index)
  return string
```

图 **17** 基于邻接表的二叉树节点数据组织结构

上述算法所涉及的模型的各层的节点的实际数据结构如上所示，我们采用邻接表的方式实现。其对应的二叉树的数据结构采用基于数组的方式管理和访问。如图 17 所示，每个节点（Node）包含信息为：左子树，右子树和节点编号。同时我们还定义了节点的

打印字符串函数（即“__repr__”），方便我们检查数据结构是否正确实现。

2）二元聚类算法[1]。它也是一个由底至顶的层次聚类算法，使用 Bi-gram 上下文信息来确定单词的分布相似性，将相似的单词放置在二叉树的附近位置 [9, 50]。算法主要的计算代价就是在于初始化的时候，算法需要计算两两单词之间的距离，对于大词表来说，不断统计 Bi-gram 概率和不断计算两两单词距离是非常费时的。在将簇大小指定为 1 之后，它从底部到顶部合并具有一个节点的单词，生成的单词路径正是二叉树结构的分布.

3）语义信息聚类算法[2]。这种聚类算法与上面的算法非常相似，其差异点就是：该算法使用单词间的距离尺度是语义向量的余弦相似度（Cosine Distance）。我们需要先调用 Word2vec 或者 Fasttext[3] 生成所有单词的词向量，然后再初始化两两单词间的距离矩阵。我们需要看到这里的问题是语言模型的其中一个任务也是生成词向量，所以这个操作可以不断自身迭代来提升最后的模型的效果 [51]。

3.5　本章小结

本章首先定义了极性编码的概念，同时给出了模型所涉及的参数的详细含义。接下来逐步推导模型的单个节点的概率公式，单个词的概率公式和模型的代价函数。另一方面，我们将提出的 p-tHSM 算法和传统的线性 tHSM 算法进行的比较。通过比较两者计算的差异性证明我们提出的算法具有更高的计算吞吐量，速度更快。因为基于二叉树的概率计算方案和传统的 softmax 计算方案不同，不能直接输出单个词的概率或者计算最佳的候选单词，我们进一步还讨论了模型在测试的时候所需的推理算法，所以我们分别针对排序和打分两个任务提出相应的策略。最后，由于单词在二叉树上的分布需要初始化，我们概述了传统的霍夫曼聚类算法，布朗聚类算法和语义向量聚类算法。

[1]https://github.com/percyliang/brown-cluster

[2]https://code.google.com/archive/p/word2vec/

[3]https://github.com/facebookresearch/fastText

第四章　类别层次概率计算模型

上一章介绍了将预测端的词用二叉树的分层的形式表示，在本章中我们将这个目标词表示扩展到一个分层的形式，使它们适配基于类的分层概率计算。首先，我们提出了一个在分层结构上建模的单词编码方案。进而推导出该模型的紧凑的代价函数及其梯度。同时，我们在实验中注意到类别划分的单词分布对其层次概率模型的性能有很大的影响，应需要在训练阶段之前被定义。因为在本次实验中并不考虑那些在训练过程中单词动态交换算法，即一边训练模型一边动态改变了单词群结构。所以在本实验中，我们考虑了几种词表分割的策略，包括：基于单词或者文本的统计、句法和语义知识来初始化其层次结构，使模型能达到稳定和高效的计算性能。

另一方面，在模型推理过程中，不同于传统的 softmax 情况，得到最好的候选者自然是可行的，层次推理不能直接用 softmax 方法来实现。我们讨论基于类的搜索策略的两种不同的推理情况：1）打分：输出给定序列的概率；2）排序：在给定的上下文中获取得分最高的一个候选单词。

4.1　词表划分编码

表 **3** 并行层次概率计算模型的符号助记表

符号	含义	取值范围
$\theta^c \in \mathbb{R}^m$	表示单词的类别向量	Float32
θ^o	表示路径 l^w 表示划分后的单词三维张量	Float32
Γ'	路径查找表，给定单词元祖序列，可以获得对应的单词	—

为了提高 GPGPU 架构的加速比，并支持广义的词汇分割方法，我们提出了一种基于类的分层 softmax 方法的高效通用结构。我们首先来介绍广义的词汇分割算法，词表分割算法能将词表划分成很多个分组。

扁平化的词汇表被分成类别向量 θ^c 和输出矩阵 θ^o 结构，其中前者定义了不同类别的概率 p^c，而后者则模拟了该分区组内的单词概率 p^o，如图 18 所示。输出矩阵的维度是：类维度和分组词维度。如果将词汇 $\mathcal{V}$ 划分为不等大小的组，我们首先将词组分类为等长矩阵，填充零掩码。对于不在这个组中的剩余节点，我们使用掩码向量 θ^m 来消

除其对概率计算和成本汇总过程的影响。因此，分组的文字长度是最大的组长度。如果我们应用相同大小的词汇分割算法，掩码向量则 θ^m 可以被忽略，并且分组的词维度是 $|\mathcal{V}|/|C|$，其中 $|C|$ 表示类维度。这里我们需要注意的是，如果 $|\mathcal{V}|$ 不能被 $|C|$ 整除，那么最后一组中的实际的单词数量应该小于该组的大小。但我们并不打算采用特殊的结构来处理这个问题。因为尽管这些不存在的单词会稀释该组中其他单词的概率大小。但是一方面不影响其他类别的概率计算，另一方面也不影响该组中，其他单词的概率顺序关系。因此实际影响比较小。

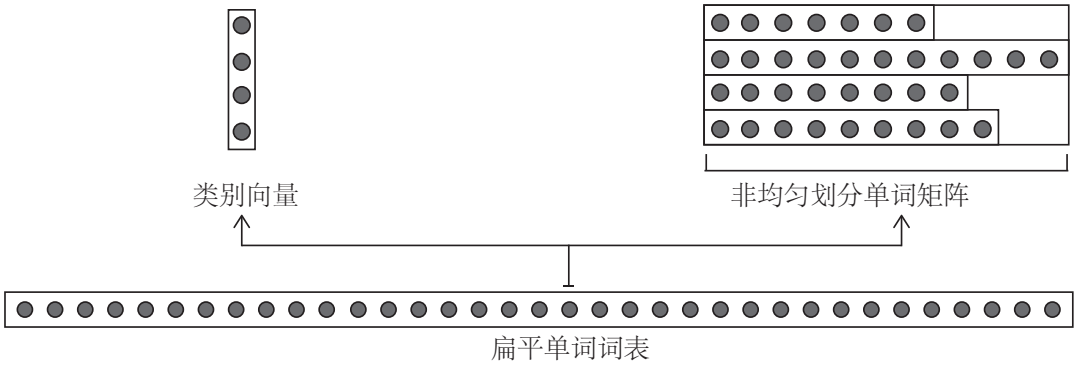

图 **18** 非均匀词表划分结构示意图

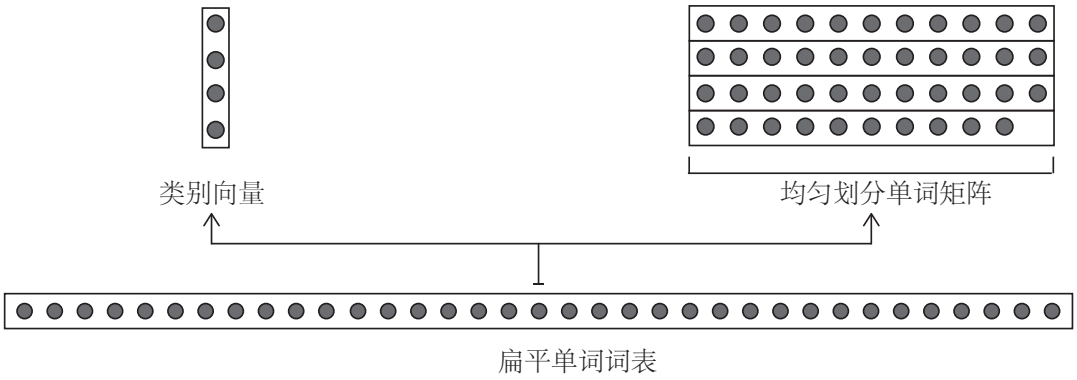

图 **19** 均匀词表划分结构示意图

更重要的是，我们将一维的单词索引替换成一个元组单词索引 (w^c, w^o)，可以利用该元组从我们预先定预先定义的查找表 Γ 中检索给定单词索引 w。除此之外，w^c 指代的是单词的类别编号，w^c 表示该组内的本地单词索引。

$$\theta^m = \begin{cases} \text{单位矩阵}, & \text{若均匀划分} \\ \text{掩码矩阵}, & \text{否则} \end{cases} \tag{4.1}$$

如图 18 所示，扁平化的词汇可以形成右上矩阵，矩阵的大小应该大于或等于词汇表，不需要外部掩码。另外，拼合词汇可以形成左上矩阵，需要外部掩码，矩阵的分布

算法依赖于具体的词表分割算法。

4.2 基于类别的代价函数和导数

当我们定义了单词基于类别的二元编码后，我们接下来讨论基于这种编码的代价函数和导数计算公式。给定最后一个隐藏层输出 h，那么这个分区组中的每个组和每个单词的概率可以被定义为:

$$\begin{aligned} \log p^c(c|h) &= \theta^c h - \log \sum \exp(\theta^c h) \\ \log p^o(w|w^c, h) &= \theta^o h - \log \sum \exp\left(\theta^o h\right) \end{aligned} \tag{4.2}$$

这里我们需要注意的是 p^c 和 p^o 是分别计算而不是并行计算的，因为主要的计算瓶颈是局部标准化的单词概率 p^o，所以就算这两个概率的并行计算也不会达到时间效率提升。所以我们需要把我们的主要工作集中在如何计算 p^o 对应的计算量而不是想方设法使得 p^c, p^o 能够并行计算。例如，假设我们将词表大小为 10000 的单词均匀划分，那么类别向量的维度是 100，单词矩阵的维度是 $[100, 100]$。如果两者能够并行计算，而不是顺序计算，它所带来的计算效率增益可以如下计算：

$$\frac{\frac{1}{100*100}}{\frac{1}{100*100+100}} - 1 = 1\% \tag{4.3}$$

从上面的计算结果来看，我们只能提升 $1/\%$ 的性能。那么如果我们每次只计算对应单词的单词矩阵的概率值那么我们的计算效率可以如下计算：

$$\frac{\frac{1}{100+100}}{\frac{1}{100*100+100}} - 1 = 49.5 \tag{4.4}$$

可以发现，只要我们能避免计算全局概率，那么我们就能实现 49.5 倍的计算效率增益。尽管我们使用二叉树状概率分布模型能获得更好的加速比，但是需要注意的是使用分类概率模型能直接将模型的计算瓶颈给削弱到很少一部分计算时间。而且相对于二叉树状概率模型，分类概率模型的优点有以下两点：1）模型鲁棒性更高，单词的分组分布对模型的精确度影响比二叉树模型要小很多；2）模型实现容易，适应性强。我们只需要将所有单词划分几个分组即可，不需要做层次二叉聚类算法，更况层次二叉聚类算法计算时间复杂度非常高，不适合实际使用。

尽管我们上面介绍了词表的非均匀划分算法，但是需要注意的是掩码矩阵不能直接应用于这个分词矩阵，应该应用在对数概率归一化（log softmax）的计算程序中。为了说明，在 softmax 函数中：$p(x_i) = \exp(x_i)/\sum_j \exp(x_j)$，如果 $x_k = 0$，但其概率 $p(x_k) > 0$

因为 $\exp(x_k) > 0$。即：对于不存在的类别中的单词，我们仍然有概率分布，这样对于那些实际存在的单词概率来说，至多有 $|\mathcal{V}|/\mathcal{C} - 1$ 的类别存在这样的概率稀释问题。

因此，该组中确切的后验词对数概率计算如下：

$$\log p^o(w|w^c, h) = \theta^o h - \log \sum \theta^m \exp(\theta^o h) \tag{4.5}$$

那么，这个模型的损失函数可以形式化地定义成如下形式：

$$\ell(\theta|h) = \log p^c(w^c|h) + \log p^o(w^o|w^c, h) \tag{4.6}$$

其中在类别层次损失等于负对数似然函数（代价函数），并且单词级别的损失需要在计算其损失时指定其类别 w^c。在模型训练过程中，w^c 是需要预先定义的，然而在测试过程中，w^c 需要预先计算最有可能的类别，即由公式 $\tilde{w}^c = \arg\max_c p^c$ 计算获得获取。

当我们定义完模型的代价函数之后，我们接下来将推导得出模型的参数的导数，以便于模型求解优化。我们的模型的所有的参数包括：$\{\theta^c, \theta^o, h\}$，对于模型的代价函数的导数计算公式如下：

$$\begin{aligned}
\frac{\partial \ell}{\partial \theta^c} &= (\delta_{ij} - p(c|h))h \\
\frac{\partial \ell}{\partial \theta^o} &= (\delta_{ij} - p(w|c, h))h \\
\frac{\partial \ell}{\partial h} &= (\delta_{ij} - p^c(c|h))\theta^c + (\delta_{ij} - p^o(w|c, h))\theta^o
\end{aligned} \tag{4.7}$$

这里，我们将词汇分成相互排斥的词组的一个主要优点是：1）避免了在整个词汇表上归一化概率。由于 p^c 是在类维度上概率归一化（Normalization）的，p^o 是在最大组大小维度上进行概率归一化的。所以在第二个方程中，多余的其他组被忽略，在最大的文本数据集中二者都不会超过 1000；2）与基于树的结构相比，它在词汇表上的结构性约束（Structural Constraint）更少，在分解步骤中丢失的信息更少；3）与子字级方法相比，它不会增加序列长度，也不会影响循环神经网络的长距离依赖建模问题。

4.3　基于类别的测试推理

在推理阶段，对于 cHSM 模型来说，序列的概率评分这个任务要容易得多。类似于以前的方法，我们可以通过重新研究我们的训练模型来解决第一个问题：

$$\log p(w_1, \cdots, w_T) = \sum_t^T \log p(w_t|h_t) = \sum_{t=1}^T \log p^c(w_t^c|h_t) + \log p^o(w_t^o|w_t^c, h_t) \tag{4.8}$$

我们发现这种类型的操作比传统的 softmax 方法有效得多，该方法涉及 $O(|\mathcal{H}|\sqrt{|\mathcal{V}|})$ 计算复杂度。

输入: 隐藏层输出 h
for $c \in C$ **do**
  // 计算类别的概率;
  Calculate $\log p^c(c|h)$;
  for $w \in c$ **do**
    // 计算单词的条件概率;
    Calculate $\log p^o(\hat{w}^o|\hat{y}^c, h)$;
    // 计算每个单词全局概率;
    $\log p(w|h) = \log p^c(w^c|h) + \log p^o(\hat{w}^o|\hat{w}^c, h)$;
  end
end
$w = \arg\max_w \log p(w|h)$;
输出: 概率最大的候选单词 w。

算法 3: 基于 cHSM 算法的全局 arg max 算法

其次，对于 arg max 的情况，我们可以在选择最佳候选项之前计算词汇表中所有单词的概率，这在直观上是正确的，但在，因为涉及到整个单词的分割矩阵。此外，我们仍然可以在算法 4 中对上述方法进行少量修改，然后计算确切的最高候选人。

此外，cHSM 算法的性能对词表划分算法有些敏感，因为某些方法可能会产生高度不平衡的字组，并且这种单词不均匀分布会在算法中产生标签偏差问题（Label Bias）。第一个本地 $\arg\max_o$ 进程 [52]。然而，在大多数情况下，如果选择合适的参数，则可以考虑不平衡的问题，本文考虑平衡词汇分区的广义形式。

我们接下来来详细说明标签偏差问题。考虑两个类 c_p 和 c_q，这两个类的包含的单词数量是不同的，我们不妨假设 $|c_p| \leq |c_q|$。在计算了最后一个隐藏层输出 h 与类图层参数的相似性之后，我们继续计算 h 与每个组的内部单词 w 的相似性得分，这些单词在每个特定组中都没有进行局部规范化整个词汇。当 $|c_p| \approx |c_q|$ 表示我们希望将词汇聚类成等大小的群组，而不是高度倾斜的群组分布时，可以减轻标签偏差问题，其中 $|c_p| \ll |c_q|$。更具体地说，对于分布不均的情况，对于类 c_q 中的单词来说这个概率被一大群单词稀释是不公平的，这样算法更有可能以较高的概率取出这个小组中的单词，放弃在其他大集团有更多的潜在的话。

我们可以用局部贪心算法搜索次优结果，而不是搜索确切的全局最优结果，而是建议将这个 arg max 进程分解为两个阶段：1）计算类概率，并剔除顶端一个 $\hat{c}$；2）计算该

输入: 隐藏层输出 h
// 挑选每个类中概率最大的类别;
$\hat{w}^o = \arg\max_o \log p^o(w|c,h)$;
// 在这些已经被栅选单词中挑选最佳的单词
$\tilde{w}^c = \arg\max_c \log p^c(w^c|h) + \log p^o(\hat{w}^o|\hat{y}^c,h)$;
通过查找表 Γ' 将 $(\tilde{w}^c, \hat{w}^o[\tilde{w}^c])$ 替换成单词 w ;
输出: 概率最大的候选单词 w。

算法 4: 基于 cHSM 算法的正确 arg max 算法

输入: 隐藏层输出 h;
// 挑选概率最大的类别;
$\hat{w}^c = \arg\max_c \log p^c(c|h)$;
$\hat{w}^o = \arg\max_o \log p^o(w|\hat{w}^c,h)$;
// 在这个类别下面，挑选概率最大的单词;
通过查找表 Γ' 将 $(\hat{w}^c, \hat{w}^o)$ 替换成单词 w ;
输出: 最佳的候选单词 $\hat{w}$.

算法 5: 基于 cHSM 模型伪 arg max 算法

类别的单词概率 $\hat{c}$，并选择具有最高本地单词概率的单词。这个算法会给 psudo 最好的候选人，但是与原始算法 4相比，它的运行速度要快得多。而且，由于分组词在本地进行归一化，标签偏差问题可以在一定程度上缓解。在实验研究中将讨论算法 4 和 5 的详细不同性能。

4.4　词表划分算法

由于 cHSM 模型的性能与其词汇分割算法密切相关，我们将聚类算法的现有工作进行汇总，并将可能的方法分类如下：

4.4.1　均匀词表划分算法

1）随机初始化。这种直观的方法忽略了单词的所有外部信息，因此单词与随机随机播放过程是等分的。这是揭示其他聚类方法下界的最坏情况，也可以揭示应用高级聚类策略的相对收益。

2）字母顺序。这种方法根据字符级别的信息对单词进行排序，同一组中的单词共享一个相似的子字符串。

3）一元单词聚类。这些单词首先根据它们在文本中的频率排序，然后通过放置边界使得每个类别占总概率质量的恒定部分，从而形成连续单词块。这种方法具有这样的性质：较低编号的类比较高编号的类具有更少的成员，因为它们的成员更频繁 [19]。

4.4.2 非均匀词表划分算法

1）二元单词聚类。它是指布朗聚类方法，这是历史适用于基于 n-gram 的基于类的模型 [9, 50]。单词使用相同的 bigram 上下文分组到相同的行中。

2）结构聚类[1]。根据文本中的单词的词性和句法结构划分词表 [53] 。

3）语义聚类。我们将传统的 kmeans 聚类方法应用到预训练的词嵌入，使得我们可以通过指定聚类的大小将词汇分成不同的形状。

4.5 本章小结

本章首先定义了基于词表划分的编码概念，同时给出了模型所涉及的参数的详细涵义。接下来，我们逐步推导模型的单个节点的概率公式，单个词的概率公式和模型的代价函数。另一方面，我们将提出的 p-cHSM 算法和传统的线性 cHSM 算法进行的比较。通过比较两者计算的差异性证明我们提出的算法更适合在 GPGPU 等高并行设备上运算。进一步的，我们还讨论了模型在测试的时候所需的推理算法，因为基于划分词表的概率计算方案和传统的 softmax 计算方案不同，不能直接输出单个词的概率或者计算最佳的候选单词，所以我们分别针对这两个任务提出推理算法。最后，由于单词在单词划分矩阵上的分布需要初始化，我们讨论了传统的霍夫曼硬聚类算法，布朗软聚类算法和语义向量软聚类算法等等。同时也讨论三种算法的实际应用过程。

[1]https://github.com/AlonDaks/unsupervised-authorial-clustering

第五章　语言建模实验结果与分析

前面两章分别介绍了基于树状和类别的层次概率模型的具体建模过程。在本章中，为了比较所提出的这两种分层概率模型与已有的算法（Baselines）在计算效率和精确性上的差异，我们将在三个标准文本数据集上进行循环语言建模任务的实验研究和结果分析。本章将讨论分析不同并行层次概率算法的实验结果，还会与其他大词表问题的优化加速算法相互对比。接下来还将从效率、可扩展性、参数大小等方面，对这些已有的方法进行实验研究并作讨论分析。

5.1　实验设置

这一小节将介绍实验所需要用到的文本数据集，还有语言建模实验的两种主要的评测指标和实验模型的实际参数配置和训练过程。

5.1.1　实验数据集

本实验所采用的数据集主要是依照两个目的选取的：1）首先为了便于实验复现和互相对比，需要在小数据集上反映出参数变化对模型的最后的效果产生的影响；2）同时还需要大数据集上，展示模型参数在最佳参数配置下，比较模型之间的最优结果。所以在本次实验中，我们选取了三个标准文本数据集：Wikitext-2，Wikitext-103 和 One Billion Words 数据集。如表 4 所示，表中列举了这三个数据集的相关统计指标，包括：单词数量、句子数量、词表大小和词表外单词的比例（Out-of-Vocabulary）。需要注意的是，词表大小是不能改变，因为不同词表规模的模型之间理论上是没有互相比较的意义，其次是由于接下来要计算的一个重要的评测指标和词表大小成负相关关系。所以表 4 中展示的数据已经预先固定了，不会再做任何的修改，例如：不能对数据集做单词大小写变化、数词转换操作或者分词操作。

其中，对于 Wikitext-2 和 Wikitext-103 数据集，训练、验证和测试集都是预先划分固定的，并且其词汇表大小也已经被定义[1]。这两个数据集拥有相同的验证和测试集，而 Wikitext-103 的训练集则比 Wikitext-2 的训练集大得多。所以我们还可以间接测算出增

[1]https://metamind.io/research/the-wikitext-long-term-dependency-language-modeling-dataset/

表 4 **WikiText-2, WikiText-103** 和 **One Billion Words** 数据集统计指标

数据集	类型	文章数	句子数量	单词数量	词表大小	OOV （%）
Wikitext-2	训练集	600	36,718	2,088,628	33,278	2.6%
	验证集	60	3,760	217,646		
	测试集	60	4,358	245,569		
Wikitext-103	训练集	28,475	1,801,350	103,227,021	267,735	0.4%
	验证集	60	3,760	217,646		
	测试集	60	4,358	245,569		
One Billion Words	训练集	—	30,301,028	768,646,526	793,471	0.28%
	验证集	—	6,075	153,583		
	测试集	—	6,206	159,354		

大训练集对测试集上的提升效果。对于第三个 “One Billion Words” 数据集[2]，它是由历史机器翻译数据集积累的文本融合而成，文本数据也取自维基百科（Wikipedia）[3]。为了保证评价的标准性和实验的可互比性，官方还提供了一套数据预处理脚本，同时指定了该脚本所需要的 Perl 语言版本，可见要求极为严苛。因为对于语言模型来说，Perl 内置函数版本不同，处理出来的文本也略有不同，语言模型之间的结果就不在同一个基准之上。通常来说，词表小的模型更占优势。当用官方提供的脚本处理完数据后，我们将 “./train/” 目录中的所有数据视为训练集，选择 “./holdout/” 目录下面的第一和第二个数据集作为相应的验证和测试集。这些数据集的详细统计指标在表 4 中进行了说明。

5.1.2 实验评价指标

在本次实验研究中，当实验模型在验证数据集上训练到收敛后，我们就需要评价不同模型的性能差异，针对语言模型的两个标准评估度量标准来揭示针对不同的优化方法的优劣：困惑度（Perplexity，PPL）和单词误差率（Word Error Rate，WER）。其中，PPL 是一个内在度量指标（Intrinsic Metric），代表了在不同语境中选择下一个候选单词时的困惑程度。语言模型困惑度较低，意味着在相同词表下拥有更好的可预测性。此外，在整个测试集上，PPL 数值是与平均负对数似然值（NLL）呈指数相关关系，这表明训练

[2]http://www.statmt.org/lm-benchmark/

[3]https://en.wikipedia.org/wiki/Main_Page/

过程中模型直接优化了 PPL 评测指标，其数学定义如下所示：

$$\mathrm{PPL}(w_1,\cdots,w_T)=\sqrt[T]{\frac{1}{\prod_{t=1}^{T}p(w_t|w_{1:t-1})}} \tag{5.1}$$

此外，WER 表示单词的莱文斯坦距离（Levenshtein Distance），用于衡量参考句子（Reference，记作 r）和预测句子（Hypothesis，记作 h）之间的相似度。它是编辑距离的一种衍生类型，被定义为错误识别的单词（删除，插入，替换）占总单词的百分比[4]：

$$\mathrm{WER}=\frac{\text{插入的单词数}+\text{删除的单词数}+\text{替换的单词数}}{\text{全部单词数量}} \tag{5.2}$$

其数学形式的定义公式为：

$$d_{r,h}(i,j)=\begin{cases}\max(i,j) & \text{如果 } \min(i,j)=0\\ \min\begin{cases}d_{r,h}(i-1,j)+1, \text{//插入的单词}\\ d_{r,h}(i,j-1)+1, \text{//删除的单词}\\ d_{r,h}(i-1,j-1)+1\{r_i\neq h_j\}, \text{//替换的单词}\end{cases} & \text{否则}\end{cases} \tag{5.3}$$

其中 $1\{r_i\neq h_j\}$ 指代的是示性函数，当且仅当 $r_i=h_j$ 的时候取值为 0，否则该函数取值为 1。$d_{r,h}(i,j)$ 表示的是参考句子 r 的第一个 i 个字符与预测句子 r 的第一个 j 个字符之间的距离。同时莱文斯坦距离还满足度量的延展性关系（三角不等式），即：两个字符串的距离不大于分别与第三个字符串的距离之和：

$$d_{r,h}+d_{r,s}\geq d_{r,s} \tag{5.4}$$

其中 s 代表另外一个生成的句子，$d_{r,h}$ 表示的是句子 r 和句子 h 之间的编辑距离。

除了以上的定量的度量指标外，训练时间效率，词表可伸缩性和运行时内存消耗等定性度量指标也应该被视为衡量不同模型的重要衡量维度。因此，我们在实验中分别从 GPGPU 和 CPU 的实际运算结果分析了不同优化算法在不同评估指标上的具体性能。最后，统计和评测了所有模型在三个标准文本数据集上的最终结果。

5.1.3　模型训练和参数配置

接下来来介绍实验模型的参数设置和训练过程。在具体算法实现中，每个模型都是使用 Theano 框架实现，而且都运行在一个独立的 GPGPU 设备上，模型不在多显卡上运算，模型之间不互相干扰。GPGPU 设备具有 12GB 的显存（设备型号是 Nvidia K40m），

[4]https://martin-thoma.com/word-error-rate-calculation

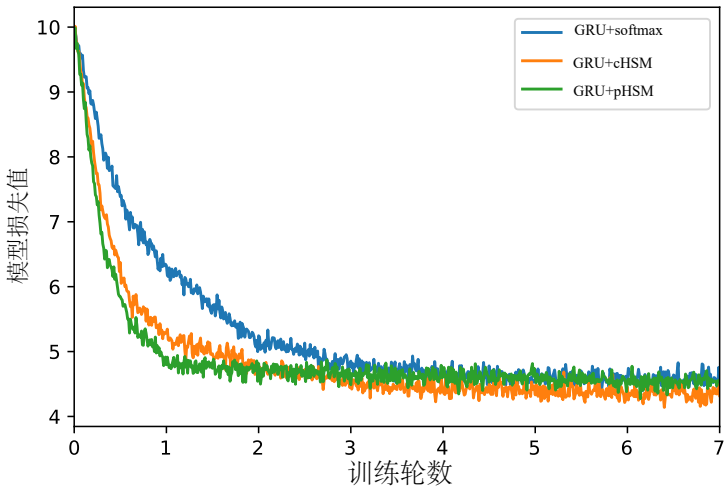

图 **20** 模型训练曲线图

保证能进行大矩阵乘法的运算，所以我们能测算出大词表问题的具体计算代价。然而我们发现，在实际试验中随着模型参数维数的增加，单个 GPGPU 显存资源被快速利用殆尽。

然后是针对数据集做必要的预处理。对于 Wikitext-2 数据集，最大句子长度固定为 256；对于 Wikitext-103 数据集，最大句子长度固定为 100；对于 OBW 数据集来说，最大句子长度固定为 50，因为第三个数据集的词表最大，需要占用更多的显存资源，所以句子长度有所缩减。对于长度超过阈值的那部分字符串将直接删除，由于测量的是语言模型的单词级损失（Word-Level Loss）而不是句子级的分数（Sentence-Level Loss），因此删除部分的那句子对模型训练的影响是很小的。

另外在实验中，Adam 优化器（Optimizer）配合两种不同的学习率（Learning Rate）作为模型的优化函数，即设置 $\mathrm{lr} = 0.06$ 和 0.001。两种词表层次分解方法收敛速度较慢需要采用较大的学习率，而传统的 softmax 和采样近似方则需要采用较小的学习率。除此以外，为了减弱模型收敛过程中的损失值的振荡现象，每隔一定步数（$\mathrm{Step} = 100$），学习率还需要逐渐缩小：$\mathrm{lr} \leftarrow \mathrm{lr} * 0.9$。

对于在 Wikitext-2 数据集上运行的实验，我们设置批处理大小（Batch Size）为 20 最大遍历周期（Epoch）为 15，直到我们观察到验证集上的最小 PPL 结果。验证集上的损失数值约 4.8，这是最好的验证上的损失值。而对于 Wikitext-103 数据集，在训练集上优化参数需要大约 5-10 个轮数，因为 Wikitext-103 训练集大约是 Wikitext-2 的 50 倍。此外，训练集损失数值大约在 5.1 左右，因为在 wikitext-103 数据集上更高的原因是我们预测的词汇量要大得多，所以混淆程度（即训练损失）应该更高。虽然我们发现模型可以很

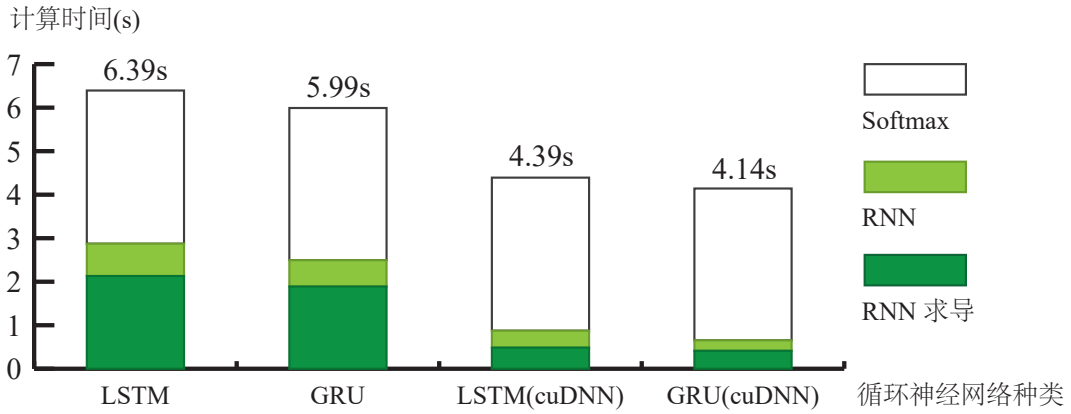

图 **21** **wikitext-103** 数据集上测量语言模型三个部分的计算时间比较

容易地收敛到局部最小点，验证误差在这个局部最小值的范围内振荡，但是 Wikitext-2 和 Wikitext-103 的唯一区别是训练数据的大小，表示收集很多训练数据不一定能保证模型在相同的验证和测试集上学得更好，训练集大小增大的同时词表也增大了 3 倍。

此外，我们在实验中发现在 OBW 数据集上运行实验相当具有挑战性，因为它需要更大的参数来拟合。所以我们应用 CUDA 实现的 RNN 模型 [54]，这将会把 RNN 部分所需要的计算时间降到最低，然而等待模型收敛到训练集最小值仍然需要 480 小时。

5.2　影响因素比较

这部分将讨论语言模型的大词表问题在具体实验中瓶颈和各种不同优化策略对对该问题的计算效率和性能的提升和分析。

5.2.1　词表层次化比较

首先在 wikitext-103 数据集，我们统计了语言模型中每个模块的计算时间消耗，如图 21 所示。我们计算运行不同的 RNN 模型（即 LSTM 模型和 GRU 模型）及其梯度（即，RNN Grad）所需的时间，以及 Wikitext-103 数据集的大词汇表上的常规 softmax，这词汇表大小与我们平时所采用的数据集的词表相当。我们分别用 Theano 框架和 CUDA 实现了 RNN 单元，一个采用 python 语言实现，第二个使用并行 C++ 语言实现。目前基于 CuDNN 实现的 RNN 模型计算时间最快，它里面的 RNN 的运行时间可以缩短到最短。我们将代码重复了 100 遍，统计了每个模块的总时间占用，然后分别计算其平均时间占用。这样做的目的是通过多次实验来保证实验数据准确性。

从图中可以看出，使用优化的 CUDA，softmax 模块的影响比 RNN 单元及其梯度更重要。Softmax 计算时间占用总计算时间随着词表的增大占比越来越大，并且已经超过

表 **5 Wikitext-103** 数据集上 **GPGPU** 和 **CPU** 的运行时内存和计算时间比较

算法	运行时内存占用	总计算时间 (ms)		前向计算时间 (ms)	
		CPU	GPGPU	CPU	GPGPU
Softmax	$\|\mathcal{HV}\|$	510.4	262.1	352.2	62.9
cHSM	$2\|\mathcal{H}\|\sqrt{\|V\|}$	506.5	**40.6**	28.7	14.6
tHSM	$\|\mathcal{H}\|$	1,004.0	444.4	8.1	5.6
p-tHSM	$\|\mathcal{H}\|\log\|\mathcal{V}\|$	**383.5**	86.4	**7.0**	**1.4**

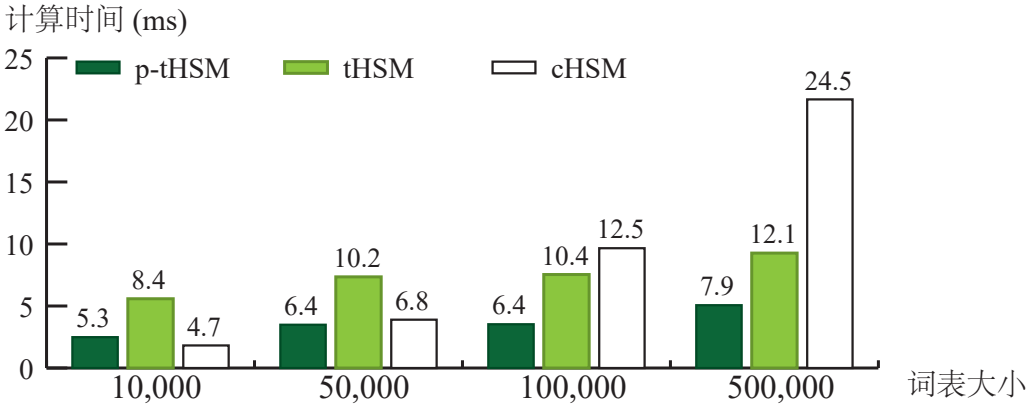

图 **22 cHSM, tHSM** 和 **p-tHSM** 算法随着词表大小的计算时间影响

了 50% 的计算时间，这就需要我们详细讨论 softmax 优化。

为了对用于训练相同批处理数据的经验时间复杂性和内存消耗进行基准测试，我们使用这些算法在表 5 中收集了 GPGPU 和 CPU 上的详细结果。我们尝试使用这些算法处理 WikiText-103 数据集，并计算处理一个批处理数据所需的平均时间。另外，输入句子的最大长度，隐藏层，输出词汇和批量大小分别设置为 $\{50, 256, 267735, 20\}$。此外，“总时间”过程表示前向传播和后向梯度优化的过程，“前进时间”过程表示从输入数据到计算模型成本所需的时间消耗。此外，我们计算了上述算法训练期间加载所需的内存。$|\mathcal{V}|$ 表示词汇大小，$|\mathcal{H}|$ 是隐藏层维度。在训练期间，tHSM 只消耗最小的存储器资源，而 p-tHSM 算法覆盖了更大的存储器集合，并且 p-tHSM 在考虑存储器消耗和速度时采用更大的存储器并且获得了更好的加速比。

为了验证与词汇大小相关的 cHSM，tHSM 和 p-tHSM 算法的可扩展性，结果展示在图 22 中。为了显示 p-tHSM 算法的影响，我们在这里不包含“Softmax”方法，因为与其他算法相比，它消耗了更多的计算时间，无法被放在这个图里面。很显然，cHSM 随着词汇大小的平方根（即，$O(|\mathcal{H}|\sqrt{|V|})$）进行缩放，而 p-tHSM 随着词汇大小的增加

呈现出稳定的表现。tHSM 算法随着此表大小的对数进行变化（即，$O(|\mathcal{H}|\log|\mathcal{V}|)$），我们的 p-tHSM 算法所需要的计算时间词表变大，与 tHSM 算法的差异越来越大。这一结果证明我们提出的计算模型更加优越。

在这些实验的基础上，我们可以得出结论：所提出的 p-tHSM 方法胜过历史记录 $O(|\mathcal{H}|\log|\mathcal{V}|)$），并且取得了令人满意的分层 softmax 方法的加速比。这种性能归因于基于 GPGPU 并行性的加速，也是由于 p-tHSM 方法的基本结构，能够将目标字树进行并行地计算。

5.2.2　搜索策略的影响

由于我们提出了三个关于推理阶段搜索策略的算法，其影响可以通过 WER 度量来观察。在这个结构化预测过程中，一个合理的搜索规则将帮助模型获得最小风险的最佳候选人，如表 7 所示。算法 1表示我们计算所有单词的得分，单词的概率用整个词汇全局归一化。

对于 p-tHSM 系列算法，我们比较了所提出的算法 2和传统的算法 1。算法 2比算法 1方法计算结果花费的时间更少。由于基于树的模型在逐层访问路径的过程中更容易出现错误情况，所以算法 1的 WER 分数要比算法 2 要好。

表 **6**　**Wikitext-2** 数据集上 **p-tHSM** 算法针对不同搜索算法的 **WER** 评测结果

	算法类别	计算时间（ms）	验证集（WER）	测试集（WER）
p-tHSM	全局计算（算法 1）	161	**76.67%**	**75.35%**
	贪心计算（算法 2）	**30**	79.61%	79.32%

值得注意的是，对于 cHSM 方法算法 2 比算法 4 得到更好的 WER，尽管后一种方法获得了词汇表上的确切顶级候选。由于类结构中存在标签偏差问题，排名最高的词容易出现算法模型无法建模的小组。最后但并非最不重要的是，算法 4 和 1获得相同的单词排序，唯一的区别是算法 4 避免了冗余计算。所以他们达到了可比的 WER 得分，但算法 4 需要更少的时间。

5.2.3　循环网络模型的影响

从表 8 中可以看出，拥有门控单元（即 LSTM 和 GRU 单元）的 RNN 模型在 PPL 和 WER 指标方面比传统的 RNN Relu 和 RNN Tanh 模型表现得更好。因为门控功能可以避免梯度消失的问题，虽然它需要稍微多一些的时间进行计算。此外，LSTM 的计算

表 7 **Wikitext-2** 数据集上 **cHSM** 算法针对不同搜索算法的 **WER** 评测结果

	算法类别	计算时间 (ms)	验证集（WER）	测试集（WER）
cHSM	算法 3	102	80.00%	80.02%
	算法 4	63	80.00%	80.02%
	算法 5	**44**	**77.09%**	**77.07%**

表 **8** **Wikitext-2** 数据集上不同循环网络针对 **PPL**、**WER** 和计算时间的影响

循环神经网络	计算时间（ms）	验证集（PPL / WER）	测试集（PPL / WER）
1×RNN Relu [55]	176.4	260.52 / 80.00%	238.75 / 80.02%
1×RNN Tanh [38]	176.2	250.57 / 79.61%	230.98 / 79.32%
1×LSTM [23]	**189.5**	180.98 / 77.16%	165.60 / 76.67%
1×GRU [56]	191.3	**179.59 / 77.09%**	**165.32 / 77.07%**
2×RNN Relu [55]	266.3	190.52 / 73.01%	198.75 / 73.02%
2×RNN Tanh [38]	266.3	189.57 / 72.62%	260.98 / 72.32%
2×LSTM [23]	**279.4**	164.98 / 71.17%	165.60 / 71.67%
2×GRU [56]	281.2	**158.59 / 70.08%**	**155.32 / 70.07%**

公式在 GPGPU 上比 GRU 单元更容易并行运行，因此 LSTM 比 GRU 模型需要更少的计算时间，尽管 LSTM 模型计算量更大。

对于传统的 RNN 模型，通过时间反向传播（Back Propagation Through Time，BPTT）进行训练，梯度计算步骤中的努力花费在该训练批次中最长序列的长度上是线性的。因此，小批量生产可能效率低下，因为批次中的次序可能有不同的长度。这个问题的一个经典的解决方案是截断 BPTT 算法，因此 RNN 模型的梯度在截断长度上是恒定的，并且对于 GPGPU 设备上的小批量处理是高效的。而前向概率计算步骤仍然是相同的，唯一的影响是避免梯度反向传播步骤。从图中可以看出，较小的截断 BPTT 在时间效率上取得了较好的效果，较长的截断 BPTT 长度在 PPL 度量方面效果更好。

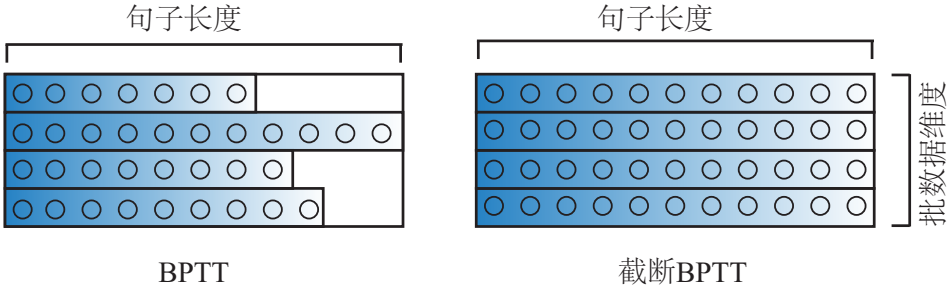

图 **23** **BPTT** 和截断 **BPTT** 算法示意图

在训练过程中，如果某些依赖关系存在在被截断的语句中，则模型可以学习这种关系；如果这些依赖关系总是跨越不同的语句块，那么截断的语句已经打破了这些依赖，模型将无法学习这种关系。这是由于模型的参数导数无法回溯到最初的跨段的那个单词，并告诉前段中的某些内容对下一段是有用的。如果在一个语句段中存在这种关系的话，模型能学习到这种关系是由于模型的参数能求导到前面对应的单词。与之相区别的是，在测试时模型将能够在跨语句段中预测，因为模型的计算步骤是不断向前传播，不存在模型反向求导的计算过程。

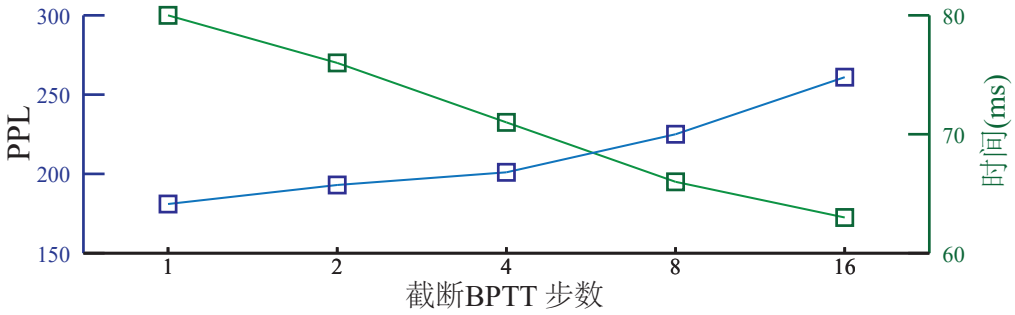

图 **24** **Wikitext-2** 数据集上 **BPTT** 和截断 **BPTT** 算法对 **RNN** 的影响

5.2.4　采样近似算法比较

基于抽样的算法的效率和准确性与样本大小密切相关，我们在这小节评测了两种采样算法的性能。我们测试不同采样样本数 k 对模型的训练和测试的结果的影响，结果展示在图 25中。从图中分析，Blackout 算法收敛性比传统的 NCE 模型相对更好，但是 NCE 算法计算效率比 Balckout 更高。所以单位时间里面 NCE 收敛速度更快，单位参数迭代更新步骤中 Balckout 算法收敛更快，这与 Ji 等人的实验结果是一致的 [38]。同时，我们还发现，当 $k = 1$ 时，采样函数计算的就是真正的二元分类概率，此时采样算法振荡现象很剧烈，无法收敛。当逐渐增大采样样本数 k，采样算法才能逐渐开始收敛。两种算法的加速比是 V/k，与 k 呈负相关关系。

5.2.5　单词聚类策略分析

Chen 等人发现 cHSM 和 ptHSM 算法的性能对单词的层次聚类算法很敏感 [30]。同样，为了获得较为稳定的 cHSM 算法和 p-tHSM 方法，我们考虑了几种现有的聚类准则，如表 9 和表 10 所示。

在对于基于二叉树的层次概率算法初始化过程中，我们采用了霍夫曼聚类，布朗聚类和词向量聚类方法来生成不同的单词路径分布，详细的结果在表 9 中给出。与能调

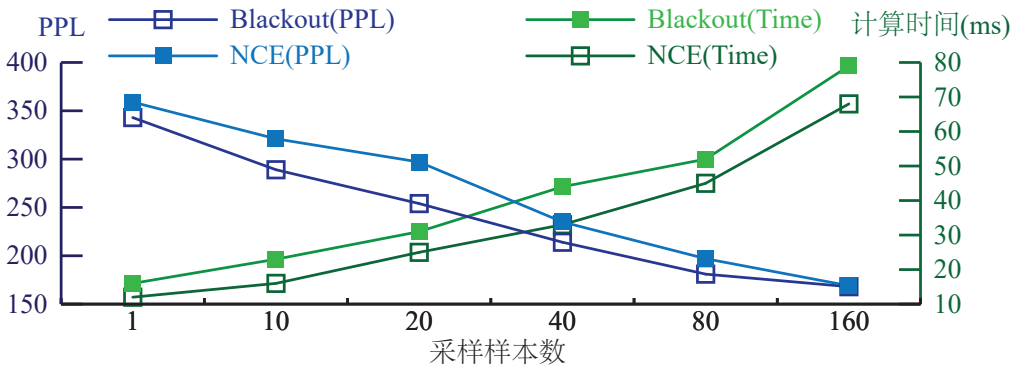

图 **25** **Wikitext-2** 上测试不同采样数量对 **NCE** 和 **Blackout** 算法的影响

整分支因子的 cHSM 方法不同，p-tHSM 算法由于聚类结果最后都合并到根节点，所以没有相关变量可以测量。Uni-gram 聚类方法（即按照词频合并）在创建单词层次结构方面效率更高，而 Bi-gram 和语义聚类需要花费巨额时间来计算词汇表中单词之间的相似度矩阵。尽管如此，对于单词树合并的规则，Bi-gram 方法考虑了二元共现单词的统计信息，语义聚类方法测度的是特征空间中的欧氏距离。在困惑度量指标衡量下，采用 Bi-gram 和语义聚类的方法比采用霍夫曼聚类的方法有更好的结果。最后，与 cHSM 方法相比，更深层次的树模型更适合于词汇聚类，适当的聚类可以提高树层次的效率。

表 **9** **Wikitext-2** 上评价不同聚类方法对 **p-tHSM** 算法的 **PPL** 影响

算法	建树时间	最大树深度	验证集（PPL）	测试集（PPL）
Uigram	3 分钟	12	218.42	216.05
Bigram	35 小时	21	186.23	189.58
Semantic	26 小时	18	**163.12**	**178.78**

对于表 10，我们将前面所有提到的聚类方法应用在 Wikitext-2 数据集上，，同时比较了不同的分支因子对算法效果的影响。从该表的结果分析得出，相比于其他算法随机方法的性能最差，因为它们没有提供有效的先验信息。此外还观察到，该方法比其他人更难以接受可接受的训练损失，因此在训练集上花费更多的时间来优化参数。然后，发现在对类结构注入外部单字和双字的知识之后，模型确实在目标空间中学习了一个明确定义的单词分布。因此，它对随机和字母表系列取得了更好的结果。而且，用语法和语义算法进行分词的词汇比上述方法得到的结果要好得多，代价是计算分割方法。从实验结果可以看出，向具有先验知识的类结构注入会增强该方法的稳定性，通过平衡聚类时间和模型的准确性，可以调整分支因子，达到预期的结果。

表 **10 Wikitext-2** 数据集上不同聚类算法对 **cHSM** 算法的 **PPL** 和 **WER** 影响

聚类算法	均匀划分？	分支数	训练轮数	测试集（PPL / WER）	耗时（ms）
Random	是	10/3330	145	211.15 / 78.55	791
		20/1664	123	228.72 / 78.89	565
		40/832	103	234.36 / 79.21	321
		80/417	78	243.12 / 79.64	171
		160/208	57	253.38 / 80.08	92
		182/183	48	268.63 / 80.11	88
Alphabet	是	10/3330	141	199.01 / 78.07	773
		20/1664	120	211.34 / 78.23	551
		40/832	100	238.75 / 79.02	313
		80/417	90	241.75 / 79.34	174
		160/208	56	248.35 / 79.62	97
		182/183	45	258.57 / 80.02	87
Uni-gram	是	10/3330	134	211.51 / 77.41	788
		20/1664	122	220.01 / 77.71	549
		40/832	113	236.56 / 77.95	302
		80/417	91	241.12 / 78.25	170
		160/208	55	247.25 / 79.21	93
		182/183	42	253.35 / 79.92	86
Bi-gram	否	10/3672	150	208.11 / 77.32	801
		20/1923	121	217,34 / 77.64	621
		40/1123	102	228.87 / 78.14	588
		80/572	89	246.32 / 78.43	186
		160/340	76	252.33 / 79.51	97
Syntactic	否	10/3612	152	214.31 / 78.11	810
		20/1972	130	220.19 / 78.86	633
		40/996	101	232.33 / 79.33	543
		80/545	89	241.34 / 79.84	179
		160/235	70	262.34 / 80.14	134
Semantic	否	10/3570	133	208.77 / 77.41	819
		20/1873	114	218.31 / 77.78	641
		40/1092	91	225.38 / 78.35	521
		80/561	69	238.45 / 78.91	174
		160/244	44	256.75 / 79.41	103

5.3　模型总体评价

如表 11 所示，我们收集了上述三种标准语料库的验证和测试数据集的所有困惑和错误率结果。值得注意的是，我们采用了一层 GRU 单元作为所有这些算法的上下文表示，其维数设置为 256。另外，对于 NCE 和 Blackout 近似，超参数 k 是对较小的 Wikitext-2 和 Wikitext-103 数据集设置为 $|\mathcal{V}|/20$，对于较大的 One Billion Words 数据集，设置为

$k = |\mathcal{V}|/200$。此外，对于 cHSM 方法，我们根据单词的单字分布来划分词汇。

考虑到 Wikitext-2 数据集上的结果，最初的 softmax 比其他算法获得了最好的分数，因为在 Blackout 和 NCE 近似中没有引入任何 cHSM 和 p-tHSM 算法的结构损失或基于抽样的变分损失。

表 **11** 所有模型在三个数据集上的 **PPL** 和 **WER** 的性能评测

数据集	算法	验证集（PPL/WER）	测试集（PPL/WER）
WikiText-2	GRU + Softmax	172.64 / 77.49%	162.09 / 77.07%
	GRU + NCE [55]	217.84 / 78.26%	199.54 / 78.02%
	GRU + Blackout [38]	221.15 / 77.72%	199.56 / 77.50%
	GRU + cHSM + Uni-gram [30]	253.18 / 78.25%	236.61 / 78.02%
	GRU + p-tHSM + Uni-gram [19]	218.42 / 78.15%	216.05 / 78.15%
	GRU + p-tHSM + Bi-gram [9]	186.23 / 78.15%	189.58 / 78.15%
WikiText-103	GRU + Softmax	130.38 / 72.15%	136.83 / 72.37%
	GRU + NCE [55]	164.78 / 73.22%	165.01 / 73.34%
	GRU + Blackout [38]	163.99 / 73.18%	162.76 / 74.22%
	GRU + cHSM + Uni-gram [30]	171.81 / 73.42%	166.74 / 73.18%
	GRU + p-tHSM + Uni-gram [19]	165.70 / 73.53%	166.11 / 72.44%
	GRU + p-tHSM + Bi-gram [9]	164.15 / 78.15%	163.55 / 77.85%
One Billion Words	GRU + Softmax	330.38 / 88.15%	330.83 / 88.37%
	GRU + NCE [55]	272.07 / 84.83%	276.11 / 84.34%
	GRU + Blackout [38]	268.67 / 84.23%	266.11 / 84.18%
	GRU + cHSM + Uni-gram [30]	225.36 / 80.32%	224.11 / 79.42%
	GRU + p-tHSM + Uni-gram [19]	231.44 / 87.53%	236.11 / 82.53%

对于第二个 Wikitext-103 数据集，布朗聚类的 p-tHSM 方法不仅获得了比霍夫曼聚类方法更好的结果，而且表现也比其他方法好。另外，cHSM 模型能够获得与 p-tHSM 变体类似的结果，表明我们可以用其他合适的用于 cHSM 方法的聚类算法获得更好的结果。由于 Wikitext-103 和 Wikitext-2 数据集共享相同的测试集，因此发现最初的 softmax 通过更大的训练数据被收敛到更好的结果。此外，对于采样方法，它收敛于比 softmax 方法好得多的结果，同时提高了时间效率。

综上所述，在用字极性编码方案替代 tHSM 中的传统霍夫曼编码方案并且实现了基于紧密的树型模型 p-tHSM 之后，我们证明了这种新颖的编码方案允许在 GPGPU 上并

行运行计算。这将原始 tHSM 的时间复杂度从 $O(|\mathcal{H}|\log|\mathcal{V}|)$ 减少到 $O(|\mathcal{H}||\mathcal{V}|)$ 并获得了最佳的加速比对于大量的词汇问题。此外，为了稳定 p-tHSM 模型的性能，我们测试了几种现有的层次聚类算法，发现基于树模型的词聚类与内部节点的二元分类密切相关。

5.4　本章小结

在本章中，首先定量分析了大词表问题，接下来比较了我们提出的模型的计算效率和传统算法之间的差异。针对层次模型的初始化算法，我们讨论并评估三种不同的层次聚类算法，以更有效的方式组织结构化的词表。结果表明，与其他概率归一化方法相比，我们提出的层次概率算法具有很好的加速比，与其他基于抽样的优化算法相比，它性能相对更好。

参 考 文 献

[1] 王建翔. 面向可读性评估的词向量技术研究及实现 [D]. 南京, 中国 : 南京大学, 2017.

[2] Hinton G E, Salakhutdinov R R. Reducing the dimensionality of data with neural networks[J]. Science, 2006, 313(5786) : 504 – 507.

[3] Wang Z, Wang D. A Joint Training Framework for Robust Automatic Speech Recognition[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2016, 24(4) : 796 – 806.

[4] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate[A]. Proceedings of 2nd International Conference on Learning Representations[C]. 2015.

[5] Bengio Y, Ducharme R, Vincent P. A Neural Probabilistic Language Model[A]. Proceedings of 14th Annual Conference on Neural Information Processing Systems[C]. 2000 : 932 – 938.

[6] Elman J L. Finding Structure in Time[J]. Cognitive Science, 1990, 14(2) : 179 – 211.

[7] Mikolov T, Karafiát M, Burget L, et al. Recurrent neural network based language model[A]. Proceedings of 11th Annual Conference of the International Speech Communication Association[C]. 2010 : 1045 – 1048.

[8] Mikolov T, Kombrink S, Burget L, et al. Extensions of recurrent neural network language model[A]. Proceedings of the 2011 IEEE International Conference on Acoustics, Speech, and Signal Processing[C]. 2011 : 5528 – 5531.

[9] Brown P F, Pietra V J D, de Souza P V, et al. Class-Based n-gram Models of Natural Language[J]. Computational Linguistics, 1992, 18(4) : 467 – 479.

[10] Baroni M, Dinu G, Kruszewski G. Don’t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors[A]. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics[C]. 2014 : 238 – 247.

[11] Bell R M, Koren Y. Lessons from the Netflix prize challenge[J]. SIGKDD Explorations, 2007, 9(2) : 75 – 79.

[12] Bengio Y, Courville A C, Vincent P. Representation Learning: A Review and New Perspectives[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013, 35(8): 1798–1828.

[13] Bengio Y, Simard P Y, Frasconi P. Learning long-term dependencies with gradient descent is difficult[J]. IEEE Transactions on Neural Networks, 1994, 5(2): 157–166.

[14] 赵林, 杨保安, 谢志鸣. 一种新的基于结构的神经网络规则抽取方法 [J]. 计算机应用与软件, 2007, 24(6): 28–29.

[15] Mnih A, Hinton G E. Three new graphical models for statistical language modelling[A]. Proceedings of 24th International Conference on Machine Learning[C]. 2007: 641–648.

[16] Mnih A, Kavukcuoglu K. Learning word embeddings efficiently with noise-contrastive estimation[A]. Proceedings of 27th Annual Conference on Neural Information Processing Systems[C]. 2013: 2265–2273.

[17] Mnih A, Teh Y W. A fast and simple algorithm for training neural probabilistic language models[A]. Proceedings of the 29th International Conference on Machine Learning[C]. 2012.

[18] Mikolov T. Statistical language models based on neural networks[J]. Presentation at Google, Mountain View, 2nd April, 2012.

[19] Mikolov T, Sutskever I, Chen K, et al. Distributed Representations of Words and Phrases and their Compositionality[A]. Proceedings of 27th Annual Conference on Neural Information Processing Systems[C]. 2013: 3111–3119.

[20] Chien J, Ku Y. Bayesian Recurrent Neural Network for Language Modeling[J]. IEEE Transactions on Neural Networks and Learning Systems, 2016, 27(2): 361–374.

[21] Bengio Y, Ducharme R, Vincent P, et al. A Neural Probabilistic Language Model[J]. Journal of Machine Learning Research, 2003, 3: 1137–1155.

[22] 贾玉祥, 王浩石, 昝红英, et al. 汉语语义选择限制知识的自动获取研究 [J]. 中文信息学报, 2014, 28(5): 66–73.

[23] Greff K, Srivastava R K, Koutník J, et al. LSTM: A Search Space Odyssey[J]. IEEE Transactions on Neural Networks and Learning Systems, 2017, 28(10): 2222–2232.

[24] Chung J, Kastner K, Dinh L, et al. A Recurrent Latent Variable Model for Sequential Data[A]. Proceedings of 29th Annual Conference on Neural Information Processing Systems[C]. 2015 : 2980 – 2988.

[25] Bradbury J, Merity S, Xiong C, et al. Quasi-Recurrent Neural Networks[A]. Proceedings of 4th International Conference on Learning Representations[C]. 2017.

[26] Hochreiter S, Schmidhuber J. Long Short-Term Memory[J]. Neural Computation, 1997, 9(8) : 1735 – 1780.

[27] 陈凯. 深度学习模型的高效训练算法研究 [D]. 安徽, 中国 : 中国科学技术大学, 2016.

[28] Pezeshki M. Sequence Modeling using Gated Recurrent Neural Networks[J]. CoRR, 2015, abs/1501.00299.

[29] Duda R O, Hart P E, Stork D G. Pattern Classification (2Nd Edition)[M]. Indianapolis, USA : Wiley-Interscience, 2000.

[30] Chen W, Grangier D, Auli M. Strategies for Training Large Vocabulary Neural Language Models[A]. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics[C]. 2016.

[31] Schwenk H. Continuous space language models[J]. Computer Speech & Language, 2007, 21(3) : 492 – 518.

[32] Tucker R C F, Carey M J, Parris E S. Automatic language identification using sub-word models[A]. Proceedings of the 1994 IEEE International Conference on Acoustics, Speech and Signal Processing[C]. 1994 : 301 – 304.

[33] Sennrich R, Haddow B, Birch A. Neural Machine Translation of Rare Words with Subword Units[A]. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics[C]. 2016.

[34] Gage P. A New Algorithm for Data Compression[J]. The C Users Journal, 1994, 12(2) : 23 – 38.

[35] Józefowicz R, Vinyals O, Schuster M, et al. Exploring the Limits of Language Modeling[J]. CoRR, 2016, abs/1602.02410.

[36] Kim Y, Jernite Y, Sontag D, et al. Character-Aware Neural Language Models[A]. Proceedings of the 30th AAAI Conference on Artificial Intelligence[C]. 2016 :

2741 – 2749.

[37] Bengio Y, Senecal J. Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model[J]. IEEE Transactions on Neural Networks, 2008, 19(4): 713 – 722.

[38] Ji S, Vishwanathan S V N, Satish N, et al. BlackOut: Speeding up Recurrent Neural Network Language Models With Very Large Vocabularies[A]. Proceedings of 3rd International Conference on Learning Representations[C]. 2016: 1 – 14.

[39] Zoph B, Vaswani A, May J, et al. Simple, Fast Noise-Contrastive Estimation for Large RNN Vocabularies[A]. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies[C]. 2016: 1217 – 1222.

[40] Gutmann M, Hyvärinen A. Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics[J]. Journal of Machine Learning Research, 2012, 13: 307 – 361.

[41] 王龙, 杨俊安, 陈雷, et al. 基于循环神经网络的汉语语言模型并行优化算法 [J]. 应用科学学报, 2015, 33(3): 253 – 261.

[42] Goodman J. Classes for fast maximum entropy training[A]. Proceedings of the 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing[C]. 2001: 561 – 564.

[43] Morin F, Bengio Y. Hierarchical Probabilistic Neural Network Language Model[A]. Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics[C]. 2005.

[44] Cline D, Razdan A, Wonka P. A Comparison of Tabular PDF Inversion Methods[J]. Computer Graphics Forum, 2009, 28(1): 154 – 160.

[45] Kronmal R A, Peterson A V. On the Alias Method for Generating Random Variables from a Discrete Distribution[J]. The American Statistician, 1979, 33(4): 214 – 218.

[46] Vaswani A, Zhao Y, Fossum V, et al. Decoding with Large-Scale Neural Language Models Improves Translation[A]. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing[C]. 2013: 1387 – 1392.

[47] Minka T. Algorithms for maximum-likelihood logistic regression[A]. Virtual Communities and Social Capital. Social Science Computer Review[C]. 2001.

[48] Dugas C, Bengio Y, Bélisle F, et al. Incorporating Second-Order Functional Knowledge for Better Option Pricing[A]. Proceedings of 14th Annual Conference on Neural Information Processing Systems[C]. 2000 : 472 – 478.

[49] 牛雪婷. Huffman 算法的分析与应用 [J]. 中国科教创新导刊, 2009(28) : 87 – 87.

[50] Liang P. Semi-supervised learning for natural language[J]. Master's Thesis Mit, 2005.

[51] Aggarwal C C, Zhai C. Mining Text Data[M]. Berlin, Germany : Springer, 2012.

[52] Lafferty J D, McCallum A, Pereira F C N. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data[A]. Proceedings of the 18th International Conference on Machine Learning[C]. 2001 : 282 – 289.

[53] Daks A, Clark A. Unsupervised Authorial Clustering Based on Syntactic Structure[A]. Proceedings of the ACL 2016 Student Research Workshop[C]. 2016 : 114 – 118.

[54] Appleyard J, Kociský T, Blunsom P. Optimizing Performance of Recurrent Neural Networks on GPUs[J]. CoRR, 2016, abs/1604.01946.

[55] Gutmann M, Hyvärinen A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models[A]. Proceedings of the 30th International Conference on Artificial Intelligence and Statistics[C]. 2010 : 297 – 304.

[56] Chung J, Gülçehre Ç, Cho K, et al. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling[J]. CoRR, 2014, abs/1412.3555.

[57] Fu R, Guo J, Qin B, et al. Learning Semantic Hierarchies via Word Embeddings[A]. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics[C]. 2014 : 1199 – 1209.

[58] Chen Y, Wang W Y, Rudnicky A I. Learning semantic hierarchy with distributed representations for unsupervised spoken language understanding[A]. Proceedings of 16th Annual Conference of the International Speech Communication Association[C]. 2015 : 1869 – 1873.

[59] Chen X, Wang Y, Liu X, et al. Efficient GPU-based training of recurrent neural network language models using spliced sentence bunch[A]. Proceedings of 15th Annual Confer-

ence of the International Speech Communication Association[C]. 2014 : 641 – 645.

[60] Mnih A, Hinton G E. A Scalable Hierarchical Distributed Language Model[A]. Proceedings of 22nd Annual Conference on Neural Information Processing Systems[C]. 2008 : 1081 – 1088.

[61] Al-Rfou R, Alain G, Almahairi A, et al. Theano: A Python framework for fast computation of mathematical expressions[J]. CoRR, 2016, abs/1605.02688.

[62] Marcus M P, Santorini B, Marcinkiewicz M A. Building a Large Annotated Corpus of English: The Penn Treebank[J]. Computational Linguistics, 1993, 19(2) : 313 – 330.

[63] Paul D B, Baker J M. The design for the wall street journal-based CSR corpus[A]. Proceedings of the 2nd International Conference on Spoken Language Processing[C]. 1992.

[64] Zhou M. Softplus Regressions and Convex Polytopes[J]. CoRR, 2016, abs/1608.06383.

[65] de Brébisson A, Vincent P. The Z-loss: a shift and scale invariant classification loss belonging to the Spherical Family[J]. CoRR, 2016, abs/1604.08859.

[66] Sundermeyer M, Ney H, Schlüter R. From Feedforward to Recurrent LSTM Neural Networks for Language Modeling[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2015, 23(3) : 517 – 529.

[67] Grave E, Joulin A, Cissé M, et al. Efficient softmax approximation for GPUs[A]. Proceedings of the 34th International Conference on Machine Learning[C]. 2017 : 1302 – 1310.

[68] Hochreiter S. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions[J]. International Journal of Uncertainty Fuzziness and Knowledge-Based Systems, 1998, 6(2) : 107 – 116.

[69] Mikolov T, Chen K, Corrado G, et al. Efficient Estimation of Word Representations in Vector Space[J]. CoRR, 2013, abs/1301.3781.

[70] Derczynski L, Chester S, Bøgh K S. Tune Your Brown Clustering, Please[A]. Recent Advances in Natural Language Processing[C]. 2015 : 110 – 117.

[71] Vincent P, de Brébisson A, Bouthillier X. Efficient Exact Gradient Update for training Deep Networks with Very Large Sparse Targets[A]. Proceedings of 29th Annual Confer-

ence on Neural Information Processing Systems[C]. 2015 : 1108 – 1116.

[72] Baltescu P, Blunsom P. Pragmatic Neural Language Modelling in Machine Translation[A]. Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies[C]. 2015 : 820 – 829.

[73] Merity S, Xiong C, Bradbury J, et al. Pointer Sentinel Mixture Models[A]. Proceedings of 4th International Conference on Learning Representations[C]. 2017.

[74] Parada C, Dredze M, Sethy A, et al. Learning Sub-Word Units for Open Vocabulary Speech Recognition[A]. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies[C]. 2011 : 712 – 721.

[75] Clark A. Combining Distributional and Morphological Information for Part of Speech Induction[A]. Proceedings of 10th Conference of the European Chapter of the Association for Computational Linguistics[C]. 2003 : 59 – 66.

[76] Martin S C, Liermann J, Ney H. Algorithms for bigram and trigram word clustering[J]. Speech Communication, 1998, 24(1) : 19 – 37.

[77] Chen X, Liu X, Wang Y, et al. Efficient Training and Evaluation of Recurrent Neural Network Language Models for Automatic Speech Recognition[J]. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2016, 24(11) : 2146 – 2157.

[78] Kleinbaum D G, Klein M. Maximum likelihood techniques: An overview[J]. Logistic regression, 2010 : 103 – 127.

[79] Chelba C, Mikolov T, Schuster M, et al. One billion word benchmark for measuring progress in statistical language modeling[A]. Proceedings of 15th Annual Conference of the International Speech Communication Association[C]. 2014 : 2635 – 2639.

[80] Jean S, Cho K, Memisevic R, et al. On Using Very Large Target Vocabulary for Neural Machine Translation[A]. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics[C]. 2015 : 1 – 10.

[81] Jordan M I. Serial order: A parallel distributed processing approach[J]. Advances in psychology, 1997, 121 : 471 – 495.

[82] Gatt A, Krahmer E. Survey of the State of the Art in Natural Language Generation: Core

tasks, applications and evaluation[J]. CoRR, 2017, abs/1703.09902.

[83] Li J, Ouazzane K, Kazemian H B, et al. Neural Network Approaches for Noisy Language Modeling[J]. IEEE Transactions on Neural Networks and Learning Systems, 2013, 24(11): 1773 – 1784.

[84] Dehdari J, Tan L, van Genabith J. BIRA: Improved Predictive Exchange Word Clustering[A]. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies[C]. 2016: 1169 – 1174.

[85] Cohn T, Lapata M. An abstractive approach to sentence compression[J]. ACM Transactions on Intelligent Systems and Technology, 2013, 4(3): 41:1 – 41:35.

[86] Li Z, Tang J, Wang X, et al. Multimedia News Summarization in Search[J]. ACM Transactions on Intelligent Systems and Technology, 2016, 7(3): 33:1 – 33:20.

[87] Józefowicz R, Zaremba W, Sutskever I. An Empirical Exploration of Recurrent Network Architectures[A]. Proceedings of the 32nd International Conference on Machine Learning[C]. 2015: 2342 – 2350.

[88] Qu K, Chai X, Liu T, et al. Computer-Aided Diagnosis in Chest Radiography with Deep Multi-Instance Learning[A]. Proceedings of 24th International Conference on Neural Information Processing[C]. 2017: 723 – 731.