北京航空航天大学计算机学院

硕士学位论文文献综述

论文题目: 神经网络语言模型的性能优化研究

专　　业: 计算机科学与技术

研究方向: 自然语言处理

作　　者: 姜 楠

学　　号: SY1506330

指导教师: 荣文戈 副教授

北京航空航天大学计算机学院

2016 年 12 月 20 号

摘要

语言模型是机器翻译和语音识别的主要应用领域。随着循环神经网络的发明，将其应用到语言模型上会使得语言建模更加精确，但是因此而带来的是模型速度下降。随着文本数据增大，单词的词表会变得十分巨大，导致计算代价非常大。历史上人们采用双层分类函数和多层分类函数来解决该问题，但是因此而带来的单词聚类问题变得更加苛刻。

本文首先介绍了语言模型的主要历史发展，回顾了自然语言处理领域中对上下文的建模、大词表问题的现有方案，以及历史上所采用的聚类模型主要模拟方法和算法。在建模方法中着重介绍了基于神经网络建模方法，即基于循环神经网络的建模方法。然后重点讨论了多层聚类算法的相关研究成果。最后本文还展望了对于不同算法的结合的成果和拟未来发展的几个方向。

关键词: 神经语言模型；循环神经网络；层次多元概率模型

Abstract

Historically, Language models has huge impact on machine translation or speech recognition. with the development of recent recurrent neural network, many researchers apply these advances on language model. Thus, there comes recurrent language models while they are lack of efficiency. To be specific, when meeting with larger language model dataset, the vocabulary size becomes overwhelming time-consuming.

Traditionally, researchers utilise class-based softmax and tree-based hierarchical softmax for solving large vocabulary problem, which depends on word clustering algorithms. So we investigate possible hierarchical clustering method for this application. To conclude, we analyse every possible unit within neural language model for speeding up its efficiency and keep its precision and accuracy. Furthermore, we also depict several possible direction for future study.

Keyword: Neural Language Model; Recurrent Neural Network; Hierarchical Softmax

目录

1 引言

近年来，随着 Web2.0 的兴起，互联网上的数据急剧膨胀。根据国际数据公司（International Data Corp.,IDC）的统计和预测，2011 年全球网络数据量已经达到 1.8ZB(1.8×10^6TB)，到 2020 年，全球数据总量预计还将增长 50 倍。大量无标注数据的出现，也让研究人员开始考虑，如何利用算法从这些大规模无标注的文本数据中自动挖掘规律，得到有用的信息。2006 年, Hinton 提出的深度学习 (Deep Learning, DL)[1], 为解决这一问题带来了新的思路。在之后的发展中，基于神经网络的表示学习技术开始在各个领域崭露头角。尤其在图像和语音领域的多个任务上，基于表示学习的方法在性能上均超过了传统方法。

近年来，深度学习逐渐在自然语言处理中 (Natural Language Processing, NLP) 得到应用. 研究者提出用神经网络 (Neural Network, NN) 来训练语言模型并进行了相关探索 [2]. 历史上，主要的语言建模方案 (Language Modeling, LM) 主要分为: 前馈神经网络语言模型、对数双线性语言模型 (Log Bi-Linear, LBL) 和循环神经网络语言模型。其中，基于循环神经网络的语言模型建模方法引起了研究者极大的兴趣 [3]. 网络通过学习能够将当前词的历史信息存储起来，以词的整个上下文 (Context) 作为依据，来预测下一个词出现的概率，克服了 n-gram 语言模型无法利用语句中长距离上下文信息的缺点. 另外，在模型训练的过程中，由于词的历史信息被映射到低维连续空间，语义相似的词被聚类，在语料中出现次数较少的词仍然能够得到很好的训练，不再需要额外的数据平滑技术 (Smoothing). 迄今为止，采用 (Recurrent Neural Network, RNN) 训练的语言模型在模型困惑度 (Perplexity, PPL) 和识别系统的识别率上都取得了最好的效果 [4].

RNN 建模方法虽然表现出极大的优越性，却以牺牲计算复杂度为代价. 若训练大规模的文本语料，则需要花费很长的时间，制约了 RNN 语言模型训练效率. 为克服这一不足，文献 [5] 提出了多种优化策略来降低网络的计算复杂度，如缩短模型训练周期、减少训练数据集的规模、降低训练词典的大小、减少隐含层的节点数等，这些方法都在一定程度上降低了网络的运算量，提高了模型的训练效率，但同时也牺牲了较多的模型性能和冗余度. 另外，在网络结构层面上，文献 [3] 研究了一种基于分类的循环神经网络 (Class-based RNN) 结构，网络的输出层被分解为两部分，增加的一部分称为分类层，从结构上降低了整个网络的计算复杂度，使得模型训练效率有了一定的提升且模型性能没有大的变化. 然而，在大词汇量连续语音识别系统中，采用此结构训练大规模语料语言模

型仍需要花费大量时间. 因此，模型训练效率有待进一步优化.

因此探讨研究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暂时延缓该问题的后果，但是一旦应用到大数据集上，即使是目前最好的中央处理单元 (Central Processing Units, CPUs) 或者图像处理单元 (Graphical Processing Unit,GPU), 仍然需要三五天时间才能训练完善。因此，在保证原有模型的准确率的目的下，如何提高模型的训练速度是我们主要讨论的内容。为此我们讨论了三个不同的方向：一种是通过采样技术 (Importance Sampling, IS) 来减少必要的训练时间; 一种是通过基于类别的多元分类 (Class-based Hierarchical Softmax, cHSM) 来加速模型; 最后一种是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 GPU 设备分别进行探讨。因为传统的线性运算模型在流行的 GPU 并行运算方案中并不适用，需要结合不同的运算设备分别讨论可行的方案。

2 语言模型概述

2.1 N-gram 语言模型

语言模型可以对一段文本的概率进行估计，对信息检索 [4]、机器翻译 [5]、语音识别等任务有着重要的作用。形式化讲，统计语言模型的作用是为一个长度为 m 的字符串确定一个概率分布 $P(w_1; w_2; \cdots; w_m)$，表示其存在的可能性，其中 w_1 到 w_m 依次表示这段文本中的各个词。一般在实际求解过程中，通常采用下式计算其概率值：

$$\begin{aligned} P(w_1; w_2; \cdots; w_m) = P(w_1)P(w_2|w_1)P(w_3|w_1; w_2) \cdots P(w_i|w_1; w_2; \cdots; w_{i-1}) \\ \cdots P(w_m|w_1; w_2; \cdots; w_{m-1}) \end{aligned} \tag{1}$$

在实践中，如果文本的长度较长，公式 1 右部 $\cdots P(w_m|w_1; w_2; \cdots; w_{m-1})$ 的估算会非常困难, 因为出现 $w_1; w_2; \cdots; w_{m-1}; w_m$ 的语段非常少，进而该模型的稀疏性特别严重。因此，研究者们提出使用一个简化模型：n 元模型 (n-gram model)。在 n 元模型中估算条件概率时，距离大于等于 n 的上文词会被忽略，也就是对上述条件概率做了以下近似：

$$P(w_i|w_1; w_2; \cdots; w_{i-1}) \approx P(w_i|w_{i-(n-1)}; \cdots; w_{i-1}) \tag{2}$$

当 $n=1$ 时又称一元模型（unigram model），公式2 右部会退化成 $P(w_i)$，此时，整个句子的概率为：$P(w_1;w_2;\cdots;wm)=P(w_1)P(w_2)\cdots P(w_m)$。从式中可以知道，一元语言模型中，文本的概率为其中各词概率的乘积。也就是说，模型假设了各个词之间都是相互独立的，文本中的词序信息完全丢失。因此，该模型虽然估算方便，但性能有限。

当 n = 2 时又称二元模型（bigram model），将 n 代入公式2 中，右部为 P$(w_i|w_{i-1})$。常用的还有 n = 3 时的三元模型（trigram model），使用 $P(w_i|w_{i-2};w_{i-1})$ 作为近似。这些方法均可以保留一定的词序信息。

2.2 前馈神经网络语言模型

N-gram 语言模型的一个显著缺陷是：基于 2 式，新词和低频词难以得到有效的概率统计。基于此，人们发明了各种平滑算法，如 discount, back-off, interpolation 等。这些方法在一定程度上改善了 n-gram 在低频词上的性能，但基于模型本身的缺陷，这一困难始终无法从根本上解决。

随着神经网络的兴起，人们开始尝试利用神经网络构造语言模型。与 n-gram 不同，神经网络对参数进行高度共享，因此对低频词具有天然的平滑能力。神经网络语言模型 (Neural Network Language Model, NNLM) 的最早由 Bengio 等人在 2001 年提出 [2], 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 [3,4,5,6,8]，但总体而言, 对 NNLM 的研究还处在起步阶段。具体而言，NNLM 通过一个多层感知网络 (MultiLayer Perceptron, MLP) 来计算 2 式中概率。图 1 给出一个典型的 NNLM 语言模型。神经网络语言模型采用普通的三层前馈神经网络结构，其中第一层为输入层。Bengio 提出使用各词的词向量作为输入以解决数据稀疏问题，因此输入层为词 $w_{i-(n-1)};\cdots;w_{i-1}$ 的词向量的顺序拼接：

$$x=[e(w_{i-(n-1)};\cdots;e(w_{i-2});e_{(w_{i-1})}] \tag{3}$$

当输入层完成对上文的表示 x 之后，模型将其送入剩下两层神经网络，依次得到隐藏层 h 和输出层 y：

$$\begin{aligned} h=&tanh(b(1)+Hx)\\ y=&b(2)+Wx+Uh \end{aligned} \tag{4}$$

其中 $H\in\mathbb{R}^{|h|\times(n-1)|e|}$ 为输入层到隐藏层的权重矩阵，$U\in\mathbb{R}^{|\mathrm{V}|\times(n-1)|h|}$ 为隐藏层到输出层的权重矩阵，|V| 表示词表的大小，$|e|$ 表示词向量的维度，$|g|$ 为隐藏

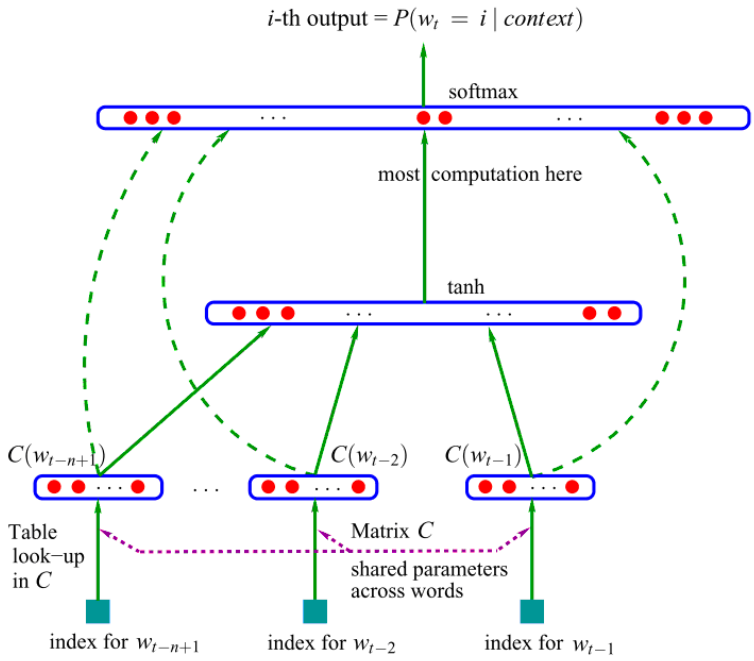

图 1: 前馈神经网络语言模型

层的维度。$b(1), b(2)$ 均为模型中的偏置项。矩阵 $W \in \mathbb{R}^{|\mathcal{V}| \times (n-1)|e|}$ 表示从输入层到输出层的直连边权重矩阵。由于 W 的存在，该模型可能会从非线性的神经网络退化成为线性分类器。Bengio 等人在文中指出，如果使用该直连边，可以减少一半的迭代次数；但如果没有直连边，可以生成性能更好的语言模型。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，下文也直接忽略这一项。如果不考虑 W 矩阵，整个模型计算量最大的操作，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操作的优化。

2.3 log 双线性语言模型

2007 年，Mnih 和 Hinton 在神经网络语言模型（NNLM）的基础上提出了 log 双线性语言模型（Log-Bilinear Language Model，LBL）[79]。LBL 与 NNLM 的区别正如它们的名字所示，LBL 的模型结构是一个 log 双线性结构；而 NNLM 的模型结构为神经网络结构。具体来讲，LBL 模型的能量函数为：

$$E(w_i; w_{i-(n-1):i-1}) = b^{(2)} + e(wi)^\top b^{(1)} + e(w_i)TH[e(w_{i-(n-1)}); \cdots; e(w_{i-1}) \quad (5)$$

LBL 模型的能量函数（公式 2.14）与 NNLM 的能量函数（公式 2.11）主要有两个区别。一、LBL 模型中，没有非线性的激活函数 tanh，而由于 NNLM 是非线

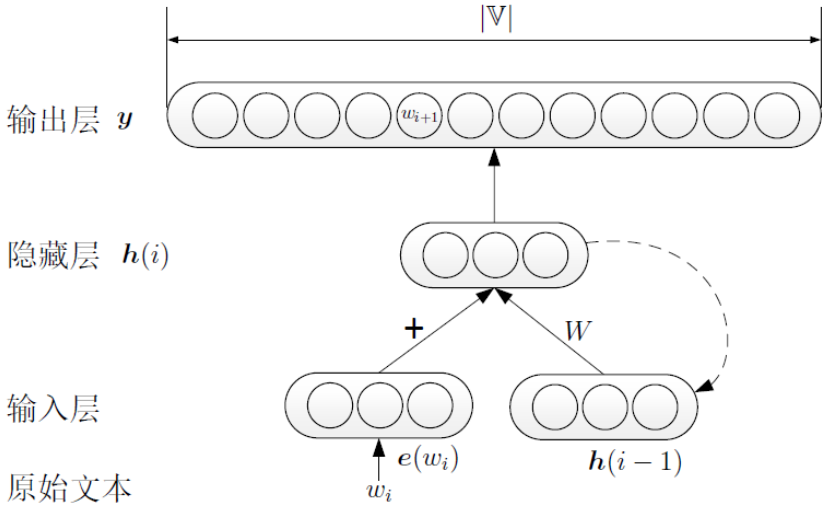

图 2: 循环神经网络语言模型（RNNLM）模型结构图

性的神经网络结构，激活函数必不可少；二、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），只在原版的 LBL 模型中存在，后续的改进工作均不包含这一特点。

后来的几年中，Mnih 等人在 LBL 模型的基础上做了一系列改进工作。其中最重要的模型有两个：层级 log 双线性语言模型（Hierarchical LBL，HLBL）[80] 和基于向量的逆语言模型（inverse vector LBL，ivLBL）[81]。以下分别介绍这两个模型所用的技术。

2.4 循环神经网络语言模型

Mikolov 等人提出的循环神经网络语言模型 (Recurrent Neural Network based Language Model，RNNLM) 则直接对 $P(w_i|w_1; w_2; \cdots; w_{i-1})$ 进行建模，而不使用公式2对其进行简化 [6, 7]。因此，RNNLM 可以利用所有的上文信息，预测下一个词，其模型结构如图2 所示。

RNNLM 的核心在于其隐藏层的算法:

$$h(i) = \phi(e(w_i) + Wh(i-1)) \tag{6}$$

其中，ϕ 非线性激活函数。但与 NNLM 不同，RNNLM 并不采用 n 元近似，而是使用迭代的方式直接对所有上文进行建模。在公式6 中，h(i) 表示文本中第 i 个词 w_i 所对应的隐藏层，该隐藏层由当前词的词向量 $e(w_i)$ 以及上一个词对应的隐藏层 $h(i-1)$ 结合得到。

隐藏层的初始状态为 $h(0)$，随着模型逐个读入语料中的词 $w_1; w_2; \cdots$, 隐藏层不断地更新为 $h(1); h(2); \cdots$ 。根据公式6，每一个隐藏层包含了当前词的信息以及上一个隐藏层的信息。通过这种迭代推进的方式，每个隐藏层实际上包含了此前所有上文的信息，相比 NNLM 只能采用上文 n 元短语作为近似，RNNLM 包含了更丰富的上文信息，也有潜力达到更好的效果。RNNLM 的输出层计算方法与 NNLM 的输出层一致。

3 国内外研究现状及发展动态

3.1 对上下文信息建模策略

依照上章节的分析，本章节主要介绍我们实验中所要涉及的模型，主要分为：普通循环神经网络节点、长短记忆网络 (Long shrot-term memory, LSTM) 和门限记忆节点 (Gated Recurrent Unit,GRU)。

LSTM 的计算公式定于如下：

- 输入门: 输入门：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：

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i)$$

- 忘记门：决定是否清楚或者保持单一部分的状态

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$

- 变换输出和前一状态到最新状态

$$g_t = \phi(W^g x_t + U^g h_{t-1} + b^g)$$

- 输出门: 计算 cell 的输出

$$o_t = \sigma(W^o x_t + U^o h^{t-1} + b^o)$$

- cell 状态更新步骤：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：

$$s_t = g_t \odot i_t + s_{t-1} \odot f_t$$

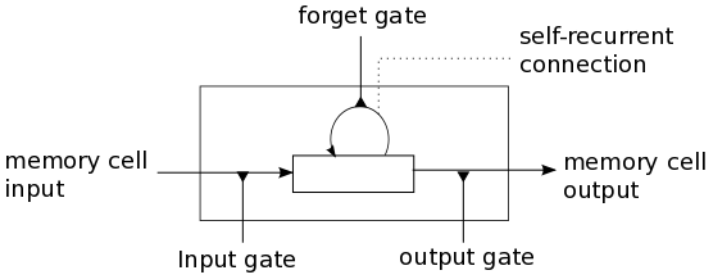

图 3: LSTM 模型

- 最终 LSTM 的输出：使用一个对当前状态的 tanh 变换进行重变换：

$$h_t = s_t \odot \phi(o_t)$$

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

$$\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \sigma(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

GRU 可以看成是 LSTM 的变种，GRU 把 LSTM 中的 forget gate 和 input gate 用 update gate 来替代。把 cell state 和隐状态 h_t 进行合并，在计算当前时刻新信息的方法和 LSTM 有所不同。下图是 GRU 更新 h_t 的过程:，具体定义如下：

- 更新门 z_t: 定义保存多少以前的信息。

$$z_t = \sigma(W^z x_t + U^z h_{t-1})$$

- 重置门 r_t: 决定保留多少输入信息.

$$r_t = \sigma(W^r x_t + U^r h_{t-1})$$

- 节点内部更新值 $\tilde{h}_t$: 其次是计算候选隐藏层（candidate hidden layer）$\tilde{h}_t$，这个候选隐藏层和 LSTM 中的 $\tilde{c}_t$ 是类似，可以看成是当前时刻的新信息，其中 r_t 用来控制需要保留多少之前的记忆，如果 r_t 为 0，那么 $\tilde{h}_t$ 只包含当前词的信息：

$$\tilde{h}_t = \tanh(W^h x_t + U^h(h_{t-1} \odot r_t))$$

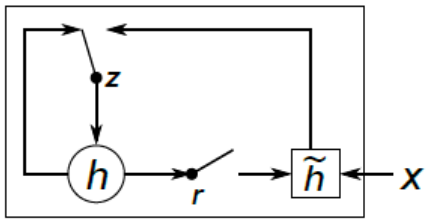

图 4: GRU 模型示意图

- 隐藏层输出值 h_t: 最后 z_t 控制需要从前一时刻的隐藏层 h_{t-1} 中遗忘多少信息，需要加入多少当前时刻的隐藏层信息 $\tilde{h}_t$，最后得到 htht，直接得到最后输出的隐藏层信息，这里与 LSTM 的区别是 GRU 中没有 output gate：

$$h_t = (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1}$$

如果 reset gate 接近 0，那么之前的隐藏层信息就会丢弃，允许模型丢弃一些和未来无关的信息；update gate 控制当前时刻的隐藏层输出 h_t 需要保留多少之前的隐藏层信息，若 z_t 接近 1 相当于我们之前把之前的隐藏层信息拷贝到当前时刻，可以学习长距离依赖。一般来说那些具有短距离依赖的单元 reset gate 比较活跃（如果 r_t 为 1，而 z_t 为 0 那么相当于变成了一个标准的 RNN，能处理短距离依赖），具有长距离依赖的单元 update gate 比较活跃。

3.2 对多元分类模型的建模

大词表问题，主要是对 softmax 如何建模的问题。在本课题中，我们探讨 cHSM 和 tHSM 两种不同的方案所带来的影响和优劣。

3.3 单词聚类的策略

当我们使用多层分类模型的时候，我们就需要将单词按照模型的架构进行划分。其中对于 cHSM 模型，我们有以下策略可以使用：1) 基于词频划分类别 2) 基于 2-gram 的布朗聚类 (brown clustering) 进行划分.3) 按照 word-embedding 的词向量信息进行聚类。另外，我们还需要注意的是，各个类别可以包含不同的数量的单词，也可以包含数量相同的单词。对于后者，我们考虑的划分模型就是基于交换算法 (Exchange Algorithm), 以此来保证获得近似的最优解。文本聚类方法可以分为静态聚类和动态聚类，静态聚类方法包括 Top-down 方法和

Bottom-up 方法，动态聚类（Online clustering）需要判断每个新加入的样本属于已有的类还是一个新类。

K-means 聚类前提：样本之间相似度能够计算

1. 随机在图中取 K 个种子点。
2. 然后对图中的所有点求到这 K 个种子点的距离，假如点 P_i 离种子点 S_i 最近，那么 P_i 属于 S_i 点群。
3. 接下来，我们要移动种子点到属于他的“点群”的中心;
4. 然后重复第 2）和第 3）步，直到种子点没有移动。

K-means 缺点：

1. 需提前确定聚类数目
2. 对初始选取的点很敏感
3. 属于硬聚类（每个样本只能属于一类）

熵：若 X 是一个离散型随机变量，取值空间为 R，其概率分布为 $p(x) = P(X = x), x \in R$。那么，X 的熵 $H(X)$ 定义为

$$H(X) \ -\sum_{x\in R} p(x)log_2p(x) \tag{8}$$

其中约定 $0log_20\ 0$，通常将 $log_2p(x)$ 写作 $logp(x)$。熵又称为自信息，可以视为描述一个随机变量的不确定性的数量。

联合熵和条件熵定义: 若 X, Y 是一对离散型随机变量 $X, Y\ p(x, y)$，则 X, Y 的联合熵 $H(X, Y)$

$$H(X,Y) = -\sum_{x\in X}\sum_{y\in Y} p(x,y)logp(x,y) \tag{9}$$

联合熵实际就是描述一对随机变量平均所需的信息量

给定随机变量 X 的情况下，随机变量 Y 的条件熵如下：

$$H(Y|X) = -\sum_{x\in X}\sum_{y\in Y} p(x,y)logp(y|x) \tag{10}$$

条件熵和联合熵的关系：

$$H(X,Y)\ H(Y|X) + H(X) \tag{11}$$

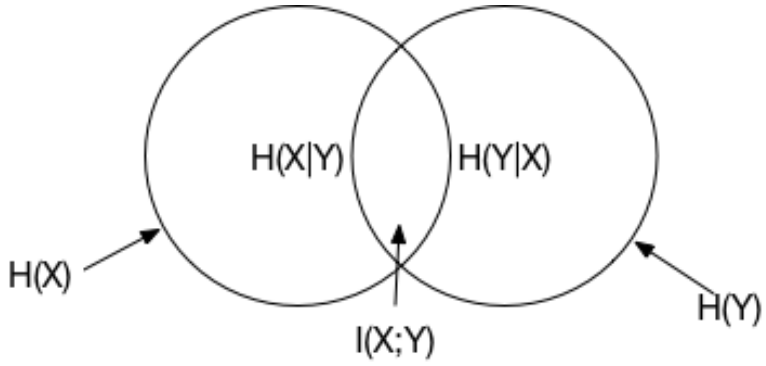

图 5: 互信息示意图

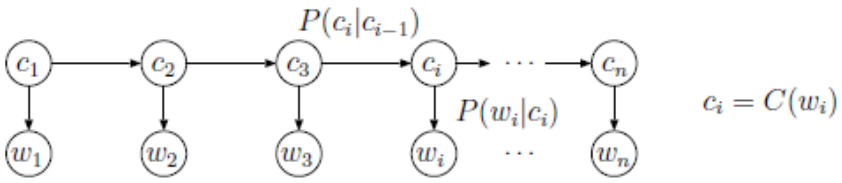

图 6: 布朗聚类示意图

推广到一般情况：

$$H(X_1, X_2, ..., X_n) = H(X_1) + H(X_2|X_1) + ... + H(X_n|X_1, ..., X_{n-1}) \tag{12}$$

互信息

$$H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y) \tag{13}$$

可得

$$H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{14}$$

这个差称为 X 和 Y 的**互信息**，记作 $I(X;Y)$，可得 $I(X;Y) = H(X) - H(X|Y)$

$I(X;Y)$ 反映的是在知道了 Y 的值以后 X 的不确定性的减少量。

$$I(X;Y) = \sum_{x,y} p(x,y) log \frac{p(x,y)}{p(x)p(y)} \tag{15}$$

布朗聚类：布朗聚类是一种自底而上的层次聚类算法，基于 n-gram 模型和马尔科夫链模型，是一种硬聚类，每个词都在且只在唯一的一个类中 [8]。

w 是词，c 是类，不同于词性标注，此处 c 是未知的。布朗聚类的输入是一个语料库，这个语料库是一个词序列，输出是一个二叉树，树的叶子节点是一个个词，树的中间节点是我们想要的类（中间结点作为根节点的子树上的所有叶

子为类中的词）。

$$\begin{aligned} Quality(C) =& \frac{1}{n} logP(w_1, \cdots, w_n) = \frac{1}{n} logP(w_1, \cdots, w_n, C(w_1), \cdots, C(w_n)) \\ =& \frac{1}{n} log \prod_{i=1}^{n} P(C(w_i)|C(w_{i-1}))P(w_i|C(w_i)) \end{aligned} \tag{16}$$

$C(w_0)$ 是种特殊的 **Start** 类，我们要找到这样的分类方式 C，使得 $Quality(C)$ 尽可能大

$$\begin{aligned} Quality(C) =& \frac{1}{n} \sum_{i=1}^{n} P(C(w_i)|C(w_{i-1}))P(w_i|C(w_i)) \\ =& \sum_{w',w} \frac{n(w,w')}{n} logP(C(w')|C(w)))P(w'|C(w')) \\ =& \sum_{c,c'} \frac{n(c,c')}{n} log \frac{n(c,c')}{n(c)n(c')} + \sum_{w'} \frac{n(w')}{n} log \frac{n(w')}{n} \end{aligned} \tag{17}$$

$n(w)$ 是 w 在文本中出现次数，$n(w,w')$ 是二元对 w,w' 在文本中的出现次数，$n(c) = \sum_{w \in c} n(w)$，$n(c,c') = \sum_{w \in c} n(w,w')$

$$Quality(C) = I(C) - H \tag{18}$$

简单算法：以 k 长度文本为例：每个字均为一类，词典大小为 $|V|$，找到俩类，合并后使得互信息变得最大，重复合并步骤使得最后都归为一类，整个算法的时间复杂度为 $O(|V|^5)$

优化算法：开始设置一个参数 m，比如 $m = 1000$，我们按照词汇出现的频率对其进行排序然后把频率最高的 m 个词各自分到一个类中，对于第 $m+1$ 到 $|V|$ 个词进行循环：

1. 对当前词新建一个类，我们现在又 $m+1$ 个类了
2. 从这 $m+1$ 个类中贪心选择最好的两个类合并，现在我们剩下 m 个类
3. 最后我们再做 $m-1$ 词合并。算法时间复杂度 $O(|V| * m^2)$

哈夫曼树：在一般的数据结构的书中，树的那章后面，著者一般都会介绍一下哈夫曼 (HUFFMAN) 树和哈夫曼编码。哈夫曼编码是哈夫曼树的一个应用。哈夫曼编码应用广泛，如 JPEG 中就应用了哈夫曼编码。

首先介绍什么是哈夫曼树：哈夫曼树又称最优二叉树，是一种带权路径长度最短的二叉树。所谓树的带权路径长度，就是树中所有的叶结点的权值乘上其

到根结点的路径长度（若根结点为 0 层，叶结点到根结点的路径长度为叶结点的层数）。树的带权路径长度记为

$$WPL = (W1 * L1 + W2 * L2 + W3 * L3 + ... + Wn * Ln) \tag{19}$$

N 个权值 Wi(i=1,2,...n) 构成一棵有 N 个叶结点的二叉树，相应的叶结点的路径长度为 Li(i=1,2,...n)。可以证明哈夫曼树的 WPL 是最小的。

哈夫曼在上世纪五十年代初就提出这种编码时，根据字符出现的概率来构造平均长度最短的编码。它是一种变长的编码。在编码中，若各码字长度严格按照码字所对应符号出现概率的大小的逆序排列，则编码的平均长度是最小的。（注：码字即为符号经哈夫曼编码后得到的编码，其长度是因符号出现的概率而不同，所以说哈夫曼编码是变长的编码。）

然而怎样构造一棵哈夫曼树呢？最具有一般规律的构造方法就是哈夫曼算法。一般的数据结构的书中都可以找到其描述：

1. 对给定的 n 个权值 W1,W2,W3,...,Wi,...,Wn 构成 n 棵二叉树的初始集合 F=T1,T2,T3,...,Ti,...,Tn，其中每棵二叉树 Ti 中只有一个权值为 Wi 的根结点，它的左右子树均为空。（为方便在计算机上实现算法，一般还要求以 Ti 的权值 Wi 的升序排列。）

2. 在 F 中选取两棵根结点权值最小的树作为新构造的二叉树的左右子树，新二叉树的根结点的权值为其左右子树的根结点的权值之和。

3. 从 F 中删除这两棵树，并把这棵新的二叉树同样以升序排列加入到集合 F 中。

4. 重复二和三两步，直到集合 F 中只有一棵二叉树为止。

哈夫曼编码 (Huffman Coding) 是一种编码方式，以哈夫曼树　即最优二叉树，带权路径长度最小的二叉树，经常应用于数据压缩。在计算机信息处理中，“哈夫曼编码”是一种一致性编码法（又称” 熵编码法”），用于数据的无损耗压缩。这一术语是指使用一张特殊的编码表将源字符（例如某文件中的一个符号）进行编码。这张编码表的特殊之处在于，它是根据每一个源字符出现的估算概率而建立起来的（出现概率高的字符使用较短的编码，反之出现概率低的则使用较长的编码，这便使编码之后的字符串的平均期望长度降低，从而达到无损压缩数据的目的）。这种方法是由 David.A.Huffman 发展起来的。

例如，在英文中，e 的出现概率很高，而 z 的出现概率则最低。当利用哈夫曼编码对一篇英文进行压缩时，e 极有可能用一个位 (bit) 来表示，而 z 则可能花去 25 个位（不是 26）。用普通的表示方法时，每个英文字母均占用一个字节（byte），即 8 个位。二者相比，e 使用了一般编码的 1/8 的长度，z 则使用了 3 倍多。倘若我们能实现对于英文中各个字母出现概率的较准确的估算，就可以大幅度提高无损压缩的比例。

霍夫曼算法证明：构建一棵二叉树，使得最小。

1. 首先分析问题的所有解：在叶子数目一定的情况下，二叉树的所有可能是有限的。

2. 假设霍夫曼树是其中最优的，那么它一定比其他树好。我们来进行一次比较。把叶子节点互换，假设这两个叶子节点的频率为 $f1$ 和 $f2$，深度为 $d1$ 和 $d2$，互换时，其他叶子节点的 cost 不变，另为 C，那么互换前总 cost 为 C+f1d1+f2d2，互换后为 C+f1d2+f2d1，如果互换后的树变“差”了，那么 $f_1d_1 + f_2d_2 \leq f_1d_2 + f_2d_1$，变换一下即为 $f_1(d_1 - d_2) \leq f_2(d_1 - d_2)$ 也就是如果 di<d2，那么 f1 必然 >f2，即叶子节点的深度越高，频率必然越低。那么我们就可以确定最小的俩个叶子节点在最底层。

3. 已经找到了最小的俩个叶子应该怎么放，接下来考虑第三小的叶子。可以继续在底层，也可以放往上一层走。为了解决这个问题，我们进行另一种比较。交换子树，跟（2）中同样的思路，$C + (F_1 + F_2) * d_1 + F_3 * d_2 <= C + F_3 * d_1 + (F_1 + F_2) * d_2$. 可以得到，在最优霍夫曼树当中，两个内部节点 n1 和 n2，如果 n1 比 n2 更深，那么 n1 下面的所有叶子的频率之和必然要小于 n2 下面所有叶子的频率之和。这样的话，我们可以把内部节点的所有叶子的频率之和标在它旁边，那么整棵树的每一个节点就有了一个数值。那么第三小的叶子的情况便可以确定。我们考虑从（f1+f2），f3，f4 之选最小的俩个结合为兄弟。

4. 这样就证明霍夫曼算法的递归过程。

4 总结与展望

1) 数学背景和理论背景。尽管本实验题目定义范围比较小，但是我们也需要很好的数学理论知识，包括：矩阵论, 概率论。还有，我们还需要极强的阅读

外文文献知识和编码实现能力，都是不可或缺的基本要求。矩阵在许多学科领域中都有应用，在很多时候，除了需要知道矩阵的理论性质以外，还需要计算矩阵的数值。为了矩阵的计算能够足够精确与快捷，数值线性代数中专门有研究矩阵的数值计算方法 [40]。与其它的数值计算一样，矩阵的数值计算注重的主要也是算法的复杂度和数值稳定性。矩阵的数值计算可以使用直接计算，也可以用迭代算法，例如在计算方块矩阵的特征值时，可以从一个非零向量 x_0 开始，通过特定迭代方法得到一个逼近某个特征向量的向量序列 [41]。而概率论，作为统计学的数学基础，概率论对诸多涉及大量数据定量分析的人类活动极为重要 [3]，概率论的方法同样适用于其他方面，例如是对只知道系统部分状态的复杂系统的描述——统计力学，而二十世纪物理学的重大发现是以量子力学所描述的原子尺度上物理现象的概率本质 [4]。

2) 基于 theano 框架的建模方案。因为基于 python 的深度学习库比较完善，适合建模。本实验拟采用 theano 的建模语言，来帮助我们快速建模和调参。Theano 是在 BSD 许可证下发布的一个开源项目，是由 LISA 集团（现 MILA）在加拿大魁北克的蒙特利尔大学（Yoshua Bengio 主场）开发。它是用一个希腊数学家的名字命名的。Python 的核心 Theano 是一个数学表达式的编译器。它知道如何获取你的结构，并使之成为一个使用 numpy、高效本地库的非常高效的代码，如 BLAS 和本地代码（C++），在 CPU 或 GPU 上尽可能快地运行。它巧妙的采用一系列代码优化从硬件中攫取尽可能多的性能。如果你对代码中的数学优化的基本事实感兴趣，看看这个有趣的名单。Theano 表达式的实际语法是象征性的，可以推送给初学者用于一般软件开发。具体来说，表达式是在抽象的意义上定义，编译和后期是用来进行计算。它是为深度学习中处理大型神经网络算法所需的计算而专门设计的。它是这类库的首创之一（发展始于 2007 年），被认为是深度学习研究和开发的行业标准。

3) 同时本实验也需要对 linux 的 bash 脚本有一定的熟悉，以方便将模型的数据结果正确的统计和运行模型的开发环境配置。

4) 试验结果图表统计和绘制. 本实验的结果需要精良的语言来控制，而 R 语言的 ggplot2 框架就很适合我们的试验结果图表的绘制工作。

5) 基于 GPU 的 cuda 的模型优化也是我们需要考虑的问题之一。CUDA（Compute Unified Device Architecture，统一计算架构 [1]）是由 NVIDIA 所推出的一種整合技術，是該公司对于 GPGPU 的正式名稱。透过这个技术，使用者可利用 NVIDIA 的 GeForce 8 以後的 GPU 和較新的 Quadro GPU 进行计算。

亦是首次可以利用 GPU 作為 C-编译器的开发环境。NVIDIA 行銷的時候 [2]，往往將编译器与架构混合推廣，造成混乱。实际上，CUDA 可以相容 OpenCL 或者自家的 C-编译器。无论是 CUDA C-語言或是 OpenCL，指令最終都會被驱动程序轉換成 PTX 代码，交由显示核心計算。

在 GPUs（GPGPU）上使用圖形 APIs 进行传统通用計算，CUDA 技术有下列几個优点：

- 分散读取——代码可以从内存的任意位址读取
- 统一虛拟内存（CUDA 6）
- 共用内存——CUDA 公开一個快速的共用存储区域（每个处理器 48K），使之在多个进程之間共用。其作为一個用戶管理的快取内存，比使用纹理查找可以得到更大的有效频宽。
- 与 GPU 之間更快的下载与回读
- 全面支持整型位与操作，包括整型纹理查找

限制

- CUDA 不支持完整的 C 语言标准。它在 C++ 编译器上運行主機代码時，會使一些在 C 中合法（但在 C++ 中不合法）的代碼無法編譯。
- 不支持紋理渲染（CUDA 3.2 及以後版本通過在 CUDA 陣列中引入“表面寫操作”——底層的不透明数据结构——來进行处理）
- 受系统主线的频宽和延遲的影響，主機與設備記憶體之間資料複製可能會導致性能下降（通過過 GPU 的 DMA 引擎處理，非同步記憶體傳輸可在一定範圍内緩解此現象）
- 当執行緒總數為數千時，执行程序按至少 32 個一組來運行才能獲得最佳效果。如果每組中的 32 個進程使用相同的執行路徑，則程式分支不會顯著影響效果；在處理本質上不同的任務時，SIMD 執行模型將成為一個瓶頸（如在光線追蹤演算法中遍歷一個空間分割的資料結構）
- 与 OpenCL 不同，只有 NVIDIA 的 GPUs 支援 CUDA 技術
- 由于编译器需要使用优化技术來利用有限的资源，即使合法的 C/C++ 有時候也會被標記並中止编译

- CUDA（計算能力 1.x）使用一個不包含回槊、函数指標的 C 語言子集，外加一些簡單的扩展。而單個進程必須運行在多個不相交的記憶體空間上，這與其它 C 語言運行環境不同。

- CUDA（計算能力 2.x）允许 C++ 类功能的子集，如成員函數可以不是虛擬的（這個限制將在以後的某個版本中移除）[参见《CUDA C 程式設計指南 3.1》－附錄 D.6]

- 双精度浮点（CUDA 计算能力 1.3 及以上）与 IEEE754 標準有所差異：倒数、除法、平方根僅支持舍入到最近的偶數。单精度中不支持反常值（denormal）及 sNaN（signaling NaN）；只支持两种 IEEE 舍入模式（舍位与舍入到最近的偶数），這些在每条指令的基楚上指定，而非控制字码；除法/平方根的精度比单精度略低。

参考文献

[1] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[2] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 932–938, 2000.

[3] Peter F. Brown, Vincent J. Della Pietra, Peter V. de Souza, Jennifer C. Lai, and Robert L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[4] Rong Jin, Alex G. Hauptmann, and Cheng Xiang Zhai. Title language model for information retrieval. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '02, pages 42–48, New York, NY, USA, 2002. ACM.

[5] Paul Baltescu and Phil Blunsom. Pragmatic neural language modelling in machine translation. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 820–829, 2015.

[6] Tomáš Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

[7] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048, 2010.

[8] Leon Derczynski, Sean Chester, and Kenneth S Bøgh. Tune your brown clustering, please. In *International Conference Recent Advances in Natural Language*

Processing, RANLP, volume 2015, pages 110–117. Association for Computational Linguistics, 2015.