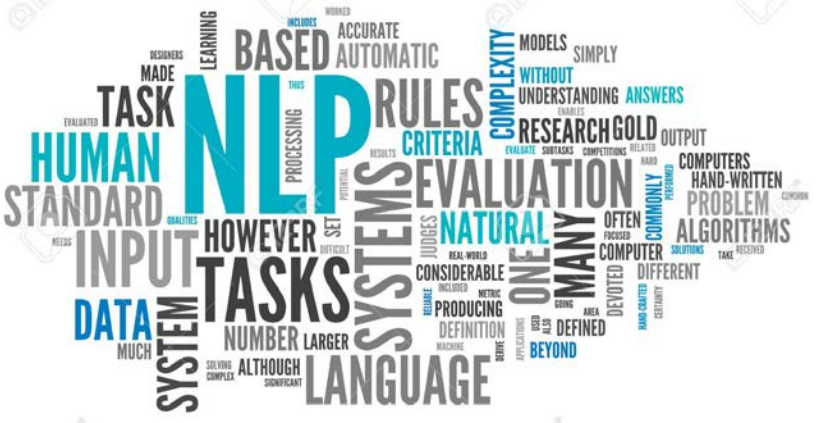
硕士毕业论文答辩

基于神经网络的高性能语言建模技术研究

答辩人：姜楠
(nanjiang@buaa.edu.cn)
指导老师：荣文戈 副教授
(w.rong@buaa.edu.cn)

北京航空航天大学计算机学院
School of Computer Science & Engineering, Beihang University



1

课题背景

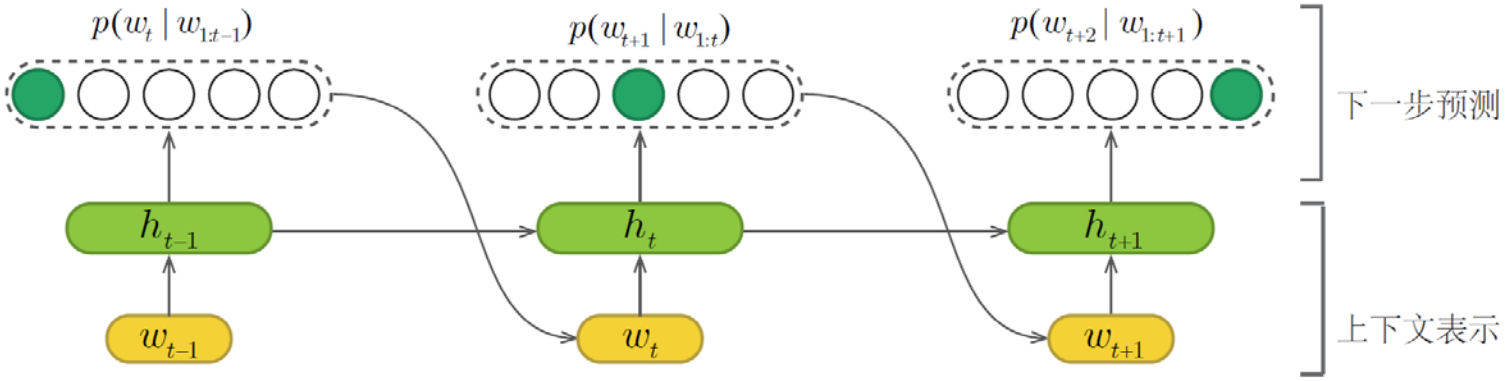
语言模型历史

大词表问题

1.1 语言建模

$$w_1, \cdots, w_{t-1} --> w_t$$

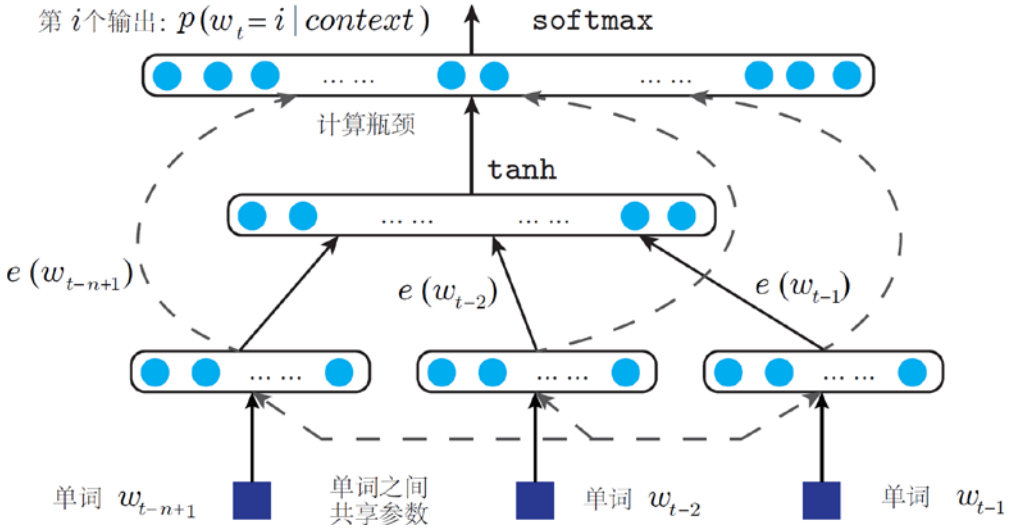
组织结构：1）上下文表示；2）预测层表示



信息抽取（Information Extraction，IE）、关系抽取（Relation Extraction，RE）、命名实体识别（Named Entity Recognition，NER）、词性标注（Part Of Speech Tagging，POS）、指代消解（Coreference Resolution）、句法分析（Parsing）、词义消歧（Word Sense Disambiguation，WSD）、语音识别（Speech Recognition）、语音合成（Text To Speech，TTS）、机器翻译（Machine Translation，MT）、自动文摘（Automatic Summarization）、问答系统（Question Answering）、自然语言理解（Natural Language Understanding）、光学字符识别（Optical Character Recognition，OCR）、信息检索（Information Retrieval，IR）

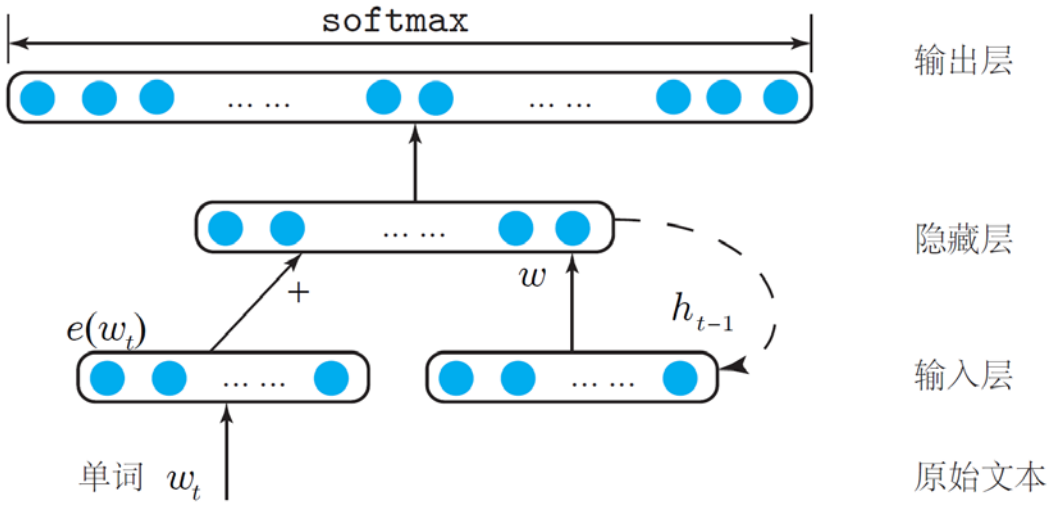
1.2 上下文表示

- 前馈神经网络
- 双线性神经网络



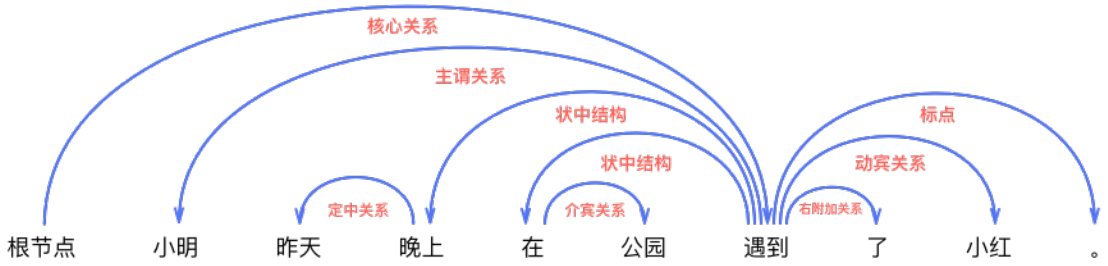
1.2 上下文表示

- 前馈神经网络
- 双线性神经网络
- 循环神经神经网络

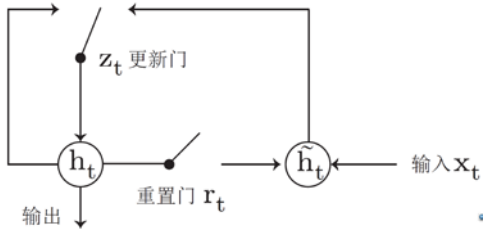


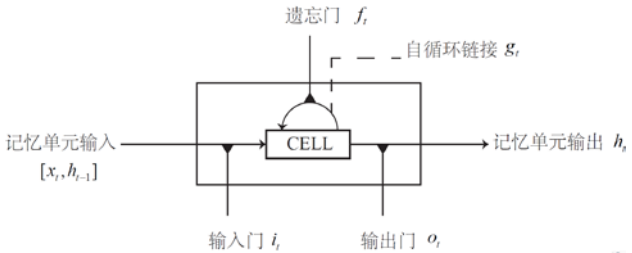
1.2 上下文表示 循环神经网络

- 文本中存在单词依赖关系 → 需要用RNN建模



- 历史上提出的RNN模型可以分为:
  - RNN ReLU模型
  - RNN tanh模型
  - Quasi RNN模型
  - LSTM模型
  - GRU模型





1.3 预测层表示

- Softmax 函数 $$p(w|h) = \frac{\exp(h^\top v_w)}{\sum_{w_j \in \mathcal{V}} \exp(h^\top v_{w_j})}$$
- Log-softmax 函数 $$\log p(w|h) = \theta^w h - \log \sum_{u \in \mathcal{V}} \exp(\theta^u h)$$

- 若维度设置为512，词表大小为267735，Float32占用4字节softmax矩阵占用522MB，softmax导数矩阵也占用522MB，一共占用1.02GB
- 然而LSTM占用内存却是4MB

1.3 预测层表示 计算瓶颈

$$\frac{\exp(h^{\top} v_w)}{\sum_{w_j \in \mathcal{V}} \exp(h^{\top} v_{w_j})}$$

操作步骤	符号表示	计算时间（%）
计算矩阵乘法（Multiply）	$h^{\top} v_w$	~80%
计算概率归一化（Sum） 除法函数（Divide）	$\sum_{w_j \in \mathcal{V}}$ $\frac{A}{B}$	~20%

1.3 预测层表示 计算瓶颈

$$\frac{\exp(h^\top v_w)}{\sum_{w_j \in \mathcal{V}} \exp(h^\top v_{w_j})}$$

操作步骤	符号表示	计算时间（%）
计算矩阵乘法（Multiply）	$h^\top v_w$	~80%
计算概率归一化（Sum） 除法函数（Divide）	$\sum_{w_j \in \mathcal{V}}$ $\frac{A}{B}$	~20%

大词表问题！

1. **矩阵乘法**需要的时间 > 线性求和计算的时间
2. 需要优化算法减少矩阵乘法操作。

1.4 大词表问题 单词拆分算法

- 单词级别：只保留较少的单词用来训练神经网络语言模型
- 子词级别：将单词划分成更小的单元子词.
- 字符级别：“*a-zA-Z@ !#*”

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

	优点	缺点
单词级别	简单，易行	速度提升有限
子词级别	效果提升明显	破坏了句子的结构，句子长度加倍
字符级别	运算速度最快（94个预测字符）	字串结构被打破，句子长度增长十几倍

1.4 大词表问题 采样近似算法

$$\log p(w|h) = \theta^w h - \log \sum_{u \in V} \exp(\theta^u h)$$

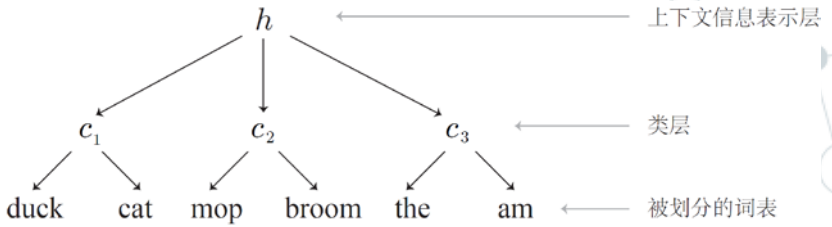
- 主要算法分类:
  - 重要性采样[2008]：用N-gram 概率估计。
  - 噪声误差估计算法[2010-2012]：用 Uni-gram采样，同时将多元酚类转化成二元分类
  - Blackout 采样[2016]：在NCE基础上改进，对正例和负例作加权平均。

- Questions:
  - 如何更快的采样？ Alias method

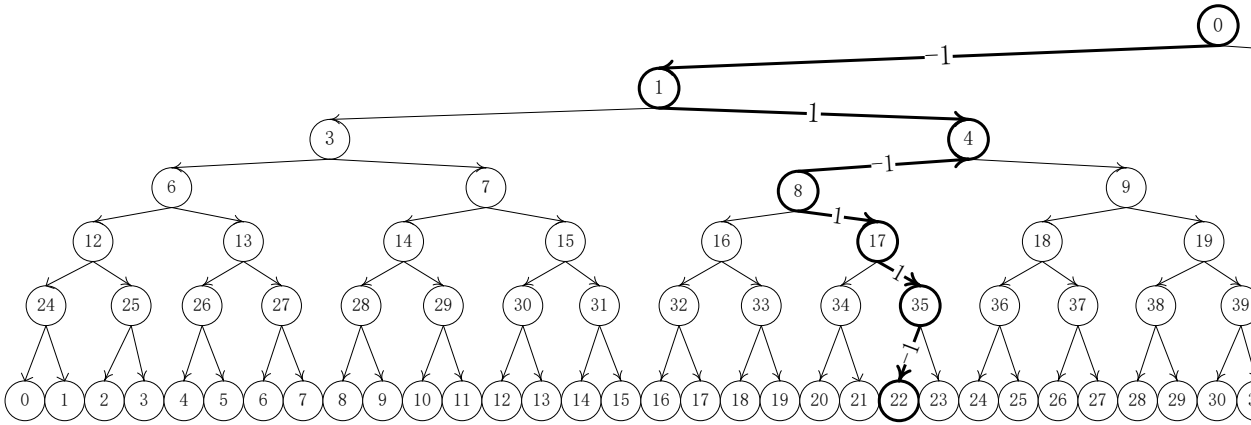
	优点	缺点
重要性采样	----	模型不收敛
NCE	算法计算快，收敛快	带有采样函数，GPU并行困难
Blackout	估计更精确	GPU并行困难，计算复杂度比NCE高

1.4 大词表问题 词表层次分解

- 单词类别分解.

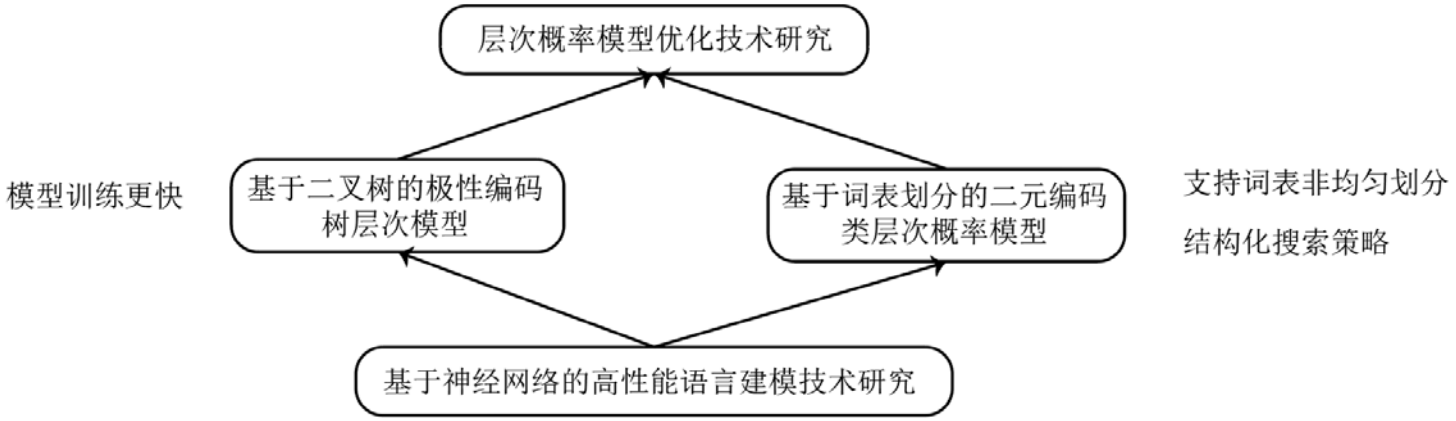


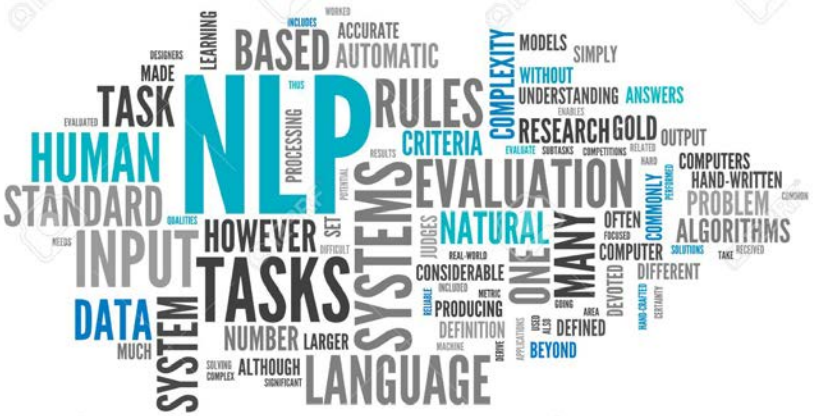
- 树状分解



1.5 本文研究内容

	优点	缺点
二叉树结构	算法计算最快	模型并行度不高（Indexing操作过多），模型精度不高
类别层次	计算速度比采样算法快 稳定性比二叉树结构高	不支持非均匀划分，测试时的策略尚未讨论





2.

层次概率模型

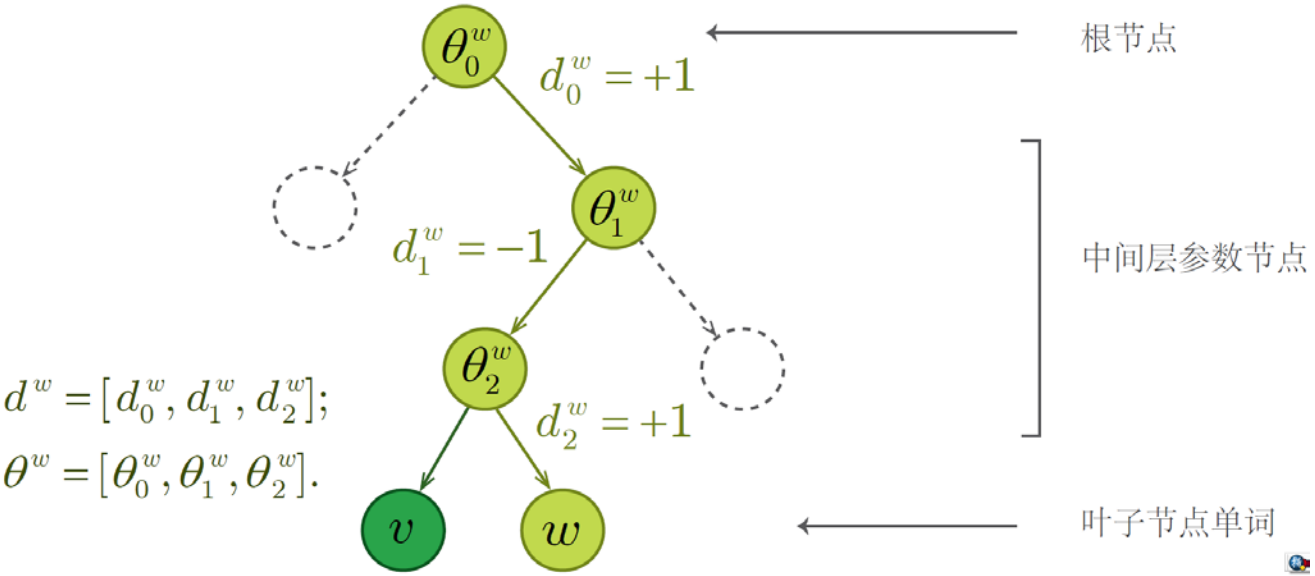
基于二叉树解构

基于类别划分

2.1 基于二叉树的层次概率模型

原单节点计算函数：

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



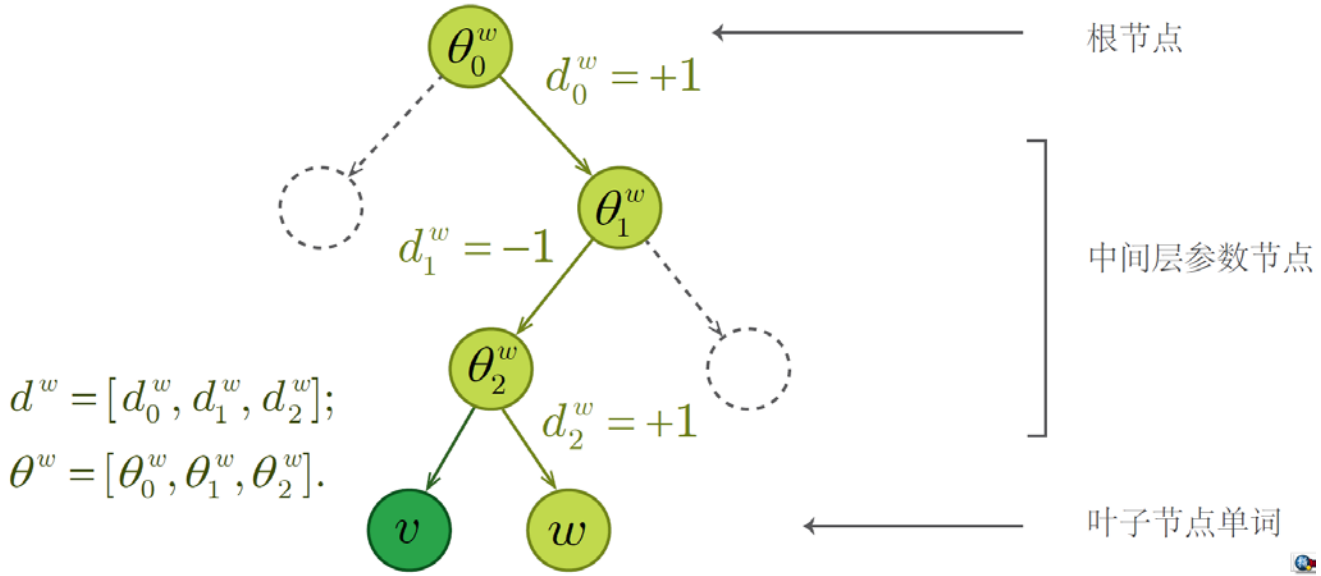
2.1 基于二叉树的层次概率模型

原单节点计算函数：

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

转化后单节点函数：

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



2.1 基于二叉树的层次概率模型

原单节点计算函数：

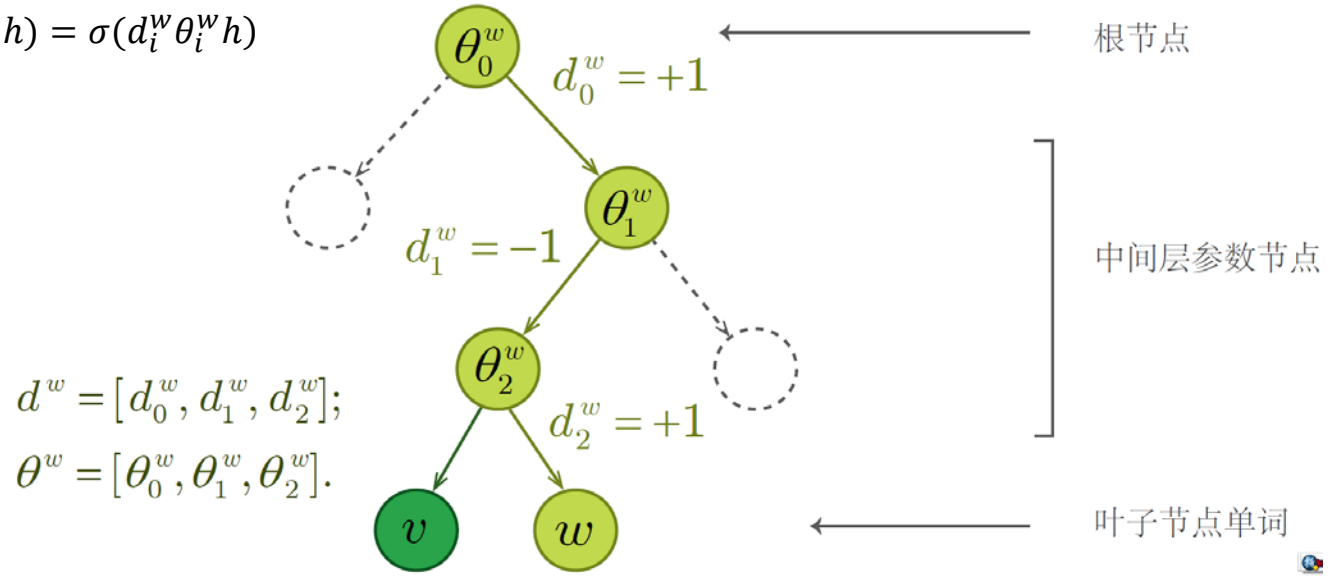
$$p(d_i^w|\theta_i^w,h)=\sigma(\theta_i^w h)^{d_i^w}\times\left(1-\sigma(\theta_i^w h)\right)^{1-d_i^w},d_i^w\in[0,1]$$

转化后单节点函数：

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

单节点概率公式：

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



2.1 基于二叉树的层次概率模型

- 推导后模型的代价函数

$$\ell(\theta|h,w) = -\log \prod_{i=0}^{l^w-1} \sigma(d_i^w \theta_i^w h) = -\log\sigma(d_i^w \theta_i^w h)$$
$$= \log\left(1+\exp(-d^{w\top}\theta^w h)\right) = \zeta(-d^{w\top}\theta^w h)$$

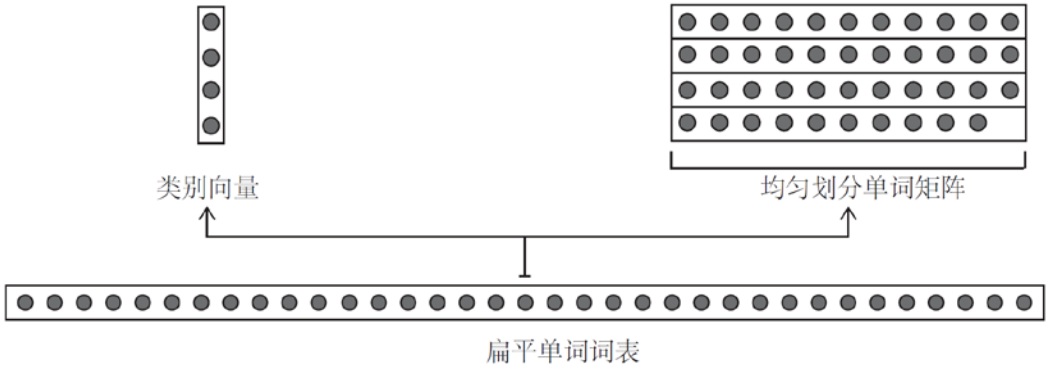
- 原代价函数

$$\ell(\theta|h,w) = \sum_{i=0}^{l^w-1} \left\{\left(1-d'^w_i\right)\log\left(\sigma(\theta_i^w h)\right) + d'^w_i \log\left(1-\sigma(\theta_i^w h)\right)\right\}$$

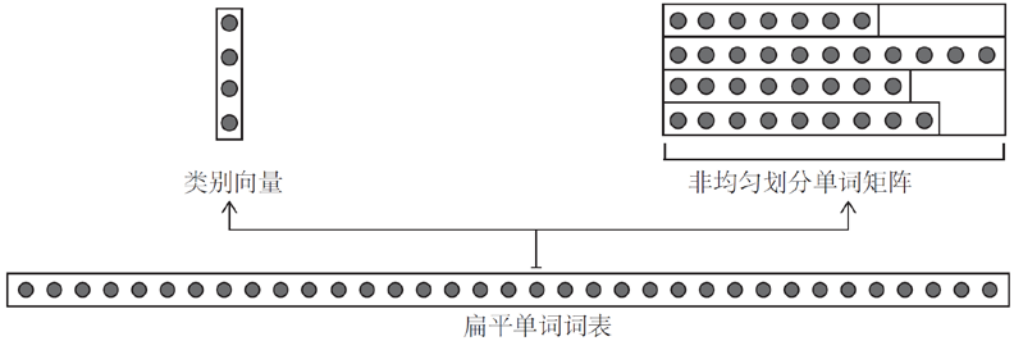
- 1) 原算法涉及许多微小的矩阵乘法操作，而在新算法中我们直接将所有参数以矩阵形式加载进内存，以内存消耗为代价来提高模型单位时间的计算量，我们进而这个参数矩阵的乘法；
- 2) 除了我们的新模型的损失函数更加紧凑外，这条路径上所有节点的概率不是逐层计算而是同时计算的，这为我们的新算法提供更好的时间效率。

2.2 基于类别的层次概率模型

- 均匀划分



- 非均匀划分：
  - 类别偏差问题



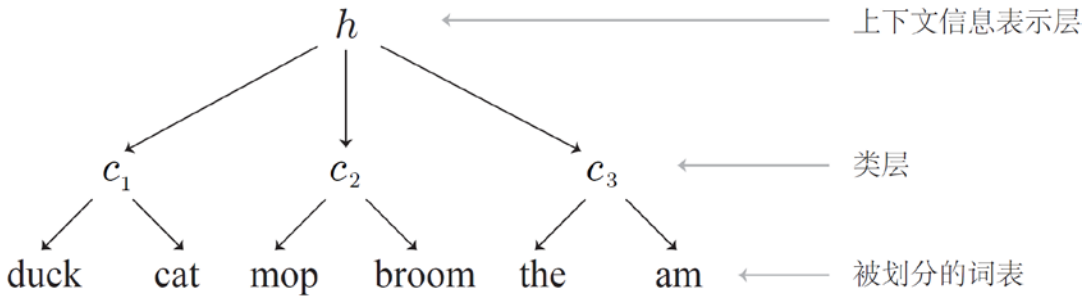
2.2 类别层次概率模型 代价函数

类层的损失 $\log p^c(c|h) = \theta^c h - \log \sum \exp(\theta^c h)$

被划分的词损失 $\log p^o(w|w^c, h) = \theta^o h - \log \sum \exp(\theta^o h)$

被划分填补的词损失 $\log p^o(w|w^c, h) = \theta^o h - \log \sum \theta^m \exp(\theta^o h)$

总代价函数 $\ell(\theta|h) = \log p^c(w^c|h) + \log p^o(w^o|w^c, h)$

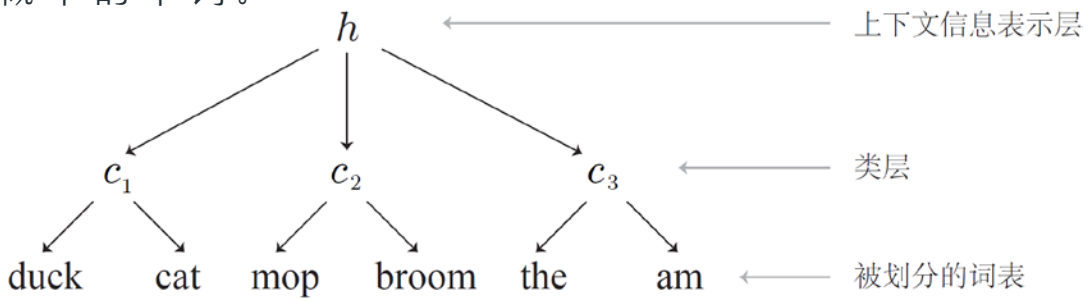


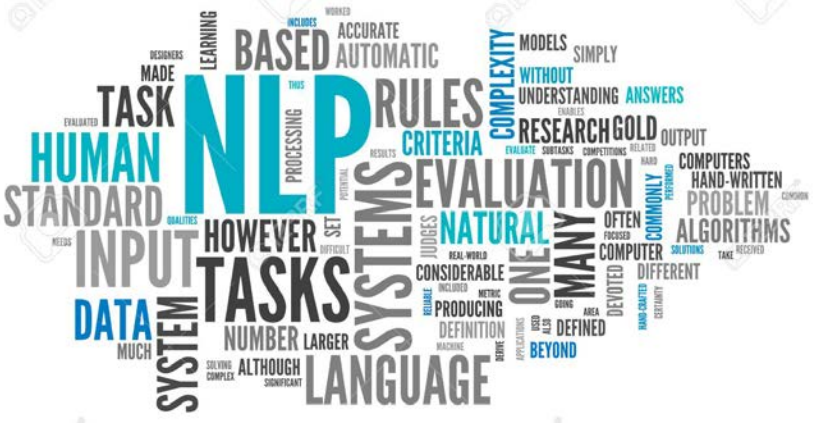
2.2 类别层次概率模型 结构化预测argmax

算法3. 计算全部单词的概率，然后排序；

算法4. 挑选每个类中概率最大的单词，
在这些已经被栅选单词中挑选最佳的单词；

算法5. 计算最大概率的类别；
在该类别取局部最大概率的单词。





3. 实验与分析

层次模型比较

采样模型比较

3.1 实验配置 数据集

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		

3.1 实验配置 评价指标

- 定量度量指标

  单词困惑度：

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

  单词误差率：

$$\mathrm{WER}=\frac{\text{插入的单词数}+\text{插入的单词数}+\text{插入的单词数}}{\text{全部单词数量}}$$

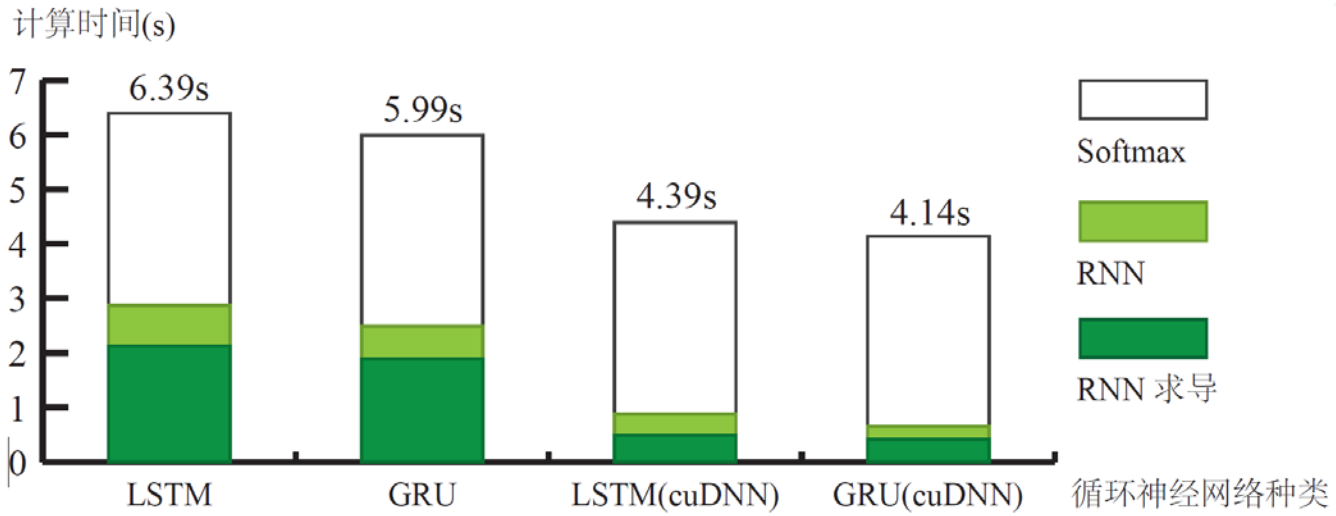
- 定性度量指标

  训练时间效率；

  算法对词表可伸缩性；

  运行时内存消耗。

3.2 运算效率分析 大词表问题



wikitext-103 数据集上测量语言模型三个部分的计算时间比较

- Softmax模块占据时间在3,4组中已经超过了80%的计算时间时间。

3.2 运算效率分析 设备影响

	Nvidia Titan X		Nvidia K40M	
	Train (μs)	Forward only (μs)	Train (μs)	Forward only (μs)
Theano LSTM	289.6	99.1	275.2	63.4
cuDNN Theano LSTM	118.8	59.5	86	32.1

- 不同GPU设备型号之间计算时间有差异，为了统一起见，我们均采用K40M的配置。

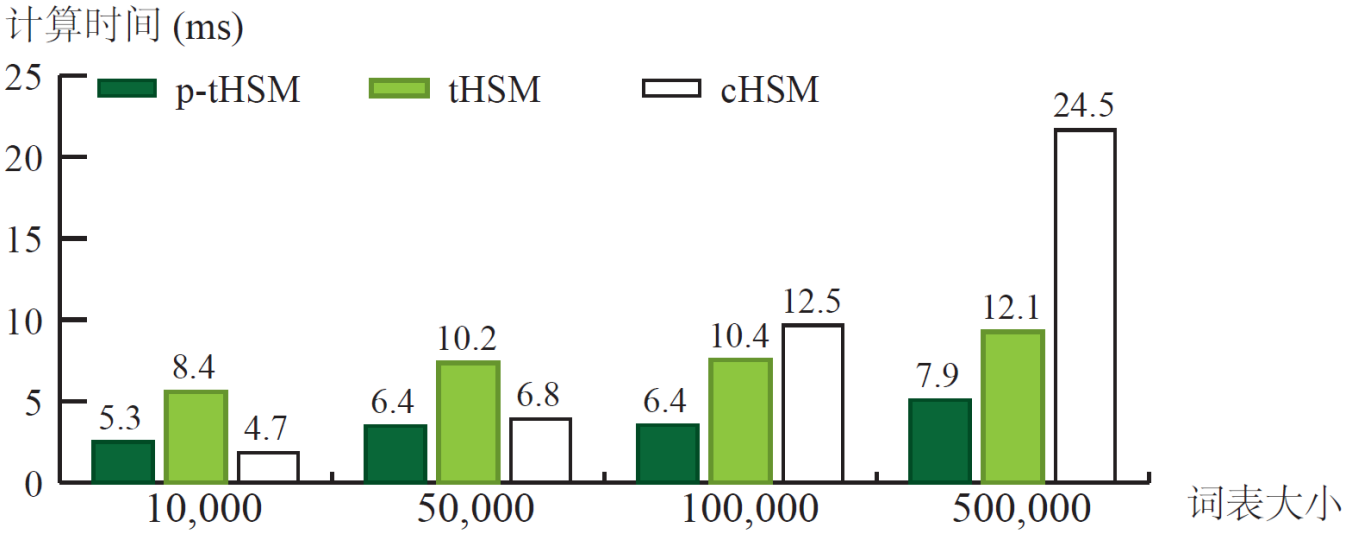
3.2 运算效率分析 层次概率算法计算效率分析

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	$\|\mathcal{H}\mathcal{V}\|$	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**

- 我们提出的p-tHSM算法的运算速度在三项评测中速度最快
- tHSM因为过多的indexing操作，导致实际运算速度不高
- cHSM算法运算效率较高

3.2 运算效率分析 层次概率算法对词表伸缩性试验



cHSM, tHSM 和 p-tHSM 算法随着词表大小的计算时间影响

- 我们提出的p-tHSM算法对词表的伸缩性更好
- 这里没有列举softmax，因为其数值过大

3.3 精确度分析 不同RNN模型的影响分析

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 [54]	176.4	260.52 / 80.00%	238.75 / 80.02%
1×RNN Tanh [37]	176.2	250.57 / 79.61%	230.98 / 79.32%
1×LSTM [22]	**189.5**	180.98 / 77.16%	165.60 / 76.67%
1×GRU [55]	191.3	**179.59 / 77.09%**	**165.32 / 77.07%**
2×RNN Relu [54]	266.3	190.52 / 73.01%	198.75 / 73.02%
2×RNN Tanh [37]	266.3	189.57 / 72.62%	260.98 / 72.32%
2×LSTM [22]	**279.4**	164.98 / 71.17%	165.60 / 71.67%
2×GRU [55]	281.2	**158.59 / 70.08%**	**155.32 / 70.07%**

- LSTM计算效率比GRU高，GRU精度更好；
- RNN ReLU和 RNN Tanh计算相对较快，但是精度低很多。

3.3 精确度分析 不同层次概率模型比较

数据集	算法	验证集(PPL/WER)	测试集(PPL/WER)
WikiText-2	**cHSM + Uni-gram**	**203.18 / 78.25%**	**206.61 / 78.02%**
	p-tHSM + Uni-gram	218.42 / 78.15%	216.05 / 78.15%
WikiText-103	**cHSM + Uni-gram**	**161.81 / 73.42%**	**156.74 / 73.18%**
	p-tHSM + Uni-gram	165.70 / 73.53%	166.11 / 72.44%
One Billion Words	**cHSM + Uni-gram**	**225.36 / 80.32%**	**224.11 / 79.42%**
	p-tHSM + Uni-gram	231.44 / 87.53%	236.11 / 82.53%

- cHSM算法在PPL和WER上面比p-tHSM更好。
- 树状模型结构性过强，很难将特征空间做迭代的二元划分。

3.3 精确度分析 不同结构化预测算法比较

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	全局 argmax（算法 3）	102	80.00%	80.02%
	贪心 argmax（算法 4）	87	80.00%	80.02%
	伪贪心 argmax（算法 5）	**44**	82.09%	82.07%

- 算法3和算法4 WER一样，但是算法更快；
- 算法5每次都取最佳值，运算速度快，但是求解的是局部解，整体误差率更大。

谢谢！

请老师各位批评指正！