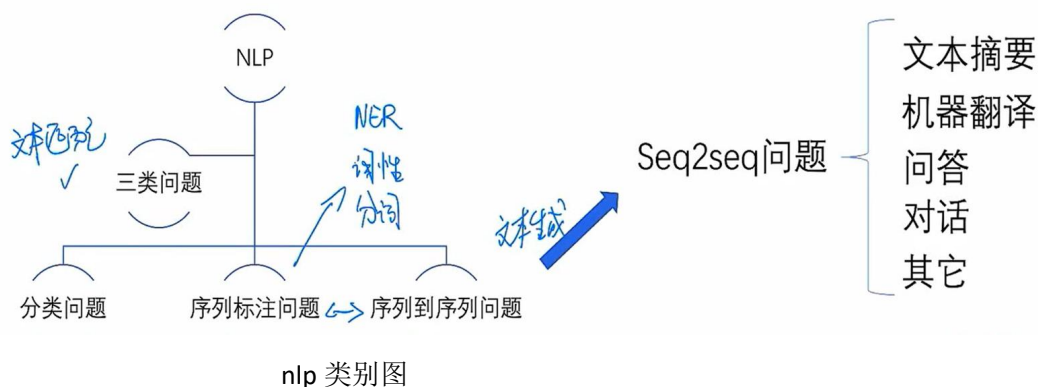


语言模型

一、项目导论

1.1 nlp 项目的类别

nlp 项目的类别可分为四种：文本匹配问题、文本分类问题、序列标注问题（ner/词性/分词）、序列到序列问题（文本摘要/机器翻译/问答/对话）



二、语言模型到 WordEmbedding

2.1 语言模型

(1) 如何表达一句话的好坏、合理与通顺？

答：我们采用概率的表达来表示，概率越高表示越通顺、合理。

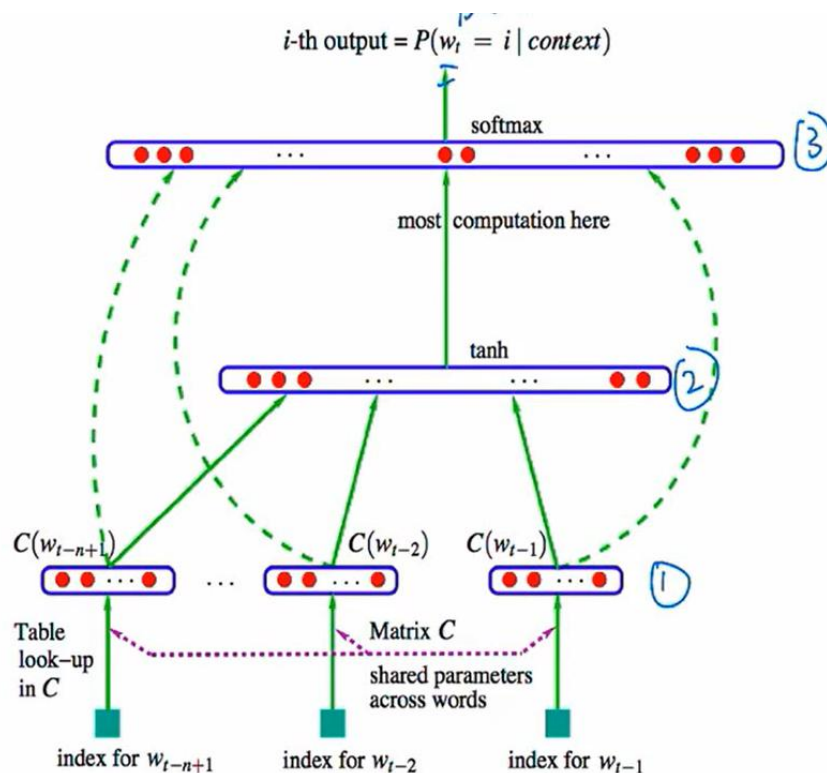
- 今天欢迎同学们来到后厂理工学院学习课程。
- 今天欢迎同学们后厂理工学院来到学习课程。
- 今天欢迎同学们来到后厂理工学院课程学习。

$$\begin{aligned} \uparrow P(s) &= P(w_1 w_2 \dots w_n) \\ &= P(w_1) P(w_2 | w_1) P(w_3 | w_1 w_2) \dots P(w_n | w_1 w_2 \dots w_{n-1}) \end{aligned}$$

其中， w_i 表示分词后的词， s 表示句子，采用联合概率表示，最终表示，当前词不仅自身相关，还与之前的所有词相关。

此外，引入马尔可夫假设，当前词只与上一个词相关，与其他词无关，这一种方式也叫 2-gram，如果相互独立，叫 1-gram；与前 2 个叫 3-gram...

(2) 最初的神经网络语言模型（NNLM）结构？



解释：总共分为 3 层：输入、隐层、输出；输入前 $n-1$ 个 token 采用编码（one-hot），乘以矩阵 C ，得到词的表征，将所有表征 concat，传入隐层，输出层采用 softmax 求出词概率。

(3) 语言的表达方式？

- ① 离散的表达方式：one-hot (先进行分词，构建 vocab, 求 one-hot)
缺陷：太稀疏、缺乏语义之间的相关性
- ② 通过上下文表达（通过周围词表达中间词）
- ③ word2vec（词向量方式）

(4) Word2vec

Word embedding 实现将一个词嵌入到一个空间的目的；具体：需要一个巨大的词表库，词表中每一个词可以向量表示，有一个中心词 c 和一个输出词 o ，计算 o 与 c 的相似度，保证同时出现的概率最大。

1.4.1 Word2vec 的目标函数：

$$L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

\downarrow
 参数

目的是根据中心词输入 w_t 与参数 θ ，保证周围词乘积的概率最大，遍历所有中心词 w_t 进行连乘即可，即 $L(\theta)$ 越大越好。

乘积可以表示相似度，中心词来预测周围词的概率可表示为：

$$P(o|c) = \text{softmax}(u_o^T v_c)$$

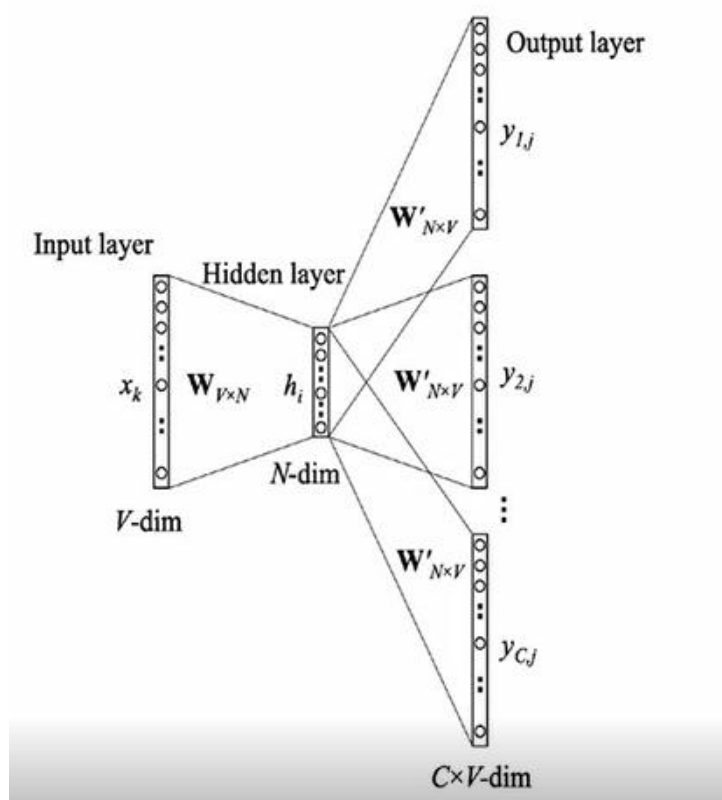
损失函数则取负数，再求 \log （保证浮点数不会上下溢）

$$J(\theta) = -\frac{1}{T} \log(L(\theta)) \quad \downarrow$$

1.4.2 词向量的模型的设计

(1) Skip-gram 模型

Skip-gram 目的是采用中心词来预测周围词，数据格式采用['中心词','左周围词','右周围词']，超参数 window 为滑窗，此时为 1；

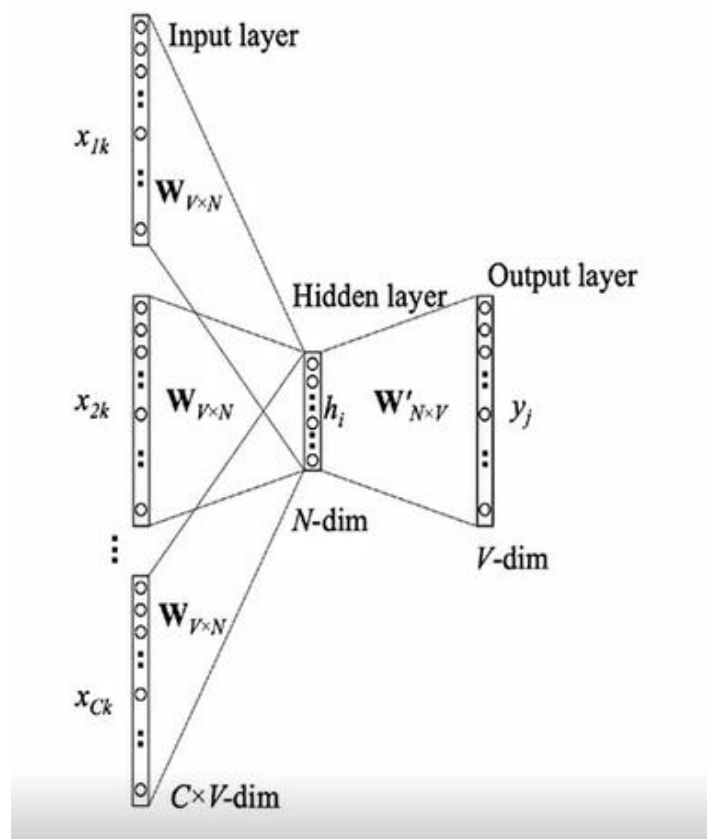


Skip-gram 结构图

解析：输入的代表采用 one-hot，隐层为一个矩阵，即词向量，输出层为周围词的预测，loss 采用交叉熵，进行 $\text{window} \times 2$ 次反向传播。

(2) cbow 模型

cbow 是一个连续的词袋模型，目的是采用周围的词来预测中心词，数据格式与 skip-gram 相反，与周围词相乘的 W 都是同一个 W ，即词向量，对所有周围进行加权平均，再传入隐层，最后预测中心词。



cbow 结构图

(3) skip-gram 与 cbow 的对比

①时间复杂度

Skip-gram 的时间复杂度为 $O(k \cdot v)$ ，因为需要 k 次反向传播，cbow 的时间复杂度为 $O(v)$

②周围词影响

Skip-gram 的效果较好一些，因为 cbow 需要周围词进行加权平均，收到许多周围词影响，效果比较差一点。

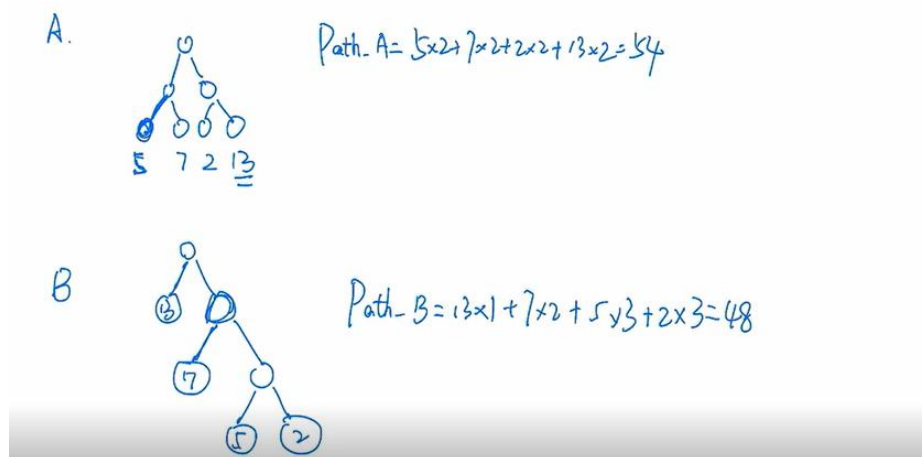
1.4.3 模型的改进与优化

为什么要进行优化？

答：在进行 skip-gram 或者 cbow 的时候，最后一层均采用 softmax 计算，当 vocab 很大时，softmax 分母的计算将会非常大，因此对最后的计算进行优化。

(1) Huffman Tree (哈夫曼树)

- 正常树与哈夫曼树的比较



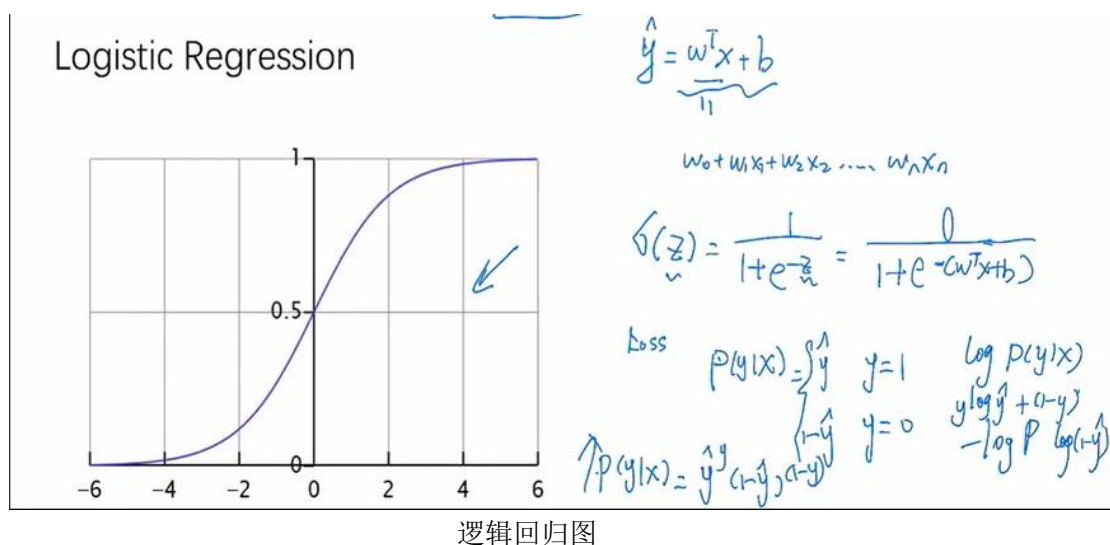
一般的树没有考虑词频，类似 A 树；然而哈夫曼树是路径最短的树，会将词频高放在顶层叶子，词频低的放在底层叶子。

- 哈夫曼编码

在哈夫曼树构建之后，将左节点置为 0，右节点置为 1，寻求路径的编码成为哈夫曼编码；

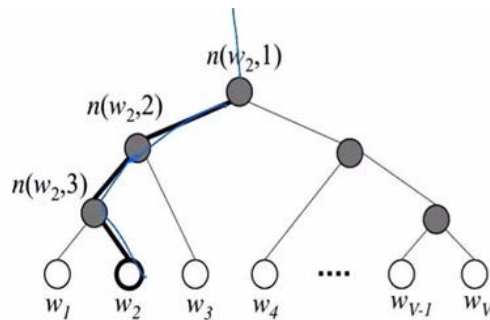
- Logistic Regression

采用逻辑回归来判断当前节点是往左走，还是往右走。



- Hierarchical Softmax

Hierarchical Softmax 则是将逻辑回归嵌入到哈夫曼树中；词表库一开始就是固定，说明哈夫曼树一开始也是定死的。所以一旦构建好哈夫曼树，后面的计算将会很快。



优点：计算复杂度会降至 $O(\log 2V)$ 、参与计算的参数更少

缺点：更偏向于高频词

(1) Negative Sampling (负采样)

负采样是每次让一个训练样本仅仅更新一小部分的权重；其中正样本为待预测词，负样本随机选择 k 个，小规模下， $5 \leq k \leq 20$ ，大规模下， $2 \leq k \leq 5$ ；

负采样的选择与词频一定的关联，概率公式如下：

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^n f(w_j)^{3/4}}$$

为何采用的词频的 $3/4$ 呢？

答：上图可得，词频的 $3/4$ 提升了低频词被选中的概率。

三、词向量的实践与应用

- 采用 gensim 调用词向量，具体百度
- 采用 AnnoyIndex 求词的最相似词检索，时间大大降低

原理：给数据建立 KD-tree 的索引空间

代码：

1、根据词向量构建 KD-树

```

In [36]: from annoy import AnnoyIndex

In [37]: wv_index = AnnoyIndex(256)
          词向量长度
          /Users/zn-nlp/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: The default argument for
          metric will be removed in future version of Annoy. Please pass metric='angular' explicitly.
          """Entry point for launching an IPython kernel.

In [ ]: i = 0
        for key in wv_model.vocab.keys():
            v = wv_model[key]
            wv_index.add_item(i, v)
            i += 1
          构建id-->词向量

In [ ]: wv_index.build(10)
          构建树，模型保存

In [ ]: wv_index.save('wv_index_build10.index')

```

2、根据新的词向量，查找与它最相似的 k 个词向量对应的 id

```

In [ ]: reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
          新的词-->id

In [ ]: for item in wv_index.get_nns_by_item(word_index[u'车'], 11):
          print(reverse_word_index[item])
          按词id寻找相似的词id，前11个

```