人工智能之NLP

NLP基础一

主讲人：GerryLiu

课程要求

- 课上课下“九字”真言
  - 认真听，**善摘录，勤思考**
  - **多温故，乐实践**，再发散
- 四不原则
  - 不懒散惰性，不迟到早退
  - 不请假旷课，不拖延作业
- 一点注意事项
  - 违反“四不原则”，不推荐就业

课程内容

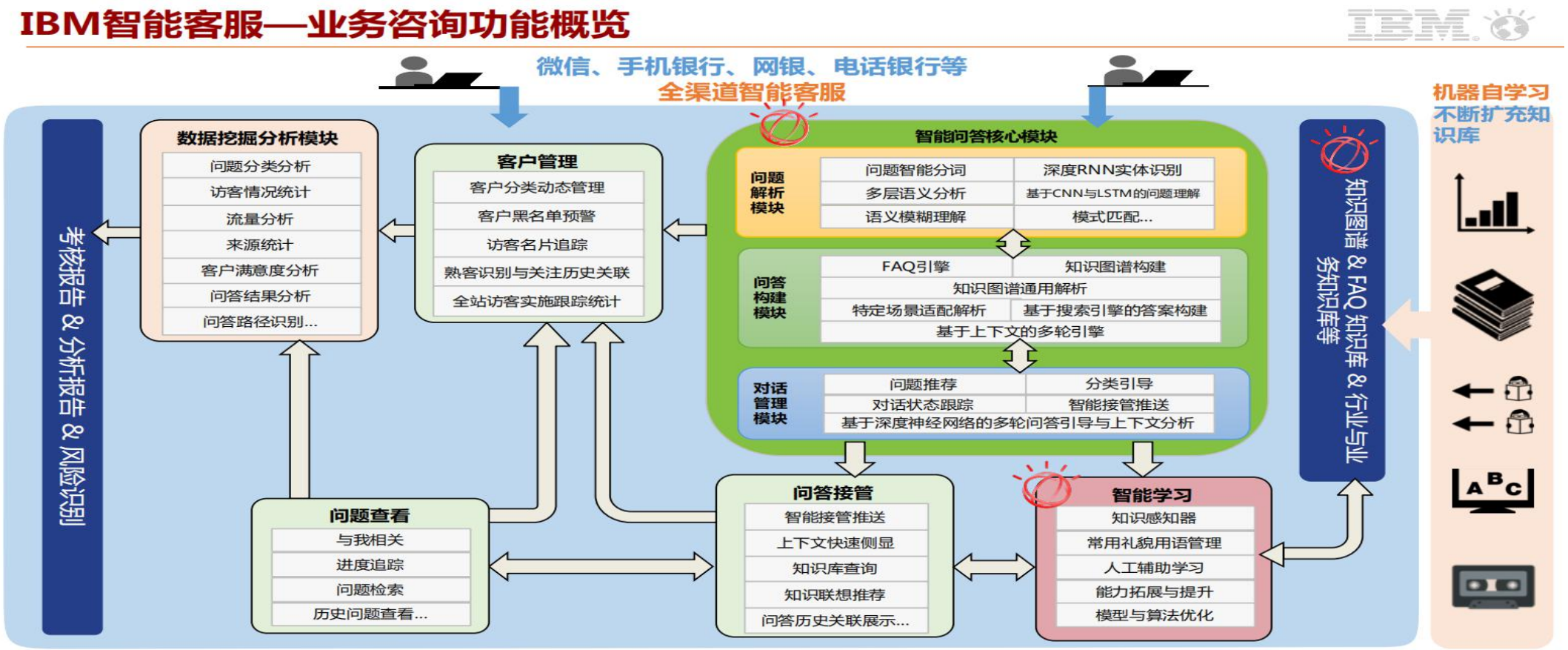
- NLP概述
- 分词
- 新词发现
- 词向量

NLP概述

- Natural Language Processing(NLP, 自然语言处理)
  - 目的：让计算机处理或“理解”自然语言，以执行语言翻译和问题回答等任务。
  - 常见应用：
    - 关键词提取、概要抽取
    - 命名实体识别(提取价格、日期、人员、公司等)
    - 关系抽取
    - 分类：文本分类、情感分析等
    - 机器翻译
    - 语音文本转换、图像文本转换
    - 问答系统

NLP概述

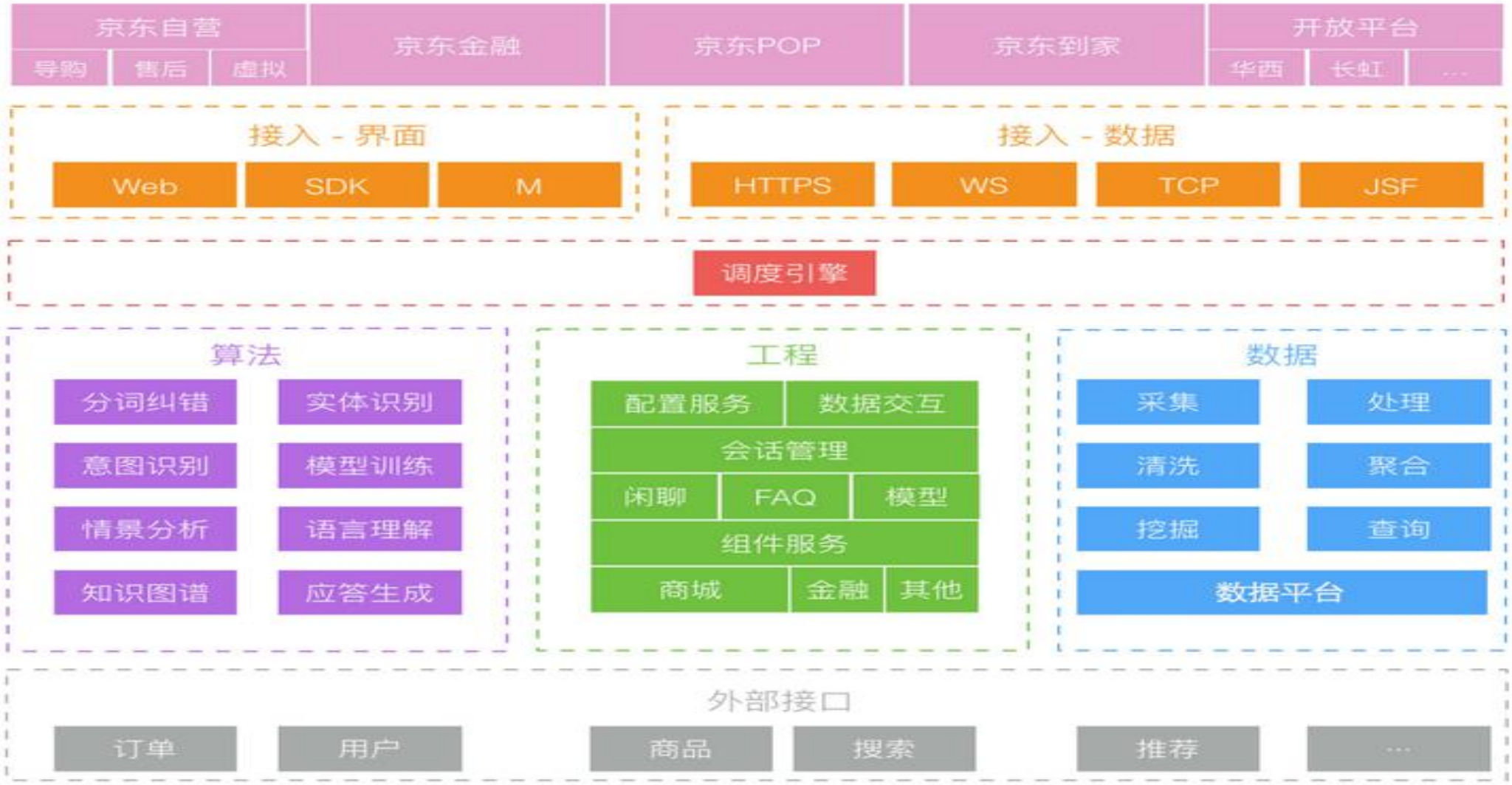
- IBM Waston



NLP概述

- 京东

NLP框架



NLP概述

- 美团点评

NLP框架

NLP概述

- NLP常用基本工具
  - jieba:
    - https://github.com/fxsjy/jieba
  - HanLP:
    - http://hanlp.com/
    - https://github.com/hankcs/pyhanlp
  - gensim
    - https://radimrehurek.com/gensim/
  - jiagu
    - https://github.com/ownthink/Jiagu
  - ltp:
    - http://ltp.ai/demo.html
    - https://github.com/HIT-SCIR/ltp

NLP基础_分词

- 分词是指将文本数据转换为一个一个的单词，是NLP自然语言处理过程中的基础；因为对于文本信息来讲，我们可以认为文本中的单词可以体现文本的特征信息，所以在进行自然语言相关任务的时候，第一步操作就是需要将文本信息转换为单词序列，使用单词序列来表达文本的特征信息。
- 分词：通过某种技术将连续的文本分隔成更具有语言语义学上意义的词。这个过程就叫做分词。
- Python中汉字分词包：jieba
  - 安装方式： pip install jieba
  - Github：https://github.com/fxsjy/jieba

```
C:\Users\ibf>pip install jieba
Collecting jieba
  Downloading jieba-0.39.zip (7.3MB)
    100% |████████████████████████████████|
Building wheels for collected packages: jieba
  Running setup.py bdist_wheel for jieba ...
  Stored in directory: C:\Users\ibf\AppData\L
811b0d
Successfully built jieba
Installing collected packages: jieba
Successfully installed jieba-0.39

C:\Users\ibf>python
Python 3.6.0 |Anaconda 4.3.1 (64-bit)| (defau
Type "help", "copyright", "credits" or "licen
>>> import jieba
```

NLP基础_分词_jieba

- "**Jieba**" Chinese text segmentation: built to be the best Python Chinese word segmentation module.
- Jieba常用的一种Python语言的**中文分词**和**词性标注**工具；算法基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图，然后采用动态规划查找最大概率路径，找出基于词频的最大切分组合；对于未登录词/新词和词性标注使用HMM的Viterbi算法来进行构造。

NLP基础_分词_jieba

- 分词模式：
  - **精确模式**
    - jieba.cut(str)
    - 试图将句子最精确地切开，适合文本分析；
  - **全模式**
    - jieba.cut(str,cut_all=True)
    - 把句子中所有的可以成词的词语都扫描出来, 速度非常快，但是不能解决歧义；
  - **搜索引擎模式**
    - jieba.cut_for_search(str)
    - 在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

NLP基础_分词_jieba

- 基础功能：
  - 1. 分词
    - 前缀字典匹配、HMM模型Viterbi算法
  - 2. 自定义词典添加
  - 3. 关键词抽取
    - TF-IDF、TextRank
  - 4. 词性标注
    - HMM模型Viterbi算法
  - 5. 并行分词
    - 当前版本不支持windows

NLP基础_分词

- 常规的分词技术：
  - 规则分词
    - 通过维护词典，通过切分语句的时候，将语句的每个字符串与词表中的词进行逐一匹配，找到则切分，否则不予切分。主要包括：
      - 正向最大匹配法
      - 逆向最大匹配法
      - 双向最大匹配法
  - 统计分词
    - 通过统计各个词在训练文本中出现的次数得到词的可信度，当连续的各个字出现的频度超过某个值的时候，就可以认为这个连续的各个字属于一个词。主要包括：
      - n-gram模型
      - HMM、CRF
  - 混合分词

NLP基础_分词_正向最大匹配法

- 正向最大匹配法(Maximum Match Method)
  - 假定词典中最长词有m个字符；
  - 从待分词文本中从左往右获取m个字符作为匹配字段；
  - 和词典进行匹配，若匹配成功，则将这个匹配字段作为词切分出来，若匹配不成功，那么将匹配字段的最后一个字符删除，形成新的匹配字段，重新进行匹配；
  - 重复上述操作完成所有词的切分。

NLP基础_分词_正向最大匹配法

- 正向最大匹配法
  - 原文本：长沙市长岳高速出现事故
  - 分词结果：长沙市长 岳 高速 出现 事故

单词
长沙市
长沙市长
长岳高速
出现
事故
长
岳
高速

NLP基础_分词_逆向最大匹配法

- 逆向最大匹配法(Reverse Maximum Match Model)
  - 假定词典中最长词有m个字符；
  - 从待分词文本中从右往左获取m个字符作为匹配字段；
  - 和词典进行匹配，若匹配成功，则将这个匹配字段作为也词切分出来，若匹配不成功，那么将匹配字段的第一个字符删除，形成新的匹配字段，重新进行匹配；
  - 重复上述操作完成所有词的切分。
  - NOTE：由于汉语的偏正结构偏多，采用从后往前的逆向最大匹配法可以适当的提高分词的精度。

NLP基础_分词_逆向最大匹配法

- 逆向最大匹配法
  - 原文本：长沙市长岳高速出现事故
  - 分词结果：长沙市 长岳高速 出现 事故

单词
长沙市
长沙市长
长岳高速
出现
事故
长
岳
高速

NLP基础_分词_双向最大匹配法

- 双向最大匹配法(Bi-directction Matching Method)
  - 将正向最大匹配法得到的分词结果和逆向最大匹配法得到的结果进行比较，按照最大匹配原则，选取**次数切分最少**的作为结果。
  - 90%左右的句子结果是一致的；9%左右的句子结果不一致，但是其中一个一定是正确的(歧义检测成功)；只有1%左右的句子是不对的。
  - 句子:”长沙市长岳高速出现事故”，逆向最大匹配法结果:“长沙市 长岳 高速 出现 事故”(4)，正向最大匹配法结果:“长沙市长 岳 高速 出现 事故”(5)，选择切分次数少的作为最终分词结果:“长沙市 长岳高速 出现 事故”

NLP基础_分词_N-Gram

- N-Gram模型是一种基于统计的模型，又称为语言模型，基本算法思想是**将文本的内容按字符进行大小为N的窗口滑动，形成长度为N的字符序列**，每个字符序列称为Gram，然后进行统计操作。该模型是基于马尔可夫假设，也就是**第N个词的出现只与前N-1个词相关，而和其它任何词都不相关**。
- 语言模型在自然语言处理中占有重要的地位，在语音识别，机器翻译，汉语自动分词和句法分析等都有应用。因为这些模型都会有噪声，都会有几种不同的结果等着我们去选择，这时候就需要知道每种结果的概率，来帮助我们选择。

NLP基础_分词_N-Gram

- 文本序列S：

$$S = w_1, w_2, \ldots, w_T$$

- 文本S出现的概率值：

$$P(S) = P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t | w_1, w_2, \ldots, w_{t-1})$$

- 利用马尔可夫假设，将条件概率的求解进行简化。

$$p(w_t | w_1, w_2, \ldots, w_{t-1}) = p(w_t | w_{t-n+1}, \ldots, w_{t-1})$$

NLP基础_分词_N-Gram

- 常见的N-Gram模型有：
  - 1-Gram： unigram model

$$P(S)=\prod_{t=1}^{T}p(w_t)$$

  - **2-Gram： bigram model**

$$P(S)=\prod_{t=1}^{T}p(w_t|w_{t-1})$$

  - 3-Gram: trigram model

$$P(S)=\prod_{t=1}^{T}p(w_t|w_{t-2},w_{t-1})$$

采用极大似然估计也就是大数定理去统计这个概率值。

NLP基础_分词_N-Gram

- bigram model

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$$P(\texttt{I} \mid \texttt{<s>}) = \tfrac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \tfrac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \tfrac{2}{3} = .67$$

$$P(\texttt{</s>} \mid \texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \tfrac{1}{3} = .33$$

NLP基础_分词_N-Gram

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

NLP基础_分词_N-Gram

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

NLP基础_分词_N-Gram

- N-Gram模型的应用场景：
  - 分词
  - 中文分词消歧
  - 输入法/关联词
  - 文本相似度
  - 机器翻译

NLP基础_分词

- N-Gram应用场景：分词
  - 原文本：长沙市长岳高速出现事故
  - 首先统计出训练集词典中各个字的频率；然后，对待切分文本的进行某种策略的切分，递归选择切分概率最大的子切分序列，最后回溯得到最大概率切分。
  - eg:
    - p1 = p(长沙市)*p(长岳高速出现事故|长沙市)
    - p2 = p(长沙市长)*p(岳高速出现事故|长沙市长)

NLP基础_分词_N-Gram

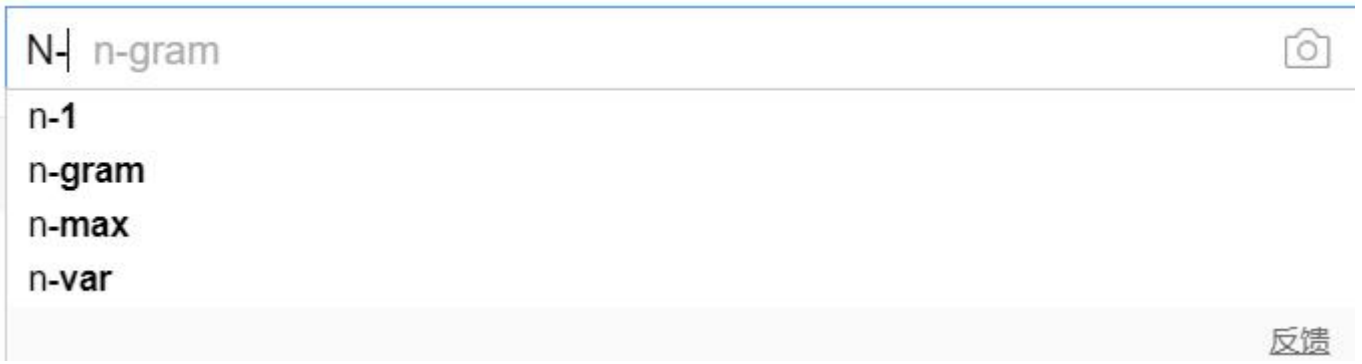
- N-Gram模型的应用场景：中文分词消歧
  - 先使用其它方式产生不同的分词组合，然后通过N-Gram来计算不同组合的概率值，选择概率值大的作为最终结果，从而达到消歧的作用。
    - 原文本：长沙市长岳高速出现事故
    - 正向最大匹配结果：长沙市长 岳 高速 出现 事故
    - 逆向最大匹配结果：长沙市 长岳高速 出现 事故

$$p_1 = p(\text{长沙市长}) * p(\text{岳} | \text{长沙市长}) * p(\text{高速} | \text{长沙市长}, \text{岳}) * p(\text{出现} | \text{岳}, \text{高速}) * p(\text{事故} | \text{高速}, \text{出现})$$

$$p_2 = p(\text{长沙市}) * p(\text{长岳高速} | \text{长沙市}) * p(\text{出现} | \text{长沙市}, \text{长岳高速}) * p(\text{事故} | \text{长岳高速}, \text{出现})$$

NLP基础_分词_N-Gram

- N-Gram模型的应用场景：输入法/关联词



NLP基础_分词_N-Gram

- N-Gram模型的应用场景：文本相似度
  - 使用N对文本进行划分，然后通过jaccard相似度计算重复的比例作为相似度的度量。
    - 1. 机器学习算法
    - 2. 深度学习算法框架
    - 3. 大数据实时处理框架

	1	2	3
1	1.0	3/9	0
2	3/9	1.0	1/14
3	0	1/14	1.0

1	机器、器学、学习、习算、算法
2	深度、度学、学习、习算、算法、法框、框架
3	大数、数据、据实、实时、时处、处理、理框、框架

NLP基础_分词_N-Gram

- N-Gram模型的应用场景：机器翻译
  - I love my family so much --> 我非常爱我的家人

英文单词	中文单词
I	我
love	爱、喜好、热爱、爱情
my	我的
family	家、家庭、家人、家族
so	这样、那么、如此、非常
much	非常

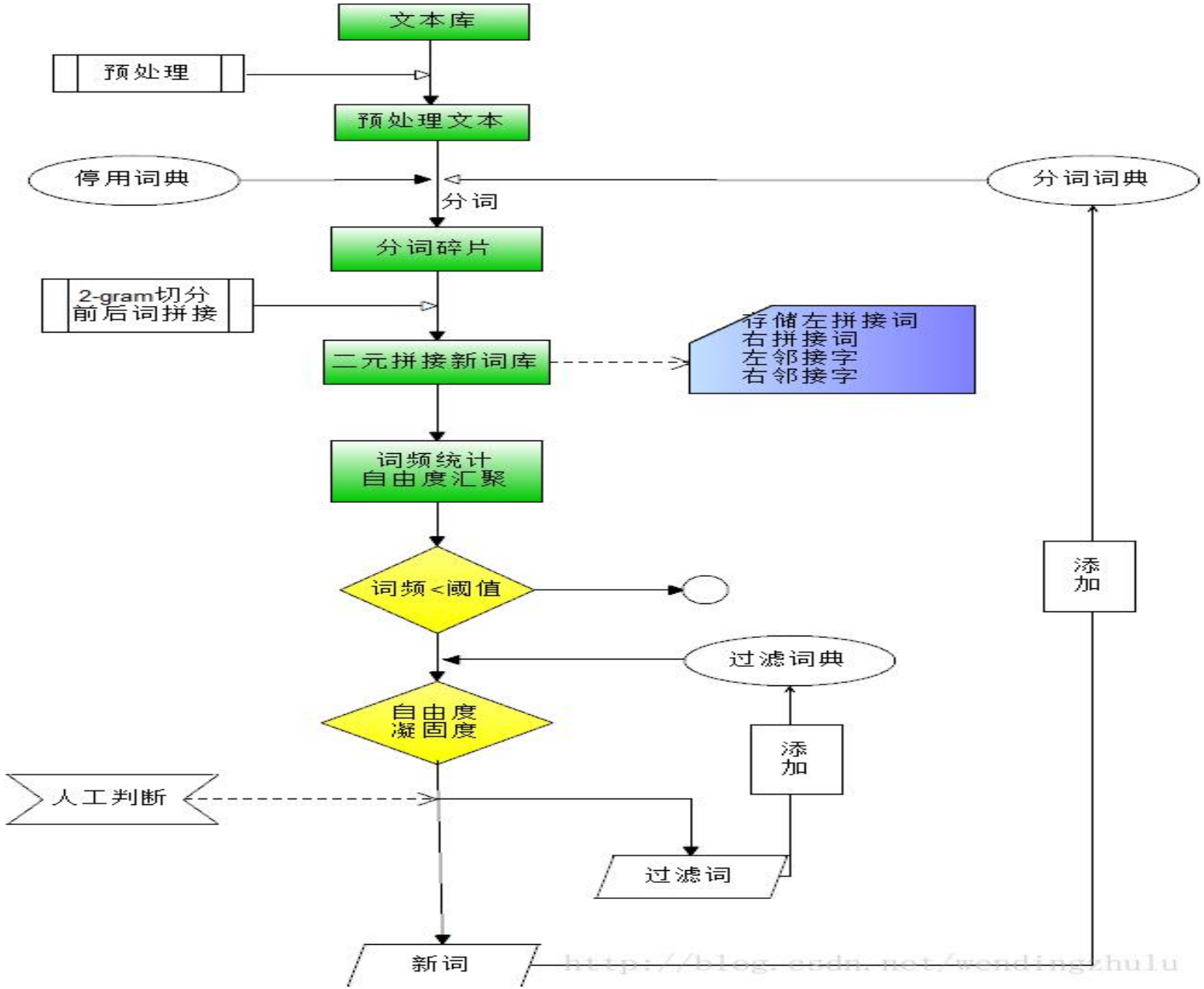
NLP基础_分词_HMM/CRF

- HMM/CRF是将分词作为字在字串中的序列标注任务来实现，使用Viterbi算法来预测各个字对应的隐状态，然后基于隐状态来进行分词操作。规定每个字对应的隐状态是：B(词首)、M(词中)、E(词尾)、S(单独成词)。
  - eg: 自然语言的基础是中文分词

自	然	语	言	的	基	础	是	中	文	分	词
B	M	M	E	S	B	E	S	B	E	B	E

NLP基础_新词发现

- 把文本中出现过的所有长度不超过d的子串当做潜在词，计算该词出现的**频率数**、**凝固程度**以及**自由程度**，然后提取出超过阈值的候选词作为新词。



NLP基础_新词发现

- **频率数**：单词出现的频次数目；
- **凝固程度**：凝固度是指一个新词单独出现的概率和其组合词出现概率的商值；当凝固度越高的时候，越表示属于一个单词。比如：词A、B组成词C，同时P(C)/(P(A)*P(B))的值非常大，那就表示词C是一个独立的词。
- **自由程度**：左/右邻接词出现的概率除以当前单词的概率越小，表示当前单词独立出现的可能性越高，也就是可以认为是独立词。

NLP基础_新词发现

- 丰富分词词库
- 当日热点词
- 挖掘不同群体的热词

男性爱说的词

兄弟、篮球、男篮、米兰、曼联、足球、蛋疼、皇马、比赛、国足、超级杯、球迷、中国、老婆、政府、航母、踢球、赛季、股市、砸蛋、牛逼、铁道部、媳妇、国际、美国、连败、魔兽、斯内德、红十字、经济、腐败、程序、郭美美、英雄、民主、鸟巢、米兰德比、官员、内涵、历史、训练、评级、金融、体育、记者、事故、程序员、媒体、投资、事件、社会、项目、伊布、主义、决赛、操蛋、纳尼、领导、喝酒、民族、新闻、言论、和谐、农民、体制、城管……

女性爱说的词

一起玩、蛋糕、加好友、老公、呜呜、姐姐、嘻嘻、老虎、讨厌、妈妈、呜呜呜、啦啦啦、便宜、减肥、男朋友、老娘、逛街、无限、帅哥、礼物、互相、奶茶、委屈、各种、高跟鞋、指甲、城市猎人、闺蜜、巧克力、第二、爸爸、宠物、箱子、吼吼、大黄蜂、狮子、胃疼、玫瑰、包包、裙子、游戏、遇见、嘿嘿、灰常、眼睛、各位、妈咪、化妆、玫瑰花、蓝精灵、幸福、陪我玩、任务、怨念、舍不得、害怕、狗狗、眼泪、温暖、面膜、收藏、李民浩、神经、土豆、零食、痘痘、戒指、巨蟹、晒黑……

NLP基础_向量化

- 在分词之后，对于文本类型的特征属性，需要进行文本数据转换，也就是需要将文本数据转换为数值型数据。常用方式如下：
  - 序号化(LabelEncode)、哑编码(OneHot)、词袋法(BOW/TF)
  - TF-IDF(Term frequency-inverse document frequency)
  - 主题模型(LSA、LDA等)
  - Word2Vec、Char2Vec
  - Doc2Vec
  - FastText、cw2vec
- 常见词向量转换工具：
  - gensim
    - https://github.com/RaRe-Technologies/gensim
    - https://radimrehurek.com/gensim/index.html

```
C:\Users\ibf>pip install gensim==3.4.0 -i https://pypi.tuna.tsinghua.edu.cn/simple/
Collecting gensim==3.4.0
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/e8/c8/e2e6cb141aea53927aa1d554ab5919e202e61c
.0-cp35-cp35m-win_amd64.whl
Requirement already satisfied: six>=1.5.0 in c:\python3.5\lib\site-packages (from gensim==3.4.0)
Requirement already satisfied: numpy>=1.11.3 in c:\python3.5\lib\site-packages (from gensim==3.4.0)
Requirement already satisfied: scipy>=0.18.1 in c:\python3.5\lib\site-packages (from gensim==3.4.0)
Requirement already satisfied: smart-open>=1.2.1 in c:\python3.5\lib\site-packages (from gensim==3.4.0
Requirement already satisfied: boto>=2.32 in c:\python3.5\lib\site-packages (from smart-open>=1.2.1->g
Requirement already satisfied: requests in c:\python3.5\lib\site-packages (from smart-open>=1.2.1->gen
Requirement already satisfied: boto3 in c:\python3.5\lib\site-packages (from smart-open>=1.2.1->gensim
Requirement already satisfied: idna<2.9,>=2.5 in c:\python3.5\lib\site-packages (from requests->smart-
Requirement already satisfied: certifi>=2017.4.17 in c:\python3.5\lib\site-packages (from requests->sm
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\python3.5\lib\site-packag
2.1->gensim==3.4.0)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\python3.5\lib\site-packages (from requests-
)
Requirement already satisfied: botocore<1.13.0,>=1.12.199 in c:\python3.5\lib\site-packages (from botc
.0)
Requirement already satisfied: s3transfer<0.3.0,>=0.2.0 in c:\python3.5\lib\site-packages (from boto3-
)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in c:\python3.5\lib\site-packages (from boto3->s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in c:\python3.5\li
13.0,>=1.12.199->boto3->smart-open>=1.2.1->gensim==3.4.0)
Requirement already satisfied: docutils<0.15,>=0.10 in c:\python3.5\lib\site-packages (from botocore<1
en>=1.2.1->gensim==3.4.0)
Installing collected packages
Successfully installed gensi
You are using pip version 9.
You should consider upgradin
```

```
C:\Users\ibf>python
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017,
Type "help", "copyright", "credits" or "license"
>>> import gensim
C:\Python3.5\lib\site-packages\gensim\utils.py:1
  warnings.warn("detected Windows; aliasing chur
>>> gensim.__version__
'3.4.0'
```

NLP基础_词向量_gensim

- Gensim属于一款开源的第三方Python工具包，用于从原始的非结构化的文本数据中，无监督的学习到文本隐层的主题向量表达形式，也即是词向量/文本向量转换；并且在向量转换的基础上提供了相似度计算、信息检索等功能的API。
  - 语料(Corpus)：一组原始文本集合，用于无监督训练文本主题的数据。在gensim中，corpus通常是一个可迭代对象；
  - 向量(Vector)：由一组文本特征构成的列表，是一段文本在Gensim中的内部表达；
  - 稀疏向量(SparseVector)：略去向量中多余的0元素。此时，向量中的每一个元素是一个(key, value)的元组；
  - 模型(Model)：定义了两个向量空间的转换。也就是将向量表达形式从某一种变量变成另外一种变量表达。

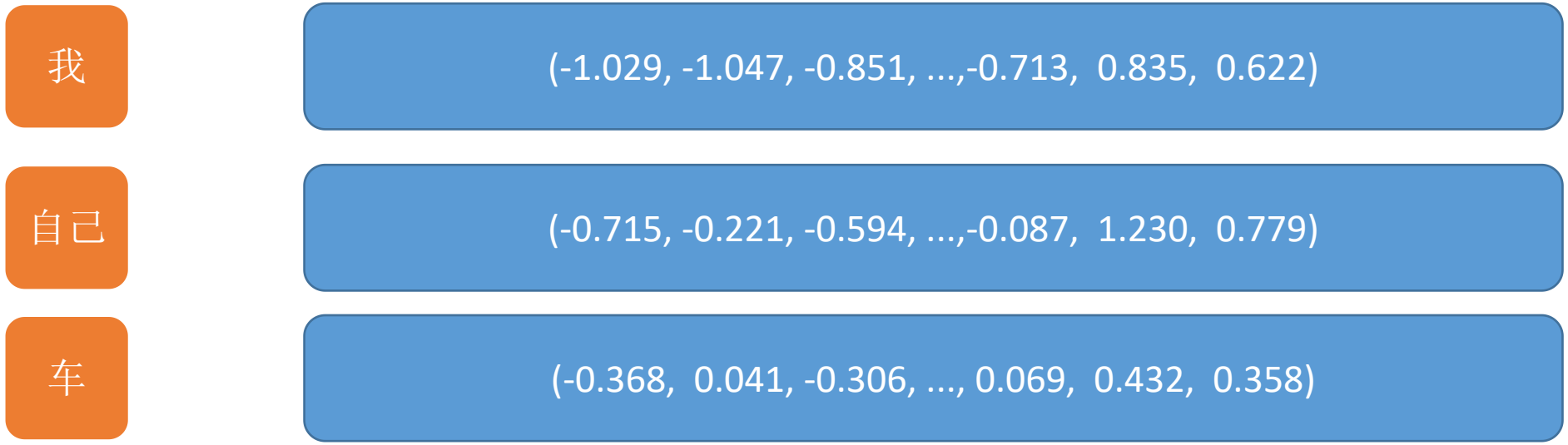
NLP基础_词向量_gensim

- gensim： https://radimrehurek.com/gensim/index.html
- 主要模型：
  - models.ldamodel
  - models.nmf
  - models.tfidfmodel
  - models.word2vec
  - models.doc2vec
  - models.fasttext

- models.ldamodel – Latent Dirichlet Allocation
  - Usage examples
- models.ldamulticore – parallelized Latent Dirichlet Allocation
  - Usage examples
- models.nmf – Non-Negative Matrix factorization
- models.lsimodel – Latent Semantic Indexing
- models.ldaseqmodel – Dynamic Topic Modeling in Python
- models.tfidfmodel – TF-IDF model
- models.word2vec – Word2vec embeddings
  - Other embeddings
  - Usage examples
- models.keyedvectors – Store and query word vectors
  - Why use KeyedVectors instead of a full model?
  - How to obtain word vectors?
  - What can I do with word vectors?
- models.doc2vec – Doc2vec paragraph embeddings
  - Usage examples
- models.fasttext – FastText model
  - Usage examples
  - Implementation Notes

NLP基础_词向量

- 词向量转换也就是将文本数据转换为数值型数据的一种方式，在人工智能中，这种方式统称为**Word Embedding**；**Word Embedding**实际上就是一种**映射**，将文本空间中的某个**word**，通过一定的方法，**映射**或者**嵌入(Embedding)**到另外一个数值向量空间。之所以称为embedding，是因为这种映射往往伴随着降维的思想。



NLP基础_词向量_One-Hot

- 使用一个非常稀疏的向量来表示单词的特征向量信息，假设现在有n个单词，那么转换的特征向量就是n维，仅在对应位置为1，其它位置全部为0。

我	1,0,0,0,0,0,0,0
来	0,1,0,0,0,0,0,0
自	0,0,1,0,0,0,0,0
湖	0,0,0,1,0,0,0,0
南	0,0,0,0,1,0,0,0
张	0,0,0,0,0,1,0,0
家	0,0,0,0,0,0,1,0
界	0,0,0,0,0,0,0,1

NLP基础_词向量_词袋法

- 词袋法(Bag of words, BOW)是最早应用于NLP和IR领域的一种文本处理模型，该模型忽略文本的语法和语序，用一组无序的单词(words)来表达一段文字或者一个文档，词袋法中**使用单词在文档中出现的频数/频率来表示文档，使用所有文档中出现的单词作为特征属性**。

d1:this is a sample is a sample

d2:this is anthoer example anthoer example

dict(词典、字典、特征属性): this sample anthoer example

	this	another	sample	example
d_1	1	0	2	0
d_2	1	2	0	2

NLP基础_词向量_TFIDF

- 在词袋法的基础上加入单词重要性的影响系数：
  - 单词的重要性随着它在文本中出现的次数成正比增加，也就是单词的出现次数越多，该单词对于文本的重要性就越高。
  - 同时单词的重要性会随着在语料库/训练数据中出现的频率成反比下降，也就是单词在语料库中出现的频率越高，表示该单词越常见，也就是该单词对于文本的重要性越低。

Variants of term frequency (TF) weight

weighting scheme	TF weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \ / \ \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Variants of inverse document frequency (IDF) weight

weighting scheme	IDF weight ($n_t = \|\{d \in D : t \in d\}\|$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(1 + \frac{N}{n_t}\right)$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

NLP基础_词向量_TFIDF

- 有两个文档，单词统计如下，请分别计算各个单词在文档中的TF-IDF值以及这些文档使用单词表示的特征向量。

单词	单词次数
this	1
is	1
a	2
sample	1

单词	单词次数
this	2
is	1
another	2
example	3

$$tf("this", d_1) = \frac{1}{5} \quad tf("this", d_2) = \frac{2}{8}$$

$$idf("this", D) = \log\left(\frac{2+1}{2+1}\right) = 0$$

$$tfidf("this", d_1) = tf("this", d_1) * idf("this", D) = 0$$

$$tfidf("this", d_2) = 0$$

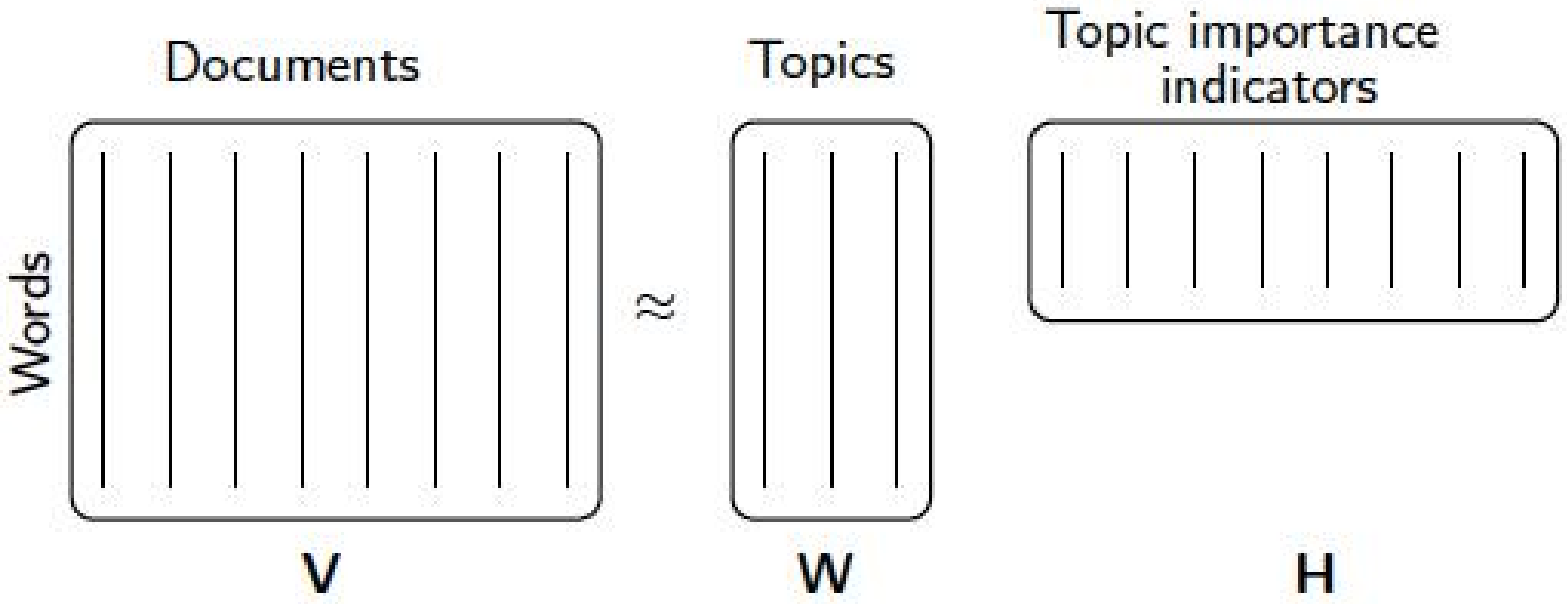
	this	is	a	another	sample	example
d_1	0	0	0.191	0	0.095	0
d_2	0	0	0	0.119	0	0.179

NLP基础_词向量

- 两个词向量之间的距离（例如，任意两个向量之间的L2范式距离或更常用的余弦距离）一定程度上表征了的词之间的语义关系；
- 例如，“椰子”和“北极熊”是语义上完全不同的词，所以它们的词向量在一个合理的嵌入空间的距离将会非常遥远。但“厨房”和“晚餐”是相关的话，所以它们的词向量之间的距离会相对小。
- 但是哑编码、词袋法以及TF-IDF这些方式都不能达到这个效果，故提出了基于矩阵分解的主题模型算法以及基于神经网络的Word2Vec的词向量转换方式。

NLP基础_词向量_主题模型

- 直接使用机器学习的主题模型相关算法将词袋法转换的文本单词特征属性句子转换为文本主题特征属性矩阵以及单词主题特征属性矩阵即可得到单词对应的特征向量以及文本对应的特征向量。



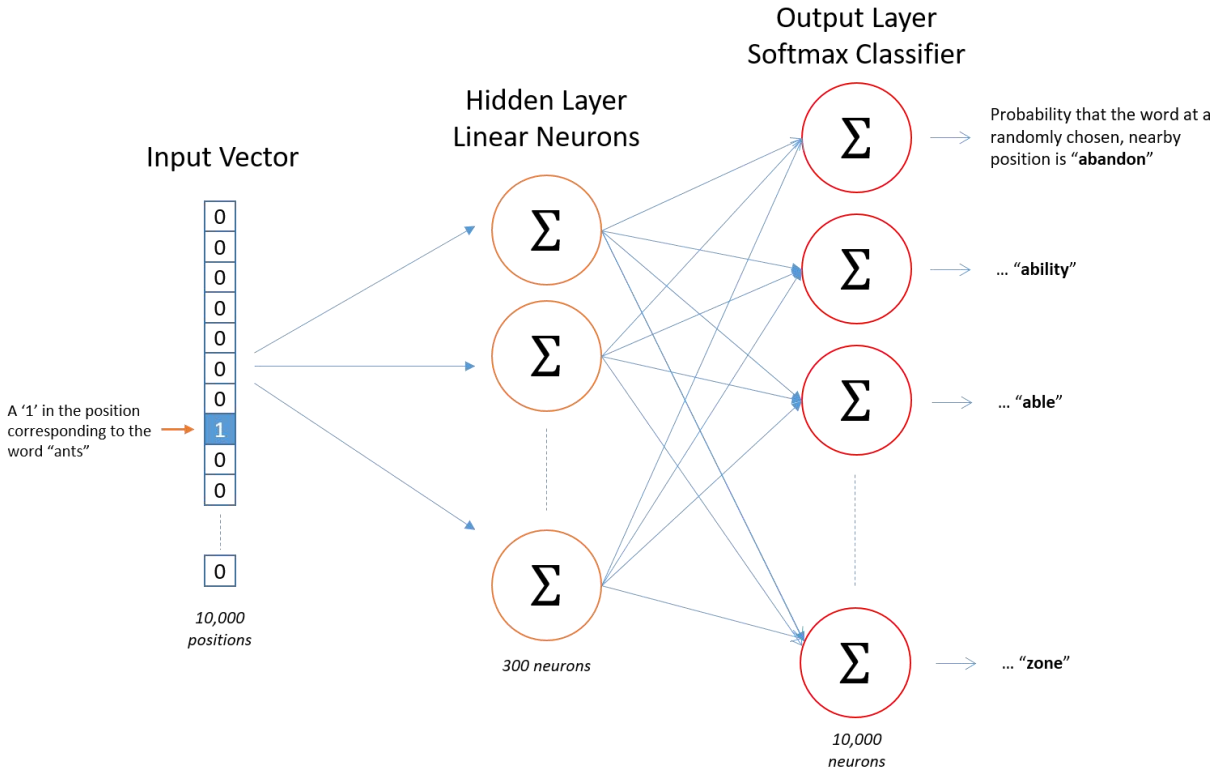
NLP基础_词向量_Word2Vec

- 词向量方法是无监督学习的少数几个成功应用之一，优点在于不需要人工进行预料的标注，直接使用未标注的文本训练集作为输入，输出的词向量可以用于下游的业务处理。
- 词向量真正的推广是开始于2013年Mikolov开发出来的Word2Vec，2014年Pennington发布了GloVe，进一步让让词向量的应用成为NLP领域中的主流。
- NOTE: Word2Vec可以认为是应用最广泛的词向量转换技术。

NLP基础_词向量_Word2Vec

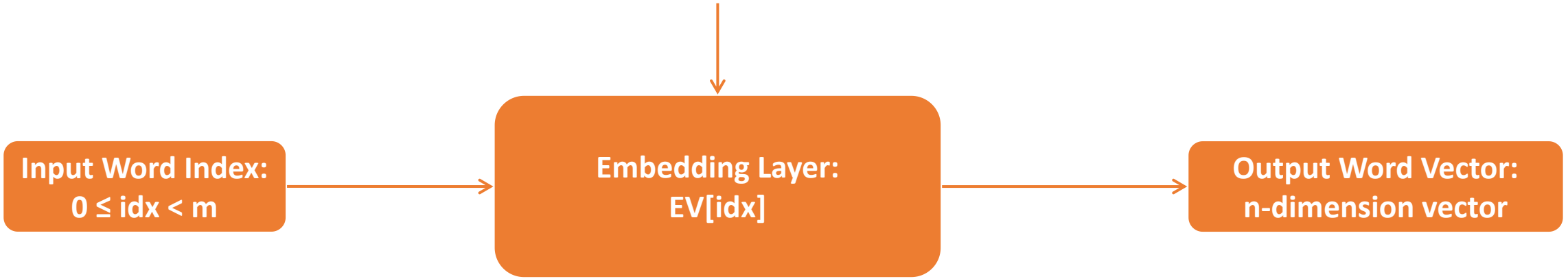
- 神经网络将词表中的词语作为输入(一般输入哑编码的单词)，输出一个低维度的向量表示这个词语，然后用反向传播的方法不断优化参数。输出的低维向量是神经网络第一层的输出，这一层通常也称作**Embedding Layer**。

$$\begin{bmatrix}0 & 0 & 0 & 1 & 0\end{bmatrix} \times \begin{bmatrix}8 & 2 & 1 & 9\\ 6 & 5 & 4 & 0\\ 7 & 1 & 6 & 2\\ 1 & 3 & 5 & 8\\ 0 & 4 & 9 & 1\end{bmatrix} = \begin{bmatrix}1 & 3 & 5 & 8\end{bmatrix}$$



NLP基础_词向量_Word2Vec

$$EV_{m,n} = \begin{bmatrix} -1.0152 & .... & .... & 0.1254 \\ .... & .... & .... & .... \\ .... & .... & .... & .... \\ 1.0251 & .... & .... & 0.2535 \end{bmatrix}$$

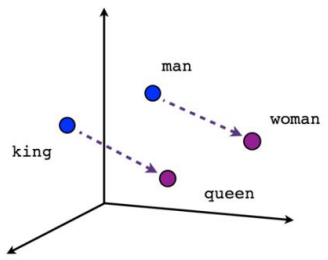


NLP基础_词向量_Word2Vec

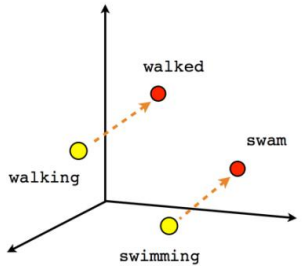
- 生成词向量的神经网络模型分为两种，一种是类似Word2Vec的方式，这类模型的目的就是生成词向量，另一种是将词向量作为副产品产生，两者的区别在于计算量不同。若词表非常庞大，用深层结构的模型训练词向量需要许多计算资源。
- Word2Vec和GloVe(Global Vectors for Word Representation)的目的是训练可以表示语义关系的词向量，它们能被用于后续的任务中；如果后续任务不需要用到语义关系，则按照此方式生成的词向量并没有什么用。另一种模型则根据特定任务需要训练词向量。当然，若特定的任务就是对语言建模，那么两种模型生成的词向量非常相似了。

NLP基础_词向量_Word2Vec

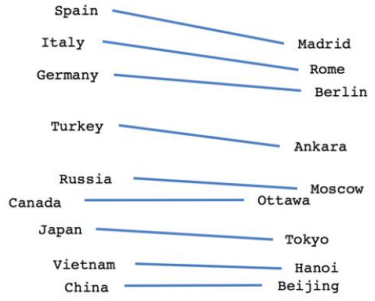
- Word2Vec(Efficient Estimation of Word Representations in Vector Space)
  - CBOW
    - predicts the current word based on the context；
  - Skip-Gram
    - predicts surrounding words given the current word；



Male-Female

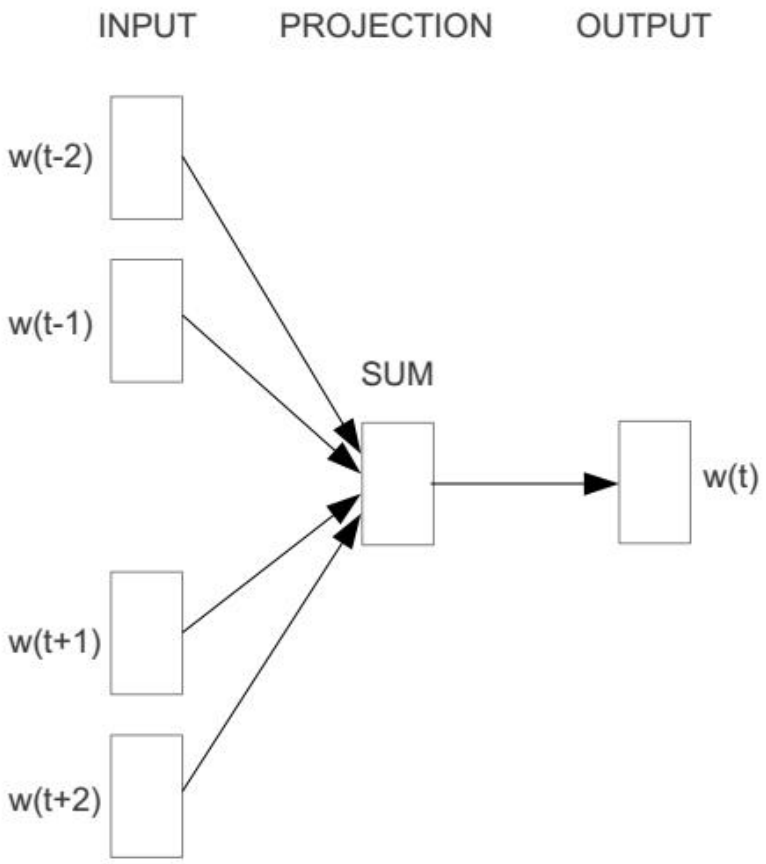


Verb tense

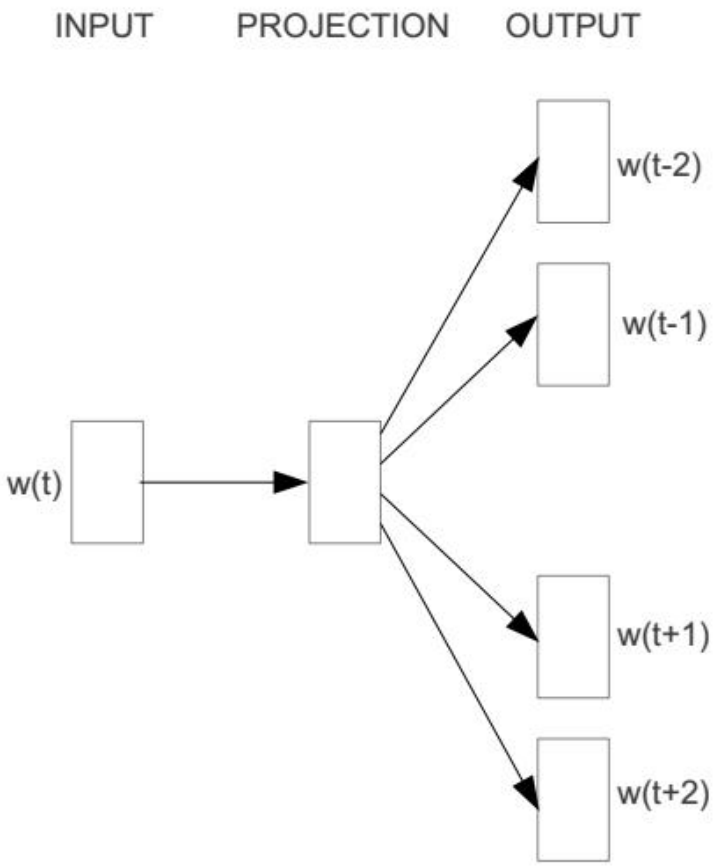


Country-Capital

NLP基础_词向量_Word2Vec

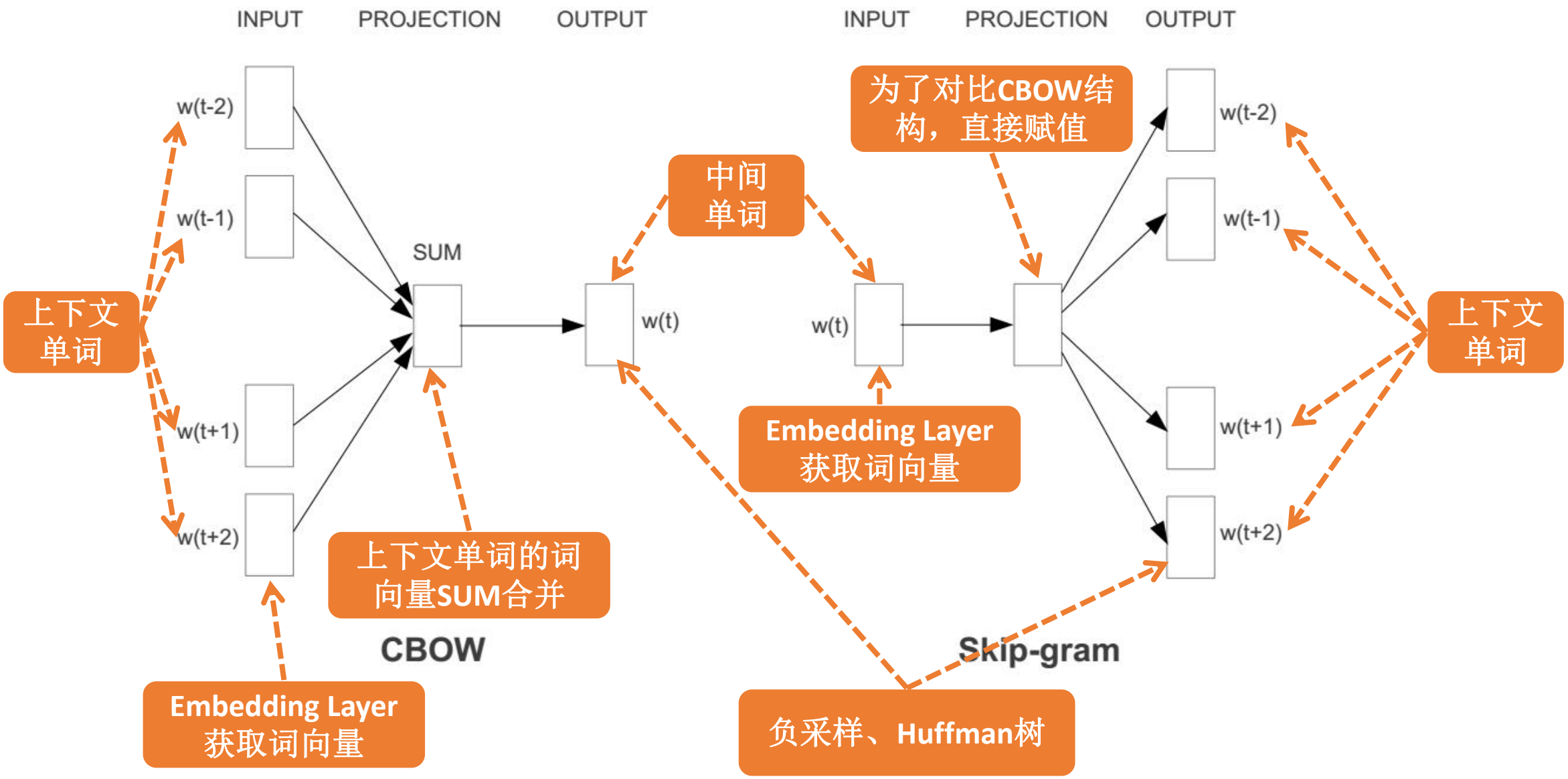


CBOW



Skip-gram

NLP基础_词向量_Word2Vec

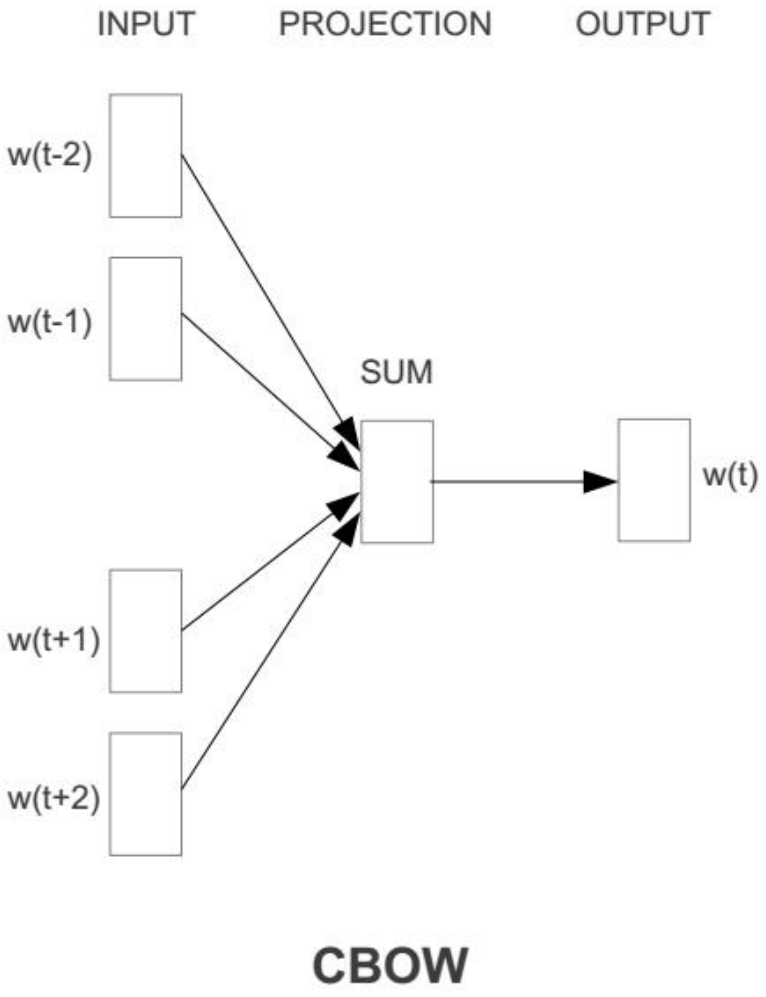


NLP基础_词向量_Word2Vec_CBOW

- CBOW(Continuous Bag of Words)
  - 给定上下文预测目标词的概率分布，

    例如，给定｛The，cat，()，over，the，puddle｝预测中心词是jumped的概率，

    模型的结构如下:
  - 比如：”一只猫坐在地上”，中间词为”猫”，那么上下文词为: [“一只”, “坐在”, “地上”]



NLP基础_词向量_Word2Vec_CBOW

- 前向过程：

$$v_{c-k} = Uw_{c-k}$$

词向量转换
Embedding Layer

$$v_c = v_{c-m} + \ldots + v_{c-1} + v_{c+1} + \ldots + v_{c+m}$$

$$z_c = Vv_c$$

负采样

$$\hat{y}_c = soft\max(z_c)$$

- 损失函数：

$$y_c = one\_hot(w_c)$$

$$loss = -y_c \log(\hat{y}_c)$$

NLP基础_词向量_Word2Vec_CBOW

$$
\begin{aligned}
\text{minimize } J &= -\log P(w_c | w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}) \\
&= -\log P(u_c | \hat{v}) \\
&= -\log \frac{\exp(u_c^T \hat{v})}{\sum_{j=1}^{|V|} \exp(u_j^T \hat{v})} \\
&= -u_c^T \hat{v} + \log \sum_{j=1}^{|V|} \exp(u_j^T \hat{v})
\end{aligned}
$$

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

1. **输入层**: 包含 $Context(w)$ 中 $2c$ 个词的词向量 $\mathbf{v}(Context(w)_1), \mathbf{v}(Context(w)_2), \cdots, \mathbf{v}(Context(w)_{2c}) \in \mathbb{R}^m$. 这里, m 的含义同上表示词向量的长度.

2. **投影层**: 将输入层的 $2c$ 个向量做求和累加, 即 $\mathbf{x}_w = \sum_{i=1}^{2c} \mathbf{v}(Context(w)_i) \in \mathbb{R}^m$.

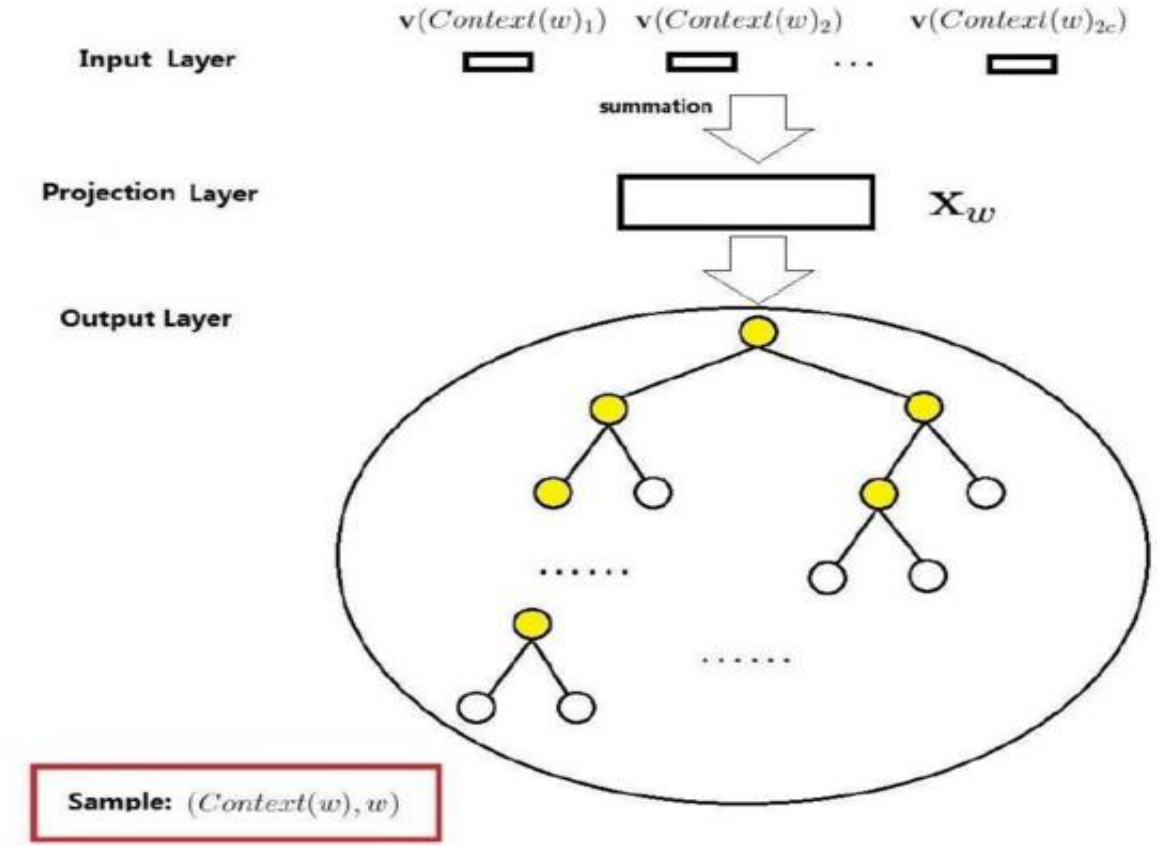


图 10 CBOW 模型的网络结构示意图

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

3. **输出层**: 输出层对应一棵二叉树, 它是以语料中出现过的词当叶子结点, 以各词在语料中出现的次数当权值构造出来的 Huffman 树. 在这棵 Huffman 树中, 叶子结点共 N $(=|\mathcal{D}|)$ 个, 分别对应词典 $\mathcal{D}$ 中的词, 非叶子结点 $N-1$ 个 (图中标成黄色的那些结点).

对比 §3.3 中神经概率语言模型的网络图 (见图 4) 和 CBOW 模型的结构图 (见图 10), 易知它们主要有以下三处不同:

1. (从输入层到投影层的操作) 前者是通过拼接, 后者通过**累加求和**.

2. (隐藏层) 前者有隐藏层, 后者**无隐藏层**.

3. (输出层) 前者是线性结构, 后者是**树形结构**.

在 §3.3 介绍的神经概率语言模型中, 我们指出, 模型的大部分计算集中在隐藏层和输出层之间的矩阵向量运算, 以及输出层上的 softmax 归一化运算. 而从上面的对比中可见, CBOW 模型对这些计算复杂度高的地方有针对性地进行了改变, 首先, 去掉了隐藏层, 其次, 输出层改用了 Huffman 树, 从而为利用 Hierarchical softmax 技术奠定了基础.

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

§4.1.2 梯度计算

Hierarchical Softmax 是 wor2vec 中用于提高性能的一项关键技术. 为描述方便起见, 在具体介绍这个技术之前, 先引入若干相关记号. 考虑 Huffman 树中的某个叶子结点, 假设它对应词典 $\mathcal{D}$ 中的词 w, 记

1. p^w: 从根结点出发到达 w 对应叶子结点的路径.
2. l^w: 路径 p^w 中包含结点的个数.
3. $p_1^w, p_2^w, \cdots, p_{l^w}^w$: 路径 p^w 中的 l^w 个结点, 其中 p_1^w 表示根结点, $p_{l^w}^w$ 表示词 w 对应的结点.
4. $d_2^w, d_3^w, \cdots, d_{l^w}^w \in \{0,1\}$: 词 w 的 Huffman 编码, 它由 $l^w - 1$ 位编码构成, d_j^w 表示路径 p^w 中第 j 个结点对应的编码 (根结点不对应编码).
5. $\theta_1^w, \theta_2^w, \cdots, \theta_{l^w-1}^w \in \mathbb{R}^m$: 路径 p^w 中**非叶子结点**对应的向量, θ_j^w 表示路径 p^w 中第 j 个非叶子结点对应的向量.

注 4.1 按理说, 我们要求的是词典 $\mathcal{D}$ 中每个词 (即 *Huffman* 树中所有叶子节点) 的向量, 为什么这里还要为 *Huffman* 树中每一个非叶子结点也定义一个同长的向量呢? 事实上, 它们只是算法中的辅助向量, 具体用途在下文中将会为大家解释清楚.

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

好了，引入了这么一大堆抽象的记号，接下来，我们还是通过一个简单的例子把它们落到实处吧，看图 11，仍以预备知识中例 2.1 为例，考虑词 $w =$ “足球”的情形.

图 11 中由 4 条红色边串起来的 5 个节点就构成路径 p^w，其长度 $l^w = 5$. $p_1^w, p_2^w, p_3^w, p_4^w, p_5^w$ 为路径 p^w 上的 5 个结点，其中 p_1^w 对应根结点. $d_2^w, d_3^w, d_4^w, d_5^w$ 分别为 1, 0, 0, 1，即“足球”的 Huffman 编码为 1001. 此外，$\theta_1^w, \theta_2^w, \theta_3^w, \theta_4^w$ 分别表示路径 p^w 上 4 个非叶子结点对应的向量.

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

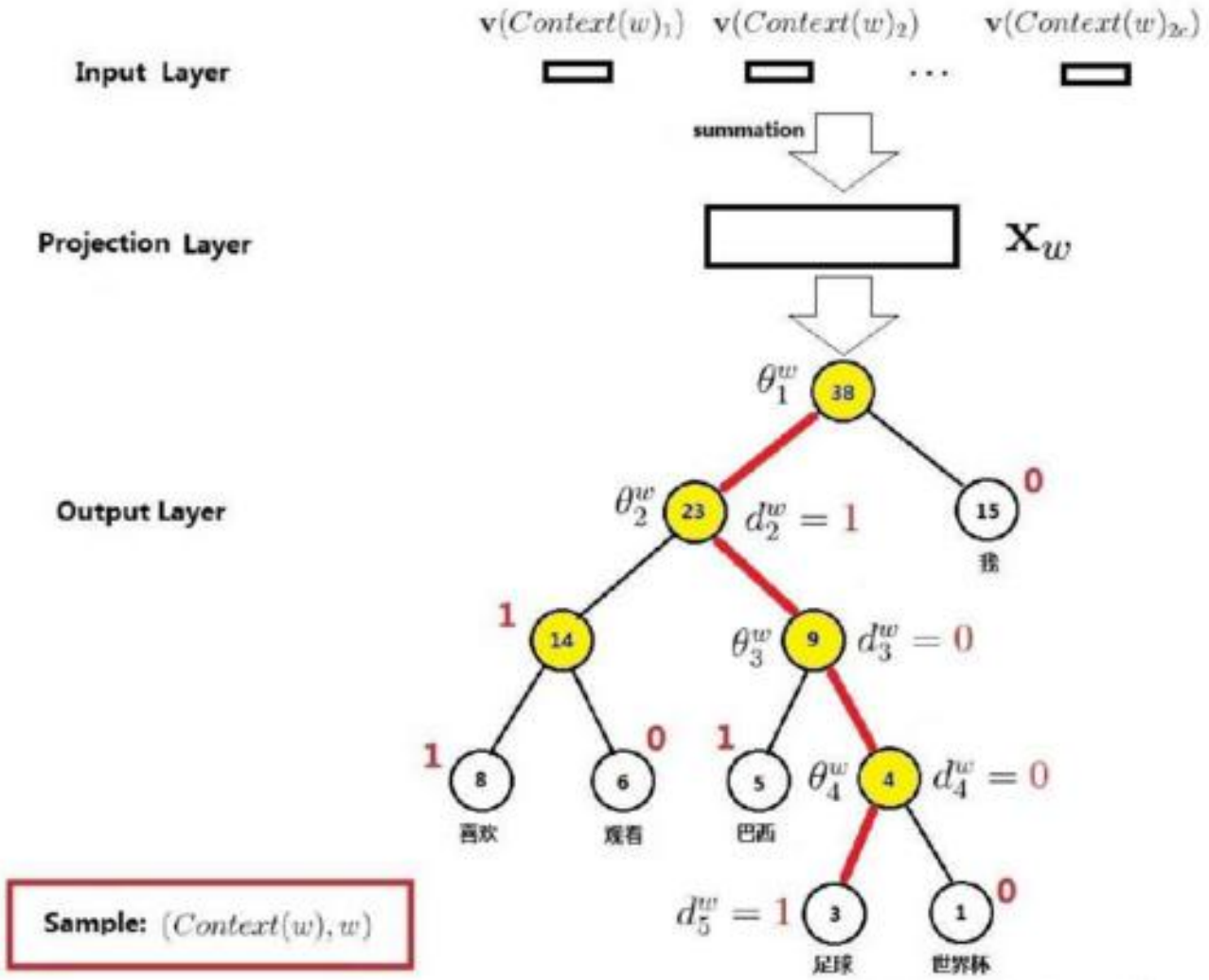


图 11 $w =$ “足球” 时的相关记号示意图

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

那么, 在如图 10 所示的网络结构下, 如何定义条件概率函数 $p(w|Contex(w))$ 呢? 更具体地说, 就是如何利用向量 $\mathbf{x}_w \in \mathbb{R}^m$ 以及 Huffman 树来定义函数 $p(w|Contex(w))$ 呢?

以图 11 中词 $w =$“足球”为例, 从根结点出发到达“足球”这个叶子节点, 中间共经历了 4 次分支 (每条红色的边对应一次分支), 而每一次分支都可视为进行了一次**二分类**.

既然是从二分类的角度来考虑问题, 那么对于每一个非叶子结点, 就需要为其左右孩子结点指定一个类别, 即哪个是正类 (标签为 1), 哪个是负类 (标签为 0). 碰巧, 除根结点以外, 树中每个结点都对应了一个取值为 0 或 1 的 Huffman 编码. 因此, 一种最自然的做法就是将 Huffman 编码为 1 的结点定义为正类, 编码为 0 的结点定义为负类. 当然, 这只是个约定而已, 你也可以将编码为 1 的结点定义为负类, 而将编码为 0 的结点定义为正类. 事实上, word2vec 选用的就是后者, 为方便读者对照着文档看源码, 下文中统一采用后者, 即约定

$$Label(p_i^w) = 1 - d_i^w, i = 2, 3, \cdots, l^w.$$

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

简言之就是，**将一个结点进行分类时，分到左边就是负类，分到右边就是正类.**

根据预备知识 §2.2 中介绍的逻辑回归，易知，一个结点被分为正类的概率是

$$\sigma(\mathbf{x}_w^\top\theta)=\frac{1}{1+e^{-\mathbf{x}_w^\top\theta}},$$

被分为负类的概率当然就等于

$$1-\sigma(\mathbf{x}_w^\top\theta),$$

注意，上式中有个叫 θ 的向量，它是待定参数，显然，在这里非叶子结点对应的那些向量 θ_i^w 就可以扮演参数 θ 的角色 (这也是为什么将它们取名为 θ_i^w 的原因).

对于从根结点出发到达“足球”这个叶子节点所经历的 4 次二分类，将每次分类结果的概率写出来就是

1. 第 1 次: $p(d_2^w|\mathbf{x}_w,\theta_1^w)=1-\sigma(\mathbf{x}_w^\top\theta_1^w)$;
2. 第 2 次: $p(d_3^w|\mathbf{x}_w,\theta_2^w)=\sigma(\mathbf{x}_w^\top\theta_2^w)$;
3. 第 3 次: $p(d_4^w|\mathbf{x}_w,\theta_3^w)=\sigma(\mathbf{x}_w^\top\theta_3^w))$;
4. 第 4 次: $p(d_5^w|\mathbf{x}_w,\theta_4^w)=1-\sigma(\mathbf{x}_w^\top\theta_4^w)$,

但是，我们要求的是 $p(\text{足球}|Contex(\text{足球}))$，它跟这 4 个概率值有什么关系呢? 关系就是

$$p(\text{足球}|Contex(\text{足球}))=\prod_{j=2}^{5}p(d_j^w|\mathbf{x}_w,\theta_{j-1}^w).$$

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

至此, 通过 $w =$ "足球" 的小例子, Hierarchical Softmax 的基本思想其实就已经介绍完了. 小结一下: **对于词典 $\mathcal{D}$ 中的任意词 w, Huffman 树中必存在一条从根结点到词 w 对应结点的路径 p^w (且这条路径是唯一的). 路径 p^w 上存在 $l^w - 1$ 个分支, 将每个分支看做一次二分类, 每一次分类就产生一个概率, 将这些概率乘起来, 就是所需的 $p(w|Context(w))$.**

条件概率 $p(w|Context(w))$ 的一般公式可写为

$$p(w|Context(w)) = \prod_{j=2}^{l^w} p(d_j^w|\mathbf{x}_w, \theta_{j-1}^w), \tag{4.3}$$

其中

$$p(d_j^w|\mathbf{x}_w, \theta_{j-1}^w) = \begin{cases} \sigma(\mathbf{x}_w^\top \theta_{j-1}^w), & d_j^w = 0; \\ 1 - \sigma(\mathbf{x}_w^\top \theta_{j-1}^w), & d_j^w = 1, \end{cases}$$

或者写成整体表达式

$$p(d_j^w|\mathbf{x}_w, \theta_{j-1}^w) = [\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{1-d_j^w} \cdot [1 - \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{d_j^w}.$$

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

注 4.2 在 §3.3 中，最后得到的条件概率为

$$p(w|Context(w)) = \frac{e^{y_{w,i_w}}}{\sum_{i=1}^{N} e^{y_{w,i}}}.$$

具体见 (3.6) 式，由于这里有个归一化操作，因此显然成立

$$\sum_{w\in\mathcal{D}} p(w|Context(w)) = 1.$$

然而，对于由 (4.3) 定义的概率，是否也能满足上式呢？这个问题留给读者思考.

将 (4.3) 代入对数似然函数 (4.1)，便得

$$\begin{aligned}\mathcal{L} &= \sum_{w\in\mathcal{C}} \log \prod_{j=2}^{l^w} \left\{ [\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{1-d_j^w} \cdot [1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{d_j^w} \right\} \\ &= \sum_{w\in\mathcal{C}} \sum_{j=2}^{l^w} \left\{ (1-d_j^w) \cdot \log[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] + d_j^w \cdot \log[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] \right\}, \end{aligned} \tag{4.4}$$

为下面梯度推导方便起见，将上式中双重求和符号下花括号里的内容简记为 $\mathcal{L}(w,j)$，即

$$\mathcal{L}(w,j) = (1-d_j^w) \cdot \log[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] + d_j^w \cdot \log[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]. \tag{4.5}$$

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

至此，已经推导出对数似然函数 (4.4)，这就是 CBOW 模型的目标函数，接下来讨论它的优化，即如何将这个函数最大化. word2vec 里面采用的是**随机梯度上升法***. 而梯度类算法的关键是给出相应的梯度计算公式，因此接下来重点讨论梯度的计算.

随机梯度上升法的做法是：每取一个样本 $(Context(w), w)$，就对目标函数中的所有（相关）参数做一次刷新. 观察目标函数 $\mathcal{L}$ 易知，该函数中的参数包括向量 $\mathbf{x}_w, \theta_{j-1}^w$，$w \in \mathcal{C}$，$j = 2, \cdots, l^w$. 为此，先给出函数 $\mathcal{L}(w, j)$ 关于这些向量的梯度.

首先考虑 $\mathcal{L}(w, j)$ 关于 θ_{j-1}^w 的梯度计算.

$$
\begin{aligned}
\frac{\partial \mathcal{L}(w,j)}{\partial \theta_{j-1}^w} &= \frac{\partial}{\partial \theta_{j-1}^w}\left\{(1-d_j^w)\cdot \log[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] + d_j^w \cdot \log[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]\right\} \\
&= (1-d_j^w)[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]\mathbf{x}_w - d_j^w \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)\mathbf{x}_w \quad \text{(利用 (2.1) 式)} \\
&= \left\{(1-d_j^w)[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] - d_j^w \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)\right\}\mathbf{x}_w \quad \text{(合并)} \\
&= \left[1 - d_j^w - \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)\right]\mathbf{x}_w.
\end{aligned}
$$

于是，θ_{j-1}^w 的更新公式可写为

$$\theta_{j-1}^w := \theta_{j-1}^w + \eta\left[1 - d_j^w - \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)\right]\mathbf{x}_w,$$

其中 η 表示**学习率**，下同.

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

接下来考虑 $\mathcal{L}(w,j)$ 关于 $\mathbf{x}_w$ 的梯度. 观察 (4.5) 可发现, $\mathcal{L}(w,j)$ 中关于变量 $\mathbf{x}_w$ 和 θ_{j-1}^w 是**对称**的 (即两者可交换位置), 因此, 相应的梯度 $\frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}$ 也只需在 $\frac{\partial \mathcal{L}(w,j)}{\partial \theta_{j-1}^w}$ 的基础上对这两个向量交换位置就可以了, 即

$$\frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w} = \left[1 - d_j^w - \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)\right] \theta_{j-1}^w.$$

到这里, 细心的读者可能已经看出问题来了: 我们的最终目的是要求词典 $\mathcal{D}$ 中每个词的词向量, 而这里的 $\mathbf{x}_w$ 表示的是 $Context(w)$ 中各词词向量的累加. 那么, 如何利用 $\frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}$ 来对 $\mathbf{v}(\widetilde{w}), \widetilde{w} \in Context(w)$ 进行更新呢? word2vec 中的做法很简单, 直接取

$$\mathbf{v}(\widetilde{w}) := \mathbf{v}(\widetilde{w}) + \eta \sum_{j=2}^{l^w} \frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}, \quad \widetilde{w} \in Context(w),$$

*求最小值用 (随机) 梯度下降法, 求最大值用 (随机) 梯度上升法, 这种基于梯度的方法通常统称为 (随机) 梯度下降法. 这里强调“上升”是想提醒大家这是在求最大值.

即把 $\sum_{j=2}^{l^w} \frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}$ 贡献到 $Context(w)$ 中每一个词的词向量上. 这个应该很好理解, 既然 $\mathbf{x}_w$ 本身就是 $Context(w)$ 中各词词向量的累加, 求完梯度后当然也应该将其贡献到每个分量上去.

注 4.3 当然, 读者这里需要考虑的是: 采用**平均**贡献会不会更合理？即使用公式

$$\mathbf{v}(\widetilde{w}) := \mathbf{v}(\widetilde{w}) + \frac{\eta}{|Context(w)|} \sum_{j=2}^{l^w} \frac{\partial \mathcal{L}(w,j)}{\partial \mathbf{x}_w}, \quad \widetilde{w} \in Context(w),$$

其中 $|Context(w)|$ 表示 $Context(w)$ 中词的个数.

NLP基础_词向量_Word2Vec_CBOW

- CBOW扩展：

下面以样本 $(Context(w), w)$ 为例, 给出 CBOW 模型中采用随机梯度上升法更新各参数的伪代码.

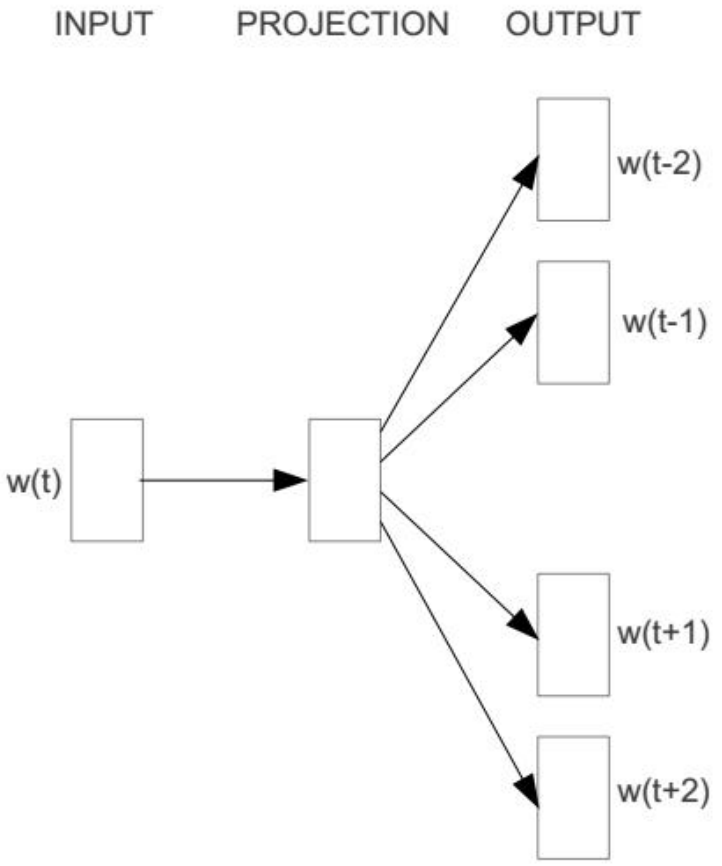
1. $\mathbf{e} = 0$.
2. $\mathbf{x}_w = \sum_{u \in Context(w)} \mathbf{v}(u)$.
3. **FOR** $j = 2 : l^w$ **DO**
   {
   3.1 $q = \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)$
   3.2 $g = \eta(1 - d_j^w - q)$
   3.3 $\mathbf{e} := \mathbf{e} + g\theta_{j-1}^w$
   3.4 $\theta_{j-1}^w := \theta_{j-1}^w + g\mathbf{x}_w$
   }
4. **FOR** $u \in Context(w)$ **DO**
   {
   $\mathbf{v}(u) := \mathbf{v}(u) + \mathbf{e}$
   }

注意, 步 3.3 和步 3.4 不能交换次序, 即 θ_{j-1}^w 应等贡献到 $\mathbf{e}$ 后再做更新.

注 4.4 结合上面的伪代码, 简单给出其与 *word2vec* 源码中的对应关系如下: *syn0* 对应 $\mathbf{v}(\cdot)$, *syn1* 对应 θ_{j-1}^w, *neu1* 对应 $\mathbf{x}_w$, *neu1e* 对应 $\mathbf{e}$.

NLP基础_词向量_Word2Vec_Skip-gram

- Skip-gram
  - 给定中心词预测上下文各个单词的概率分布，例如，给定jumped，预测上下文单词｛The，cat，()，over，the，puddle｝的概率，模型的结构如下:
  - 比如：”一只猫坐在地上”，中间词为”猫”，那么上下文词为: [“一只”, “坐在”, “地上”]



Skip-gram

NLP基础_词向量_Word2Vec_Skip-gram

- 前向过程：

$$v_c = Uw_c$$

词向量转换
Embedding Layer

$$z_{2m} = V_{2m} v_c$$

负采样

$$\hat{y}_{2m} = soft\max(z_{2m})$$

- 损失函数：

$$y_t = one\_hot(w_t)$$

$$loss = \sum_{j=0, j\neq m}^{2m} -y_{c-m+j} \log(\hat{y}_{c-m+j})$$

NLP基础_词向量_Word2Vec_Skip-gram

$$
\begin{aligned}
\text{minimize } J &= -\log P(w_{c-m},\ldots,w_{c-1},w_{c+1},\ldots,w_{c+m}|w_c) \\
&= -\log \prod_{j=0,j\neq m}^{2m} P(w_{c-m+j}|w_c) \\
&= -\log \prod_{j=0,j\neq m}^{2m} P(u_{c-m+j}|v_c) \\
&= -\log \prod_{j=0,j\neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|}\exp(u_k^T v_c)} \\
&= -\sum_{j=0,j\neq m}^{2m} u_{c-m+j}^T v_c + 2m\log\sum_{k=1}^{|V|}\exp(u_k^T v_c)
\end{aligned}
$$

NLP基础_词向量_Word2Vec_Skip-gram

- Skip-gram扩展：

1. **输入层**: 只含当前样本的中心词 w 的词向量 $\mathbf{v}(w) \in \mathbb{R}^m$.

2. **投影层**: 这是个恒等投影, 把 $\mathbf{v}(w)$ 投影到 $\mathbf{v}(w)$. 因此, **这个投影层其实是多余的**, 这里之所以保留投影层主要是方便和 CBOW 模型的网络结构做对比.

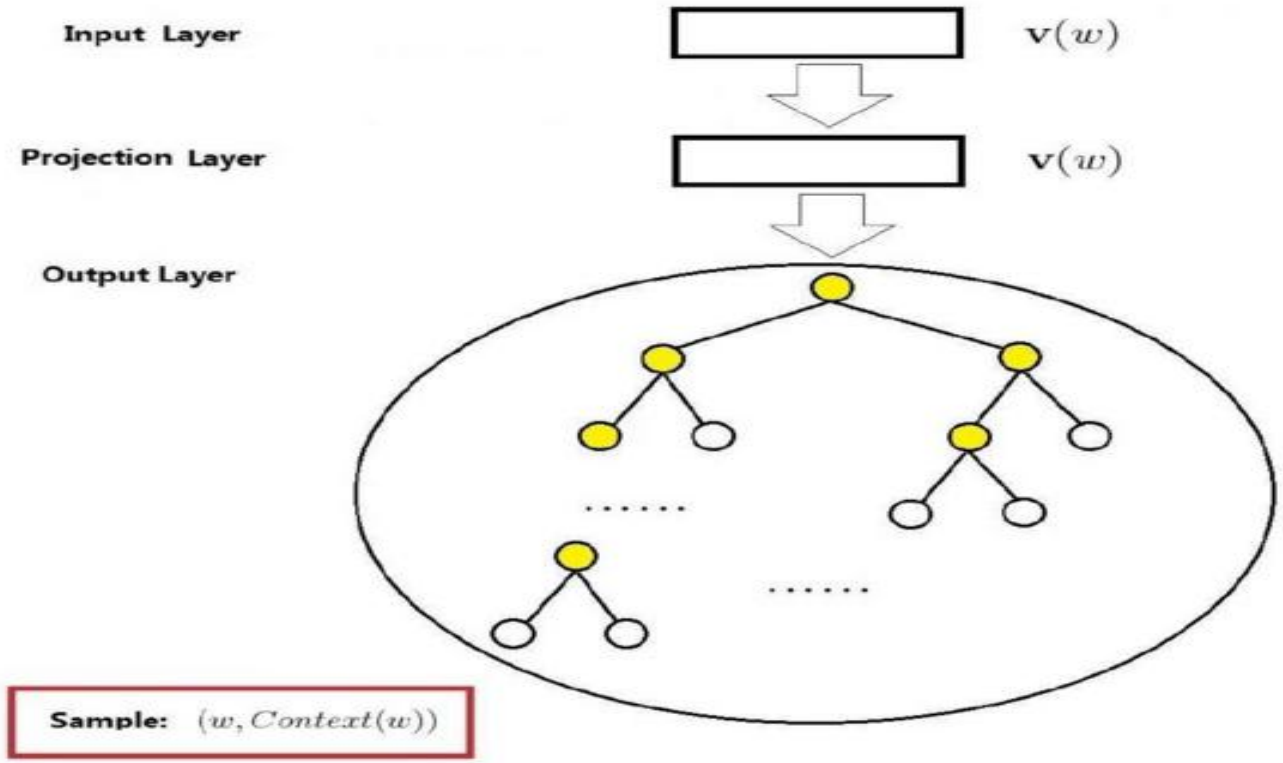


图 12 Skip-gram 模型的网络结构示意图

NLP基础_词向量_Word2Vec_Skip-gram

- Skip-gram扩展：

3. **输出层**: 和 CBOW 模型一样, 输出层也是一棵 Huffman 树.

§4.2.2 梯度计算

对于 Skip-gram 模型, 已知的是当前词 w, 需要对其上下文 $Context(w)$ 中的词进行预测, 因此目标函数应该形如 (4.2), 且关键是条件概率函数 $p(Context(w)|w)$ 的构造, Skip-gram 模型中将其定义为

$$p(Context(w)|w) = \prod_{u \in Context(w)} p(u|w),$$

上式中的 $p(u|w)$ 可按照上小节介绍的 Hierarchical Softmax 思想, 类似于 (4.3) 地写为

$$p(u|w) = \prod_{j=2}^{l^u} p(d_j^u|\mathbf{v}(w), \theta_{j-1}^u),$$

其中

$$p(d_j^u|\mathbf{v}(w), \theta_{j-1}^u) = [\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{1-d_j^u} \cdot [1 - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{d_j^u}. \tag{4.6}$$

NLP基础_词向量_Word2Vec_Skip-gram

- Skip-gram扩展：

将 (4.6) 依次代回, 可得对数似然函数 (4.2) 的具体表达式

$$\begin{aligned}\mathcal{L} &= \sum_{w\in\mathcal{C}} \log \prod_{u\in Context(w)} \prod_{j=2}^{l^u} \{[\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{1-d_j^u} \cdot [1-\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{d_j^u}\} \\ &= \sum_{w\in\mathcal{C}} \sum_{u\in Context(w)} \sum_{j=2}^{l^u} \{(1-d_j^u)\cdot \log[\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] + d_j^u \cdot \log[1-\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]\}.\end{aligned} \tag{4.7}$$

同样, 为下面梯度推导方便起见, 将三重求和符号下花括号里的内容简记为 $\mathcal{L}(w,u,j)$, 即

$$\mathcal{L}(w,u,j) = (1-d_j^u)\cdot \log[\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] + d_j^u \cdot \log[1-\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)].$$

至此, 已经推导出对数似然函数的表达式 (4.7), 这就是 Skip-gram 模型的目标函数. 接下来同样利用**随机梯度上升法**对其进行优化, 关键是要给出两类梯度.

首先考虑 $\mathcal{L}(w,u,j)$ 关于 θ_{j-1}^u 的梯度计算 (与 CBOW 模型对应部分的推导完全类似).

$$\begin{aligned}\frac{\partial \mathcal{L}(w,u,j)}{\partial \theta_{j-1}^u} &= \frac{\partial}{\partial \theta_{j-1}^u}\{(1-d_j^u)\cdot \log[\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] + d_j^u \cdot \log[1-\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]\} \\ &= (1-d_j^u)[1-\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]\mathbf{v}(w) - d_j^u \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)\mathbf{v}(w) \quad \text{(利用 (2.1) 式)} \\ &= \{(1-d_j^u)[1-\sigma(\mathbf{v}(w)^\top \mathbf{v}_{j-1}^u)] - d_j^u \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)\}\mathbf{v}(w) \quad \text{(合并)} \\ &= [1-d_j^u - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]\mathbf{v}(w).\end{aligned}$$

NLP基础_词向量_Word2Vec_Skip-gram

- Skip-gram扩展：

于是, θ_{j-1}^u 的更新公式可写为

$$\theta_{j-1}^u := \theta_{j-1}^u + \eta \left[1 - d_j^u - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)\right] \mathbf{v}(w).$$

接下来考虑 $\mathcal{L}(w, u, j)$ 关于 $\mathbf{v}(w)$ 的梯度. 同样利用 $\mathcal{L}(w, u, j)$ 中 $\mathbf{v}(w)$ 和 θ_{j-1}^w 的**对称性**, 有

$$\frac{\partial \mathcal{L}(w, u, j)}{\partial \mathbf{v}(w)} = \left[1 - d_j^u - \sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)\right] \theta_{j-1}^u.$$

于是, $\mathbf{v}(w)$ 的更新公式可写为

$$\mathbf{v}(w) := \mathbf{v}(w) + \eta \sum_{u \in Context(w)} \sum_{j=2}^{l^u} \frac{\partial \mathcal{L}(w, u, j)}{\partial \mathbf{v}(w)}.$$

下面以样本 $(w, Context(w))$ 为例, 给出 Skip-gram 模型中采用随机梯度上升法更新各参数的伪代码.

NLP基础_词向量_Word2Vec_Skip-gram

- Skip-gram扩展：

$\mathbf{e} = \mathbf{0}$

FOR $u \in Context(w)$ **DO**

{

  FOR $j = 2 : l^u$ **DO**

  {

    1. $q = \sigma(\mathbf{v}(w)^\top \theta^u_{j-1})$
    2. $g = \eta(1 - d^u_j - q)$
    3. $\mathbf{e} := \mathbf{e} + g\theta^u_{j-1}$
    4. $\theta^u_{j-1} := \theta^u_{j-1} + g\mathbf{v}(w)$

  }

}

$\mathbf{v}(w) := \mathbf{v}(w) + \mathbf{e}$

但是, word2vec 源码中, 并不是等 $Context(w)$ 中的所有词都处理完后才刷新 $\mathbf{v}(w)$, 而是, 每处理完 $Context(w)$ 中的一个词 u, 就及时刷新一次 $\mathbf{v}(w)$, 具体为

NLP基础_词向量_Word2Vec_Skip-gram

- Skip-gram扩展：

FOR $u \in Context(w)$ DO
{
  $\mathbf{e} = 0$
  FOR $j = 2 : l^u$ DO
  {
    1. $q = \sigma(\mathbf{v}(w)^\top \theta^u_{j-1})$
    2. $g = \eta(1 - d^u_j - q)$
    3. $\mathbf{e} := \mathbf{e} + g\theta^u_{j-1}$
    4. $\theta^u_{j-1} := \theta^u_{j-1} + g\mathbf{v}(w)$
  }
  $\mathbf{v}(w) := \mathbf{v}(w) + \mathbf{e}$
}

同样, 需要注意的是, 循环体内的步 3 和步 4 不能交换次序, 即 θ^u_{j-1} 要等贡献到 e 后才更新.

注 4.5 结合上面的伪代码, 简单给出其与 *word2vec* 源码中的对应关系如下: *syn0* 对应 $\mathbf{v}(\cdot)$, *syn1* 对应 θ^u_{j-1}, *neu1e* 对应 e.

NLP基础_词向量_Word2Vec

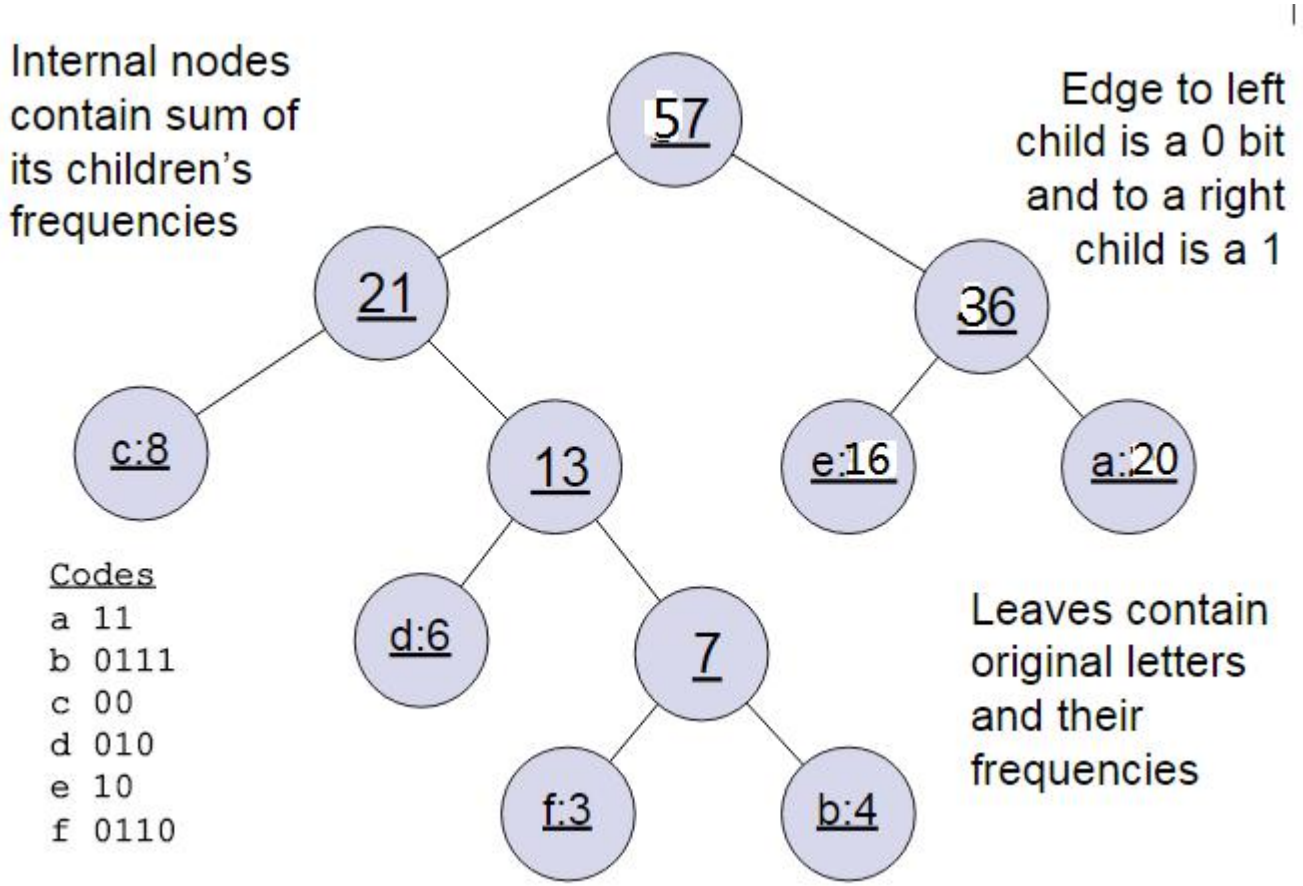
- 在传统的Word2Vec的结构中，在计算过程中需要计算所有单词(2m)和中心词之间的概率，比较耗时，并且需要判断最终预测的具体是N个单词中的那一个单词(N为词汇表大小)，所以在Word2Vec的论文(Efficient Estimation of Word Representations in Vector Space)中，作者Mikolov提出**Hierarchical softmax(哈夫曼树)**和**Negative sampling(负样本采样)**两种方法对Word2Vec的模型训练进行优化。这也是Word2Vec能够大量真正应用于NLP领域的主要原因(不需要依赖神经网络训练模型)。

NLP基础_词向量_Word2Vec

- CBOW和Skip-Gram的训练可以使用霍夫曼树加速，霍夫曼树的构建如下所示：
  - 输入： 权重值为(w1, w2, ..., wn)为n个节点
  - 输出：对应的霍夫曼树
  - 步骤：
    - 将(w1, w2, ..., wn)看成有n棵树的森林，每个树仅有一个节点；
    - 在森林中选择根节点权重最小的两棵树进行合并，得到一个新的树，这两颗树分布作为新树的左右子树，新树的根节点权重为左右子树的根节点权重之和。
    - 将之前的根节点权重最小的两棵树从森林删除，并将新树加入森林；
    - 重复步骤2和步骤3直到森林只有一个树为止。

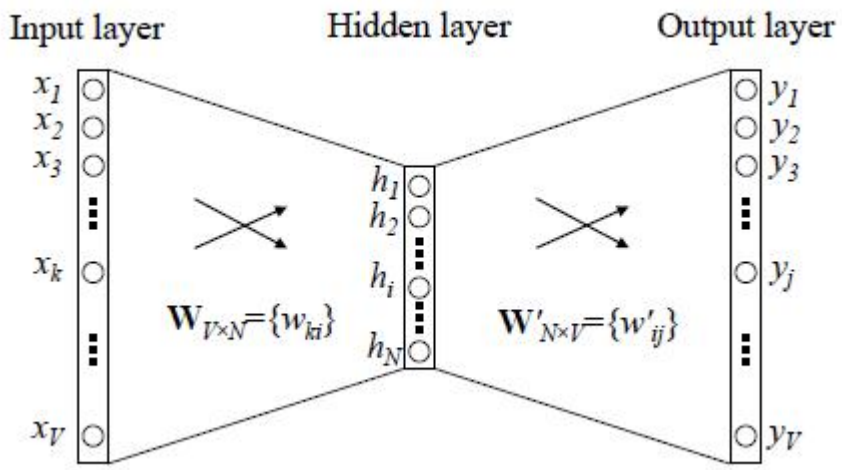
NLP基础_词向量_Word2Vec

- 六个节点: (a, b, c, d, e, f); 节点权重为: (20, 4, 8, 6, 16, 3)



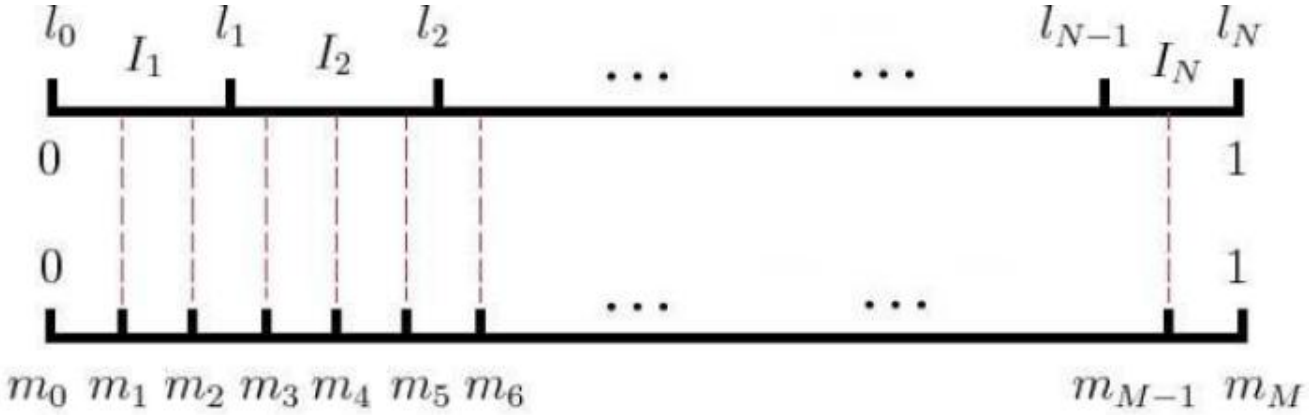
NLP基础_词向量_Word2Vec_Hierarchical Softmax

- 输入层到隐藏层之间的映射，没有采用神经网络的线性转换加激活函数的方式，而且采用简单的均值的方式来实现；
- 从隐藏层到输出的softmax层之间的计算量采用霍夫曼树进行了改进。
- 参考：https://www.cnblogs.com/pinard/p/7243513.html



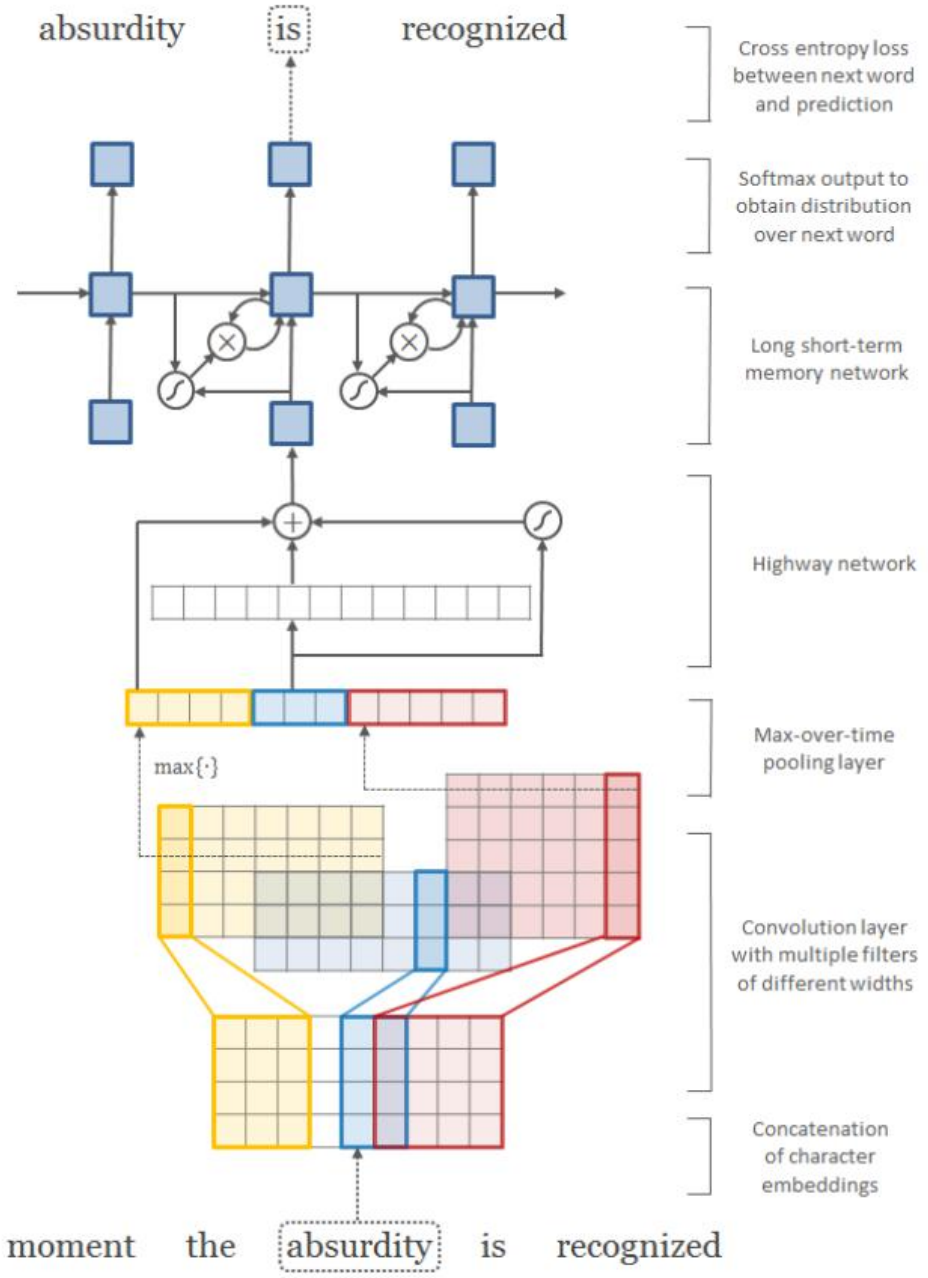
NLP基础_词向量_Word2Vec_Negative Sampling

- Hierarchical Softmax有一个非常大的缺点，就是针对很生僻的词，在霍夫曼树中，这个词所在的位置非常低，也就是路径非常长；
- Negative Sampling(负采样)将确实存在的数据(实际值)当做正样本，然后将不存在的数据当做负样本，然后进行分类。
- 参考：https://www.cnblogs.com/pinard/p/7249903.html



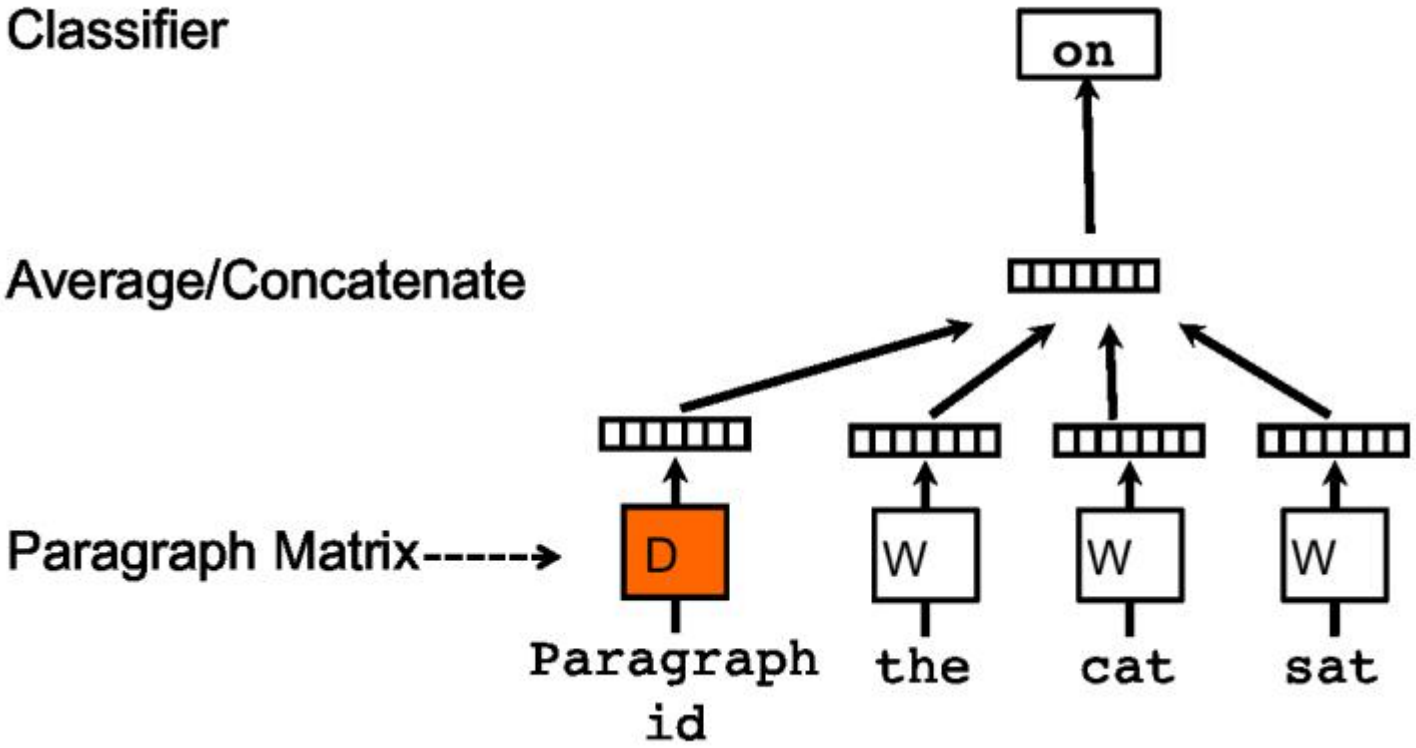
NLP基础_词向量_Char2Vec

- 和Word2Vec不一样，不进行分词，直接将字符转换为字向量，后续的模型基于字向量进行模型的构建。
- 相比于Word2Vec来讲，Char2Vec直接应用于字符集，对拼写更加宽容。现阶段NLP中应用非常多的一种向量方式。



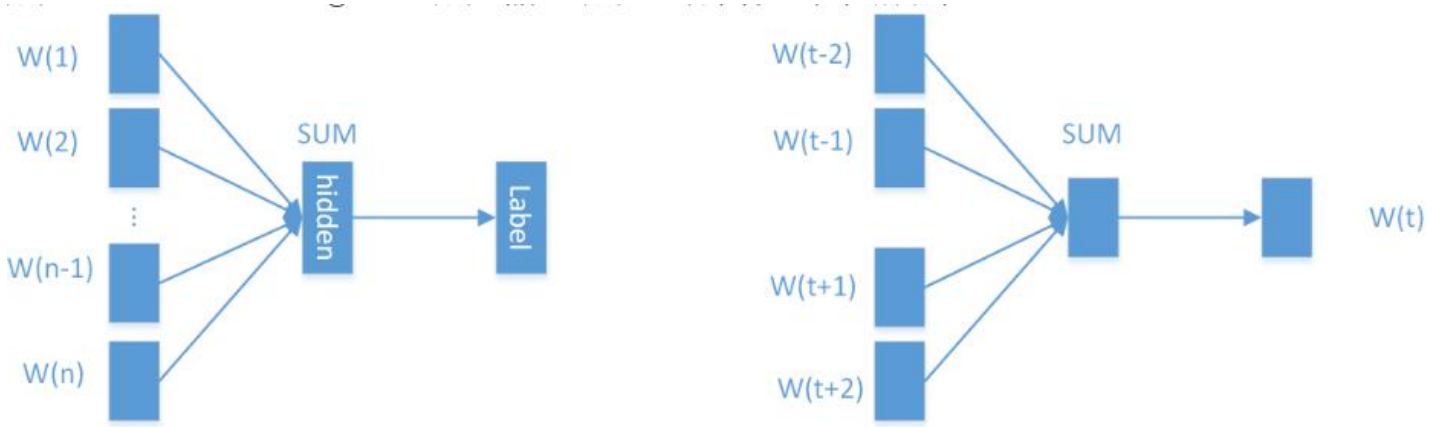
NLP基础_词向量_Doc2Vec

- Doc2Vec使用Word2Vec作为第一步输入，然后利用Word2Vec的单词向量对每个句子或者段落生成复合向量。



NLP基础_词向量_FastText

- FastText是一个文本分类和词向量训练工具，最大的特点就是模型简单，只有一层隐层和输出层；结构基本的CBOW类似，主要区别在于：
  - CBOW中输出的是词的置信度，FastText输出的是类别label；
  - CBOW中输入是当前窗口除中心词之外的所有词，而FastText输入的是文章中的所有词；



NLP基础_词向量_FastText

- 在词向量的训练过程中，增加了**subwords**特性，其实就是一个词的character-level的n-gram，比如单词”hello”，长度至少为3的character-level的ngram的'hel', 'ell', 'llo', 'hell', 'ello'以及'hello'，每个ngram都可以使用一个dense的向量zg表示，故最终一个单词可以表示为:

$$V_{hello} = \sum_{g \in \phi} z_g^T v_c$$

NLP基础_词向量_FastText

$$h = \frac{1}{n}\sum_{i=1}^{n} w_i$$

$$z = sigmoid(W_o h)$$

Negative Sampling

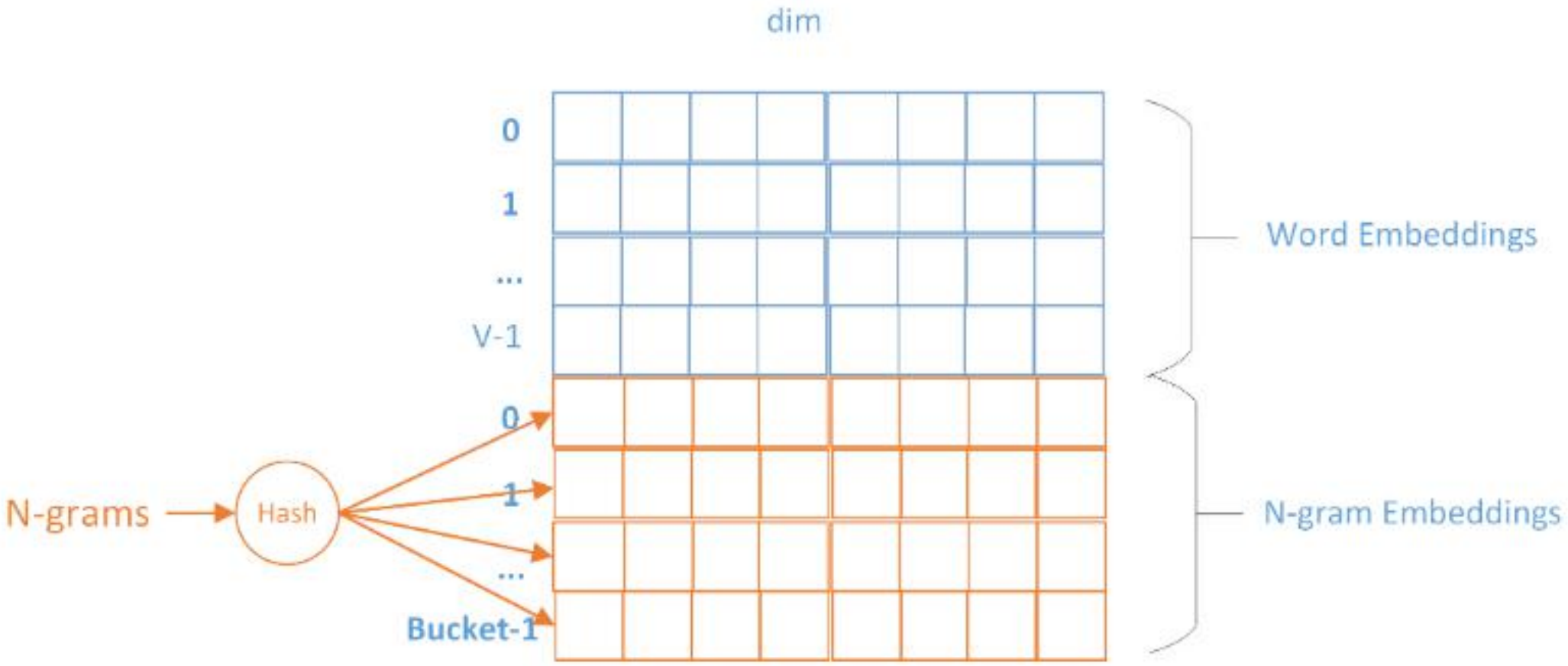
$$loss = -\frac{1}{M}\sum_{i=1}^{m}\left(log\sigma(u_o^T h_i) + \sum_{j\sim P(w)}[log\sigma(-u_j^T h_i)]\right)$$

NLP基础_词向量_FastText

- FastText在分类中也增加了词级别N-gram的特征，主要是为了通过增加N-Gram的特征信息来保留词序信息(因为隐层是通过简单的求和平均得到的)，比如：某篇文档有3个词，w1、w2、w3，N-gram取N为2，w1、w2、w3以及bigram w12、w23是新的embedding向量，那么文章的隐层表示为：

$$h = \frac{1}{5}(w_1 + w_2 + w_3 + w_{12} + w_{23})$$

NLP基础_词向量_FastText



NLP基础_词向量_cw2vec

- cw2vec(Learning Chinese Word Embeddings with Stroke n-gram Information)
  - CCKS2018阿里健康团队中文电子病历命名实体识别评测任务冠军
  - 思想：**类似FastText的思想，利用中文汉字的笔画信息，使用N-Gram的的方式来提取中文汉字对应的高阶特征信息。**
  - http://www.statnlp.org/wp-content/uploads/papers/2018/cw2vec/cw2vec.pdf

NLP基础_词向量_cw2vec

字符为单位

部首为单位

NLP基础_词向量_cw2vec

- 可以看到从偏旁部首或者字件来提取词语的信息可以改进基于汉字的词语信息的提取效果，但是在某些汉字中，偏旁的设计仅只是为了方便汉字的查询，而不是为了表达汉字的语义信息，所以在cw2vec中提出了一种基于笔画的特征信息提取。

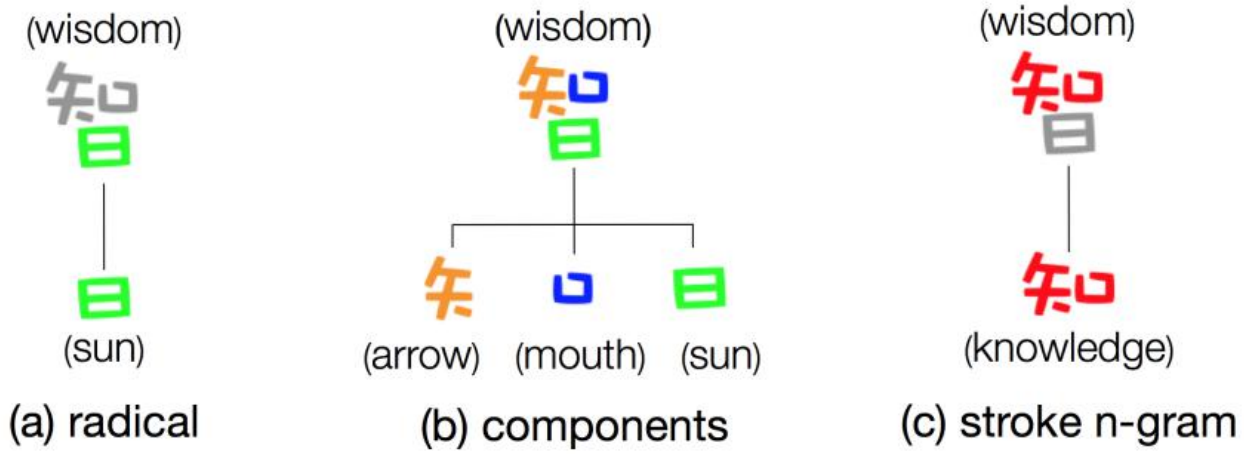


Figure 1: Radical v.s. components v.s. stroke n-gram

NLP基础_词向量_cw2vec

- 1. 将词语分割成字符；
- 2. 提取每个字符的笔画信息，然后将所有字符的笔画信息组合到一起；
- 3. 查表得到每个笔画对应的ID，组成这个词语对应的ID列表；
- 4. 产生N-Gram笔画特征。

Stroke Name	Horizontal	Vertical	Left-falling	Right-falling	Turning
Shape, ID	一 (㇀), 1	丨(亅), 2	丿, 3	乀 (丶), 4	乛(㇆), 5

Figure 3: General shapes of Chinese strokes.

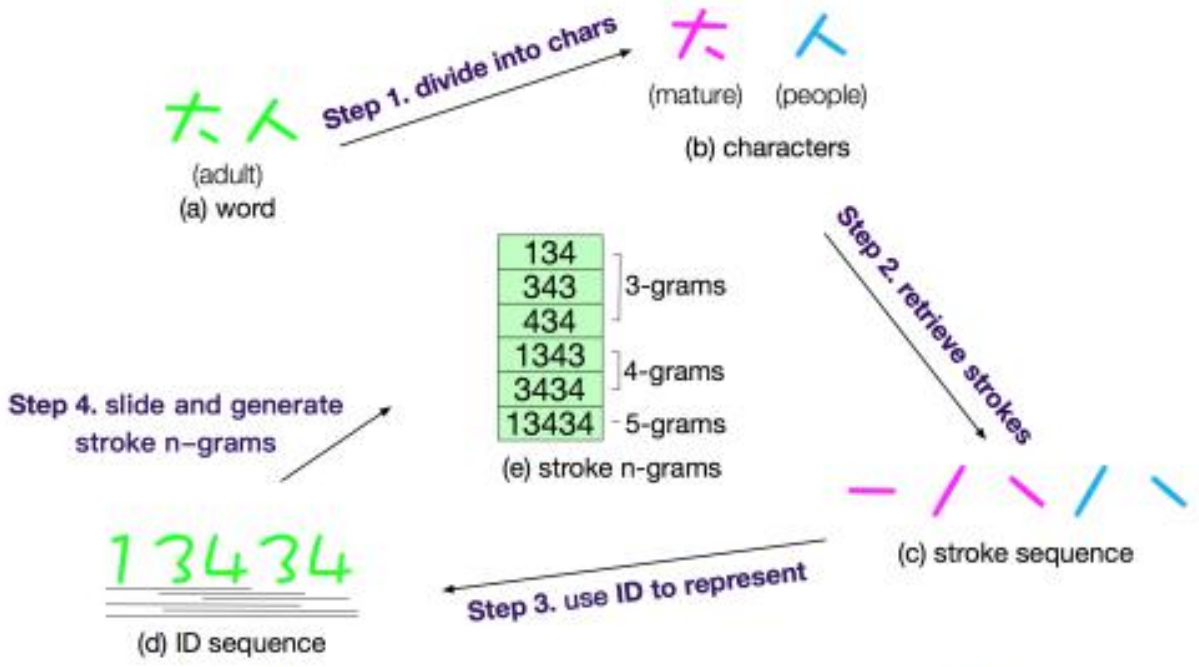


Figure 4: An illustrative example to show the procedures of the generation of stroke n-grams from a word.

NLP基础_词向量_cw2vec

- cw2vec使用和Word2Vec中的Skip-Gram的基础上进行模型训练，仅仅是将词语替换为词语的n-gram笔画特征信息来进行模型训练。
  - 短语：治理 雾霾 刻不容缓
  - 中心词：雾霾
  - 上下文词：治理 刻不容缓

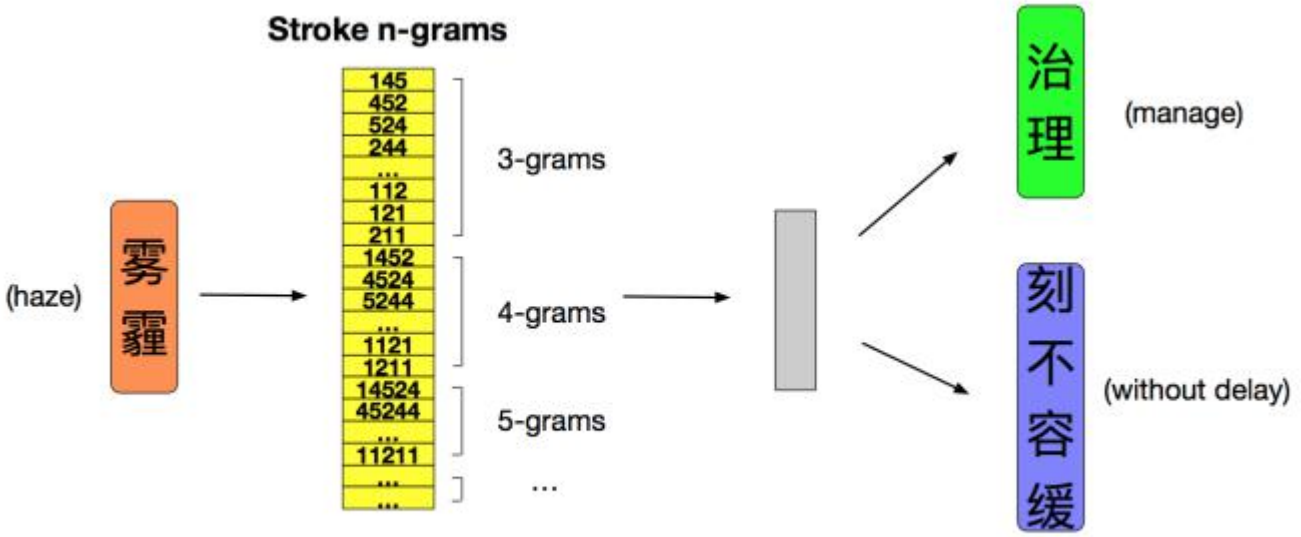


Figure 2: The overall architecture of our approach.

NLP基础_词向量_cw2vec

- 模型损失函数如下：

$$sim(w,c) = \sum_{q \in S(w)} \vec{q} \cdot \vec{c}$$

$$\mathcal{L} = \sum_{w \in D} \sum_{c \in T(w)} \log \sigma(sim(w,c)) + \lambda \mathbb{E}_{c' \sim P}[\log \sigma(-sim(w,c'))]$$

NLP基础_词向量_cw2vec

- 模型损失函数如下：

$$\mathcal{L} = \sum_{w \in D} \sum_{c \in T(w)} log\sigma(sim(w,c)) + \sum_{i=1}^{\lambda} \mathbb{E}_{c' \sim P(D)} [\![log\sigma(-sim(w,c'))]\!]$$

其中，W和C分别为当前词语和上下文词语，σ是sigmoid函数，T(w)是当前词语划窗内的所有词语集合，D是训练语料的全部文本。为了避免传统softmax带来的巨大计算量，这篇论文也采用了负采样的方式。C'为随机选取的词语，称为“负样例”，λ是负样例的个数，而$\mathbb{E}_{c' \sim P(D)} [\![\cdot]\!]$则表示负样例C'按照词频分布进行的采样，其中语料中出现次数越多的词语越容易被采样到。相似性sim(·,·)函数被按照如下构造：

$$sim(w,c) = \sum_{q \in S(w)} \vec{q} \cdot \vec{c}$$

其中，$\vec{q}$为当前词语对应的一个n元笔画向量，而$\vec{c}$是其对应的上下文词语的词向量。这项技术将当前词语拆解为其对应的n元笔画，但保留每一个上下文词语不进行拆解。S(w)为词语w所对应的n元笔画的集合。在算法执行前，这项研究先扫描每一个词语，生成n元笔画集合，针对每一个n元笔画，都有对应的一个n元笔画向量，在算法开始之前做随机初始化，其向量维度和词向量的维度相同。

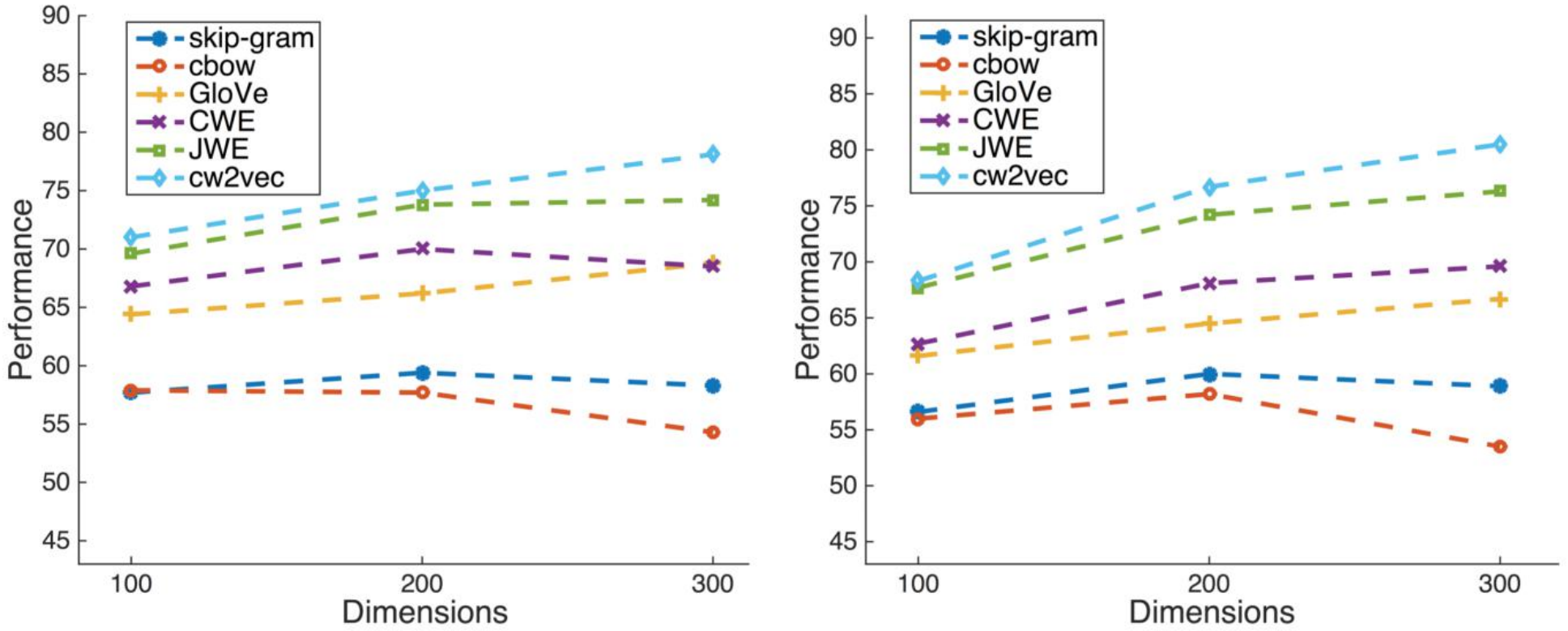
NLP基础_词向量_cw2vec

Model	Word Similarity		Word Analogy		Text Classification	Named Entity Recognition
	wordsim-240	wordsim-296	3CosAdd	3CosMul		
skip-gram (Mikolov et al. 2013b)	44.2	44.4	58.3	58.9	93.4	65.1
cbow (Mikolov et al. 2013b)	47.0	50.2	54.3	53.5	93.4	59.6
GloVe (Pennington, Socher, and Manning 2014)	45.2	44.3	68.8	66.7	94.2	66.0
CWE (Chen et al. 2015)	50.0	51.5	68.5	69.6	93.2	65.8
GWE (Su and Lee 2017)	50.0	49.1	50.8	50.6	94.3	65.5
JWE (Xin and Song 2017)	48.0	**52.7**	74.2	76.3	94.2	67.9
cw2vec (stroke n-grams)	**50.4**	**52.7**	**78.1**	**80.5**	**95.3**	**71.7**

Table 1: Performance on word similarity, word analogy task, text classification and named entity recognition. The embeddings are set as 300 dimensions. The evaluation metric is $\rho \times 100$ for word similarity, accuracy percentage for word analogy and text classification, $F1$-measure for named entity recognition task.

wordsim-240/wordsim-296：词相似度数据集
3CosAdd/3CosMul：词对比任务数据集
Glove：2014年斯坦福发表的词向量生成方式
CWE：2015清华大学中文汉字词向量生成方式
GWE：2017年台湾大学在CWE的基础上汉字词向量生成方式
JWE：2017年香港科技大学基于CBOW的汉字词向量生成方式

NLP基础_词向量_cw2vec



(a) 3CosAdd

(b) 3CosMul

Figure 5: Performance on word analogy over dimensions

NLP基础_词向量_cw2vec

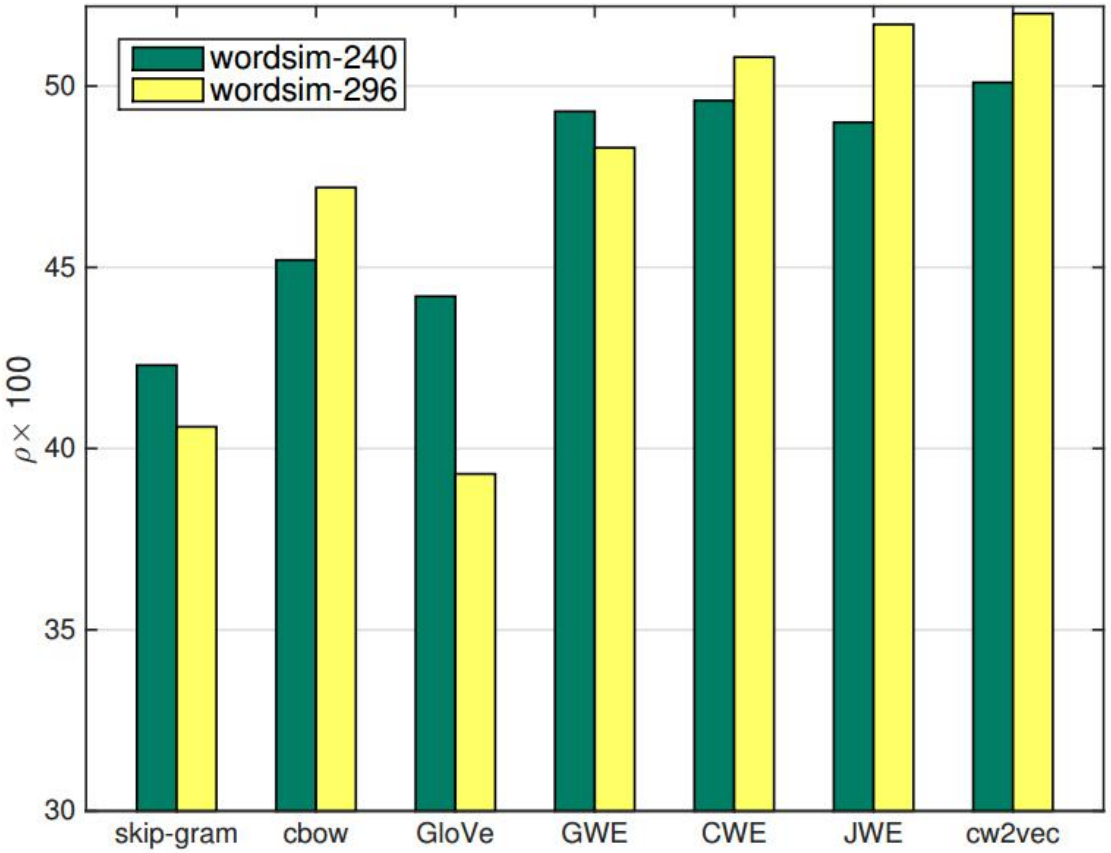


Figure 6: Performance on word similarity, trained on the front 20% wikipedia articles. The embeddings are set as 100 dimensions.

cw2vec实现参考：https://github.com/zhang2010hao/cw2vec-pytorch