**词向量**

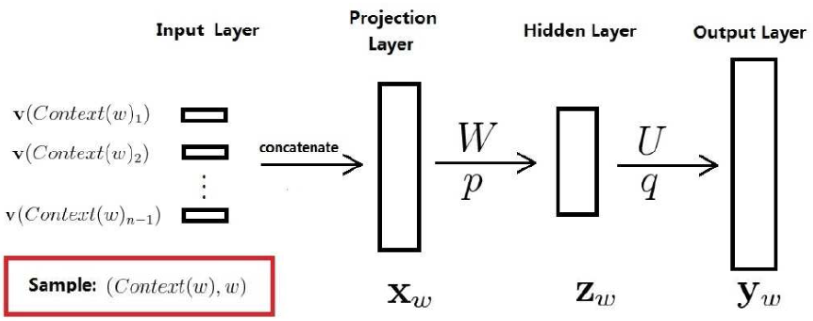
词向量的思想是：对于词典D中任意的一个词w，将其映射成一个固定维度的实值向量。而且这些词向量隐含地表征了词与词之间的一些关联；比如意思相近的词在空间中的夹角会比较小，意思相反的词则夹角就比较大。

“词向量”的应用场景有很多，比如：1. 在深度学习模型的embedding层使用“预训练”好的词向量，通常都会在很大程度上提高模型的性能。2. 在k-mean聚类的时候使用词向量，可以获得非常大的提升。

1. **Bengio (2003年)提出的“神经概率语言模型”做词嵌入：**

首先介绍一篇2003年的论文：a neural probabilistic language model，它介绍了一种用神经网络语言模型的方法做词嵌入。

整个网络分为4层，分别为输入层，投影层，隐藏层，输出层。模型的思想是，用前面(n-1)个词作为输入，第n个词作为输出，来构建训练集；也即对于词w构建的sample为，其中表示w前面的(n-1)个词。



* **输入层：**对于词w，取前面的(n-1)个词作为输入。
* **投影层：**每个词映射成k维向量，然后将这(n-1)个k维向量按顺序拼接，形成一个(n-1)k维的长向量。（要注意，词向量一开始是随机的，最后经训练确定，属于模型参数）
* **隐藏层：**隐藏层的节点根据需要可以进行调节
* **输出层：**N维向量（N是词典的大小），表示预测为每个词的概率

前向传播的表达式为：

**隐含层：**

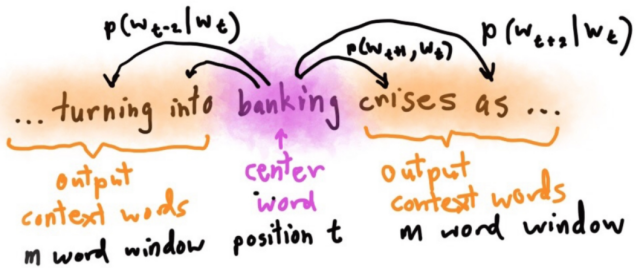
**输出层：**

其中，是(n-1)k维的输入向量；是N维的输出概率向量，表示由前面(n-1)个词，预测后面一个词的概率，（属于多分类问题）

注意：这个网络的参数是由两部分组成的

1. 神经网络的连接权重
2. 词向量：每个词一开始都是随机初始化为一个k维向量，最后再经训练确定（传统的机器学习算法输入是已知的，这里是需要训练得到），因此“词向量”可以看作神经网络语言模型训练完之后产生的副产品。
3. **CS224N讲解的原始ski-gram模型**

skip-gram是根据中间词w来预测上下文context(w)的概率；



模型的参数为矩阵U和矩阵V，分别表示，每个单词作为“中间词”和作为“周围词”的词向量。也就是一个单词用两个向量表示，最终输出的词向量可以把这个词向量相加。

优化的目标是，在语料库T中，每个单词都尽可能地预测出周围的单词（也即前后各m个单词），因此目标函数为：；

概率使用softmax归一化表示，其中u,v表示各个单词对应的词向量，“点乘”表示两个词的相似度（使用向量的“点乘”能粗糙地表示相似度，因为余弦相似度的分子就是计算呢“点乘”）

为了方便求导，将目标函数取反并取对数，得到损失函数：

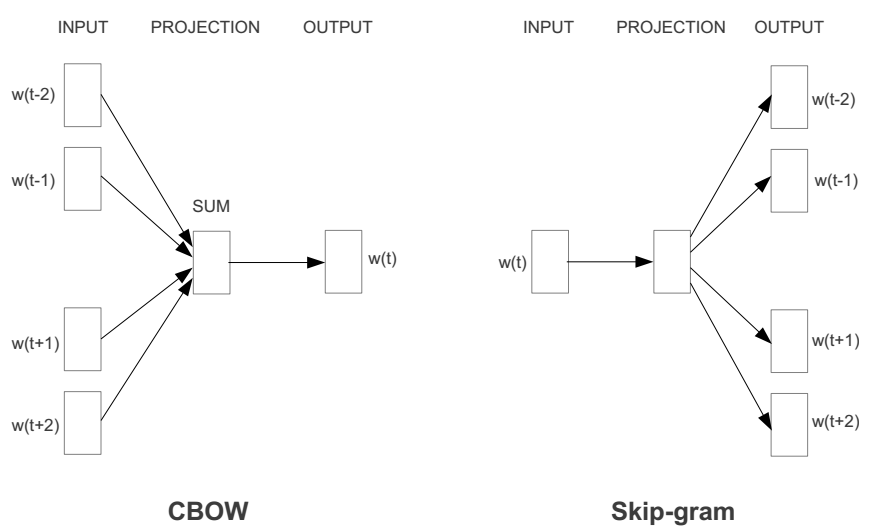


采用SGD的方法进行优化，损失函数为；同时优化U和V，使得损失函数最小，便得到词向量。

**三、Mikolov (2013年)提出的“Word2Vec模型”（本章重点1）**

W2V也是一种“**浅层神经网络语言模型**”；它有两种模型，分别为CBOW和skip-gram；CBOW是根据上下文context(w) 来预测中间词w的概率，而skip-gram是根据中间词w来预测上下文context(w)的概率；他们都有3层结构——输入层，投影层，输出层（注：无隐藏层）。每种模型又有两种实现方式，分别为基于Huffman树和基于负例采样。

注：这个模型在gensim有实现的包，直接调用便可，效果非常不错



**（一）CBOW：**

首先讲解最朴素的做法：

* **输入层：**上下文2c个词的词向量，每个词指定为k维。
* **投影层：**将这个2c个向量直接求和，
* **输出层**：N维（N是字典的大小），与一般的多分类问题一样，使用softmax函数来表示预测为每个单词的概率。

用周围词（）预测出中间词（w）的概率表示为：；我们希望，对于语料T，所有的周围词都尽可能地预测到对应的中间词（也就是尽可能地共现）；因此，需要**极大化**的目标函数为：



对数目标函数为：

其中,均是需要优化的模型参数，是神经网络的权重矩阵，是投影层的向量（分摊到每个词便可得每个词的词向量）；这两个参数一开始都需要随机初始化，然后使用mini\_GD来进行优化，优化结束后便可得到词向量。

问题是这样朴素的解法复杂度太高了，因此衍生了下面的两种解法。

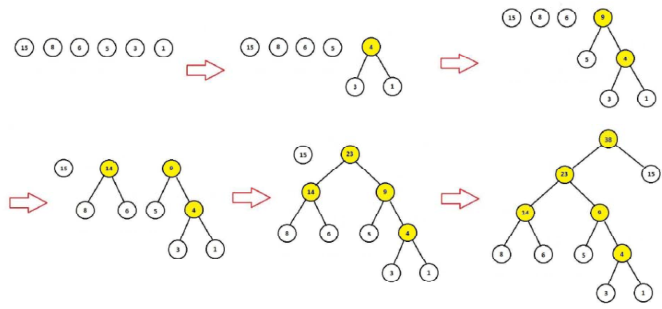
**1、基于Huffman树的实现**

* 1. **Huffman树**

huffman树是一棵特殊的二叉树，在这课二叉树中，高频词的路径短，低频词的路径长。其构造方法是：以词典每个词作为叶子节点，词的频次作为叶子节点的权重，不断地向上合并；具体为：

1. 将词典的N个词，看成N棵树的森林
2. 将“根节点权值“最小的两棵树合并成一棵新树，原来的树视为新树的左右子树，新树的“根节点权值”为左右子树的“根节点权值”之和
3. 重复2），直到森林中只剩下一棵树

整个Huffman树构建完成以后，叶节点的个数为N（词典大小），非叶节点的个数为(N-1)；以[“我”，“喜欢”，“观看”，“巴西”，“足球”，“世界杯”]这句话为例，其词频分别为：15,8,6,5,3,1，其构造过程如下：

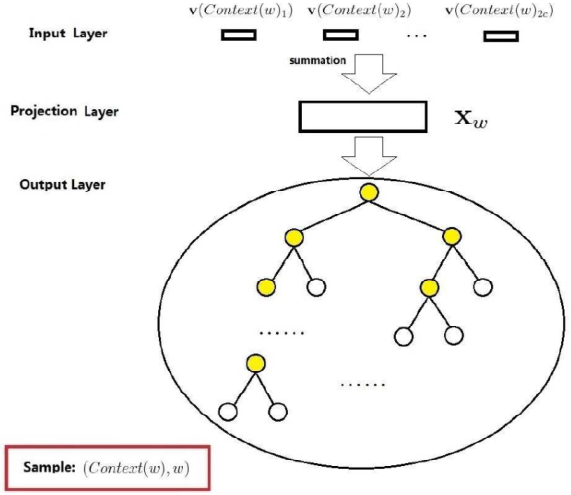


另外约定，左节点编码为1，右节点编码为0。

* 1. **网络结构**

H\_CBOW的网络结构有3层，分别为输入层，投影层，输出层；以样本(context(w),w) (context(w),w)为例，做简要说明。

* **输入层：**上下文的2c个词的词向量（假设上文和下文分别为c各词）
* **投影层：**将这个2c个向量直接求和，
* **输出层：**对应一棵huffuman树，它以词典中的词为叶节点，以词出现的次数为叶节点的权重来构建；而且每个非叶节点对应一个二分类器。

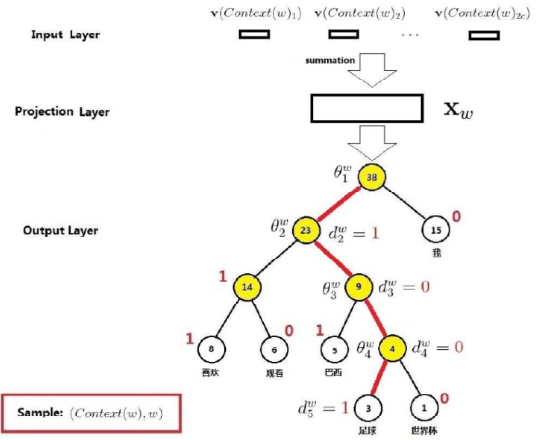


图：CBOW模型结构示意图

* 1. **目标函数推导**

为了方便后面的公式描述，先约定一些符号：

* ：词所在路径的每个节点的编码。（注：无根节点）；其中表示词w在路径上的节点个数；这些编码就是二分类器的正确预测。
* ：词w的路径上，某个内部节点的权重向量。（注：无叶节点）；因为每个内部节点对应一个二分类器，这些权重向量就是二分类器的参数



图：词“足球”的相关记号示意图

给定，预测目标词的概率，相当于沿着huffuman树进行搜索，每经过一个内部节点，就进行一次二分类并生成一个概率，将路径上的所有概率连乘起来，就表示预测为目标词的概率；

对于语料库T，我们希望所有的周围词都尽可能地预测到对应的中间词

* 每个二分类器是一个逻辑回归分类器，预测为“对应编码”的概率表示为：



注：这里有点傻逼，表示预测为右边的概率，但右边编码为0；

* 因此，根据上下文的投影，生成目标单词的概率为：



* 我们的优化目标是，对于语料库T，希望所有的周围词都尽可能地预测到对应的中间词，因此目标函数为：



* 对数目标函数为：



**1.4 梯度上升法求解参数**

W2V采用随机梯度上升法进行参数优化：即每取一个样本，就更新w所在路径上的所有分类器的参数；也即优化的目标函数是。

1. **更新二分类器的参数：**

，而

因此：；可以发现，每个二分类器的参数可以独立地优化，也即让每个二分类器尽可能地预测到属于它的编码。（可以看到取log把连乘变成连加是很重要的）



因此权重更新公式为：

1. **更新词向量：**

由于和是对称的，因此梯度的计算换一下变量就可：



因此，context中每个词的词向量更新公式为：；即把贡献到context(w)每个词的词向量上。

**1.5 伪代码**

以样本为例（每取一个样本就对所有的参数进行更新），下面是H\_CBOW的参数更新过程：

|  |
| --- |
| 3. #更新路径上的每个二分类器的权重      1. #更新上下文每个词的词向量 |

**2、基于负例采样（Negative Sampling ）的实现**

**2.1 模型讲解**

NS\_CBOW；最重要的是负样本的采样；具体做法是：给定的语料库T，对于目标单词w，取作为正样本，在词典中用“负例采样算法”采样neg个词作为负样本，即；而整个样本集为

一个“样本集”中，有all个样本，指定目标词为正样本，其他neg个词为负样本；每个词对应一个二分类器，也即有1个“正二分类器”和neg个“负二分类器”。

我们希望每个二分类器尽可能预测出与它匹配的类别，也即希望“正二分类器”预测为正样本的概率尽可能高，同时“负二分类器”预测为负样本的概率尽可能高。

**2.2 目标函数推导**

* 对于目标单词w，它属于正样本，其余采样出的neg个单词属于负样本。每个样本预测为属于自己类别的概率表示为：



其中是词对应的一个辅助向量，属于模型参数；

直观地理解是：假设一个样本集有all个样本，对每个样本都训练一个二分类模型，是样本对应的模型参数

* 将上式写成整体表达形式：



其中：，表示目标词的标签为1，其他neg个负样本的标签为0

* 我们的**优化目标**是，对于语料库T，每个context(w)都尽量预测出目标词w，而尽量避免预测到其他的词；因此，目标函数为：



* 对上式取对数，得到最终的对数似然函数：



**2.3 梯度上升法求解参数**

采用随机梯度上升法进行优化，即每取一个样本，负例采样得到一个大小为all的“样本集”，将这all个样本对应的分类器进行参数更新，也即优化的目标函数是。

1. **更新二分类器的参数：**

，而

因此：；同理，二分类器独立地优化，每个二分类器尽可能预测出属于它的类别。

****

因此，权重更新公式为：****

1. **更新词向量：**

****

因此，context中每个词的词向量更新公式为：

**2.4 伪代码**

|  |
| --- |
| 1.  2.  3.  #更新“1个样本集”中每个二分类器的权重    4.  #更新上下文每个词的词向量 |

**2.4 负例采样算法**

上文提到的，对于一个目标单词，需要采样对应的“负样本集”，这个“负样本集”就是用负例采样算法得到的；其原理是，词频越高，被采样到的概率越大。

　　在实现上，将一个长度为1的线段分成N份（N为词典的大小），每一段表示词典中的一个单词，每段的长度不一样，高频词对应的长度长，低频词对应的长度短，每个词的长度的计算公式为：

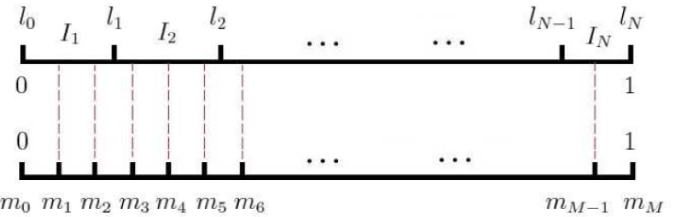
****

在W2V里，做了一点小小的处理：

****

采样前，将这段长为1的线段划分为M等分（M>>N，W2V中），则一共有M-1个位置（对应图中的红色虚线），每个位置对应一个单词；

采样时，每次生成一个[1, M-1]的随机数r，看看r对应哪个单词，就将那个单词采样出来；如果碰巧采样到了目标单词，直接跳过便可。



图：负例采样示意图

1. **Skip-gram：**
2. **Pennington(2014年)提出的Glove模型（本章重点2）**

GloVe训练出的词向量与W2V的相当，甚至比它还要好，因此这也是一个非常有必要掌握的模型。

1. **获取共现矩阵**
2. **Glove模型**

**Glove与W2V的比较**

**文本分类**

**一、TxetCNN：**

基于卷积网络设计，设计不同size的卷积核对句子扫描，相当于提取不同的n-gram特征。

****

**二、Bi-LSTM**

bi-LSTM可以捕捉序列的“历史信息”（正向LSTM）和“未来信息”（逆向LSTM），在文本分类中，两个相反的LSTM只有最后一个cell有输出；将两个输出拼接(merge=concat)，再经过dense层形成判别概率。

****

**序列标注**

**一、biLSTM-crf**

命名实体识别的任务中，最常用的模型就是biLSTM-crf，模型最大的优点是，完全不需要手工设计特征（传统的crf模型需要手工定义非常多的特征函数），而能达到或接近state of art的效果。



**1、模型的结构：**

如果把NER纯粹当做一个标签的预测问题，理论上可以直接用biLSTM，然后每个step都加一个全连接层并用softmax激活，表示每个step预测为每种标签的概率，然后分别取概率最大的标签序列即可（实际使用中，效果也尚可）

但仅仅这样没有考虑到标签之间的约束关系，因为根据“语法规则”，有些标签是不能连接在一起的，比如，，等，因此加上crf来层对标签之间的关系进行约束。

**模型分为4层：**

* **嵌入层**：(bs, seq\_len, embedd\_dim)
* **bi\_LSTM层**：(bs, seq\_len, biLstm\_dim)，用于捕捉序列信息
* **hidden层**：(bs, seq\_len, n\_tag)，表示每个step预测为每种标签的概率。
* **CRF层**：[n\_tag, n\_tag]；用于约束标签之间的关系。

**前向传播公式：（从hidden层到CRF层）**

* 输入一个句子：，其真实标签序列是：；

为方便计算，在句首和句尾分别加入“start, end”标志位，也即，，，；

* 定义由“X预测为真实Y”的得分是：
* **参数A**：标签之间的“转移得分矩阵”，它的size是，k是标签的种类，表示标签i转移到标签j的得分
* **参数E**：biLSTM输出的“发射得分矩阵”，它的size是，n是模型的长度，k是输出的维度，表示第i个单词对应标签j的得分；
* 这个定义表达的意思是：模型预测的得分 = biLSTM预测的得分总和 + 标签关系之间的得分总和。
* 归一化为概率：“X预测为真实Y”的概率是：；其中表示所有可能的标签序列。

应该说，crf最精巧的地方在于，计算概率是以路径为单位的，也即：真实路径的概率 = 真实路径的得分 / 所有路径的得分总和

**2、目标函数（损失函数）**

* 当采用bs\_GD算法时，我们的目标函数是，
* 将上式取对数并取反为，得到损失函数：



在训练过程中，最小化这个损失函数，便可求得参数A和E。参数E由bi\_LSTM的结构和参数确定，A是一个的矩阵。

**3、归一化因子的快速求解：**

上文中的损失函数，有一个归一化因子，是很难直接求解的，因为直接求解的时间复杂度是；我们依然可以用动态规划的思想来降低复杂度。

假设seq\_len=4, n\_tag=2，以s(3,1)为例子讲解：

s(3,1)表示，到达点x31的所有路径相加；它等于，到达点x21的每条路径，加上，到达x22的每条路径



* step1：
* step2：
* step3：
* step4：
* final：

可以看到，采用动态规划的方法，时间复杂度由原来的，降低到。

目前的深度学习框架都已经带有自动求导的功能，只要我们能写出可导的损失函数，就可以帮我们完成优化过程了。

**4、标签的预测：**

对于预测的过程——给定一个句子，找出得分最大的标签序列；可采用动态规划的方法；HMM，bilstm\_crf的预测问题只有细微的差别，这里一起讲解。

**动态规划——寻找最大得分的路径：**

说实话，公式的描写看起来，很容易把人绕晕，其实你只需要把握动态规划讲什么，就很容易理解公式了

1. **HMM：**

观测序列是给定的，每个step都有n\_tag个标签，每个step的n\_tag个标签生成给定观测的概率就确定了；因此，每个节点有分数——每种tag生成给定观测的概率；节点与节点的连接有分数——相邻step的各个tag之间转移的概率；

希望找到一条路径，使得这些得分相乘最大。

**2) bi\_lstm-CRF：**

序列有seq\_len个step，每个step，bilstm（经hidden层）都输出n\_tag个得分；因此，每个节点有分数——隐藏层输出的得分；节点与节点的连接有分数——crf的转移得分；

希望找到一条路径，使得这些得分加起来最大。

为了方便描述，假设seq\_len=3，n\_tag=3；节点(2, 1)：表示step2的第1个节点



* step = 1：每个节点都有一个分数
* step = 2：

对于节点(2, 1)，分别计算(1, 1)->(2,1)，(1, 2)->(2,1)，(1, 3)->(2,1)的得分，得分最大的那个是最优子路径；保存最优子路径(1,1) -> (2, 1)，及得分s(2,1)；（也就是说，对于一条长的最优路径，如果中途经过节点(2, 1)，那么他一定是从(1, 1)那里过来，不可能从其他节点过来）

对于节点(2, 2)，保存最优子路径(1, 3) -> (2, 2)，及得分s(2,2)

对于节点(2, 3)，保存最优子路径(1, 1) -> (2, 3)，及得分s(2,3)

* step = 3：

对于节点(3, 1)，如果路径是(2,1)->(3,1)，那么他一定先走到达(2,1)的最优子路径(1, 1)->(2, 1)，再从(2,1)->(3,1)；计算s(2,1)->(3,1)的得分。

同理计算s(2,2)->(3,1)，s(2,3)->(3,1)；得分最大的那个是最优子路径；保存最优子路径(1, 3) -> (2, 2) -> (3,1)，及其得分s(3,1)；

对于节点(3, 2)，保存最优子路径(1, 1) -> (2, 3)->(3,2)，及得分s(3,2)

对于节点(3, 3)，保存最优子路径(1, 1) -> (2, 3)->(3,3)，及得分s(3,3)

* 终止：看看最后一步(step3)，得分最大的路径是哪个，假如是s(3,3)，那么最优路径就是(1, 1) -> (2, 3)->(3,3)。

编程实现时，可以统一用这种方式编写循环，来计算得分：

(1, 1) -> (2, 3)的得分是：s(2,3) = x11 + t13 + x23；

(1, 1) -> (2, 3)->(3, 3)的得分是：s(3,3) = s12 + t33 + x33；

(1, 1) -> (2, 3)->(3, 3)->(4, 2)的得分是：s(4,2) = s13 + t32 + x42

可以看到，时间复杂度由原来的，降低到，非常屌！！