通俗理解word2vec

<https://www.jianshu.com/p/471d9bfbd72f?utm_campaign=haruki&utm_content=note&utm_medium=reader_share&utm_source=weixin>

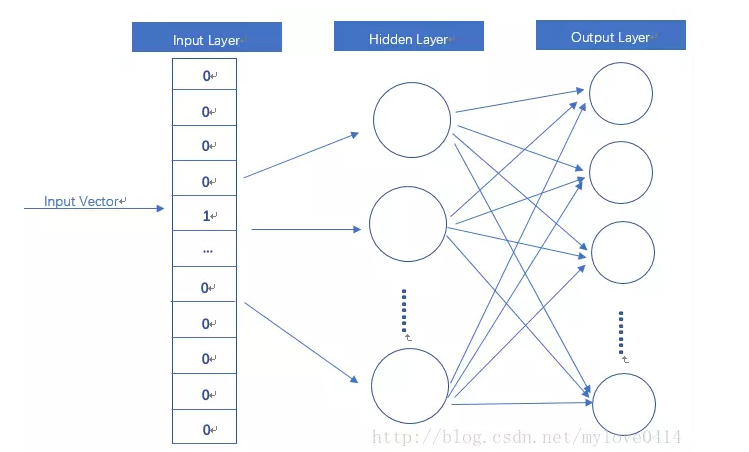
Word2vec 模型是干什么的？

给模型一段话作为输入，模型会吐出每一个词的词向量，该向量带有词的上下文信息。

也就是说只要输入是序列数据 那么通过这个模型都可以学习到它的嵌入向量，比如，图数据，行为数据等等，所谓万物皆可embedding

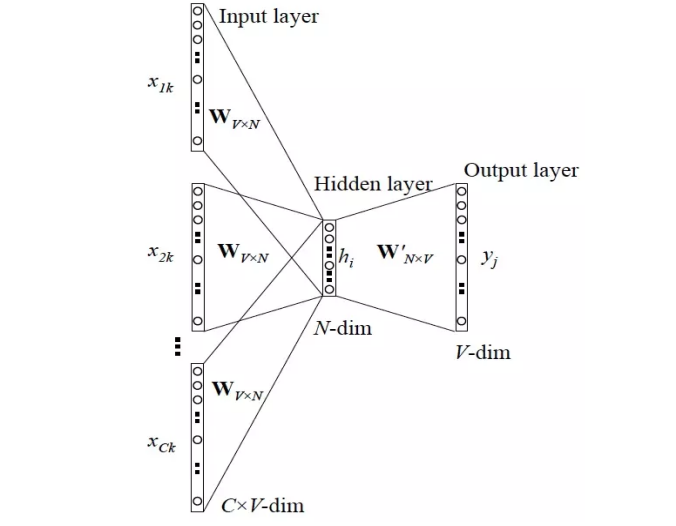
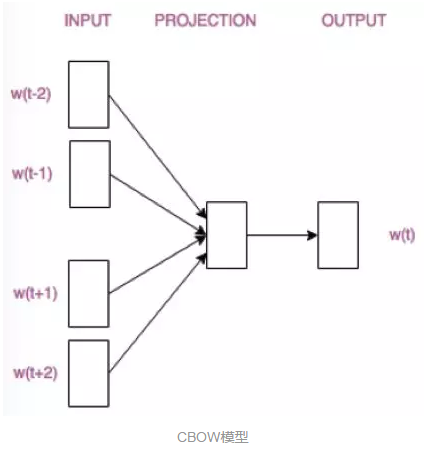
Word2vec具体原理

对于自然语言来说，句子是文本，是单词组成的，是无法直接进行训练的，通常先对文本进行onehot。然后通过神经网络进行训练



输入一句话每个单词onehot组成的向量，Hidden Layer没有激活函数也就是线性的单元。Output Layer维度跟Input Layer的维度一样，用的是Softmax回归。当这个模型训练好以后，我们并不会用这个训练好的模型处理新的任务，我们真正需要的是这个模型通过训练数据所学得的参数，例如隐层的权重矩阵。

一般word2vec有两种可选的模型，CBOW 与Skip-Gram两种模型，CBOW对小型数据库比较合适，而Skip-Gram在大型语料中表现更好。



CBOW

通过当前单词周围的词，利用上下文信息来预测当前单词，即 输入上下文的onehot vec

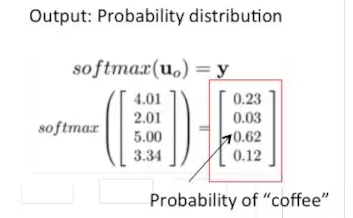
输出模型对这个单词预测的vec 真实vec与预测vec之间差值就是损失值。

1 输入层：上下文单词的onehot. {假设单词向量空间dim为V，上下文单词个数为C}  
2 所有onehot分别乘以共享的输入权重矩阵W. {VN矩阵，N为自己设定的数，初始化权重矩阵W}  
3 所得的向量 {因为是onehot所以为向量} 相加求平均作为隐层向量, size为1N.

1V \* VN = 1N C个 1N

1H = avg (C个1N )  
4 乘以输出权重矩阵W' {NV}

1N(H) \* NV = 1V  
5 得到向量 {1V} 激活函数处理得到V-dim概率分布 {PS: 因为是onehot嘛，其中的每一维斗代表着一个单词}  
6 概率最大的index所指示的单词为预测出的中间词（target word）与true label的onehot做比较，误差越小越好（根据误差更新权重矩阵）



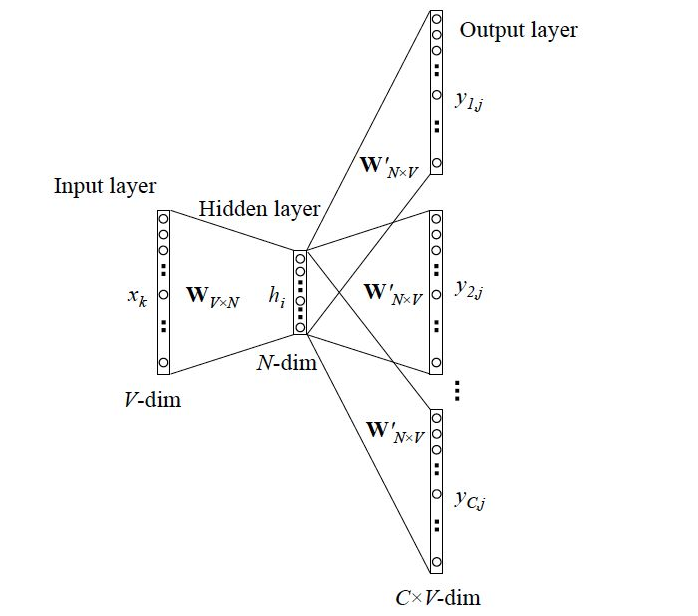
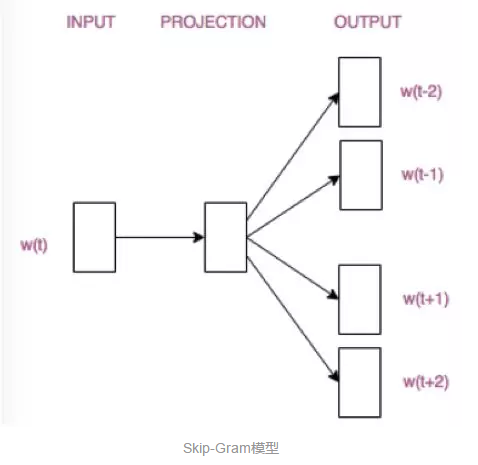
所以，需要定义loss function（一般为交叉熵代价函数），采用梯度下降算法更新W和W'.

输入层的每个单词与矩阵W相乘得到的向量的就是我们想要的词向量（word embedding），这个矩阵（所有单词的word embedding）也叫做look up table（其实聪明的你已经看出来了，其实这个look up table就是矩阵W自身），也就是说，任何一个单词的onehot乘以这个矩阵都将得到自己的词向量。有了look up table就可以免去训练过程直接查表得到单词的词向量了。

Skip-Gram

<https://zhuanlan.zhihu.com/p/27234078>

https://zhuanlan.zhihu.com/p/53194407

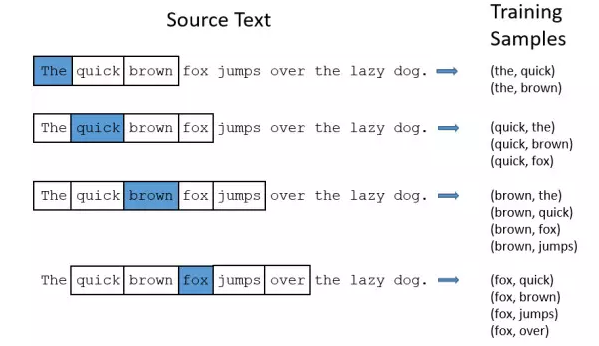


从直观上理解，Skip-Gram是给定input word来预测上下文。假如我们有一个句子“The dog barked at the mailman”。首先我们选句子中间的一个词作为我们的输入词，例如我们选取“dog”作为input word；

skip\_window，它代表着我们从当前input word的一侧（左边或右边）选取词的数量。如果我们设置skip\_window=2，那么我们最终获得**窗口中的词（包括input word在内）**就是['The', 'dog'，'barked', 'at']。skip\_window=2代表着选取左input word左侧2个词和右侧2个词进入我们的窗口，所以整个窗口大小span=2x2=4

num\_skips，它代表着我们从整个窗口中选取多少个不同的词作为我们的output word，当skip\_window=2，num\_skips=2时，我们将会得到两组 (input word, output word) 形式的训练数据，即 ('dog', 'barked')，('dog', 'the')

神经网络基于这些训练数据将会输出一个概率分布，这个概率代表着到我们词典中每个词有多大可能性跟input word同时出现,对其实预测的是成对出现的词对，



面的图中给出了一些我们的训练样本的例子。我们选定句子“The quick brown fox jumps over lazy dog”，设定我们的窗口大小为2（window\_size=2），也就是说我们仅选输入词前后各两个词和输入词进行组合。蓝色代表input word，方框内代表位于窗口内的单词。Training Samples（输入， 输出）

这个过程比如说训练The quick brown fox jumps over lazy dog这句话，需要遍历每一个单词，每一个单词都跑一次神经网络，不同的次其窗口有可能还不一样，

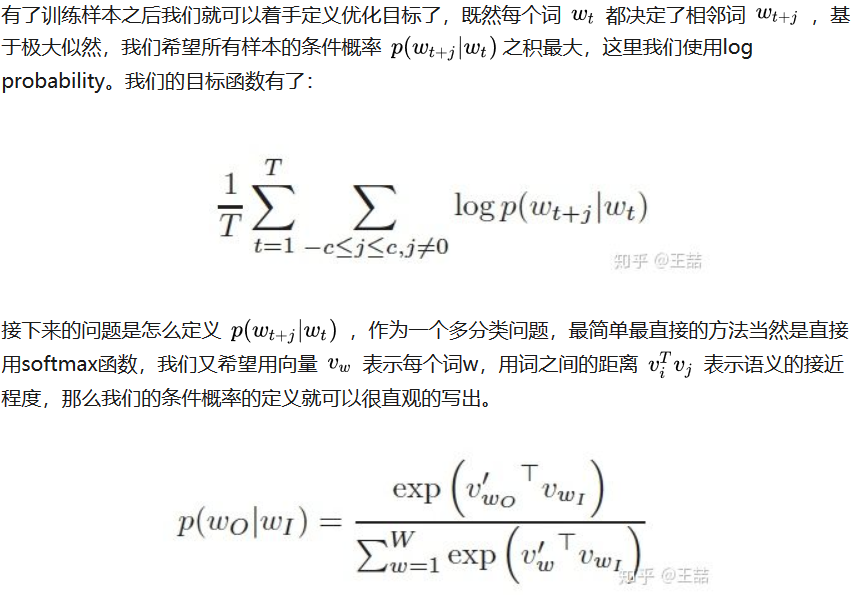
比如选定了brown这个词了，根据skip\_window=2,确定了窗口，我们前后取两个单词去预测，窗口有4窗口长度是2sw+1,窗口单词[The ,quick,brown, fox, jumps]

如果num\_skips=2 意思是我们是要预测上下文这4个单词，但是我们学习的时候呢只选择其中两个单词来学习。

为什么要选用一个一个词对来训练呢，原因就在于训练的时候我们需要使用两个词的词向量内积来度量词的接近程度

用softmax函数来代表词对出现概率p

基于极大似然，要使得所有词对出现的条件概率最大，定义目标函数



就是说我们的目标是窗口内所有词对出现的概率最大，最大似然，

4实战

上面讲了这么多理论细节，其实在真正应用的时候，只需要调用 Gensim （一个 Python 第三方库）的接口就可以。但对理论的探究仍然有必要，你能更好地知道参数的意义、模型结果受哪些因素影响

加速算法

**负采样（negative sampling）**  
训练一个神经网络意味着要输入训练样本并且不断调整神经元的权重，从而不断提高对目标的准确预测。每当神经网络经过一个训练样本的训练，它的权重就会进行一次调整。  
正如我们上面所讨论的，vocabulary的大小决定了我们的Skip-Gram神经网络将会拥有大规模的权重矩阵，所有的这些权重需要通过我们数以亿计的训练样本来进行调整，这是非常消耗计算资源的，并且实际中训练起来会非常慢。  
**负采样（negative sampling）**解决了这个问题，它是用来提高训练速度并且改善所得到词向量的质量的一种方法。不同于原本每个训练样本更新所有的权重，负采样每次让一个训练样本仅仅更新一小部分的权重，这样就会降低梯度下降过程中的计算量。  
当我们用训练样本 ( input word: "fox"，output word: "quick") 来训练我们的神经网络时，“ fox”和“quick”都是经过one-hot编码的。如果我们的vocabulary大小为10000时，在输出层，我们期望对应“quick”单词的那个神经元结点输出1，其余9999个都应该输出0。在这里，这9999个我们期望输出为0的神经元结点所对应的单词我们称为“negative” word。  
当使用负采样时，我们将随机选择一小部分的negative words（比如选5个negative words）来更新对应的权重。我们也会对我们的“positive” word进行权重更新（在我们上面的例子中，这个单词指的是”quick“）。

在论文中，作者指出指出对于小规模数据集，选择5-20个negative words会比较好，对于大规模数据集可以仅选择2-5个negative words。

回忆一下我们的隐层-输出层拥有300 x 10000的权重矩阵。如果使用了负采样的方法我们仅仅去更新我们的positive word-“quick”的和我们选择的其他5个negative words的结点对应的权重，共计6个输出神经元，相当于每次只更新IMG_256个权重。对于3百万的权重来说，相当于只计算了0.06%的权重，这样计算效率就大幅度提高。