参考链接：

<https://www.leiphone.com/news/201706/eV8j3Nu8SMqGBnQB.html>

<https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/>

Word2Vec其实就是通过学习文本来用词向量的方式表征词的语义信息，即通过一个嵌入空间使得语义上相似的单词在该空间内距离很近。Embedding其实就是一个映射，将单词从原先所属的空间映射到新的多维空间中，也就是把原先词所在空间嵌入到一个新的空间中去。

从直观上理解，Skip-Gram是给定input word来预测上下文。而CBOW是给定上下文，来预测input word

Word2Vec模型实际上分为了两个部分，**第一部分为建立模型，第二部分是通过模型获取嵌入词向量**。

隐层的权重矩阵——这些权重在Word2Vec中实际上就是我们试图去学习的“word vectors”

有了input word以后，我们再定义一个叫做skip\_window的参数，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。另一个参数叫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

神经网络基于这些训练数据将会输出一个概率分布，这个概率代表着我们的词典中的每个词是output word的可能性

当我们的模型完成训练后，给定一个单词”Soviet“作为输入，输出的结果中”Union“或者”Russia“要比”Sasquatch“被赋予更高的概率。

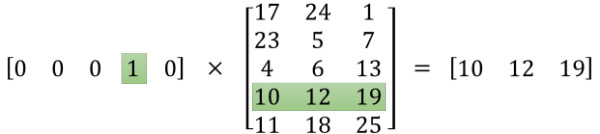
我们得想个办法来表示这些单词。最常用的办法就是基于训练文档来构建我们自己的词汇表（vocabulary）再对单词进行one-hot编码。

模型的输入如果为一个10000维的向量，那么输出也是一个10000维度（词汇表的大小）的向量，它包含了10000个概率，每一个概率代表着当前词是输入样本中output word的概率大小。

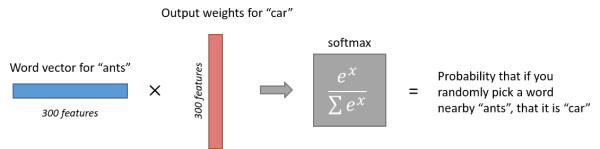
如果我们现在想用300个特征来表示一个单词（即每个词可以被表示为300维的向量）。那么隐层的权重矩阵应该为10000行，300列（隐层有300个结点）。词向量的维度是一个可以调节的超参数.

我们最终的目标就是学习这个隐层的权重矩阵

input word和output word都会被我们进行one-hot编码。仔细想一下，我们的输入被one-hot编码以后大多数维度上都是0（实际上仅有一个位置为1），所以这个向量相当稀疏。对于矩阵相乘，它会消耗相当大的计算资源，为了高效计算，它仅仅会选择矩阵中对应的向量中维度值为1的索引行。这样模型中的隐层权重矩阵便成了一个”查找表“（lookup table），进行矩阵计算时，直接去查输入向量中取值为1的维度下对应的那些权重值。隐层的输出就是每个输入单词的“嵌入词向量”。栗子：



经过神经网络隐层的计算，ants这个词会从一个1 x 10000的向量变成1 x 300的向量，再被输入到输出层。



如何在skip-gram模型上进行高效的训练:

1. 将常见的单词组合（word pairs）或者词组作为单个“words”来处理。

2. 对高频次单词进行抽样来减少训练样本的个数。

3. 对优化目标采用“negative sampling”方法，这样每个训练样本的训练只会更新一小部分的模型权重，从而降低计算负担。

对常用词抽样并且对优化目标采用“negative sampling”不仅降低了训练过程中的计算负担，还提高了训练的词向量的质量。

Word2Vec通过“抽样”模式来解决这种高频词问题。它的基本思想如下：对于我们在训练原始文本中遇到的每一个单词，它们都有一定概率被我们从文本中删掉，而这个被删除的概率与单词的频率有关。

ωi是一个单词，Z(ωi) 是 ωi 这个单词在所有语料中出现的频次。举个栗子，如果单词“peanut”在10亿规模大小的语料中出现了1000次，那么 Z(peanut) = 1000/1000000000 = 1e - 6。在代码中还有一个参数叫“sample”，这个参数代表一个阈值，默认值为0.001。这个值越小意味着这个单词被保留下来的概率越小（即有越大的概率被我们删除）

P(ωi) 代表着保留某个单词的概率：

IMG_256

随着单词出现频率的增高，它被采样保留的概率越来越小。

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。（个人理解：对于一个batch的训练样本，loss、dw、db是求一个batch的平均值，然后更新w和b）

**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“）。

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 300 x 6 = 1800 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。

我们使用“一元模型分布（unigram distribution）”来选择“negative words”。

一个单词被选作negative sample的概率跟它出现的频次有关，出现频次越高的单词越容易被选作negative words。

每个单词被选为“negative words”的概率计算公式与其出现的频次有关：

IMG_256

每个单词被赋予一个权重，即 f(ωi)， 它代表着单词出现的频次。

unigram table有一个包含了一亿个元素的数组，这个数组是由词汇表中每个单词的索引号填充的，并且这个数组中有重复，也就是说有些单词会出现多次。那么每个单词的索引在这个数组中出现的次数该如何决定呢，有公式，也就是说计算出的**负采样概率\*1亿=单词在表中出现的次数**。

有了这张表以后，每次去我们进行负采样时，只需要在0-1亿范围内生成一个随机数，然后选择表中索引号为这个随机数的那个单词作为我们的negative word即可。一个单词的负采样概率越大，那么它在这个表中出现的次数就越多，它被选中的概率就越大。

为了解决这个问题，即softmax的昂贵计算，Word2Vec使用了一种称为噪声对比估计的技术。

其基本思想是将多项分类问题（因为它是预测下一个单词的问题）转换为二元分类问题。也就是说，不是使用softmax来估计输出词的真实概率分布，而是使用二元逻辑回归（二元分类）代替。

对于每个训练样本，增强的（优化的）分类器被馈送一对真正的对（中心词和出现在其上下文中的另一个词）以及多个kk随机损坏对（由中心词和随机选择的来自词汇表的单词）。通过学习区分真正的对与被破坏的对，分类器将最终学习单词向量。

这很重要：优化分类器不是预测下一个单词（“标准”训练技术），而是简单地预测一对单词是好还是不好。

Word2Vec稍微自定义过程并将其称为负采样（negative sampling）。在Word2Vec中，负样本（用于损坏的对）的单词是从专门设计的分布中抽取的，这种分布倾向于更频繁地抽取不太频繁的单词。

对于word2vec任何给定的单词，你有一个需要类似的单词列表（积极的类），但负面的类（与目标单词不相似的单词）是通过抽样编写的。

嵌入层权重矩阵的维度是(vocab\_size, embedding\_size)。