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。Word2Vec的整个建模过程实际上与自编码器（auto-encoder）的思想很相似，即先基于训练数据构建一个神经网络，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——后面我们将会看到这些权重在Word2Vec中实际上就是我们试图去学习的“word vectors”。基于训练数据建模的过程，我们给它一个名字叫“Fake Task”，意味着建模并不是我们最终的目的。

自编码器（auto-encoder）：通过在隐层将输入进行编码压缩，继而在输出层将数据解码恢复初始状态，训练完成后，我们会将输出层“砍掉”，仅保留隐层。

训练模型的真正目的是获得模型基于训练数据学得的隐层权重。为了得到这些权重，我们首先要构建一个完整的神经网络作为我们的“Fake Task”，后面再返回来看通过“Fake Task”我们如何间接地得到这些词向量。

Fake Task它的完整过程如下:

1、首先我们选取句子中的一个词作为输入词（实际训练过程中，是依次将句子中的每个词作为输入词构建训练对的），例如选取“fox”作为input word；

2、有了input word后，我们再定义一个叫skip\_window的参数，它代表我们从当前input word的一侧选取词的数量。如果设置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', 'fox', 'jumps', 'over']。那么整个窗口大小span = = 4。另外一个参数叫num\_skips，它代表我们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('fox', 'jumps')，('fox', 'the')

3、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词典中的每个词是output word的可能性。例如，上面我们得到两组数据。我们先用一组数据('fox', 'jumps')来训练神经网络，那么模型通过前面学习这个训练样本，会告诉我们词汇表中其他单词的概率大小和“jumps”的概率大小。

模型的输出概率代表着我们的词典中每个词有多大可能跟input word同时出现。

输出层用softmax，但是要知道，如果真的用softmax来算，计算量大的惊人。所以可以用这三个方法来减少计算量：

1、将常见的单词组合（word pairs）或者词组作为单个“words”来使用。

2、对高频词进行抽样来减少训练样本的个数

3、最后最重要的一点，就是“nagative sampling”方法，这样每个训练样本只会更新一小部分模型权重，从而降低计算负担。

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“）。

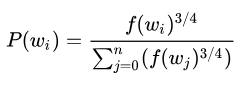
在论文中，作者指出指出对于小规模数据集，选择5-20个negative words会比较好，对于大规模数据集可以仅选择2-5个negative words。

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

要注意的一点是，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。

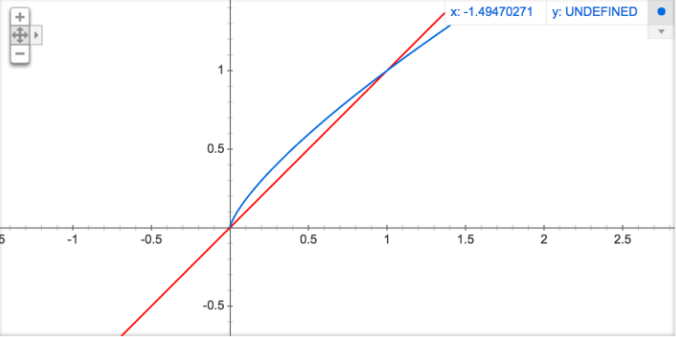
在word2vec的C语言实现中，你可以看到对于这个概率的实现公式。每个单词被选为“negative words”的概率计算公式与其出现的频次有关。

代码中的公式实现如下：



每个单词被赋予一个权重，即， 它代表着单词出现的频次。

公式中开3/4的根号完全是基于经验的，论文中提到这个公式的效果要比其它公式更加出色。你可以在google的搜索栏中输入“plot y = x^(3/4) and y = x”，然后看到这两幅图（如下图），仔细观察x在[0,1]区间内时y的取值，x^{3/4}有一小段弧形，取值在y=x函数之上。



负采样的C语言实现非常的有趣。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。

输出层并不是W的转制而是一个新的300\*10000的权重矩阵，再通过softmax 来算概率