**word2vec原理**

**1、词的独热表示one-hot**

考虑一下的三个特征：

["male", "female"]

["from Europe", "from US", "from Asia"]

["uses Firefox", "uses Chrome", "uses Safari", "uses Internet Explorer"]

将它换成独热编码后，应该是：

feature1=[01,10]

feature2=[001,010,100]

feature3=[0001,0010,0100,1000]

**优点：**一是解决了分类器不好处理离散数据的问题，二是在一定程度上也起到了扩充特征的作用。

**缺点：**

（1）任意两个词之间都是孤立的，根本无法表示出在语义层面上词语词之间的相关信息，而这一点是致命的。

（2）我们的词汇表一般都非常大，比如达到百万级别，这样每个词都用百万维的向量来表示简直是内存的灾难。能不能把词向量的维度变小呢？

**2、词的分布式表示 distributed representation**

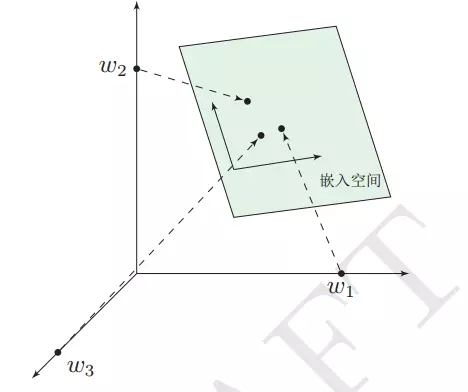
Dristributed representation可以解决One hot representation的问题，它的思路是通过训练，将每个词都映射到一个**较短的词向量**上来。所有的这些词向量就构成了向量空间，进而可以用普通的统计学的方法来研究词与词之间的关系。

比如下图我们将词汇表里的词用"Royalty","Masculinity", "Femininity"和"Age"4个维度来表示，King这个词对应的词向量可能是(0.99,0.99,0.05,0.7)(0.99,0.99,0.05,0.7)。



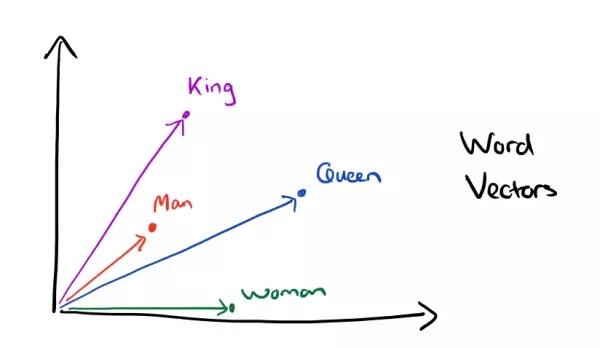
当然在实际情况中，我们并不能对词向量的每个维度做一个很好的解释。

这个过程称为**word embedding（词嵌入）**，即将高维词向量嵌入到一个低维空间：



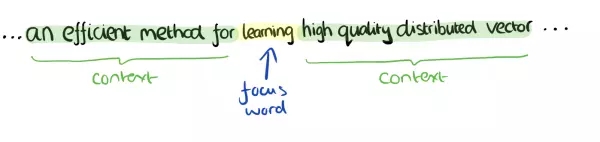
经过我们一系列的降维操作，有了用Dristributed representation表示的较短的词向量，我们就可以较容易的分析词之间的关系了，比如我们将词的维度降维到2维，有一个有趣的研究表明，用下图的词向量表示我们的词时，我们可以发现：

C:\Users\Administrator\Desktop\9285151-48358f3362496684.webp.jpg



出现这种现象的原因是，我们得到最后的词向量的训练过程中引入了词的上下文。

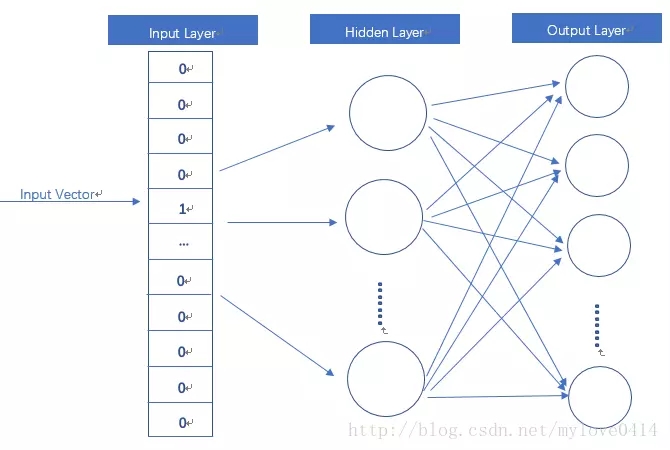
举个栗子：



你想到得到"learning"的词向量，但训练过程中，你同时考虑了它左右的上下文，那么就可以使"learning"带有语义信息了。通过这种操作，我们可以得到近义词，甚至cat和它的复数cats的向量极其相近。

**word2vec**

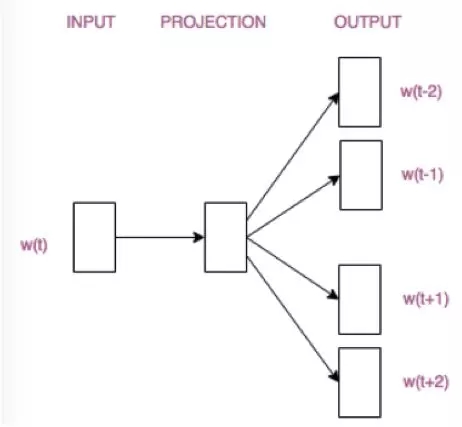
输入是One-Hot Vector，Hidden Layer没有激活函数，也就是线性的单元。Output Layer维度跟Input Layer的维度一样，用的是Softmax回归。当这个模型训练好以后，我们并不会用这个训练好的模型处理新的任务，我们真正需要的是这个模型通过训练数据所学得的**参数**，例如**隐层的权重矩阵**。



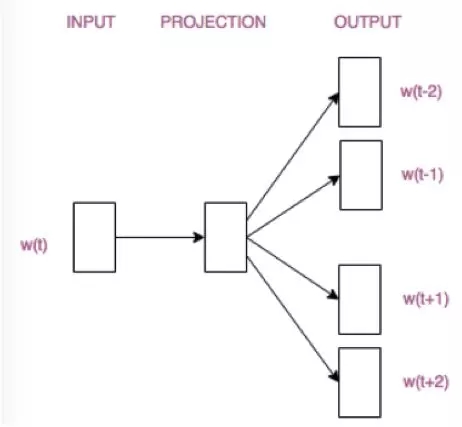
这个模型是如何定义数据的输入和输出呢？一般分为**CBOW(Continuous Bag-of-Words 与Skip-Gram**两种模型：

（1）CBOW模型的训练输入是某一个特征词的上下文相关的词对应的词向量，而输出就是这特定的一个词的词向量。

（2）Skip-Gram模型和CBOW的思路是反着来的，即输入是特定的一个词的词向量，而输出是特定词对应的上下文词向量。CBOW对小型数据库比较合适，而Skip-Gram在大型语料中表现更好。

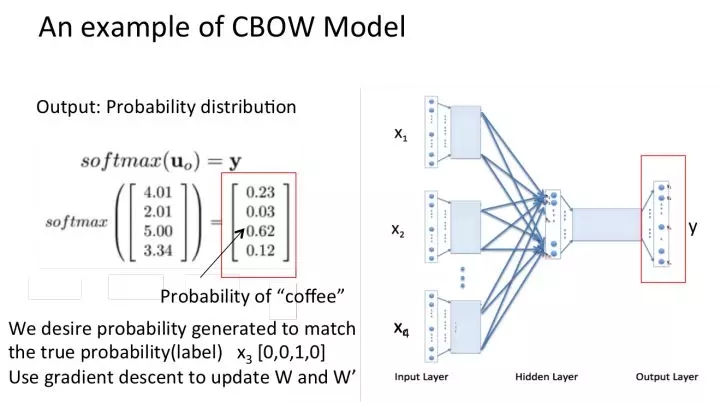
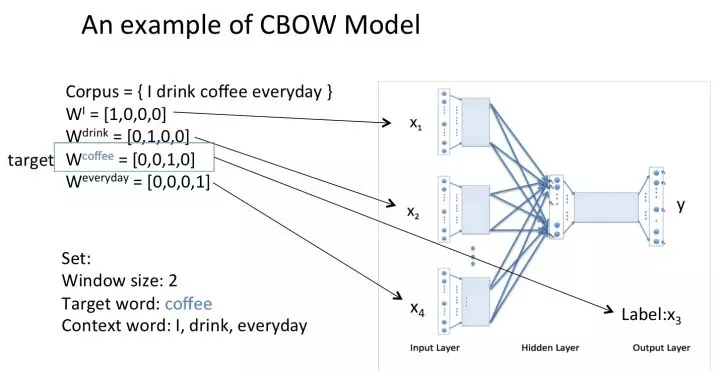


Skip-Gram



CBOW

**CBOW的训练过程**



**skip-gram的训练过程**

Skip-gram模式是根据中间词，预测前后词，CBOW模型刚好相反，根据前后的词，预测中间词。

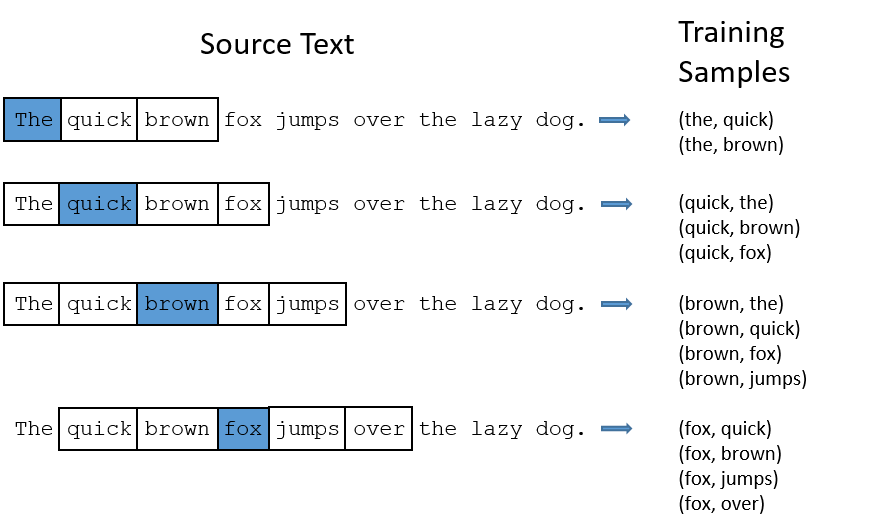
首先，我们需要定义一个窗口大小，在窗口里面的词，我们才有中间词和前后词的定义。一般这个窗口大小在5-10之间。

举个例子，我们设置窗口大小（window size）为2：

|The|quick|brown|fox|jump|

那么，brown就是我们的中间词，The、quick、fox、jump就是前后词。

我们是以什么样的格式用来训练的呢？



可以看到，我们总是以中间词放在第一个位置，然后跟着我们的前后相邻词。可以看到，每一对词都是一个输入和一个输出组成的数据对(X,Y)。其中，X是feature，Y是label。

所以，我们训练模型之前，需要根据语料，整理出所有的像上面这样的输入数据用来训练。

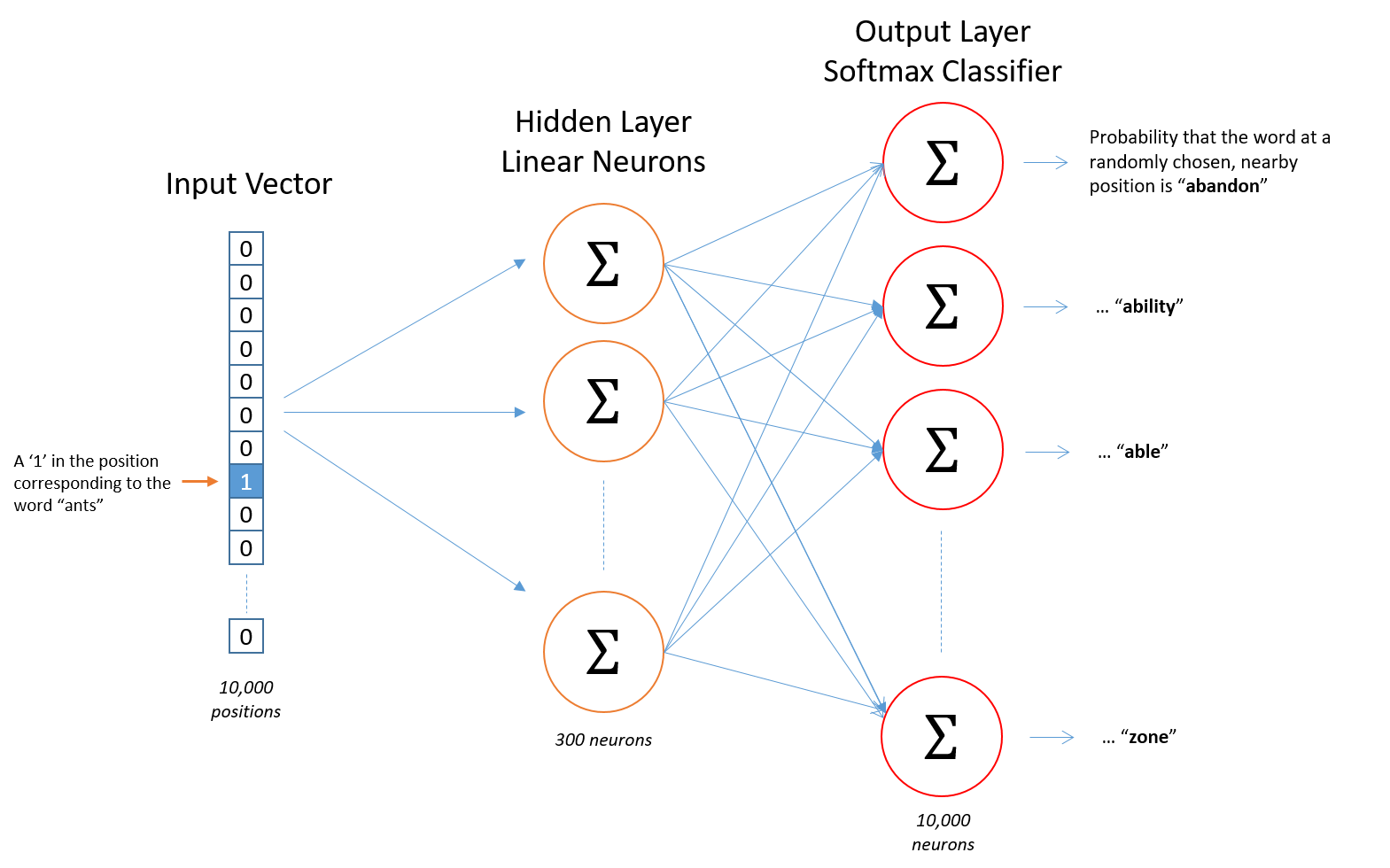
word2vec是一个简单的神经网络，有以下几个层组成：

**１个输入层、1个隐藏层、1个输出层**

输入层输入的就是上面我们说的数据对的数字表示，输出到隐藏层。

隐藏层的神经网络单元的数量，其实就是我们所说的embedding size，需要注意的是，我们的隐藏层后面不需要使用激活函数。

输出层，我们使用softmax操作，得到每一个预测结果的概率。



输入层

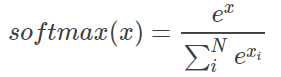
输入使用的是one-hot编码

隐藏层

隐藏层的神经单元数量，代表着每一个词用向量表示的维度大小。假设我们的hidden\_size取300，也就是我们的隐藏层有300个神经元，所以对于输入层和隐藏层之间的**权值矩阵W**，它的形状应该是**[vocab\_size, hidden\_size]**的矩阵，

输出层

输出层是一个[vocab\_size]大小的向量，每一个值代表着输出一个词的概率。我们需要知道它接下来的词的概率分布。



softmax还有一个性质，因为它函数指数操作，如果损失函数使用对数函数，那么可以抵消掉指数计算。

回顾一下我们的结构图，很显然，三个层之间会有两个权值矩阵W，同时，两个偏置项b。所以我们的整个网络的构建，可以用下面的**伪代码**：



**损失函数**

输出层，实际上就是一个softmax分类器。所以按照常规套路，损失函数就选择交叉熵损失函数。



# 损失函数

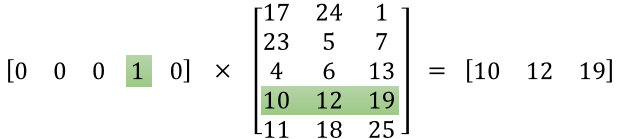
cross\_entropy\_loss = tf.reduce\_mean(-tf.reduce\_sum(y\_label \* tf.log(prediction), reduction\_indices=[1]))

# 训练操作

train\_op = tf.train.GradientDescentOptimizer(0.1).minimize(cross\_entropy\_loss)

**为啥输入使用one-hot编码？**

我们知道word2vec训练后会得到一个权值矩阵W1(暂时忽略b1)，这个矩阵就是我们的所有词的向量表示啦！这个矩阵的每一行，就是一个词的向量表示。如果两个矩阵相乘…



在矩阵相乘的时候，就是选取出矩阵中的某一行，而这一行就是我们输入这个词语的word2vec表示！。