Word2vector Summarization

Github: <https://github.com/weixsong/word2vec/>

# 1 Basic Introduction of word2vector

## 1.1 CBOW

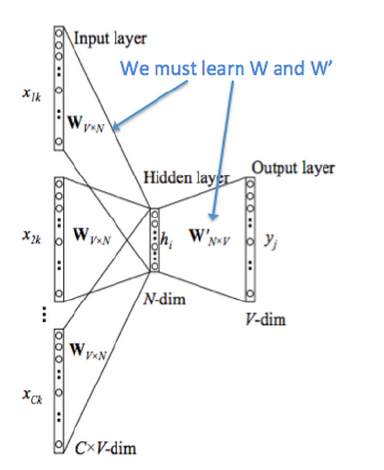


图1

上图是目前各种文档中常见的图，但是上图只是简单的给出一个CBOW模型的架构，但是本文中将要使用的一些符号会和上图中不对应，而且上图中神经网络的权重矩阵的表示也和常用的表示不太一致。所以本文中重新绘制一个CBOW模型：



图2

简单的来说，CBOW就是用一个词的上下文环境来预测当前单词的出现的概率。

让我们做一些约定：

* Wi: word i from vocab V
* V : (dim: H \* V) input word matrix
* vi : (dim: H) i-th column of V, the input vector representation of word wi
* U: (dim: V\*H), output word matrix
* ui : (dim: H), ith row of U, the output vector representation of word wi
* H: is the hidden size, and also is the dim of word embeddings
* Input Xc-1 is a one hot vector with dimension of V

Then CBOW model works in these steps:

1. Generate one hot word vectors (xc-m,…,xc-1,xc+1,…,xc+m) for the input context size of m.
2. Get our embedded words vectors for the context,
   1. vc-m = Vxc-m
   2. vc-m+1 = Vxc-m+1
   3. …
   4. vc+m = Vxc+m
3. average these vector to get = (vc-m +vc-m+1 +….+vc+m) / 2m
4. generate a score vector z = U
5. turn the score vector into probability distribution by softmax: = softmax(z)

CBOW的objective function为：

(1)

使得给定一个Context获得wt的概率最大，等价于使得cross entropy 最小，并且我们一般计算过程中都使用cross entropy作为cost function，并且使用梯度下降使得cost function逐渐减小。

对于一个训练样本，cross entropy可以表示为：

(2)

因为 y 为 one hot vector, 所以上面公式中只有一个值不为0， 所以公式2可以重新写过：

(3)

通过采用cross entropy作为cost function, 我们可以通过使得cost function最小来进行参数更新 (针对一个训练样本)：

(这里理解为cross entropy可能有点费解，但是logP(wc|Context)确实可以理解为真实的概率，所以可以理解为cross entropy,同时也可以理解为如果要想使得概率最大，那么就可以使得取负号的概率最小)

(4)

上面公式(4)中表示当前对应wt的context, 即为hidden vector 。在这个公式中，要计算一个softmax，每个对应的单词都要计算输出概率，会导致计算量比较大。另外因为用了所有的单词作为output，所以计算derivative of error with regard to the hidden layer的时候，要把每个单词的error都传回去，这样需要更新的output vector非常多，也就是相当于需要更新整个weight matrix，参数更新的时候计算量也比较大。

## 1.2 Skip-Gram

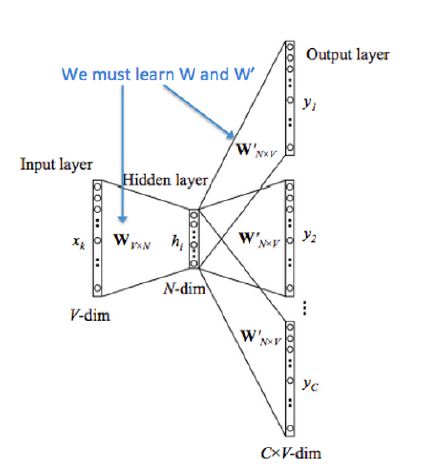


图3

同样的这个skip gram图是目前最常用的图，但是里面的表示会和我们的用法有些冲突，所以重新绘制一下skip gram：



图4

Skip gram 是用当前词预测其上下文的单词的出现的概率，使得真实的上下文的单词出现的概率最大。

We could breakdown the skip gram model in these steps:

1. Generate one hot word vector x.
2. Get our embedded words vectors for the context, vc = Vx
3. Since there is no average, just set = vc
4. Generate 2m score vectors: uc-m, …, uc-1, uc+1, …, uc+m using u = U
5. turn each score vector into probability distribution by softmax: y = softmax(u)
6. we desire our probability vector generated to match the true probability which is yc-m, …, yc-1, yc+1, …, yc+m, the one hot vector of the actual output.

对于skip-gram model我们的目标是在给定一个单词的基础上使得预测上下文单词的概率最大，所以我们的objective function为：

(5)

同CBOW一样，我们在计算中通常将使得概率最大化转换为使得cross entropy最小化，所以上面的公式通过cross entropy可以表示为（针对一个训练样本）：

（这里理解为cross entropy可能会有点费解，因为要在给定context的基础上获得预测多个上下文单词的概率，而我们常用的cross entropy只对应一个样本。这个时候可以把多个单词的上下文当做一个训练样本，这样就符合常见的cross entropy的形式了，还可以看公式的最后一步的写法，把每个上下文单词的cross entropy求和作为总的cross entropy。同样，也可以理解为将log似然函数（概率最大）取反，这样就要使得取反的log似然函数最小化最为优化目标函数）

(6)

上面的公式中，因为存在一个 1到V的求和计算，所以导致每个训练样本的计算复杂度都非常的高，所以在实际应用中都是采用hierarchary softmax 或者negative sampling。

这个公式同上面提到的CBOW Model的优化公式一样，要计算整个单词表的输出概率，这样计算复杂度较高，另外因为在神经网络计算中output layer计算了单词表的每个单词的输出概率，每个单词的输出概率的计算都依赖于hidden layer 的值，所以在back propagation的时候每个单词的error都需要back propagate回去，这就相当于一个完全的network weight matrix update。

## 1.3 Negative Sampling

因为在公式（4）和（6）中，每个样本对应的计算的工作量比较大，每个样本都需要计算所有的词典词的概率，所以大神们提出了Negative Sampling来简化计算复杂度，这就要求我们改变模型的一些参数，特别是需要改变模型的objective function，然后根据变化了得objective function进行参数更新。

暂时我们假定Negative Sampling基于skip -gram model, 那么对于一个训练样本，我们就有对应的2m个单词属于正样本，同时我们也可以进行随机采样，获取2m个负样本，这个时候我们想要做的就是训练模型使得给定当前单词的情况下获得正样本的概率最大，同时使得获取负样本的概率最小。 (We build a new objective function that tries to maxmize the probability of a word and contenxt being in the corpus data if it indeed in, and maximize the probability of a word and context not being in the corpus data if it indeed is not)

一个样本属于正样本的概率为：

(7)

同样，一个样本属于负样本的概率为：

(8)

对于Negative Sampling我们的objective function为：

(9)

公式中表示正样本集合，为负样本集合。

从上面的公式中，我们可以得到对于一个训练样本，使得得到正样本属于正样本与负样本属于负样本的概率最大化，我们对应的新的目标函数 （objective function）为：

(10)

同样的在训练的过程中，依旧采用cross entropy作为cost function, 这样我们的计算目的就转换为使得cross entropy最小化。这里先给出基本的cross entropy的公式，下面会有详细的参数更新的说明。

(11)

# 2 Parameter update of word2vector

本章主要介绍word2vector中的参数更新算法，因为CBOW和skip-gram都属于比较简单的network，所以参数更新过程比较简单。（不知道为毛这么简单的network被有些资料分类到deep learning中）

## 2.1 one word context

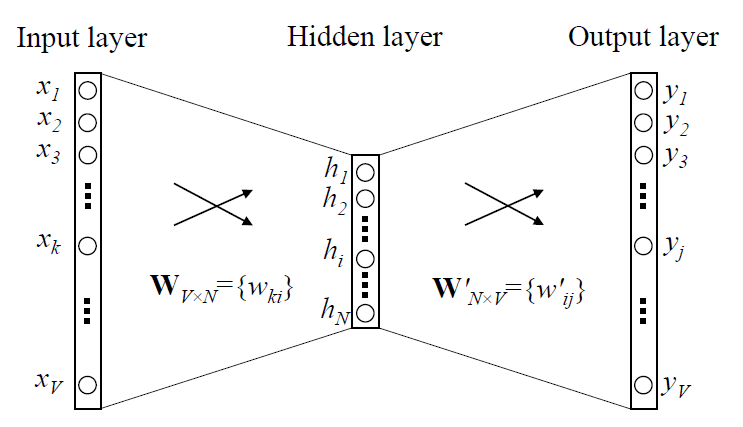


图5

这种情况下，我们的神经网络对应的是最基本的神经网络，给定输入，存在一个hidden layer，然后计算得到输出。同样上面这个图的变量的标记不是符合常用的表示，我们重新标记：



图6

输入X为one hot vector, 输出对应一个词的概率分布，这个时候神经网络的hidden state为：

(12)

这里h()表示当前单词wi的input vector （为V的第j列）, 有了hidden state, 我们可以计算得到一个概率分布，即神经网络的输出值：

(13)

公式中为词的output vector (为U的第j行), 为了得到一个概率分布，对神经网络的输出值进行softmax即可。

(14)

1. update equation for hidden-output weights

我们的目标函数为：

(15)

我们将使得概率最大cost function转换为cross entropy(针对一个样本):

(16)

Let compute the derivative of the Error E with regard to the output of output layer:

(17)

注意看公式17的第二部分，其实第二部分就是一个softmax，可以表示为 , 所以公式17可以继续简化为：

(18)

公式18中的为训练样本的ground truth. 哇塞，神奇出现了，这不就是squared error的derivative么，哈哈，可以看出机器学习其实本来就是一家。

我们把公式18矩阵化一下：

(19)

因为word2vec的output layer为linear的，所以我们不需要计算the derivative of the error with regard to input of the output layer.这个derivative和是一样的。同样的hidden layer也是linear layer，所以不需要单独计算input and output derivative.

Now let’s compute the derivative of the error with regard to the weights from hidden layer to output layers:

(20)

上面公式20中，, 所以公式20可以继续展开：

(21)

矩阵化公式21得到：

(22)

特别注意啦，单词的output vector更新来啦，请注意，我们的矩阵U为output weight, 那么U的维度为V\*H, 也就是说U的每一列都对应一个单词的output vector.

(23)

其中为learning rate, 以后就不赘述了。

刚才说了，每一行都对应一个output word vector, 所以如果将上面的参数更新公式表示为行向量，则为:

(24)

上面公式中表示word wi对应的output vector。公式24意味着对于U，每一行都需要进行更新。

Now let compute the derivative of the error with regard to the hidden layer:

(25)

因为，所以上面的公式可以继续简化：

(26)

将公式26矩阵化得到：

(27)

然后，let compute the derivative of the error with regard to the weight V from input to hidden layer.

(28)

(29)

我们得到input weight V的更新公式：

(30)

将公式29矩阵化得到：

(31)

请注意，V即为word input vector, V的每一列都表示对应的 word的input vector, 这个input vector也是最终我们要获取的word embedding，我们可以把公式30的更新，重新表示为针对一个word的更新，也就是针对V的一列的更新，注意看公式31中，x 为 one hot vector, 其中只有一个元素为1， 所以也就对应了实际上只更新了一列，其他的列（也就是其他的word对应的embeddings没有收到影响），所以针对一个单词的V的更新，可以表示为：

(32)

上面公式中表示输入单词wi对应的input word vector,即为V的第wi列，也就是我们最终要计算的得到wi的word vector.

目前为止，我们已经得到了context只有一个单词的word2vec参数更新的公式，下面将把这些参数更新的公式应用到CBOW和skip-gram model中，如果你还没有明白目前的公式推导，那么最好还是再重新看几遍。

## 2.2 CBOW Model

CBOW model与上面提到的one word context 的公式，目标函数以及参数推导基本一致，只不过是变化了context的计算方式为：= (vc-m +vc-m+1 +….+vc+m) / 2m， 之前只是使用一个word作为contenxt， 然后就没有然后了，其他的都是木有变化的。

Objective function还是：

(33)

我们将使得概率最大cost function转换为cross entropy(针对一个样本):

(34)

然后计算各种derivative of error with regard to XXX, 然后我们得到对于某个word wj的oupput vector的更新公式，同公式24一样：

(35)

然后，同公式32一样，我们可以得到word wi的input vector 的更新公式，但是，这里要特别注意，因为我们的context是多个单词，所以error传播回去，要对每一个context单词都进行vector更新，也就是说对于每个context里面的单词对应的word vector(vc-m,vc-m+1,….,vc+m)都要进行更新，这个时候我们要将derivative of error平分给每个input word对应的word vector，更新公式为：

(36)

其中 C 为Context的大小，表示context中第c个单词。

## 2.3 skip-gram Model

Skip-gram model是用一个输入的单词，预测上下文中的单词的出现的概率。所以对应的= vc, vc=Vx.

因此，skip-gram的cost function如公式6所示，这里再次给出cost function公式：

(37)

因为在skip-gram中，需要预测多个单词（context单词）的出现概率，使得context单词的出现概率最大，所以我们对每个要预测的单词分别计算error derivative （即对分别计算每个单词的error derivative，然后将他们的error derivative加起来作为整个训练样本context的error derivative）

公式37可以继续简化为：

(38)

Now lets compute the derivative of error with regard to the output of output layer:

(39)

注意啦，这个公式和CBOW中的基本上一样，只不过公式中多个符号c来表示当前的单词是在context中的。

然后将所有上下文中的单词的derivative of error累加起来获得当前训练样本的derivative of error:

(40)

然后，我们就可以和（a）中一样的计算各种derivative并且获得weight更新公式了：

(41)

有了公式41，我们可以将41应用到24中，获得output vector更新公式：

(42)

然后，

有了则可以进一步计算word input vector的更新公式，同样应用我们已经得到的公式36可以得到：

(43)

到此为止，我们就获得了skip-gram的参数更新公式。然后接下来将要介绍如果减少计算复杂度以及在减少复杂度的基础上如果重新定义cost function以及根据新的cost function推算参数更新公式。

为了写这个总结，已经使出了[洪荒之力]了，非法转载者死全家 ☺。

# 3 Optimizing Computational Efficiency

因为在公式34中存在一个softmax的计算，要计算所有的V个单词的output probability 的概率分布，这样对每个训练样本都要重新计算一遍这个概率分布，会导致计算量非常的大。所以就有一些大神用了其他的方法来降低计算复杂度。大神们提出了多个简化计算模型，这里我们主要关注Hierarchical Softmax和Negative Sampling。

# 3.1 Hierarchical Softmax

### 3.1.1 Basic Theory of Hierarchical softmax

Hierarchical Softmax （这里我们简称HS）是将整个词表按照单词的频率构造了一颗Huffman树，这样保证能够实现一个按照单词频率的最小编码。

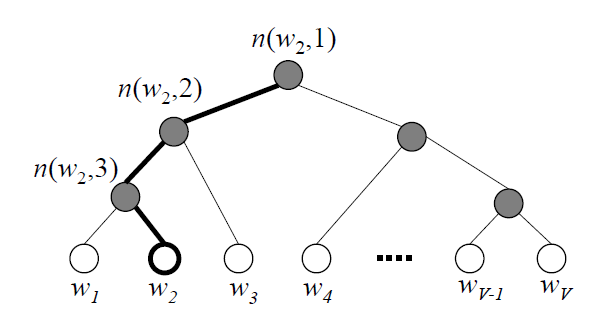


图7

HS的原理可以简单的理解为从hidden layer沿着一个huffman tree的路径找到对应的单词，我们的目标就是使得沿着路径找到这个单词的概率最大。这个时候，就可以将这个路径理解为多个logistic regression, 然后将多个logistic regression的概率相乘，作为得到target 单词的概率，我们的目标就是使得这个概率最大。所以一些资料上说可以把HS理解为multi logistic regression.

整个神经网络的结构如下(结合了Hierarchical Softmax)：

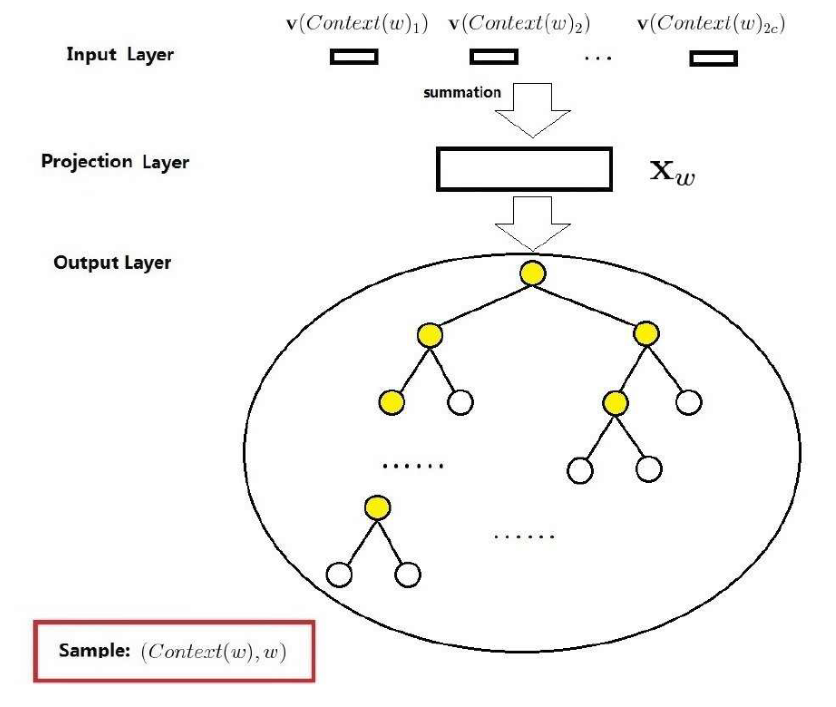


图8

让我们来继续看图7，我们有一些基本的规定：

1. Huffman tree 一共有V个叶子结点， V-1个非叶子结点 （inner node）
2. 每个叶子结点对应一个单词，每个非叶子结点对应一个output vector，但是特别注意这个时候这个output vector不再对应到某个单词，仅仅是相当于神经网络的weight。
3. 每个非叶子结点对应的vector可以表示为: , 其中j表示从根结点开始到word w的第j个非叶子结点
4. 因为是Huffman tree，所以我们可以将选择左子树作为正分类，选择右子树作为父分类。我们从hidden layer通过一个path走到对应的word要进行多次分类，将多次分类的概率相乘，就是我们得到对应单词的概率，也就是我们要优化的目标。
5. 我们用L(w)表示到达对应单词的路径长度，也就是我们要进行多少次二分类。
6. 对于CBOW, , 对于skip-gram model,

对于HS来说，我们的训练目标是是的hidden state通过L(w)次分类获得对应的单词的概率最大。

这样，我们在HS中得到正分类的概率为：

获得负样本分类的概率为：

通过上面两个公式，我们整理得到获得单词的概率为（多次分类概率乘积）：

(44)

其中, 表示到达w的路径中第j+1个结点的选择，表示到达w的路径中第j个结点的左孩子结点。也就是说，从根结点开始，如果到达w的路径走了左子树，那么我们就当成一个正样本选择，如果路径选择了右子树，那么我们就把这次二分类当成一个负样本选择。然后使得整个路径的选择概率最大。

### 3.1.2 Parameter update of Hierarchical softmax

然后就要进行目标函数优化了, 我们将公式44作为objective function，同样的可以将公式44转换为cross entropy来使得cross entropy最小化进行参数更新。

HS 不用像传统的那样要计算softmax，只需要计算path上的node的logistic regression, 这一方面降低的计算复杂度。

(45)

有了cost function， 就可以开始求导了。

(46)

特别注意啦，这里不再是compute the derivative of the error with regard to the output of output layer了，而是针对huffman tree的inner node的二分类的概率值进行求导，因为我们要跟新的值就是inner node对应的output vector, 然后叶子结点没有任何实际上的意思。

(47)

根据公式47，huffman tree中的inner node对应的output vector的更新公式为：

(48)

公式48应该应用到所有的到达w的路径结点上， i=1,2,…,L(w)-1。

如果是skip-gram model, 那么应该对每一个output context word重复公式48的更新过程。

在参数更新上，HS也不用更新每个inner node的output vector, 只需要更新path node的vector即可，在这一方面，HS又提升了速度。

在HS中，我们不用像传统的CBOW那样计算一个output layer的softmax，只需要计算从huffman tree 根结点沿着path走到对应的单词wi的路径上经过的点的logistic regression，这个path的最大长度也就是logV，所以在计算复杂度上有明显提升。另外，因为没有计算每个单词的概率，所以不需要将每个单词的error 反向传播回去，只需要将从根结点走到wi的路径上遇到的huffman tree的内部结点的error反向传播回去，这样我们只需要反向传播L(w)-1个error，也就是说在更新经过结点的内部vector即可，不用像常规network一样需要更新整个weight matrix from hidden layer to output layer. 哈哈，肿么样，有木有感觉很神奇。

Now lets compute the derivative of the error with regard to the hidden layer:

(49)

得到EH之后，对于CBOW模型而言我们就可以将EH带入公式36:

来计算input word vector的更新了。上面给出了公式36的形式。

对于skip-gram模型，因为一个训练样本对应多个context words，所以我们要分别计算每个context word的EH, 然后将每个单词对应的EH求和获得skip-gram model的EH, 然后将EH带入公式公式43进行input word vector更新，公式43如下所示：

# 3.2 Negative Sampling

Negative sampling简单来说就是给定训练样本的前提下，随机的进行采样，将随机采样得到的N个样本作为负样本，我们的训练目标就是使得network的输出对于正样本的概率尽可能的大，对于负样本的概率尽可能的小。

(We build a new objective function that tries to maxmize the probability of a word and contenxt being in the corpus data if it indeed in, and maximize the probability of a word and context not being in the corpus data if it indeed is not)

对于多个正样本和多个负样本，objective function为：

对于每个样本而言，objective function为：

Negative sampling不用像传统方法一样计算每个字典单词的概率(softmax)，而只用计算当前input word以及对应的负样本的分类概率，即logistic regression，这样就很大程度上降低了计算复杂度。

我们使用cross entropy来进行参数更新：

(50)

然后进行求导：

(51)

(52)

然后我们可以获得对于Negaive sampling的output vector的参数更新公式：

(53)

在进行output vector更新的时候，negative sampling 不用更新所有的output vectors, 只需要更新当前的正样本和负样本集合对应的output vectors即可，这样也极大的降低了计算复杂度。

这里面和U交叉使用，可能会比较混乱，其实都是一个意思，都是output vector。公式53只应用到这个比较小的集合中，而不是应用到所有的词典单词中。

对于negative sampling，我们也不用计算单词表中每个单词的输出概率，而是将输出概率转换为了分类概率，我们只需要计算一个比较小的结合的单词的分类概率，使得属于正样本的概率尽可能的大，属于负样本的概率尽可能的小。这样可以降低计算复杂度。在参数更新的时候，因为没有采用所有的单词，只是采用了一个比较小的集合的单词，所以只需要计算这个小的集合的单词的分类误差，然后将这个小的集合的derivative of error 反向传播回去。也就是说在参数更新的时候我们不需要更新一个完整的weight matrix from hidden layer to output layer，只需要更新小集合的每个单词对应的output vector，吼吼，这样就又降低了计算复杂度。

哈哈，这里是不是有点熟悉，我们又获得了EH， 同样和HS差不多，把EH带入到之前已经计算得到的公式就可以计算input word vector的更新公式啦。

对于CBOW model, 把EH带入公式36可以得到context words的input vector更新。

对于skip-gram model, 我们要分别计算每个预测的context word的EH, 然后将EH相加得到一个总的EH, 然后将总的EH代入公式43中更新word *wi*的input vector.