Word2vector Summarization

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

1 Basic Introduction of word2vector

1.1 CBOW

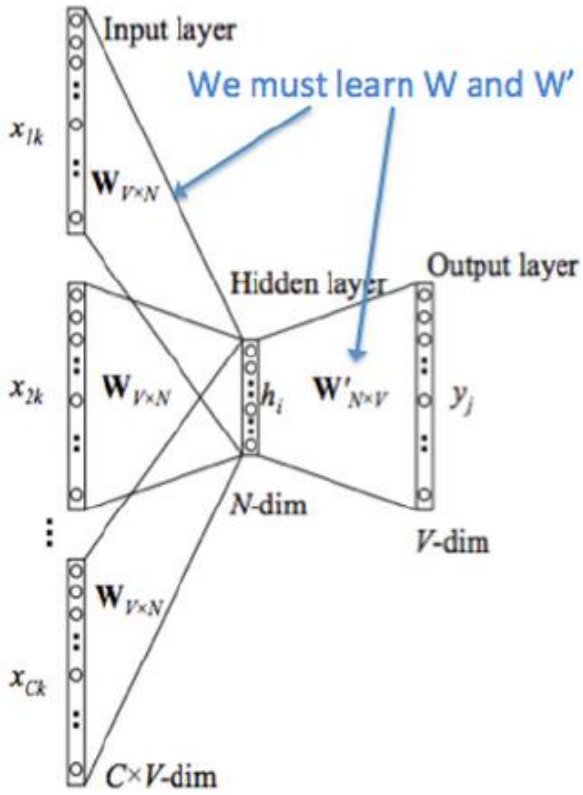

图 1

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：

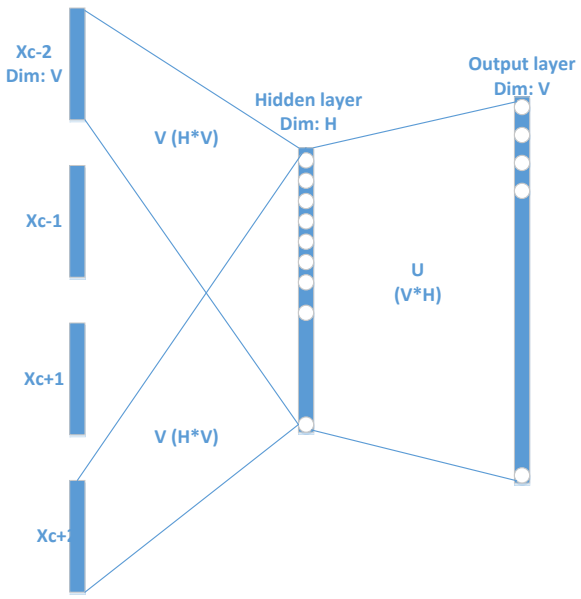

图 2

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：

- W_i: word i from vocab V
- V : (dim: H * V) input word matrix
- v_i : (dim: H) i-th column of V, the input vector representation of word w_i
- U: (dim: V*H), output word matrix
- u_i : (dim: H), ith row of U, the output vector representation of word w_i
- H: is the hidden size, and also is the dim of word embeddings
- Input X_{c-1} is a one hot vector with dimension of V

Then CBOW model works in these steps:

1. Generate one hot word vectors (x^{c-m},…,x^{c-1},x^{c+1},…,x^{c+m}) for the input context size of m.
2. Get our embedded words vectors for the context,
    a. $v_{c-m} = Vx^{c-m}$
    b. $v_{c-m+1} = Vx^{c-m+1}$
    c. …
    d. $v_{c+m} = Vx^{c+m}$
3. average these vector to get $\check{v}$= (v_{c-m} +v_{c-m+1} +….+v_{c+m}) / 2m
4. generate a score vector z = U$\check{v}$
5. turn the score vector into probability distribution by softmax: $\hat{y}$ = softmax(z)

CBOW 的 objective function 为：

$$J = \frac{1}{T}\sum_{t=1}^{T} logP(w_t|Context)$$
$$= \frac{1}{T}\sum_{t=1}^{T} logP(w_t|\check{v})$$

(1)

使得给定一个 Context 获得 w_t 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：

$$H(y,\hat{y}) = -\sum_{i=1}^{V} \log y_i(\hat{y}_i)$$

(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：

$$H(y,\hat{y}) = -\log y_i(\hat{y}_i) \quad (3)$$

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(w_c|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)

$$minimize(J) = -\log P(w_c|w_{c-m},\dots,w_{c-1},w_{c+1},\dots,w_c)$$
$$= -logP(u_c|\check{v})$$
$$= -log\frac{\exp(u_c^T\check{v})}{\sum_{j=1}^{V}\exp(u_j^T\check{v})}$$
$$= -u_c^T\check{v} + log\sum_{j=1}^{V}\exp(u_j^T\check{v})$$

(4)

上面公式(4)中$\check{v}$表示当前对应 w_t 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

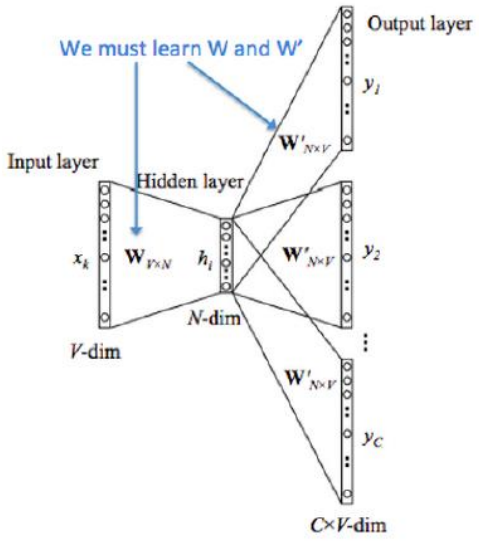

图 3

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：

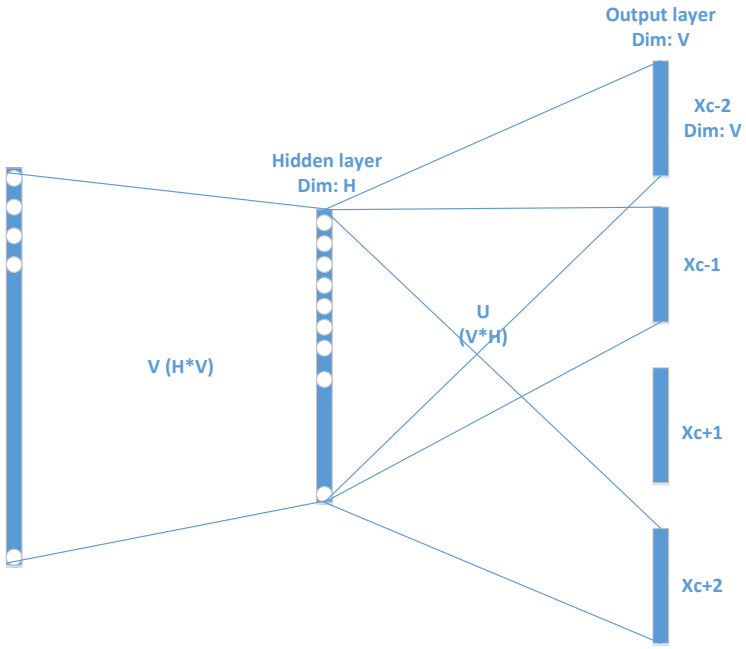

图 4

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, $v_c = Vx$
3. Since there is no average, just set $\check{v} = v_c$
4. Generate 2m score vectors: $u_{c-m}, \ldots, u_{c-1}, u_{c+1}, \ldots, u_{c+m}$ using $u = U\check{v}$
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 $y^{c-m}, \ldots, y^{c-1}, y^{c+1}, \ldots, y^{c+m}$, 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：

$$objective\ function = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j!=0} logP(w_{t+j}|w_t) \tag{5}$$

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）

$$\begin{aligned}
&minimize(J) = -\log \mathrm{P}(w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}|w_c) \\
&= -\log \prod_{j=0, j!=m}^{2m} P(w_{c-m+j}|w_c) \\
&= -\log \prod_{j=0, j!=m}^{2m} P(u_{c-m+j}|v_c) \\
&= -\log \prod_{j=0, j!=m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{V}\exp(u_k^T v_c)} \\
&= -\sum_{j=0, j!=m}^{2m} u_{c-m+j}^T v_c + 2m log \sum_{k=1}^{V}\exp(u_k^T v_c)
\end{aligned} \tag{6}$$

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：

$$\mathrm{P}(\mathrm{D}=1|\mathrm{w},\mathrm{c},\theta)=\frac{1}{1+e^{(-v_c^T v_w)}} \tag{7}$$

同样，一个样本属于负样本的概率为：

$$\begin{aligned}&\mathrm{P}(\mathrm{D}=0|\mathrm{w},\mathrm{c},\theta)=1-\mathrm{P}(\mathrm{D}=1|\mathrm{w},\mathrm{c},\theta)\\&=1-\frac{1}{1+e^{(-v_c^T v_w)}}\end{aligned} \tag{8}$$

对于 Negative Sampling 我们的 objective function 为：

$$\begin{aligned}\theta&=argmax\prod_{(w,c)\in D}P(D=1|w,c,\theta)\prod_{(w,c)\in\tilde{D}}P(D=0|w,c,\theta)\\&=argmax\prod_{(w,c)\in D}P(D=1|w,c,\theta)\prod_{(w,c)\in\tilde{D}}\big(1-P(D=1|w,c,\theta)\big)\\&=argmax\sum_{(w,c)\in D}logP(D=1|w,c,\theta)+\sum_{(w,c)\in\tilde{D}}log\big(1-P(D=1|w,c,\theta)\big)\\&=argmax\sum_{(w,c)\in D}log\frac{1}{1+\exp(-u_w^T v_c)}+\sum_{(w,c)\in\tilde{D}}log\left(1-\frac{1}{1+\exp(-u_w^T v_c)}\right)\end{aligned}$$

$$= argmax \sum_{(w,c)\in D} log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \widetilde{D}} log\left(\frac{1}{1+\exp(u_w^T v_c)}\right)$$

(9)

公式中D表示正样本集合，$\widetilde{D}$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：

$$J = \log\delta\left(u_{c-m+j}^T v_c\right) + \sum_{k=1}^{K} \log\delta\left(-u_k^T v_c\right)$$

(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。

$$\begin{aligned}&\text{cross entropy} = \mathrm{E} = -\mathrm{J}\\&= -[\log\delta\left(u_{c-m+j}^T v_c\right) + \sum_{k=1}^{K} \log\delta\left(-u_k^T v_c\right)]\end{aligned}$$

(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

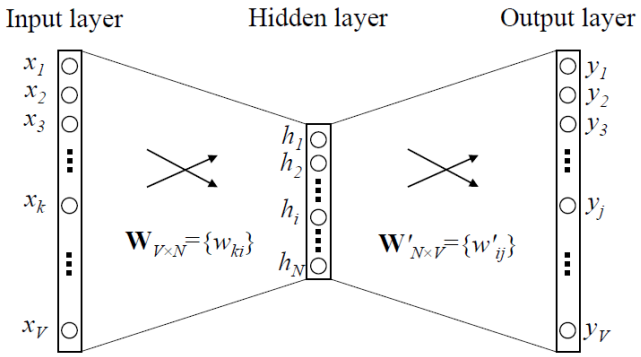

图 5

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：

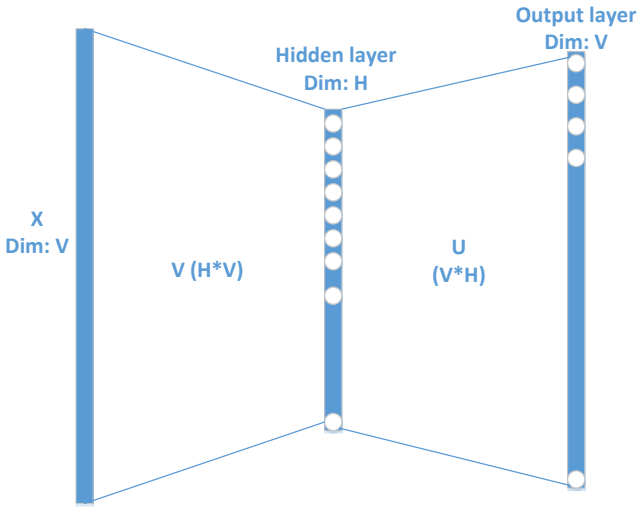

图 6

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：

$$\mathrm{h} = \mathrm{Vx} = V_{w_i} \tag{12}$$

这里 h(V_{w_i})表示当前单词 w_i 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：

$$u_j = U_j^T h \tag{13}$$

公式中U_j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。

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{i=1}^{V}\exp(u_i)} \tag{14}$$

(A) update equation for hidden-output weights

我们的目标函数为：

$$\begin{aligned} &maxP(w_O|w_I) = maxy_{j*} \\ &= maxlogy_{j*} \\ &= maxlogP(u_{j*}|\check{v}) \\ &= maxlog\frac{\exp(u_{j*}^T\check{v})}{\sum_{j=1}^{V}\exp(u_j^T\check{v})} \\ &= u_{j*}^T\check{v} - log\sum_{j=1}^{V}\exp(u_j^T\check{v}) = -E \end{aligned} \tag{15}$$

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

$$\begin{aligned} \mathrm{E} &= -u_{j*}^T\check{v} + log\sum_{j=1}^{V}\exp(u_j^T\check{v}) \\ &= -y_j + log\sum_{k=1}^{V}\exp(y_k) \end{aligned} \tag{16}$$

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

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j}(-y_j + log\sum_{k=1}^{V}\exp(y_k))$$
$$= -1 + \frac{\partial}{\partial y_j}(log\sum_{k=1}^{V}\exp(y_k))$$
$$= -1 + \frac{\exp(y_j)}{\sum_{k=1}^{V}\exp(y_k)}$$

(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 y_j, 所以公式 17 可以继续简化为：

$$\frac{\partial E}{\partial y_j} = -1 + y_j$$
$$= y_j - t_j = e_j$$

(18)

公式 18 中的t_j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：

$$\frac{\partial E}{\partial y} = y - t$$

(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$\frac{\partial E}{\partial y}$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:

$$\frac{\partial E}{\partial U_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial U_{ij}}$$
$$= (y_i - t_i)\frac{\partial y_i}{\partial U_{ij}}$$

(20)

上面公式 20 中，$y_i = \sum_{j=1}^{H} U(i,j)h_j$, 所以公式 20 可以继续展开：

$$\frac{\partial E}{\partial U_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial U_{ij}}$$

$$= (y_i - t_i)\frac{\partial y_i}{\partial U_{ij}}$$
$$= (y_i - t_i)h_j = e_i h_j$$

(21)

矩阵化公式 21 得到：

$$\frac{\partial E}{\partial U} = \frac{\partial E}{\partial y}H^T$$

(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.

$$U(i,j) = U(i,j) - \alpha(y_i - t_i)h_j$$

(23)

其中α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:

$$U_{w_i} = U_{w_i} - \alpha(y_i - t_i)H$$

(24)

上面公式中U_{w_i}表示 word w_i 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:

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V}\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial h_i}$$

(25)

因为$y_j = \sum_{i=1}^{H} U(j,i)h_i$，所以上面的公式可以继续简化：

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V}\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial h_i}$$
$$= \sum_{j=1}^{V}(y_i - t_i)\,U(j,i)$$
$$= \mathrm{EH}_i$$

(26)

将公式 26 矩阵化得到：

$$\frac{\partial E}{\partial h} = (U)^T \frac{\partial E}{\partial y}$$

(27)

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

$$h_i = \sum_{j=1}^{V} V(i,j) x_j$$

(28)

$$\begin{aligned}
\frac{\partial E}{\partial V_{i,j}} &= \frac{\partial E}{\partial h_i} \frac{\partial h_i}{\partial V_{i,j}} \\
&= \sum_{j=1}^{V} (y_i - t_i)\, U(j,i) \frac{\partial h_i}{\partial V_{i,j}} \\
&= \mathrm{EH}_i x_j \\
&= \sum_{j=1}^{V} (y_i - t_i)\, U(j,i) x_j
\end{aligned}$$

(29)

我们得到 input weight V 的更新公式：

$$\begin{aligned}
V_{i,j} &= V_{i,j} - \alpha \sum_{j=1}^{V} (y_i - t_i)\, U(j,i) x_j \\
&= V_{i,j} - \alpha \mathrm{EH}_i x_j
\end{aligned}$$

(30)

将公式 29 矩阵化得到：

$$\frac{\partial E}{\partial V} = \frac{\partial E}{\partial h} x^T$$

(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$\frac{\partial E}{\partial V}$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：

$$V_{w_i} = V_{w_i} - \alpha \frac{\partial E}{\partial h}$$

(32)

上面公式中V_{w_i}表示输入单词 w_i 对应的 input word vector,即为 V 的第 w_i 列，V_{w_i} 也就是我们最终要计算的得到 w_i 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：$\check{v} = (v_{c-m} + v_{c-m+1} + \ldots + v_{c+m}) / 2m$ ，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：

$$\begin{aligned} & maxP(w_O|w_{I,1}, w_{I,2}, \ldots, w_{I,c}) = maxy_{j*} \\ & = maxlogy_{j*} \\ & = maxlogP(u_{j*}|\check{v}) \\ & = maxlog\frac{\exp(u_{j*}^T\check{v})}{\sum_{j=1}^{V}\exp(u_j^T\check{v})} \\ & = u_{j*}^T\check{v} - log\sum_{j=1}^{V}\exp(u_j^T\check{v}) = -E \end{aligned}$$

(33)

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

$$\begin{aligned} \mathrm{E} = & -u_{j*}^T\check{v} + log\sum_{j=1}^{V}\exp(u_j^T\check{v}) \\ & = -y_j + log\sum_{k=1}^{V}\exp(y_k) \end{aligned}$$

(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：

$$U_{w_j} = U_{w_j} - \alpha(y_j - t_j)H$$

(35)

然后，同公式 32 一样，我们可以得到 word w_i 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(v_{c-m},v_{c-m+1},....,v_{c+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：

$$V_{w_{I,c}} = V_{w_{I,c}} - \alpha \frac{1}{C} \frac{\partial E}{\partial h} \tag{36}$$

其中 C 为 Context 的大小，$w_{I,c}$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$\check{v}$= v_c, v_c=Vx.

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

$$\begin{aligned}
&minimize(J) = -\log \mathrm{P}(w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m} | w_c) \\
&= -\log \prod_{j=0, j!=m}^{2m} P(w_{c-m+j} | w_c) \\
&= -\log \prod_{j=0, j!=m}^{2m} P(u_{c-m+j} | v_c) \\
&= -\log \prod_{j=0, j!=m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{V} \exp(u_k^T v_c)} \\
&= -\sum_{j=0, j!=m}^{2m} u_{c-m+j}^T v_c + 2m log \sum_{k=1}^{V} \exp(u_k^T v_c)
\end{aligned} \tag{37}$$

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 $w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}$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：

$$\mathrm{E} = -\sum_{j=0, j!=m}^{2m} y_j + 2m log \sum_{k=1}^{V} \exp(y_k) \tag{38}$$

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

$$\frac{\partial E}{\partial y_{c,j}} = -1 + y_{c,j}$$
$$= y_{c,j} - t_{c,j}$$
$$= e_{c,j}$$

(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:

$$\frac{\partial E}{\partial y_j} = \sum_{c=1}^{C} e_{c,j} = EI_j$$

(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：

$$\frac{\partial E}{\partial U_{ij}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial U_{ij}}$$
$$= \sum_{c=1}^{C} e_{c,i}\frac{\partial y_i}{\partial U_{ij}}$$
$$= EI_i h_j$$

(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：

$$U_{w_i} = U_{w_i} - \alpha EI_i H$$

(42)

然后，

$$\frac{\partial E}{\partial h} = (U)^T \frac{\partial E}{\partial y}$$

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial h_i}$$
$$= \sum_{j=1}^{V} (y_i - t_i)\, U(j,i)$$
$$= \mathrm{EH}_i$$

有了$\frac{\partial E}{\partial h}$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：

$$V_{w_I} = V_{w_I} - \alpha \frac{\partial E}{\partial h} \tag{43}$$

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。

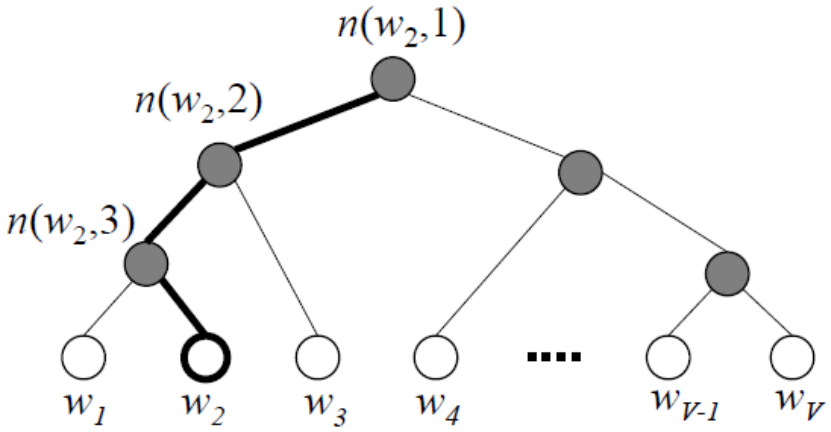

图 7

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)：

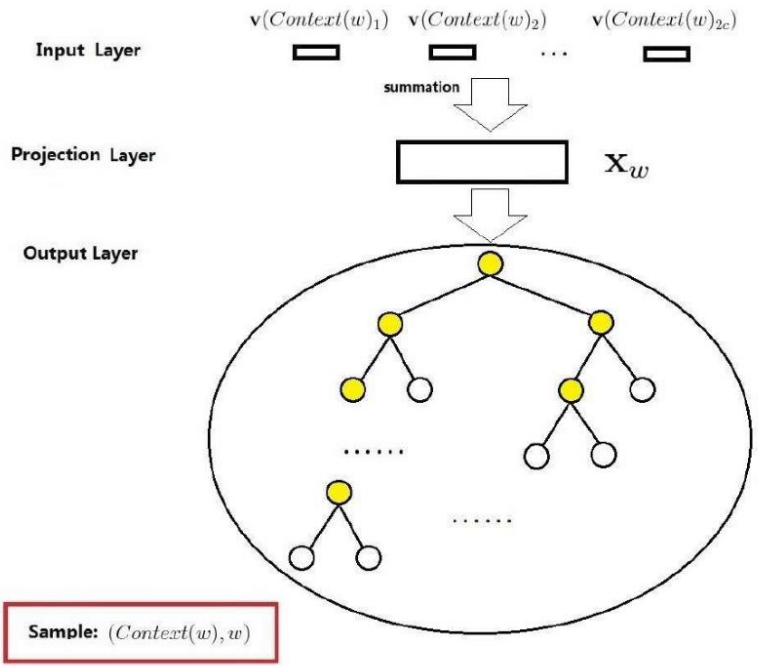

图 8

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: $V'_{n(w,j)}$, 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, $h = \frac{1}{C}\sum_{c=1}^{C} V_{w_c}$, 对于 skip-gram model, $h = V_{w_c}$

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：

$$p(n, left) = \sigma(V'_{n(w,j)}\mathrm{h})$$

获得负样本分类的概率为：

$$p(n,right) = 1 - \sigma\left(V'_{n(w,j)}\mathrm{h}\right) = \sigma\left(-V'_{n(w,j)}\mathrm{h}\right)$$

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）：

$$p(w_O|context) = \prod_{i=1}^{L(w)-1} \sigma([\![n(w,j+1) = ch(n(w,j))]\!]V'_{n(w,i)}\mathrm{h})$$

(44)

其中$[\![x]\!] = 1\ if\ x\ is\ true\ or\ 0\ is\ x\ if\ false, n(w,j+1)$表示到达 w 的路径中第 j+1 个结点的选择，$ch(n(w,j)$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。

$$\begin{aligned}
\mathrm{E} &= -logp(w_O|context) \\
&= -log \prod_{i=1}^{L(w)-1} \sigma([\![n(w,j+1) = ch(n(w,j))]\!]V'_{n(w,i)}\mathrm{h}) \\
&= -\sum_{i=1}^{L(w)-1} log\sigma([\![n(w,j+1) = ch(n(w,j))]\!]V'_{n(w,i)}\mathrm{h}) \\
&= -\sum_{i=1}^{L(w)-1} log\sigma\left([\![\cdot]\!]V'_{n(w,i)}\mathrm{h}\right)
\end{aligned}$$

(45)

有了 cost function， 就可以开始求导了。

$$\begin{aligned}
&\frac{\partial E}{\partial V'_{n(w,i)}\mathrm{h}} = -\left(1 - \sigma\left([\![\cdot]\!]V'_{n(w,i)}\mathrm{h}\right)\right)\frac{\partial [\![\cdot]\!]V'_{n(w,i)}\mathrm{h}}{\partial V'_{n(w,i)}\mathrm{h}} \\
&= \left(\sigma\left([\![\cdot]\!]V'_{n(w,i)}\mathrm{h}\right) - 1\right)[\![\cdot]\!] \\
&= \begin{cases} \sigma\left(V'_{n(w,i)}\mathrm{h}\right) - 1 & if \quad [\![\cdot]\!] = 1 \\ \sigma\left(V'_{n(w,i)}\mathrm{h}\right) & if \quad [\![\cdot]\!] = 0 \end{cases} \\
&= \sigma\left(V'_{n(w,i)}\mathrm{h}\right) - t_i
\end{aligned}$$

(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。

$$\begin{aligned}\frac{\partial E}{\partial V'_{n(w,i)}} &= \frac{\partial E}{\partial V'_{n(w,i)}\mathrm{h}}\frac{\partial V'_{n(w,i)}\mathrm{h}}{\partial V'_{n(w,i)}} \\ &= \left(\sigma\left(V'_{n(w,i)}\mathrm{h}\right) - t_i\right)\frac{\partial V'_{n(w,i)}\mathrm{h}}{\partial V'_{n(w,i)}} \\ &= \left(\sigma\left(V'_{n(w,i)}\mathrm{h}\right) - t_i\right)h\end{aligned} \tag{47}$$

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

$$V'_{n(w,i)} = V'_{n(w,i)} - \alpha\left(\sigma\left(V'_{n(w,i)}\mathrm{h}\right) - t_i\right)h \tag{48}$$

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 w_i 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:

$$\begin{aligned}\frac{\partial E}{\partial h} &= \sum_{j=1}^{L(w)-1}\frac{\partial E}{\partial V'_{n(w,i)}\mathrm{h}}\frac{\partial V'_{n(w,i)}\mathrm{h}}{\partial h} \\ &= \sum_{j=1}^{L(w)-1}\frac{\partial E}{\partial V'_{n(w,i)}\mathrm{h}}V'_{n(w,i)} \\ &= EH\end{aligned} \tag{49}$$

得到 EH 之后，对于 CBOW 模型而言我们就可以将 EH 带入公式 36:

$$V_{w_{I,c}} = V_{w_{I,c}} - \alpha\frac{1}{C}\frac{\partial E}{\partial h}$$

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：

$$V_{w_I} = V_{w_I} - \alpha \frac{\partial E}{\partial h}$$

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：

$$\begin{aligned}
\theta &= argmax \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta) \\
&= argmax \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} \big(1-P(D=1|w,c,\theta)\big) \\
&= argmax \sum_{(w,c)\in D} logP(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} log\big(1-P(D=1|w,c,\theta)\big) \\
&= argmax \sum_{(w,c)\in D} log\frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} log\left(1-\frac{1}{1+\exp(-u_w^T v_c)}\right) \\
&= argmax \sum_{(w,c)\in D} log\frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} log\left(\frac{1}{1+\exp(u_w^T v_c)}\right)
\end{aligned}$$

对于每个样本而言，objective function 为：

$$J = \log\delta\big(u_{c-m+j}^T v_c\big) + \sum_{k=1}^{K} \log\delta\big(-u_k^T v_c\big)$$

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：

$$\begin{aligned}&\text{cross entropy} = \mathrm{E} = -\mathrm{J}\\&= -[\log\delta(u_{c-m+j}^{T}v_c) + \sum_{k=1}^{K}\log\delta(-u_k^{T}v_c)]\\&= -log\delta(V'^{T}_{wo}h) - \sum_{w_i\in Wneg}\log\delta(-V'^{T}_{w_i}h)]\end{aligned} \tag{50}$$

然后进行求导：

$$\begin{aligned}&\frac{\partial E}{\partial V'^{T}_{w_i}h} = \begin{cases}\delta(V'^{T}_{w_i}h) - 1 & if\ w_i = w_O\\ \delta(V'^{T}_{w_i}h) & if\ w_i \in Wneg\end{cases}\\&= \delta(V'^{T}_{w_i}h) - t_i\end{aligned} \tag{51}$$

$$\begin{aligned}&\frac{\partial E}{\partial V'_{w_i}} = \frac{\partial E}{\partial V'^{T}_{w_i}h}\frac{\partial V'^{T}_{w_i}h}{\partial V'_{w_i}}\\&= (\delta(V'^{T}_{w_i}h) - t_i)\mathrm{h}\end{aligned} \tag{52}$$

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

$$U_{w_j} = U_{w_j} - \alpha(\delta(V'^{T}_{w_i}h) - t_i)\mathrm{h} \tag{53}$$

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V'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$w_j \in \{w_O\} \cup Wneg$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。

$$\frac{\partial E}{\partial h} = \sum_{w_j \in \{w_O\} \cup Wneg} \frac{\partial E}{\partial {V'}_{w_i}^T h} \frac{\partial {V'}_{w_i}^T h}{\partial h}$$
$$= \sum_{w_j \in \{w_O\} \cup Wneg} \left(\delta\left({V'}_{w_i}^T h\right) - t_i\right) \frac{\partial {V'}_{w_i}^T h}{\partial h}$$
$$= \sum_{w_j \in \{w_O\} \cup Wneg} \left(\delta\left({V'}_{w_i}^T h\right) - t_i\right) {V'}_{w_i}$$
$$= EH$$

(54)

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 w_i 的 input vector.

4. 参数更新公式推导 2（另外一种公式推导）

本章给出了在 CBOW 和 Skip-gram model 分别应用 Hierarchical Softmax 和 Negative Sampling 优化技术下公式推导的过程。

4.1 Hierarchical Softmax

本部分给出了 Hierarchical softmax 在 CBOW 和 skip-gram model 中应用时候参数更新的公式推导。不懂 HS 的请重新阅读第三章。

4.1.1 CBOW

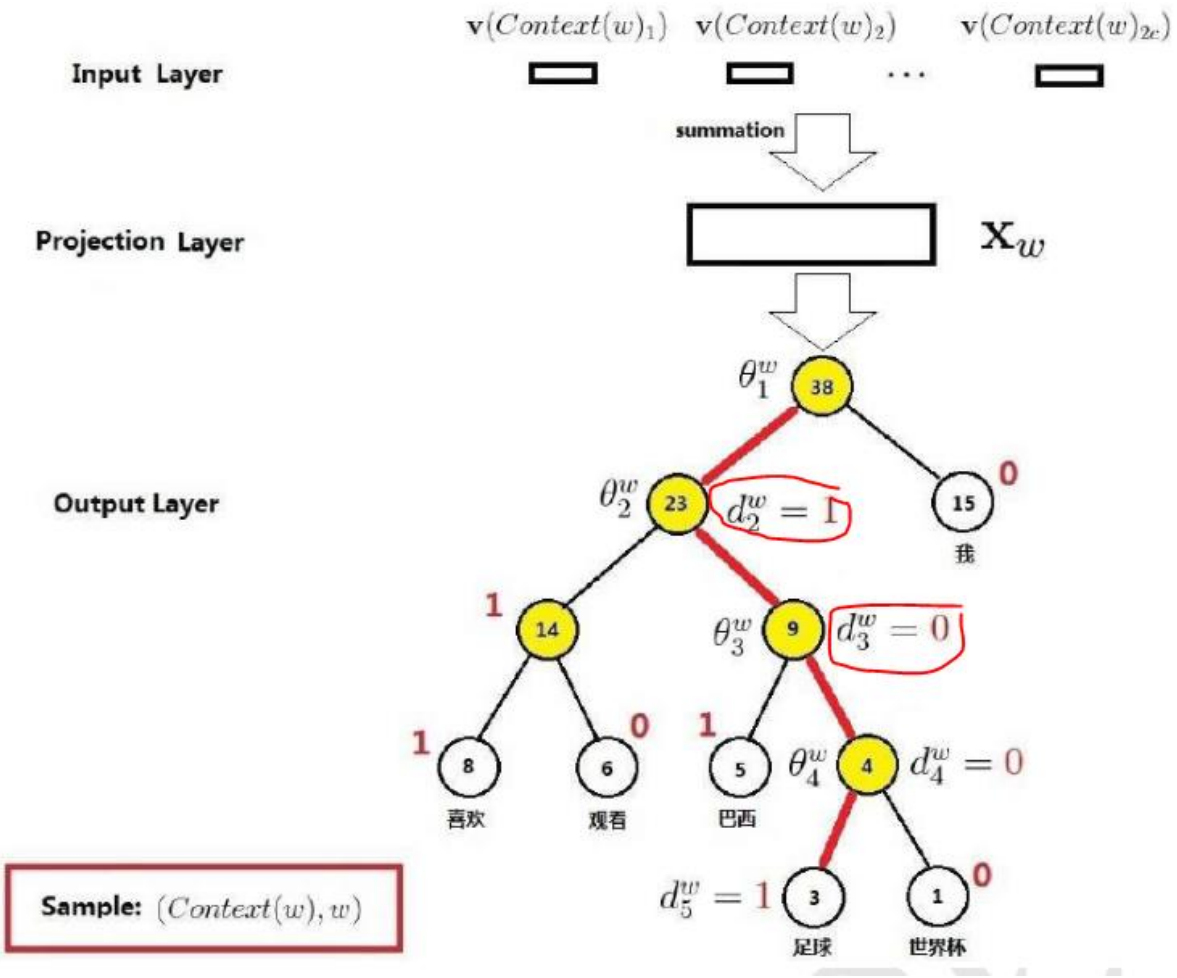

图 9

下面推导中，用的公式符号和图 9 中稍微有些不同，为的是让公式符号和全文一致。

CBOW model 中，我们的 Hierarchical softmax 的目的是通过根结点进行多次二分类，然后使得沿着二分类的路径到达我们对应的单词的概率最大，所以我们的 cost function 是一个全局概率最大化。首先我们看对于一个训练样本的 cost function (公式 44):

$$p(w_O|context) = \prod_{i=1}^{L(w)-1} \sigma([\![n(w,j+1) = ch(n(w,j))]\!] V'_{n(w,i)}\mathrm{h})$$

上面公式中，有一个符号函数，之前在推导参数更新的时候，我们直接包含了符号函数，同时我们也可以将符号函数展开写成一个完全的公式，这个时候让我们假设在进行二分类的时候选择左子树为正样本，选择右子树为负样本，让我们定义一些新的符号来标记我们在进行二分类的时候是选择了正样本还是负样本分类，让d_j^w表示当前 context 为 w 的情况下，第 j 次二分类的符号，如果选择左子树，那么$d_j^w = 1$，如果选择右子树则$d_j^w = 0$，所以进行一次二分类的概率计算可以重新写为：

$$p\left(d_j^w | h, V'_{n(w,j-1)}\right) = [\sigma(V'_{n(w,j-1)}\mathrm{h})]^{d_j^w} * [1 - \sigma(V'_{n(w,j-1)}\mathrm{h}]^{1-d_j^w} \tag{55}$$

然后，对于一个训练样本，CBOW model 下的 Hierarchical softmax 的 cost function 就可以重新写为：

$$p(w_O|context) = \prod_{i=2}^{L(w)} [\sigma(V'_{n(w,i-1)}\mathrm{h})]^{d_i^w} * [1 - \sigma(V'_{n(w,i-1)}\mathrm{h}]^{1-d_i^w} \tag{56}$$

对于整个训练样本的 cost function 则为(对每个训练样本的概率取了 log 似然函数)：

$$\begin{aligned} L &= \sum_{w \in C} log \prod_{i=2}^{L(w)} \left[\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right]^{d_i^w} * [1 - \sigma(V'_{n(w,i-1)}\mathrm{h}]^{1-d_i^w} \\ &= \sum_{w \in C} \sum_{i=2}^{L(w)} log\left[\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right]^{d_i^w} * [1 - \sigma(V'_{n(w,i-1)}\mathrm{h}]^{1-d_i^w} \\ &= \sum_{w \in C} \sum_{i=2}^{L(w)} [d_i^w * \log\left(V'_{n(w,i-1)}\mathrm{h}\right) + (1 - d_i^w) * \log(1 - \sigma(V'_{n(w,i-1)}\mathrm{h})] \end{aligned} \tag{57}$$

注意，公式 57 的目标是我们要使得L尽可能的大，同样为了计算方便我们可以将 log 似然函数取反，这样就变成了 cross entropy 了，我们就可以将目标转换为使得 cross entropy 尽可能的小。

针对 cross entropy, 上面的公式可以写为：

$$\begin{aligned} cross\ entropy &= \sum_{w \in C} -log \prod_{i=2}^{L(w)} \left[\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right]^{d_i^w} * [1 - \sigma(V'_{n(w,i-1)}\mathrm{h}]^{1-d_i^w} \\ &= -\sum_{w \in C} \sum_{i=2}^{L(w)} log\left[\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right]^{d_i^w} * [1 - \sigma(V'_{n(w,i-1)}\mathrm{h}]^{1-d_i^w} \end{aligned}$$

$$= -\sum_{w\in C}\sum_{i=2}^{L(w)}[d_i^w * \log\left(\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \log(1-\sigma(V'_{n(w,i-1)}\mathrm{h})] \tag{58}$$

让我们只看中括号内部的计算：

$$l(w,i) = d_i^w * \log\left(\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \log(1-\sigma(V'_{n(w,i-1)}\mathrm{h}) \tag{59}$$

然后我们计算$l(w,i)$关于$V'_{n(w,i-1)}$的倒数：

$$\begin{aligned}
&\frac{\partial l(w,i)}{\partial V'_{n(w,i-1)}} = \frac{\partial}{\partial V'_{n(w,i-1)}}\{d_i^w * \log\left(\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \log\left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)\} \\
&= \frac{\partial}{\partial V'_{n(w,i-1)}}\{d_i^w * \log\left(\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)\} + \frac{\partial}{\partial V'_{n(w,i-1)}}\{(1-d_i^w) * \log\left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)\} \\
&= d_i^w * \left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) * h + (1-d_i^w) * \left(-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)h \\
&= \left[d_i^w * \left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \left(-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)\right]h \\
&= \left[d_i^w - \sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right]h
\end{aligned} \tag{60}$$

公式 60 中的参数更新，和 word2vec 中的代码的参数更新有一些不同，代码中为：

```
// 'g' is the gradient multiplied by the learning rate
g = (1 - vocab[word].code[d] - f) * alpha;
// Propagate errors output -> hidden
for (c = 0; c < layer1_size; c++) neu1e[c] += g * syn1[c + l2];
// Learn weights hidden -> output
for (c = 0; c < layer1_size; c++) syn1[c + l2] += g * neu1[c];
```

图 10

这是因为在 huffman tree 中我们本文中取了二分类左边为正样本，右边为负样本，左边$d_i^w = 1$, 右边$d_i^w = 0$, 哈哈哈，其实这样也可的，只是 cost function 不一样而已。

为了与代码一致，让我们稍微改一改上面的公式 55，即改为左边为负样本，右边为正样本

$$p\left(d_j^w \middle| h, V'_{n(w,j-1)}\right) = [1-\sigma(V'_{n(w,j-1)}\mathrm{h})]^{d_j^w} * [\sigma(V'_{n(w,j-1)}\mathrm{h}]^{1-d_j^w} \tag{61}$$

然后重写我们的整体的 cost function:

$$cross\ entropy = \sum_{w\in C} -log \prod_{i=2}^{L(w)} \left[1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right]^{d_i^w} * \left[\sigma(V'_{n(w,i-1)}\mathrm{h}\right]^{1-d_i^w}$$
$$= -\sum_{w\in C}\sum_{i=2}^{L(w)} log\left[1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right]^{d_i^w} * \left[\sigma(V'_{n(w,i-1)}\mathrm{h}\right]^{1-d_i^w}$$
$$= -\sum_{w\in C}\sum_{i=2}^{L(w)} \left[d_i^w * \log\left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \log(\sigma(V'_{n(w,i-1)}\mathrm{h})\right]$$

(62)

重写括号内部的表示：

$$l(w,i) = d_i^w * \log\left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \log(\sigma(V'_{n(w,i-1)}\mathrm{h})$$

(63)

用新的公式重新计算倒数，计算$l(w,i)$关于$V'_{n(w,i-1)}$的倒数：

$$\frac{\partial l(w,i)}{\partial V'_{n(w,i-1)}} = \frac{\partial}{\partial V'_{n(w,i-1)}}\left\{d_i^w * \log\left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \log\left(\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right\}\right.$$
$$= \frac{\partial}{\partial V'_{n(w,i-1)}}\left\{d_i^w * \log\left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)\right\} + \frac{\partial}{\partial V'_{n(w,i-1)}}\left\{(1-d_i^w) * \log\left(\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right\}\right.$$
$$= d_i^w * \left(-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) * h + (1-d_i^w) * \left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)h$$
$$= \left[d_i^w * \left(-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)\right]h$$
$$= \left[1-d_i^w - \sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right]h$$

(64)

这样子是不是就与代码中的参数更新一致啦~

```
// 'g' is the gradient multiplied by the learning rate
g = (1 - vocab[word].code[d] - f) * alpha;
```

所以对于 Huffman tree 中 inner node 对应的 output vector 的更新公式为：

$$V'_{n(w,i-1)} = V'_{n(w,i-1)} - \alpha\left[1-d_i^w - \sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right]h$$

(65)

对于每个训练样本(w, context)，每个到达 w 的路径上的中间节点都需要进行公式 65 的参数更新。

接下来我们计算$l(w,i)$关于h的倒数：

$$\begin{aligned}
&\frac{\partial l(w,i)}{\partial h} = \frac{\partial}{\partial h}\{d_i^w * \log\left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \log\left(\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right\} \\
&= \frac{\partial}{\partial h}\{d_i^w * \log\left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)\} + \frac{\partial}{\partial h}\{(1-d_i^w) * \log\left(\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right\} \\
&= d_i^w * \left(-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) * V'_{n(w,i-1)} + (1-d_i^w) * \left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) V'_{n(w,i-1)} \\
&= \left[d_i^w * \left(-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right) + (1-d_i^w) * \left(1-\sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right)\right] V'_{n(w,i-1)} \\
&= \left[1-d_i^w - \sigma\left(V'_{n(w,i-1)}\mathrm{h}\right)\right] V'_{n(w,i-1)}
\end{aligned}$$

(66)

因为在 huffman tree 中我们走过多个中间节点，每个节点都依赖于 hidden state h， 所以对于一个训练样本我们要把 huffman tree 中达到 word w 的路径上每个中间节点的误差都反向传播回去：

$$\begin{aligned}
&\frac{\partial E}{\partial h} = \sum_{j=2}^{L(w)} \frac{\partial l(w,j)}{\partial h} \\
&= \sum_{j=2}^{L(w)} \left[1-d_j^w - \sigma\left(V'_{n(w,j-1)}\mathrm{h}\right)\right] V'_{n(w,j-1)}
\end{aligned}$$

(67)

然后我们就可以把公式 67 带入到公式 36 中计算 input word vectors 了。哈哈

公式 36 如下：

$$V_{w_{I,c}} = V_{w_{I,c}} - \alpha \frac{1}{C} \frac{\partial E}{\partial h}$$

综上所述，对于训练样本(context(w), w)，采用 Hierarchical softmax 的 CBOW Model 的参数更新流程如下：（*采用梯度上升*）

1. $\mathbf{e} = \mathbf{0}$.
2. $\mathbf{x}_w = \sum_{u \in Context(w)} \mathbf{v}(u)$.
3. **FOR** $j = 2 : l^w$ **DO**

{

3.1 $q = \sigma(\mathbf{x}_w^\top \theta_{j-1}^w)$

3.2 $g = \eta(1 - d_j^w - q)$

3.3 $\mathbf{e} := \mathbf{e} + g\theta_{j-1}^w$

3.4 $\theta_{j-1}^w := \theta_{j-1}^w + g\mathbf{x}_w$

}

4. **FOR** $u \in Context(w)$ **DO**

{

$\mathbf{v}(u) := \mathbf{v}(u) + \mathbf{e}$

}

4.1.2 Skip-gram

我们已知 skip-gram model 是给定一个单词作为 context，然后取预测多个上下文单词，这个其实和采用 HS 的 CBOW 区别不大，只是可以将预测每个上下文单词独立开来，将每个单词的概率乘积作为 cost function.

$$objective\ function = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c \le j \le c, j!=0} logP(w_{t+j}|w_t)$$

按照 Hierarchical softmax 的思想，P(w|u)可以类似公式 56 写为：

$$p(w|u) = \prod_{j=2}^{L(w)} p\left(d_j^w | h, V'_{n(w,j-1)}\right)$$

所以，采用 hierarchical softmax 的 skip-gram 的 cost function 可以写为(log 似然函数)：

$$L = \sum_{w \in C} log \prod_{u \in context(w)} p(u|w)$$
$$= \sum_{w \in C} log \prod_{u \in context(w)} \prod_{i=2}^{L(u)} p\left(d_j^u | h, V'_{n(u,j-1)}\right)$$

同样的我们可以将 cost function 改为 cross entropy，只需要取一个负号即可，这里我们同样和 CBOW 中一样，左边为负样本，右边为正样本：

$$
\begin{aligned}
cross\ entropy &= \sum_{w\in C}\sum_{u\in context(w)} -log\prod_{i=2}^{L(u)}\left[1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right]^{d_i^u} * \left[\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right]^{1-d_i^u} \\
&= -\sum_{w\in C}\sum_{u\in context(w)}\sum_{i=2}^{L(u)} log\left[1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right]^{d_i^u} * \left[\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right]^{1-d_i^u} \\
&= -\sum_{w\in C}\sum_{u\in context(w)}\sum_{i=2}^{L(u)} \{d_i^u * log\left[1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right] + (1-d_i^u)\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)
\end{aligned}
\tag{68}
$$

让我们只看括号中的部分：

$$
l(w,u,i) = d_i^u * \log\left(1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right) + (1-d_i^u) * \log\left(\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right.
$$

计算$l(w,u,i)$关于$V'_{n(u,i-1)}$的导数：

$$
\begin{aligned}
&\frac{\partial l(w,u,i)}{\partial V'_{n(u,i-1)}} = \frac{\partial}{\partial h}\{d_i^u * \log\left(1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right) + (1-d_i^u) * \log\left(\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right\} \\
&= \frac{\partial}{\partial V'_{n(u,i-1)}}\{d_i^u * \log\left(1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right)\} + \frac{\partial}{\partial h}\{(1-d_i^u) * \log\left(\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right\} \\
&= d_i^u * \left(-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right) * h + (1-d_i^u) * \left(1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right)h \\
&= \left[d_i^u * \left(-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right) + (1-d_i^u) * \left(1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right)\right]h \\
&= \left[1-d_i^u-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right]h
\end{aligned}
\tag{69}
$$

我擦，是不是发现这个公式和公式 64 是一样的，所以 output vector 的更新公式为：

$$
V'_{n(u,i-1)} = V'_{n(u,i-1)} - \alpha\left[1-d_i^u-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right]h
$$

然后计算$l(w,u,i)$关于 h 的倒数：

$$
\begin{aligned}
&\frac{\partial l(w,u,i)}{\partial h} = \frac{\partial}{\partial h}\{d_i^u * \log\left(1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right) + (1-d_i^u) * \log\left(\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right\} \\
&= \frac{\partial}{\partial h}\{d_i^u * \log\left(1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right)\} + \frac{\partial}{\partial h}\{(1-d_i^u) * \log\left(\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right\} \\
&= d_i^u * \left(-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right) * V'_{n(u,i-1)} + (1-d_i^u) * \left(1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right)V'_{n(u,i-1)} \\
&= \left[d_i^u * \left(-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right) + (1-d_i^u) * \left(1-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right)\right]V'_{n(u,i-1)} \\
&= \left[1-d_i^u-\sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right]V'_{n(u,i-1)}
\end{aligned}
\tag{69}
$$

因为 skip-gram model 是用一个单词作为上下文，然后预测多个该单词的上下文单词，所以每个上下文单词的 error 都应该 back propagate 回去，所以计算关于 h 的偏导数的时候要将每个预测的上下文单词的 error 累加起来：

$$\frac{\partial E}{\partial h} = \sum_{u \in context(w)} \sum_{j=2}^{L(u)} \frac{\partial l(u,j)}{\partial h}$$
$$= \sum_{u \in context(w)} \sum_{j=2}^{L(u)} \left[1 - d_i^u - \sigma\left(V'_{n(u,i-1)}\mathrm{h}\right)\right] V'_{n(u,i-1)} \tag{70}$$

所以，有了$\frac{\partial E}{\partial h}$之后，我们就可以把$\frac{\partial E}{\partial h}$带入公式 43 计算 input vector 更新了。公式 43 如下：

$$V_{w_I} = V_{w_I} - \alpha \frac{\partial E}{\partial h}$$

综上所述，给定样本(w, context(w))的情况下，采用 hierarchical softmax 的 skip-gram model 的参数更新过程为（图中为采用梯度上升，与描述算法略有不同）：

$\mathbf{e} = \mathbf{0}$
FOR $u \in Context(w)$ **DO**
{
  FOR $j = 2 : l^u$ **DO**
  {
    1. $q = \sigma(\mathbf{v}(w)^\top \theta^u_{j-1})$
    2. $g = \eta(1 - d^u_j - q)$
    3. $\mathbf{e} := \mathbf{e} + g\theta^u_{j-1}$
    4. $\theta^u_{j-1} := \theta^u_{j-1} + g\mathbf{v}(w)$
  }
}
$\mathbf{v}(w) := \mathbf{v}(w) + \mathbf{e}$

然而，在实际 word2vec 的代码中，又有一些其他的不同：

但是, word2vec 源码中, 并不是等 $Context(w)$ 中的所有词都处理完后才刷新 $\mathbf{v}(w)$, 而是, 每处理完 $Context(w)$ 中的一个词 u, 就及时刷新一次 $\mathbf{v}(w)$, 具体为

FOR $u \in Context(w)$ **DO**
{
  $\mathbf{e} = 0$
  FOR $j = 2 : l^u$ **DO**
  {
    1. $q = \sigma(\mathbf{v}(w)^\top \theta^u_{j-1})$
    2. $g = \eta(1 - d^u_j - q)$
    3. $\mathbf{e} := \mathbf{e} + g\theta^u_{j-1}$
    4. $\theta^u_{j-1} := \theta^u_{j-1} + g\mathbf{v}(w)$
  }
  $\mathbf{v}(w) := \mathbf{v}(w) + \mathbf{e}$
}

4.2 Negative Sampling

本小节主要介绍采用 negative sampling 优化方法的 CBOW 和 Skip-gram model 如何进行参数更新。

4.2.1 CBOW

Negative Sampling 这里就不赘述了，不懂的请重头再看一遍，☺.

给定训练样本(context(w), w), 我们可以 negative sampling 获得一个针对当前训练样本的正样本和负样本集合 C，获得集合 C 中的单词 u 的概率可以表示为：

$$p(u|context(w)) = \begin{cases} \delta(V'^T_{w_u} h) & if\ L(u) = 1 \\ 1 - \delta(V'^T_{w_u} h) & if\ L(u) = 0 \end{cases} \tag{71}$$

其中$L(u) = 1$表示单词 u 为正样本，$L(u) = 0$表示单词 u 为负样本。

将公式 71 写成整体表达式为：

$$p(u|context(w)) = [\delta(V'^T_{w_u} h)]^{L(u)} * [1 - \delta(V'^T_{w_u} h)]^{1-L(u)} \tag{72}$$

这样，对于一个训练样本，cost function 可以表达为：

$$g(w) = \delta\left(V'^{T}_{w_w} h\right) \prod_{u \in Neg(w)} 1 - \delta\left(V'^{T}_{w_u} h\right)$$

(73)

公式 73 的目的就是增大正样本的概率同时降低负样本的概率，对于 corpus C，cost function 为：

$$cost\ funciton = \sum_{w \in C} g(w)$$

$$= \sum_{w \in C} \delta\left(V'^{T}_{w_w} h\right) \prod_{u \in Neg(w)} 1 - \delta\left(V'^{T}_{w_u} h\right)$$

(74)

同样啦，我们取 log 似然函数，如果转换为 cross entropy 的话，将 log 似然函数取负号即可

$$cross\ entropy = \sum_{w \in C} -\log[g(w)]$$

$$= -\sum_{w \in C} \log\left[\prod_{u \in \{w\} \cap Neg(w)} \left[\delta\left(V'^{T}_{w_u} h\right)\right]^{L(u)} * \left[1 - \delta\left(V'^{T}_{w_u} h\right)\right]^{1-L(u)}\right]$$

$$= -\sum_{w \in C} \sum_{u \in \{w\} \cap Neg(w)} log\left[\delta\left(V'^{T}_{w_u} h\right)\right]^{L(u)} * \left[1 - \delta\left(V'^{T}_{w_u} h\right)\right]^{1-L(u)}$$

$$= -\sum_{w \in C} \sum_{u \in \{w\} \cap Neg(w)} \{L(u) * \log\left[\delta\left(V'^{T}_{w_u} h\right)\right] + \left(1 - L(u)\right) * \log[1 - \delta\left(V'^{T}_{w_u} h\right)]\}$$

(75)

让我们 focus 在括号中的内容：

$$l(w,u) = L(u) * \log\left[\delta\left(V'^{T}_{w_u} h\right)\right] + \left(1 - L(u)\right) * \log[1 - \delta\left(V'^{T}_{w_u} h\right)]$$

(76)

计算$l(w,u)$对于V'_{w_u}的偏导数：

$$\frac{\partial l(w,u)}{\partial V'_{w_u}} = \frac{\partial}{\partial V'_{w_u}} \{L(u) * \log\left[\delta\left(V'^{T}_{w_u} h\right)\right] + \left(1 - L(u)\right) * \log\left[1 - \delta\left(V'^{T}_{w_u} h\right)\right]\}$$

$$= L(u) * \left(1 - \delta\left(V'^{T}_{w_u} h\right)\right) * h + \left(1 - L(u)\right) * \left(-\delta\left(V'^{T}_{w_u} h\right)\right) * h$$

$$= \left(L(u) * \left(1 - \delta\left(V'^{T}_{w_u} h\right)\right) + \left(1 - L(u)\right) * \left(-\delta\left(V'^{T}_{w_u} h\right)\right)\right) h$$

$$= \left[L(u) - \delta\left(V'^{T}_{w_u} h\right)\right] h$$

(77)

所以 output vector 的更新公式为：

$$V'_{w_u} = V'_{w_u} - \alpha\left[L(u) - \left(V'^{T}_{w_u} h\right)\right] h$$

(78)

然后计算$l(w,u)$关于 h 的偏导数：

$$\begin{aligned}
&\frac{\partial l(w,u)}{\partial h} = \frac{\partial}{\partial h}\left\{L(u) * \log\left[\delta\left(V'^{T}_{w_u} h\right)\right] + \left(1 - L(u)\right) * \log\left[1 - \delta\left(V'^{T}_{w_u} h\right)\right]\right\} \\
&= L(u) * \left(1 - \delta\left(V'^{T}_{w_u} h\right)\right) * V'_{w_u} + \left(1 - L(u)\right) * \left(-\delta\left(V'^{T}_{w_u} h\right)\right) * V'_{w_u} \\
&= \left(L(u) * \left(1 - \delta\left(V'^{T}_{w_u} h\right)\right) + \left(1 - L(u)\right) * \left(-\delta\left(V'^{T}_{w_u} h\right)\right)\right) V'_{w_u} \\
&= \left[L(u) - \delta\left(V'^{T}_{w_u} h\right)\right] V'_{w_u}
\end{aligned}$$

(79)

因为 Negative Sampling 针对一个样本（Context(w), w）我们构造一个小的正样本和负样本的集合 C，每个集合 C 中的 word u 都依赖于 h，所以要把 C 中每个单词的 error back propagate 回去：

$$\frac{\partial E}{\partial h} = \sum_{u \in \{w\} \cap Neg(w)} \frac{\partial l(w,u)}{\partial h}$$

(80)

然后我们就可以把公式 80 带入到公式 36 中进行 input vector 更新了。

综上所述，基于 negative sampling 的 CBOW 的参数更新流程为：

1. $\mathbf{e} = \mathbf{0}$.
2. $\mathbf{x}_w = \sum_{u \in Context(w)} \mathbf{v}(u)$.
3. **FOR** $u = \{w\} \cup NEG(w)$ **DO**
   {
   3.1 $q = \sigma(\mathbf{x}_w^{\top} \theta^u)$
   3.2 $g = \eta(L^w(u) - q)$
   3.3 $\mathbf{e} := \mathbf{e} + g\theta^u$
   3.4 $\theta^u := \theta^u + g\mathbf{x}_w$
   }
4. **FOR** $u \in Context(w)$ **DO**
   {
   $\mathbf{v}(u) := \mathbf{v}(u) + \mathbf{e}$
   }

4.2.2 Skip-gram

Skip-gram model 是给定一个单词作为 context, 然后预测这个单词的上下文单词出现的概率，其中我们可以把预测每个单词的概率看做是独立事件，这样 skip-gram 的预测就可以写为：

$$\mathrm{P}(w_{c-m},\dots,w_{c-1},w_{c+1},\dots,w_{c+m}|w_c) = \prod_{j=0,j!=m}^{2m} P\left(w_{c-m+j}\middle|w_c\right)$$

Negative Sampling 的目标为使得正样本的概率尽可能的大，同时使得负样本的概率尽可能的小，所以给定一个训练样本(w, context(w))，cost function 为：

$$g(w) = \prod_{u\in\{w\}\cap Neg(w)} p(u|w)$$

上一节中我们知道整体的概率表达式为：

$$p\left(u\middle|context(w)\right) = [\delta\left(V'^{T}_{w_u}h\right)]^{L(u)} * [1-\delta\left(V'^{T}_{w_u}h\right)]^{1-L(u)}$$

所以 g(w)可以写为：

$$\begin{aligned} g(u) &= \prod_{z\in\{w\}\cap Neg(w)} p(z|w) \\ &= \prod_{z\in\{w\}\cap Neg(w)} \{[\delta\left(V'^{T}_{w_z}h\right)]^{L(z)} * [1-\delta\left(V'^{T}_{w_z}h\right)]^{1-L(z)}\} \end{aligned} \tag{81}$$

然后我们整个 corpus 的 cost function 为：

$$\begin{aligned} L &= \sum_{w\in C}\prod_{u\in\{context(w)\}} g(u) \\ &= \sum_{w\in C}\prod_{u\in\{context(w)\}}\prod_{z\in\{w\}\cap Neg(w)} \{[\delta\left(V'^{T}_{w_z}h\right)]^{L(z)} * [1-\delta\left(V'^{T}_{w_z}h\right)]^{1-L(z)}\} \end{aligned} \tag{82}$$

也就是说对于预料 C 中的每个单词 w，都要计算 w 的每个上下文单词 u 的一个正样本和负样本集合，然后计算单词 u 在正样本和负样本集合中的 cost。

同样我们可以将公式转换为 log 似然函数，也可以进一步转换为 cross entropy，对于每个样本，如果转换为 log 似然函数，则为(log 应该加在每个样本的概率之前)：

$$L = \sum_{w\in C} log \prod_{u\in\{context(w)\}} g(u)$$

同时，如果我们继续转换为 cross entropy:

$$
\begin{aligned}
L &= \sum_{w \in C} -log \prod_{u \in \{context(w)\}} g(u) \\
&= -\sum_{w \in C} log \prod_{u \in \{context(w)\}} g(u) \\
&= -\sum_{w \in C} \sum_{u \in \{context(w)\}} \log[g(u)] \\
&= -\sum_{w \in C} \sum_{u \in \{context(w)\}} \log \left[\prod_{z \in \{u\} \cap Neg(wu)} \left\{ \left[\delta\left(V'^{T}_{w_z} h\right) \right]^{L(z)} * \left[1 - \delta\left(V'^{T}_{w_z} h\right) \right]^{1-L(z)} \right\} \right] \\
&= -\sum_{w \in C} \sum_{u \in \{context(w)\}} \sum_{z \in \{u\} \cap Neg(wu)} log \left\{ \left[\delta\left(V'^{T}_{w_z} h\right) \right]^{L(z)} * \left[1 - \delta\left(V'^{T}_{w_z} h\right) \right]^{1-L(z)} \right\} \\
&= -\sum_{w \in C} \sum_{u \in \{context(w)\}} \sum_{z \in \{u\} \cap Neg(u)} \{L(z) * \log\left(\delta\left(V'^{T}_{w_z} h\right)\right) + \left(1 - L(z)\right) * \log(1 - \delta\left(V'^{T}_{w_z} h\right))\}
\end{aligned}
$$

(83)

然而，word2vec 中并没有基于公式 83 进行计算 cost function，因为公式 83 中，每个训练样本的 context word u 都计算一个这样本和负样本集合，这样导致计算量比较大，所以在 code 中实际上是一个训练样本 w, 只针对单词 w 进行了负采样，这样就对于一个训练样本只需要进行一次负采样即可，本质上还是 CBOW 模型，只是 context 就是一个单词 w。这个时候 cost function 为：

$$
\begin{aligned}
L &= \sum_{w \in C} -log \prod_{u \in \{context(w)\}} g(u) \\
&= -\sum_{w \in C} log \prod_{u \in \{context(w)\}} g(u) \\
&= -\sum_{w \in C} \sum_{u \in \{context(w)\}} \log[g(u)] \\
&= -\sum_{w \in C} \sum_{u \in \{context(w)\}} \log \left[\prod_{z \in \{w\} \cap Neg(w)} \left\{ \left[\delta\left(V'^{T}_{w_z} h\right) \right]^{L(z)} * \left[1 - \delta\left(V'^{T}_{w_z} h\right) \right]^{1-L(z)} \right\} \right] \\
&= -\sum_{w \in C} \sum_{u \in \{context(w)\}} \sum_{z \in \{w\} \cap Neg(w)} log \left\{ \left[\delta\left(V'^{T}_{w_z} h\right) \right]^{L(z)} * \left[1 - \delta\left(V'^{T}_{w_z} h\right) \right]^{1-L(z)} \right\} \\
&= -\sum_{w \in C} \sum_{u \in \{context(w)\}} \sum_{z \in \{w\} \cap Neg(w)} \{L(z) * \log\left(\delta\left(V'^{T}_{w_z} h\right)\right) + \left(1 - L(z)\right) * \log(1 - \delta\left(V'^{T}_{w_z} h\right))\}
\end{aligned}
$$

(84)

让我们 focus 在括号中的计算：

$$l(w,z) = L(z) * \log\left(\delta\left({V'}_{w_z}^{T} h\right)\right) + \left(1 - L(z)\right) * \log(1 - \delta\left({V'}_{w_z}^{T} h\right))$$

计算$l(w,z)$关于$\partial {V'}_{w_z}$的导数：

$$\begin{aligned}
&\frac{\partial l(w,z)}{\partial {V'}_{w_z}} = \frac{\partial}{\partial {V'}_{w_z}}\left\{L(z) * \log\left(\delta\left({V'}_{w_z}^{T} h\right)\right) + \left(1 - L(z)\right) * \log\left(1 - \delta\left({V'}_{w_z}^{T} h\right)\right)\right\} \\
&= L(z)\left(1 - \delta\left({V'}_{w_z}^{T} h\right)\right) h + \left(1 - L(z)\right)\left(-\delta\left({V'}_{w_z}^{T} h\right)\right) h \\
&= \left[L(z)\left(1 - \delta\left({V'}_{w_z}^{T} h\right)\right) + \left(1 - L(z)\right)\left(-\delta\left({V'}_{w_z}^{T} h\right)\right)\right] h \\
&= \left[L(z) - \delta\left({V'}_{w_z}^{T} h\right)\right] h
\end{aligned} \tag{85}$$

我们可以获得 output vector 的更新公式：

$${V'}_{w_z} = {V'}_{w_z} - \alpha\left[L(z) - \left({V'}_{w_z}^{T} h\right)\right] h \tag{86}$$

计算$l(w,z)$关于 h 的导数:

$$\begin{aligned}
&\frac{\partial l(w,z)}{\partial h} = \frac{\partial}{\partial h}\left\{L(z) * \log\left(\delta\left({V'}_{w_z}^{T} h\right)\right) + \left(1 - L(z)\right) * \log\left(1 - \delta\left({V'}_{w_z}^{T} h\right)\right)\right\} \\
&= L(z)\left(1 - \delta\left({V'}_{w_z}^{T} h\right)\right) \partial {V'}_{w_z} + \left(1 - L(z)\right)\left(-\delta\left({V'}_{w_z}^{T} h\right)\right) \partial {V'}_{w_z} \\
&= \left[L(z)\left(1 - \delta\left({V'}_{w_z}^{T} h\right)\right) + \left(1 - L(z)\right)\left(-\delta\left({V'}_{w_z}^{T} h\right)\right)\right] \partial {V'}_{w_z} \\
&= \left[L(z) - \delta\left({V'}_{w_z}^{T} h\right)\right] \partial {V'}_{w_z}
\end{aligned} \tag{87}$$

因为 negative sampling 中的正样本集合和负样本集合的 error 都依赖于 h，所以要将所有的 error 都 back propagate 回去，

$$\frac{\partial E}{\partial h} = \sum_{u \in \{w\} \cap Neg(w)} \frac{\partial l(w,u)}{\partial h}$$

所以可以获得 input vector 的更新公式：

$$V_{w_I} = V_{w_I} - \alpha \frac{\partial E}{\partial h}$$

综上所述，给定样本(w, context(w))的情况下，基于 negative sampling 的 skip-gram 模型的参数更新流程为：

FOR $\widetilde{w} = Context(w)$ **DO**

{

$\mathbf{e} = \mathbf{0}$.

FOR $u = \{w\} \cup NEG^{\widetilde{w}}(w)$ **DO**

{

$q = \sigma(\mathbf{v}(\widetilde{w})^{\top}\theta^{u})$

$g = \eta(L^{w}(u) - q)$

$\mathbf{e} := \mathbf{e} + g\theta^{u}$

$\theta^{u} := \theta^{u} + g\mathbf{v}(\widetilde{w})$

}

$\mathbf{v}(\widetilde{w}) := \mathbf{v}(\widetilde{w}) + \mathbf{e}$

}