## 1.REVERSE THIS

1. As we known, we have

$$x = \alpha \odot z + F_\theta(u)$$
$$v = \beta \odot u$$

Thus, we can know that

$$z = (x - F_\theta(u)) \odot \frac{1}{\alpha}$$
$$u = v \odot \frac{1}{\beta}$$
$$z = (x - F_\theta(v \odot \frac{1}{\beta})) \odot \frac{1}{\alpha}$$

2. We can get the vector Jacobian

$$J = \begin{bmatrix} \frac{\partial x}{\partial z} & \frac{\partial x}{\partial u} \\ \frac{\partial v}{\partial z} & \frac{\partial v}{\partial u} \end{bmatrix}$$

Here is how we calculate $\frac{\partial x}{\partial z}$:

$$\alpha \odot z = \begin{bmatrix} \alpha_1 z_1 \\ \alpha_2 z_2 \\ \cdots \\ \alpha_i z_i \\ \cdots \\ \alpha_n z_n \end{bmatrix}$$

$$x = \begin{bmatrix} \alpha_1 z_1 + F_\theta(u) \\ \alpha_2 z_2 + F_\theta(u) \\ \cdots \\ \alpha_i z_i + F_\theta(u) \\ \cdots \\ \alpha_n z_n + F_\theta(u) \end{bmatrix}$$

As we known, $\frac{\partial x}{\partial z}$ only related to $\alpha \odot z$.

Thus, We can get the $X = \frac{\partial x}{\partial z} = \begin{bmatrix} \frac{\partial x_1}{\partial z_1} & \frac{\partial x_1}{\partial z_2} & \cdots & \frac{\partial x_1}{\partial z_n} \\ \frac{\partial x_2}{\partial z_1} & \frac{\partial x_2}{\partial z_2} & \cdots & \frac{\partial x_2}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial z_1} & \frac{\partial x_n}{\partial z_2} & \cdots & \frac{\partial x_n}{\partial z_n} \end{bmatrix}$

$When i \neq j$ , $\frac{\partial x_i}{\partial z_i} = 0$
When $i = j$ , $\frac{\partial x_i}{\partial z_i} = a_i$
Thus, $\frac{\partial x}{\partial z}$ is a diagonal matrix. When $i = j$, $X_{ij} = a_i$

$$\frac{\partial x}{\partial z} = \begin{bmatrix} a_1 & & \\ & \ddots & \\ & & a_n \end{bmatrix} = diag(a)$$

Similarly, we can know that $\frac{\partial v}{\partial u} = diag(\beta)$
And since $v$ and $z$ have no relationship, $\frac{\partial v}{\partial z} = 0$
For $\frac{\partial x}{\partial u}$, it only relates to $F_\theta(u)$. Therefore, $\frac{\partial x}{\partial u} = \frac{\partial F_\theta}{\partial u}$
Finally,

$$J = \begin{bmatrix} \frac{\partial x}{\partial z} & \frac{\partial x}{\partial u} \\ \frac{\partial v}{\partial z} & \frac{\partial v}{\partial u} \end{bmatrix} = \begin{bmatrix} diag(\alpha) & \frac{\partial F_\theta}{\partial u} \\ 0 & diag(\beta) \end{bmatrix}$$

3. Since Jacobian is a upper-triangular matrix, the determinant of the Jacobian would be the products of the elements that on the diagonal:

$$det(J) = \prod_{i=0}^{n} \alpha_i \prod_{j=0}^{h} \beta_j$$

assuming the length of $\alpha$ is $n$, the length of $\beta$ is $j$.

## 2.MAKING SKIP-GRAM TRAINING PRACTICAL

1. We assume that the word is indexed as $i$ in the dictionary, its vector is represented as $\mathbf{v}_i \in \mathbb{R}^h$ when it is the central target word, and $\mathbf{u}_i \in \mathbb{R}^d$ when it is a context word. Let the central target word $w_c$ and context word $w_o$ be indexed as $c$ and $o$ respectively in the dictionary.
The conditional probability of generating the context word for the given central target word can be obtained by performing a softmax operation on the vector inner product:

$$P(w_o \mid w_c) = \frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)}$$

Since we use cross-entropy loss, we will consider $\log P(w_o \mid w_c)$. By definition, we have

$$\log P(w_o \mid w_c) = \mathbf{u}_o^\top \mathbf{v}_c - \log \left( \sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c) \right)$$

Then we can get the gradient of the central word vector:

$$\frac{\partial \log P(w_o \mid w_c)}{\partial \mathbf{v}_c} = \mathbf{u}_o - \frac{\sum_{j \in \mathcal{V}} \exp(\mathbf{u}_j^\top \mathbf{v}_c) \mathbf{u}_j}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} = \mathbf{u}_o - \sum_{j \in \mathcal{V}} \left( \frac{\exp(\mathbf{u}_j^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} \right) \mathbf{u}_j = \mathbf{u}_o - \sum_{j \in \mathcal{V}} P(w_j \mid w_c) \mathbf{u}_j.$$

Actually, the result of the gradient of the central word vector equals to what we expected minus what we observed.
And in other words, this gradient equals to:
the context vector of $w_o$ - the sum of (the context vector of each word j $w_j$ $\times$ the possibility that word j is the context word of the center word($w_c$))
Its computation obtains the conditional probability for all the words in the dictionary given the central target word $w_c$. We then use the same method to obtain the gradients for other word vectors.

And as we can see, the gradient computation for each step contains the sum of the number of items in the dictionary size $d$. What's more, during the calculation of the conditional possibility of each word in the output layer, before the softmax, it needs to do the weighted sum and the number of weights that need to be calculated is $h$, which means the computing complexity of the conditional possibility of each word is approximately $O(h)$. Thus, we can know that the running time of calculating this gradient would approximately be $O(hd)$.

2. One way to improve the running time is ”Hierarchical Softmax”. It uses a binary tree for data structure . And this binary tree would be a Huffman tree.

The leaf nodes of the tree representing every word in the dictionary $\mathcal{V}$. And the internal nodes would act like the hidden layer units.
The root is the word vector($v_c$) of the input center word, all the leaves are similar to the output units in the softmax layer.
In this structure, we have make all the calculation that in the softmax layer into this binary tree. And we can realize the softmax calculation by finding the related path in this tree and go through it.

We assume that $L(w)$ is the number of nodes on the path (including the root and leaf nodes) from the root node of the binary tree to the leaf node of word $w$ . Let $n(w, j)$ be the $j^{th}$ node on this path,

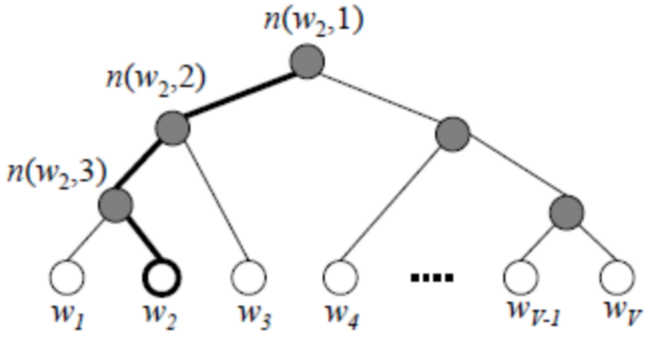

**Figure 1.** Binary Huffman Tree of Hierarchical Softmax

with the context word vector $\mathbf{u}_{n(w,j)}$.

Then we have

$$P(w_o \mid w_c) = \prod_{j=1}^{L(w_o)-1} \sigma\left( [\![ n(w_o, j+1) = \text{leftChild}(n(w_o, j)) ]\!] \cdot \mathbf{u}_{n(w_o,j)}^\top \mathbf{v}_c \right)$$

Here the $\sigma$ function is sigmoid function. And here is the rule that how should we go through the tree: if $x$ is true($[\![x]\!] = 1$) then go to the left sub-tree, else($[\![x]\!] = -1$), then go to the right sub-tree. The sigmoid function is:

$$P(+) = \sigma(x_w^T \theta) = \frac{1}{1+e^{-x_w^T \theta}}$$

Where $x_w$ is the word vector of the current internal node and $\theta$ is the parameter that we need to trained.
And

$$P(-) = 1 - P(+)$$

By comparing P(+) and P(-), we can decide to go for the left sub-tree or the right sub-tree.

For example, in the graph, if $w_2$ is a training sample output, then we will expect that: for node $n(w_2,1)$ has $P(-) > P(+)$, for node $n(w_2,2)$ has $P(-) > P(+)$, for node $n(w_2,3)$ has $P(+) > P(-)$ And the conditional possibility of $w_2$ generated based on the given central target word $w_c$ is:

$$P(w_2 \mid w_c) = \sigma(\mathbf{u}_{n(w_2,1)}^\top \mathbf{v}_c) \cdot \sigma(\mathbf{u}_{n(w_2,2)}^\top \mathbf{v}_c) \cdot \sigma(-\mathbf{u}_{n(w_2,3)}^\top \mathbf{v}_c).$$

In this way, the computation complexity decreases to $O(hlog_2 d)$ from $O(hd)$, as the height of the tree is $O(log_2 d)$, and the computing complexity of the conditional possibility of each word is approximately $O(h)$. And also since this is a Huffman tree, the word has higher word frequency would more near to the root, thus these high-frequency words would be found more faster, with shorter path to reach the leaf.

3. We can use negative sampling, which means we only train the model with the negative samples. Here is how it works:
   For example, we have one center word $w_c$, and it has $2c$ words as its context, we mark it as $context(w)$. Then we sample $K$ words $w_i, i = 1,2,3..,K$ that do not belong to the $context(w)$ . Then these $context(w)$ and $w_i$ made up a positive sample and K negative samples. By doing binary logistic regression, we can get the trained parameter of each $w_i$ and their word vectors.

Then the joint probability would be

$$\prod_{t=1}^{T} \prod_{-m\leq j\leq m,\ j\neq 0} P(w^{(t+j)} \mid w^{(t)})$$

the conditional probability is approximated to be

$$P(w^{(t+j)} \mid w^{(t)}) = P(D=1 \mid w^{(t)}, w^{(t+j)}) \prod_{k=1,\ w_k\sim P(w)}^{K} P(D=0 \mid w^{(t)}, w_k)$$

Where $P(D=1 \mid w^{(t)}, w^{(t+j)})$ is for the one positive sample, and $P(D=0 \mid w^{(t)}, w_k)$ is for the negatives samples. $P(w)$ is the distribution of the $K$ words $w_i, i = 1,2,3..,K$ that do not belong to the $context(w)$.

Let the text sequence index of word $w^{(t)}$ at time step$t$ be $i_t$ and $h_k$ for noise word $w_k$ in the dictionary. The logarithmic loss for the conditional probability above is

$$-\log P(w^{(t+j)} \mid w^{(t)}) = -\log P(D=1 \mid w^{(t)}, w^{(t+j)}) - \sum_{k=1,\ w_k\sim P(w)}^{K} \log P(D=0 \mid w^{(t)}, w_k)$$

$$= -\log \sigma\left(\mathbf{u}_{i_{t+j}}^{\top}\mathbf{v}_{i_t}\right) - \sum_{k=1,\ w_k\sim P(w)}^{K} \log\left(1-\sigma\left(\mathbf{u}_{h_k}^{\top}\mathbf{v}_{i_t}\right)\right)$$

$$= -\log \sigma\left(\mathbf{u}_{i_{t+j}}^{\top}\mathbf{v}_{i_t}\right) - \sum_{k=1,\ w_k\sim P(w)}^{K} \log \sigma\left(-\mathbf{u}_{h_k}^{\top}\mathbf{v}_{i_t}\right).$$

As we can see from the above formula,the gradient computation in each step of the training is no longer related to the dictionary size, but linearly related to $K$. When $K$ takes a smaller constant, the negative sampling has a lower computational overhead for each step.