2.1 Hierarchical Softmax

A computationally efficient approximation of the full softmax is the hierarchical softmax. In the context of neural network language models, it was first introduced by Morin and Bengio [12]. The main advantage is that instead of evaluating W output nodes in the neural network to obtain the probability distribution, it is needed to evaluate only about $\log_2(W)$ nodes.

The hierarchical softmax uses a binary tree representation of the output layer with the W words as its leaves and, for each node, explicitly represents the relative probabilities of its child nodes. These define a random walk that assigns probabilities to words.

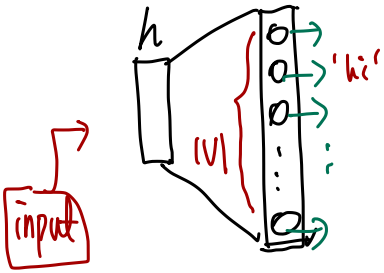
More precisely, each word w can be reached by an appropriate path from the root of the tree. Let $n(w, j)$ be the j-th node on the path from the root to w, and let $L(w)$ be the length of this path, so $n(w, 1) = \text{root}$ and $n(w, L(w)) = w$. In addition, for any inner node n, let $\text{ch}(n)$ be an arbitrary fixed child of n and let $[\![x]\!]$ be 1 if x is true and -1 otherwise. Then the hierarchical softmax defines $p(w_O|w_I)$ as follows:

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma\left([\![n(w, j+1) = \text{ch}(n(w, j))]\!] \cdot {v'_{n(w,j)}}^\top v_{w_I}\right) \tag{3}$$

where $\sigma(x) = 1/(1 + \exp(-x))$. It can be verified that $\sum_{w=1}^{W} p(w|w_I) = 1$. This implies that the cost of computing $\log p(w_O|w_I)$ and $\nabla \log p(w_O|w_I)$ is proportional to $L(w_O)$, which on average is no greater than $\log W$. Also, unlike the standard softmax formulation of the Skip-gram which assigns two representations v_w and v'_w to each word w, the hierarchical softmax formulation has one representation v_w for each word w and one representation v'_n for every inner node n of the binary tree.

The structure of the tree used by the hierarchical softmax has a considerable effect on the performance. Mnih and Hinton explored a number of methods for constructing the tree structure and the effect on both the training time and the resulting model accuracy [10]. In our work we use a binary Huffman tree, as it assigns short codes to the frequent words which results in fast training. It has been observed before that grouping words together by their frequency works well as a very simple speedup technique for the neural network based language models [5, 8].

① 기존 softmax



training time. The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^\top v_{w_I}\right)} \quad (2)$$

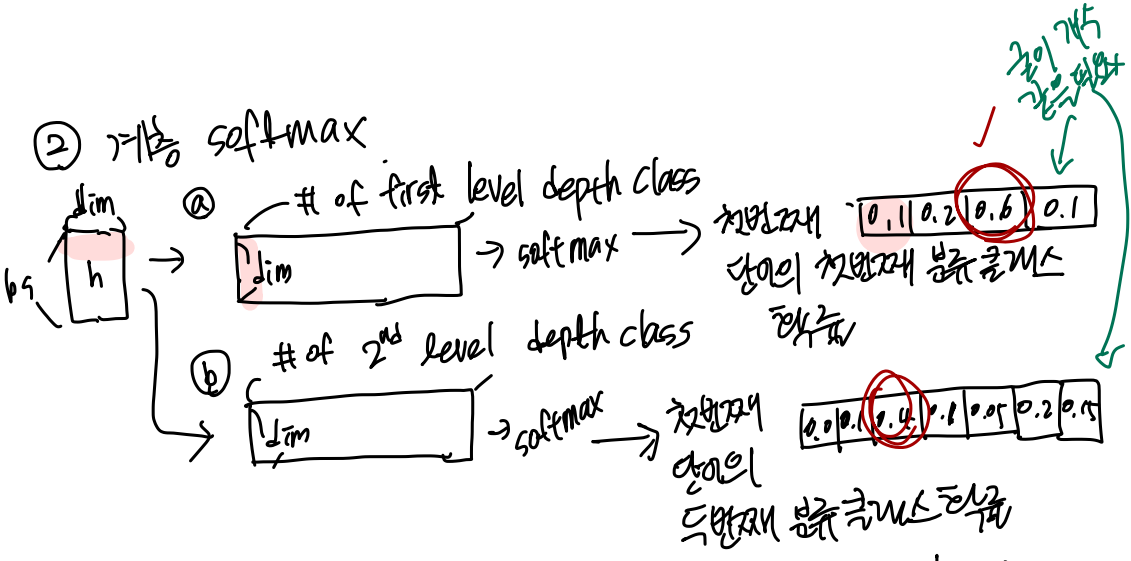
where v_w and v'_w are the "input" and "output" vector representations of w, and W is the number of words in the vocabulary. This formulation is impractical because the cost of computing $\nabla \log p(w_O|w_I)$ is proportional to W, which is often large (10^5–10^7 terms).

⇒ 'hi'라는 단어의 prob을 알기위해 hidden vector h는 output matrix $\in \mathbb{R}^{\text{hidden\_dim} \times \text{Vocab\_size}}$ 곱하고

Vocab size 만큼 softmax 수행 ⇒ Log(Vocab size)

→ 과거 기존 이거 너무 computation 크다!

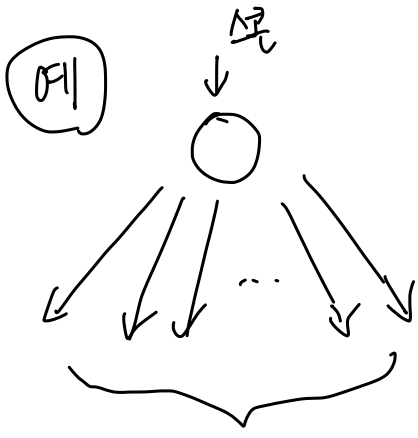
② 계층 softmax

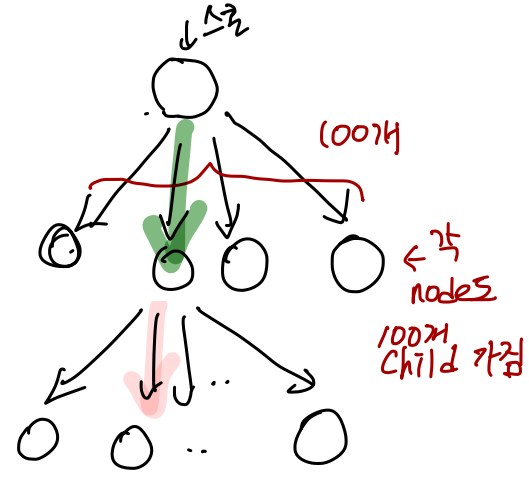


∴ 최종 $P(W_{out}|h_{in}) = P(W_{\text{첫번째계층클래스}}|h_{in}) \times P(W_{\text{두번째계층클래스}}|h_{in})$

Wout : 최종 prediction 단어

hin : input 단어 embedding

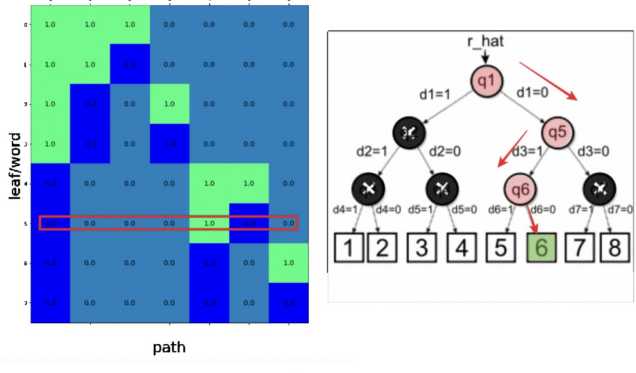




∴ output 확률 계산위해 200번 수행

Q. 이렇게 계산해서 학습에 쓰려면?

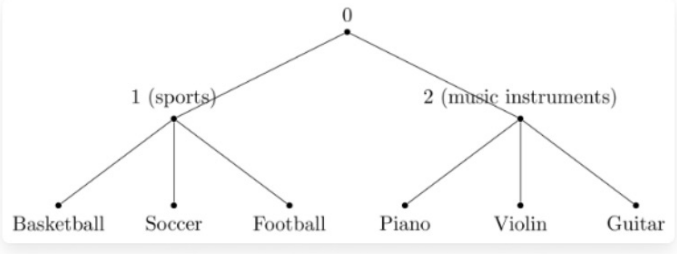
⇒ training data의 ground truth label이 대응되는 index와 함께 graph traverse path도 주어져야할 것.



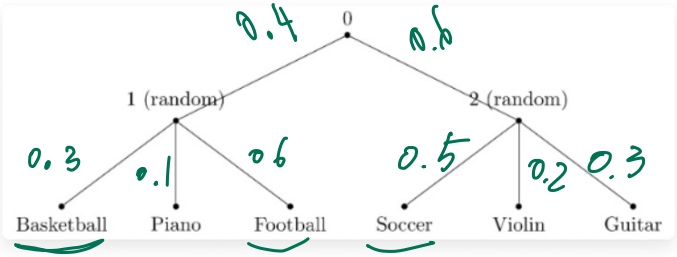
Q. 꼭 이진트리여야하나?

⇒ 대부분 하프만 tree로 구성해서 이진 tree가 많은데,
꼭 그럴 필요는 없을듯 (관련 코드 레포 참고했을때..)

⇒ 단, 어떻게 계층을 쌓는지가 성능에 영향 많이줌



Good Tree

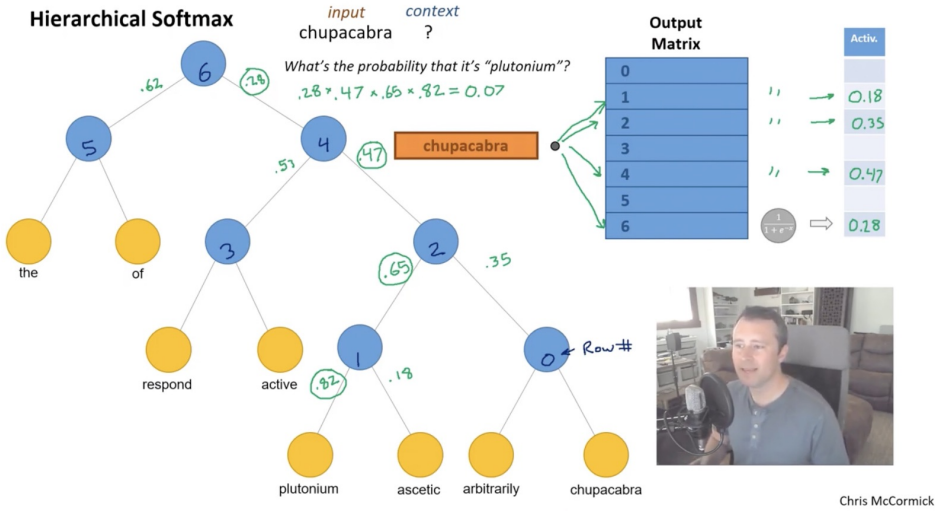


• basketball, football, soccer 이 sports로 각각 높은 확률 주었더니,

0.4 x 0.3 = 0.12
0.4 x 0.6 = 0.24
0.6 x 0.5 = 0.3

모델의 업데이트마다 soccer이 왼쪽으로 첫 layer부터 높은 확률주도록 되어야하는데, 유사한 클래스가 같이 지정된 모델이 학습된 것 같음.

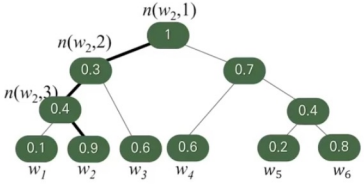
허프만 트리



4. Complexity Reduction

Efficient Estimation of Word Representations in Vector Space

- Hierarchical Softmax



$$\sum_{i=1}^{6} P(w_i = w_O) = 1 \times 0.3 \times 0.4 \times 0.1 + 1 \times 0.3 \times 0.4 \times 0.9 + 1 \times 0.3 \times 0.6 + 1 \times 0.7 \times 0.6 + 1 \times 0.7 \times 0.4 \times 0.2 + 1 \times 0.7 \times 0.4 \times 0.8 = 1$$

$P(n, left) = \sigma\left({v'_n}^T \cdot h\right)$: node n의 left로 갈 확률

$P(n, right) = \sigma\left(-{v'_n}^T \cdot h\right)$: node n의 right로 갈 확률

$P(n, left) + P(n, right) = 1$
: sigmoid이므로 위의 식이 성립

$$\sum_{i=1}^{V} P(w_i = w_O) = 1$$

: 모든 단어의 값에 대한 계산 없이

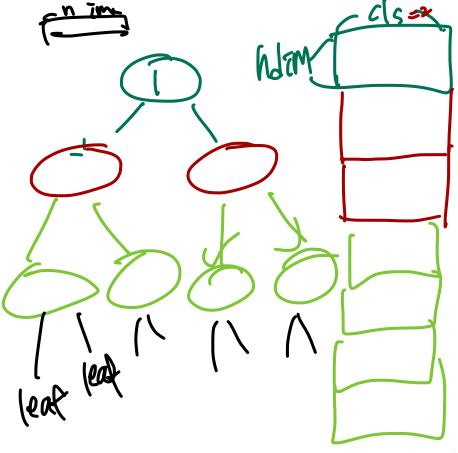
전체 합이 1이 되는 확률값을 구할 수 있다.

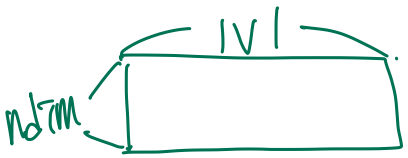
$Computational\ complexity \rightarrow D \times \ln(V)$

D: 각 step마다 길이 D의 vector 내적

$\ln(V)$: binary tree의 depth는 $\ln(V)$에 비례한다. 26

- forward 계산 좋고 backward도 좋겠지만,
- model complexity (모델 파라미터) 증가.. ㅋ 현실이 안돼.





ndim x node 개수 x 2.

intermediate node 개수 $\leq \frac{|V|}{2}$.