

FastText

Bag of Tricks for Efficient Text Classification

A. Joulin, E. Grave, P. Bojanowski, T. Mikolov

Enriching Word Vectors with Subword Information

P. Bojanowski, E. Grave, A. Joulin, T. Mikolov

AI Robotics KR NLP Study

발표자 남혜리

What is Fasttext?



Library for efficient text classification and representation learning

FastText is an open-source, free, **lightweight** library that allows users to learn **text representations** and **text classifiers**. It works on standard, generic hardware.

Models can later be reduced in size to even fit on mobile devices.

What is Fasttext?

- Facebook에서 연구 / 개발
- 근간이 된 논문 3편

Enriching Word Vectors with Subword Information

P. Bojanowski, E. Grave, A. Joulin, T. Mikolov

Bag of Tricks for Efficient Text Classification

A. Joulin, E. Grave, P. Bojanowski, T. Mikolov

FastText.zip: Compressing text classification models

A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jegou, T. Mikolov

What is Fasttext?

- Facebook에서 연구 / 개발
- 근간이 된 논문 3편

[Enriching Word Vectors with Subword Information](#)

P. Bojanowski, E. Grave, A. Joulin, T. Mikolov

[Bag of Tricks for Efficient Text Classification](#)

A. Joulin, E. Grave, P. Bojanowski, T. Mikolov

[FastText.zip: Compressing text classification models](#)

A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jegou, T. Mikolov

Bag of Tricks for Efficient Text Classification

Background Summary

NN Model(Recent Approach) vs Linear Model(Traditional Approach)

- NN(Neural Network) Model
 - 성능이 좋지만 대량의 데이터 셋에서는 상대적으로 학습/추론이 느리다는 한계가 있음
- Linear Model (EX. Logistic or SVM)
 - NN모델들에 비해 구조가 단순하기 때문에 주로 Baseline으로 쓰임
 - 하지만, 적절한 Feature와 함께 사용하면 SOTA 수준의 성능을 낼 수 있음
 - **단순한 구조 때문에 대용량 데이터 셋의 처리**에도 적합

요약

이 논문에서는 베이스라인 모델을 **Output Space가 매우 큰 대용량 데이터**에 확장 가능하도록 하는 방법(Tricks)들을 살펴봄

- Rank Constraints를 둔 Linear Model과 Fast Loss Approximation 방법을 사용하여 10억 개의 단어를 10분만에 학습시킴
 - ✓ 1) Word Representation Learning(Mikolov et al., 2013; Levy et al., 2015)의 방법론(**CBOW**)을 차용
 - ✓ 2) **Hiarachical Softmax**
- Tag prediction / Sentiment analysis에 적용한 결과를 제시

Model Architecture

- Linear Model은 파라미터를 공유하지 않기 때문에 클래스 당 데이터가 적을 때는 일반화 능력이 떨어짐
 - ✓ 일반적인 해법 : Low Rank Matrix로 분해하거나, MLP를 사용
 - ✓ 저차원의 특징을 뽑아 내는 단계(레이어)를 두어 노이즈에 오버피팅하는 것을 방지
- 논문에서는 **Lookup Table**을 사용하여 해결
(CBOW와 매우 유사한 접근법)

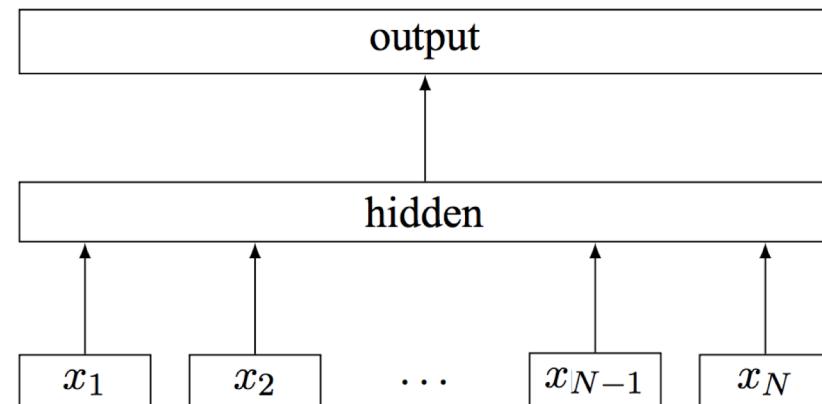
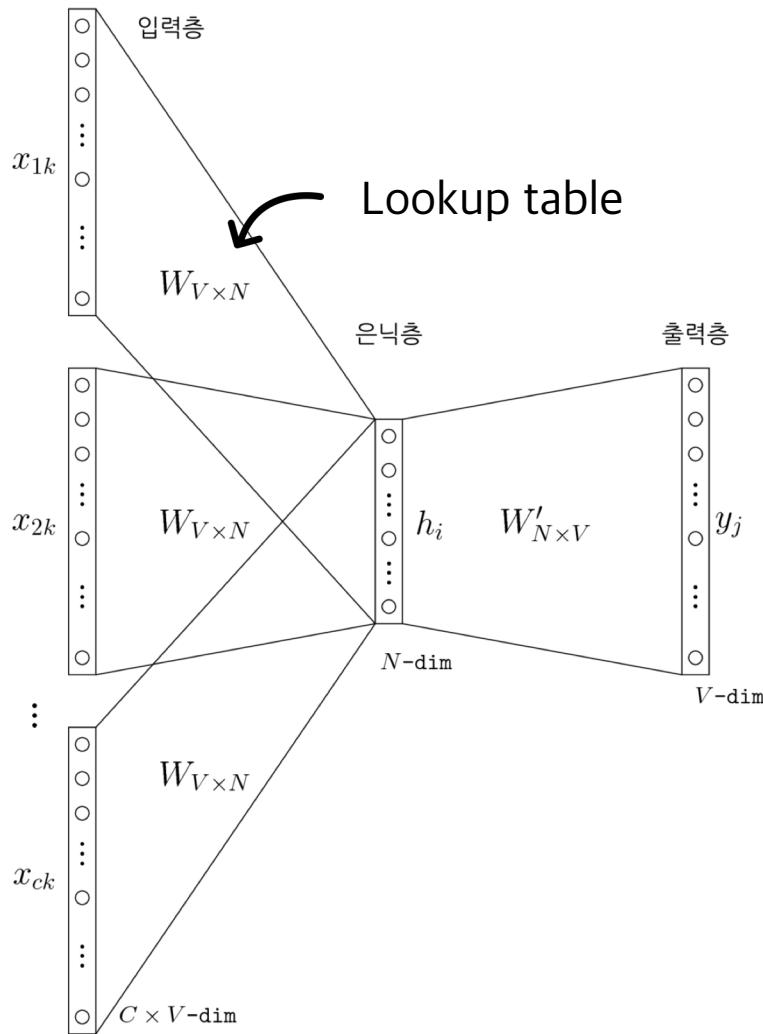


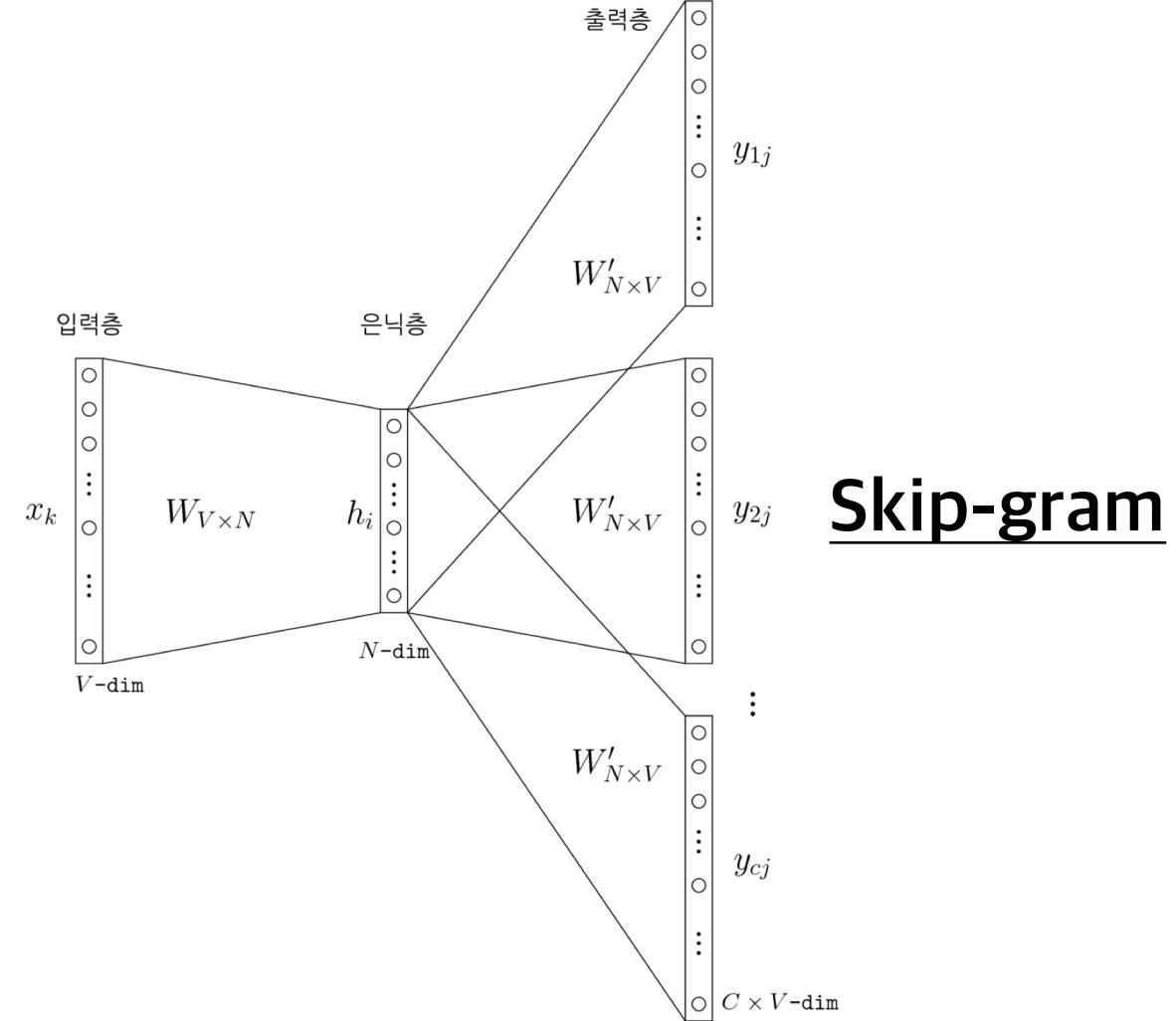
Figure 1: Model architecture of fastText for a sentence with N ngram features x_1, \dots, x_N . The features are embedded and averaged to form the hidden variable.

Model Architecture

CBOW



Skip-gram

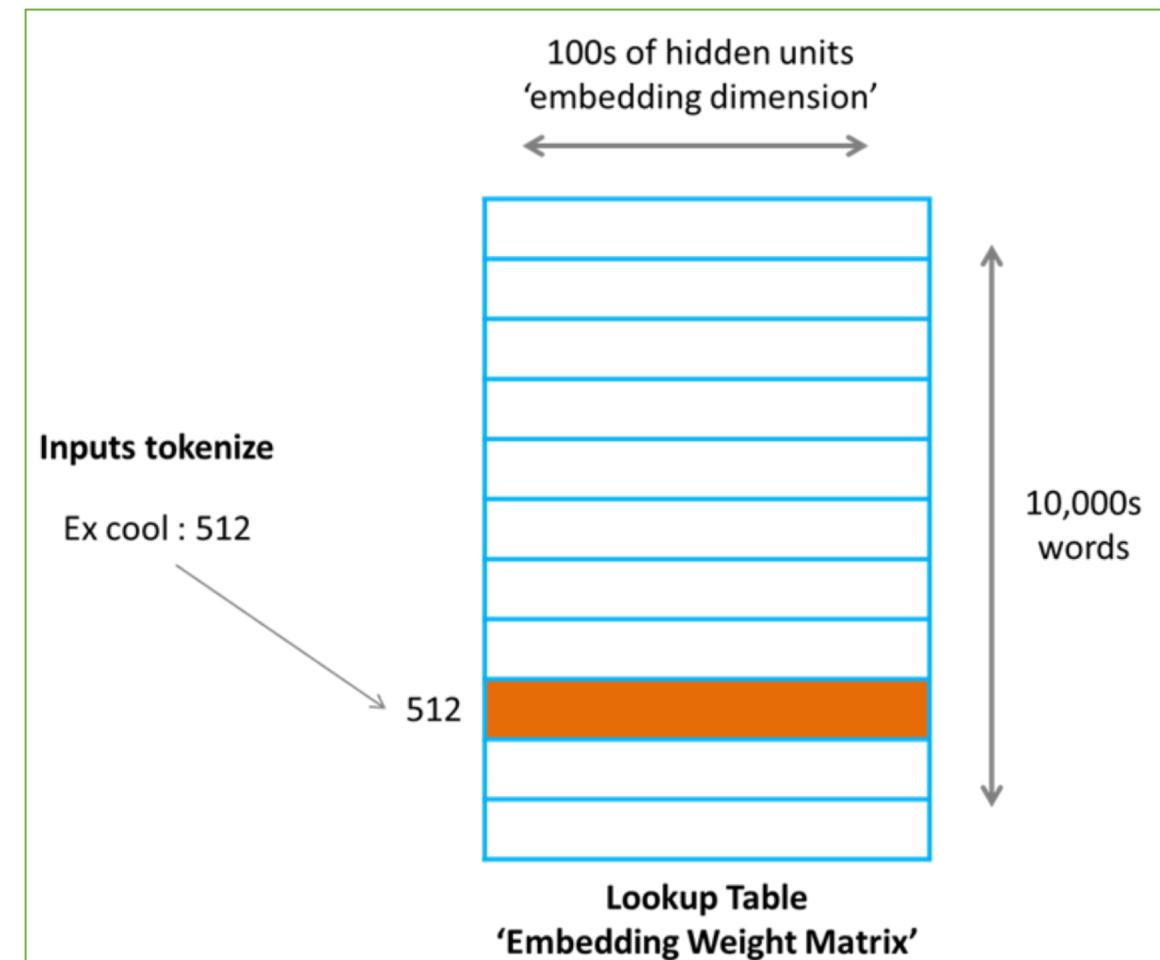


Model Architecture

- Lookup table

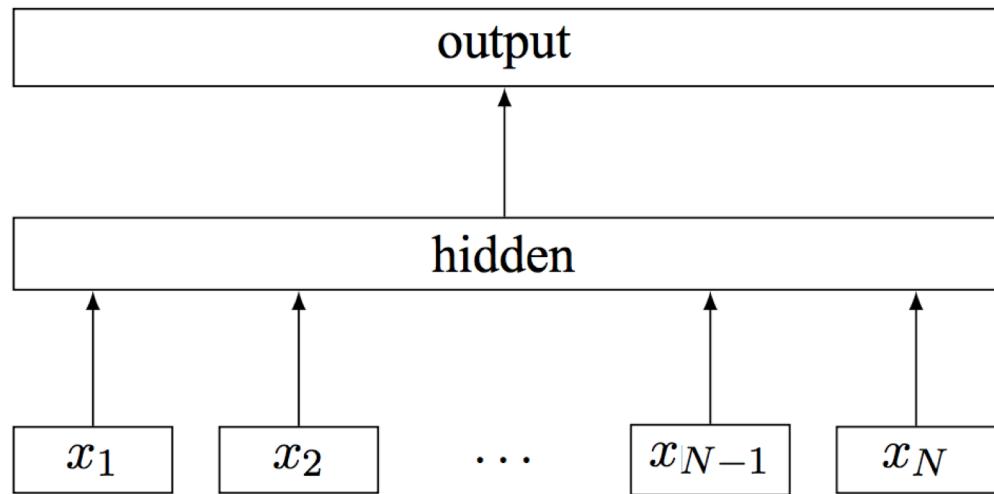
$$[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix} 8 & 2 & 1 & 9 \\ 6 & 5 & 4 & 0 \\ 7 & 1 & 6 & 2 \\ 1 & 3 & 5 & 8 \\ 0 & 4 & 9 & 1 \end{bmatrix} = [1 \ 3 \ 5 \ 8]$$

One-hot vector Hidden layer output
Embedding Weight Matrix



<https://towardsdatascience.com/what-the-heck-is-word-embedding-b30f67f01c81>

Model Architecture



Linear Classifier에 입력해 Output 계산



단어 벡터들을 평균 (Text Representation)



Lookup table을 사용해 단어의 임베딩을 구함
(Word Representation)

단어의 핵심 정보를 담고 있는
저차원의 dense representation

► Target이 중심단어이면 CBOW와 동일

Model Architecture

- Loss function (Cross Entropy)

$$-\frac{1}{N} \sum_{n=1}^N y_n \log(f(BAx_n))$$

Diagram illustrating the components of the loss function:

- # documents**: Points to the term $\frac{1}{N}$.
- Label**: Points to the term y_n .
- Weight matrices**: Points to the term B .
- Normalized bag of features of the nth doc**: Points to the term Ax_n .
- Softmax**: Points to the term f .

Hierarchical Softmax

- Softmax 레이어의 계산량을 줄이기 위한 방법

$$\text{softmax} : p(t|c) = \frac{e^{\theta_t^T h_c}}{\sum_{j=1}^k e^{\theta_j^T h_c}}$$

- ✓ Normalization을 위해 많은 계산량이 필요
 - ✓ (k 길이의 벡터의 내적을 클래스의 개수만큼 수행)
- ✓ $O(hk)$ 의 시간 복잡도 (k 는 클래스의 개수, h 는 hidden layer의 차원)
- Hierarchical Softmax를 사용하면 시간복잡도를 $O(h \log_2 k)$ 로 줄일 수 있음

Hierarchical Softmax

- 리프 노드가 클래스의 확률인 **트리 구조로 모델링**
- 각 클래스의 확률은 리프까지의 경로 상의 **Binary Decision(왼쪽/오른쪽 경로를 선택할 확률의 곱으로 모델링 됨)**

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

$$\text{where } [x] = \begin{cases} 1 & \text{if } x \text{ is true} \\ -1 & \text{otherwise} \end{cases}$$

$L(w)$: 루트 부터 w 까지의 경로에 있는 노드들의 수

$n(w, i)$ 는 $vector v'_{n(w, i)}$ 와 연관된 경로의 i 번째 노드

$ch(n)$: 노드 n 의 임의의 자식 노드

N-gram Features

- 텍스트 내의 단어의 순서를 고려하기 위해 **N-gram Feature**를 사용
- **단어 순서의 지역적 정보**를 반영
 - 예시) “나는 매주 화요일 마다 nlp 스터디에 참여한다.”
 - Bi-gram : [‘나는 매주’, ‘매주 화요일’, ‘화요일 마다’, ‘마다 nlp’ ….]
 - Tri-gram : [‘나는 매주 화요일’, ‘매주 화요일 마다’, ‘화요일 마다 nlp’….]
- N-gram은 word의 조합이므로 가지 수가 훨씬 많음
- 성능 (속도 & 메모리 효율성)을 유지하기 위해 Hashing Trick을 사용 (feature hashing)
 - Bag of n-gram (sparse vector) ► Fixed size array
 - hashing trick (Weinberger et al., 2009) with the same hashing function as in Mikolov et al. (2011)

Experiment

- Sentiment Analysis

Model	AG	Sogou	DBP	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW (Zhang et al., 2015)	88.8	92.9	96.6	92.2	58.0	68.9	54.6	90.4
ngrams (Zhang et al., 2015)	92.0	97.1	98.6	95.6	56.3	68.5	54.3	92.0
ngrams TFIDF (Zhang et al., 2015)	92.4	97.2	98.7	95.4	54.8	68.5	52.4	91.5
char-CNN (Zhang and LeCun, 2015)	87.2	95.1	98.3	94.7	62.0	71.2	59.5	94.5
char-CRNN (Xiao and Cho, 2016)	91.4	95.2	98.6	94.5	61.8	71.7	59.2	94.1
VDCNN (Conneau et al., 2016)	91.3	96.8	98.7	95.7	64.7	73.4	63.0	95.7
fastText, $h = 10$	91.5	93.9	98.1	93.8	60.4	72.0	55.8	91.2
fastText, $h = 10$, bigram	92.5	96.8	98.6	95.7	63.9	72.3	60.2	94.6

Table 1: Test accuracy [%] on sentiment datasets. FastText has been run with the same parameters for all the datasets. It has 10 hidden units and we evaluate it with and without bigrams. For char-CNN, we show the best reported numbers without data augmentation.

	Zhang and LeCun (2015)		Conneau et al. (2016)			fastText
	small char-CNN	big char-CNN	depth=9	depth=17	depth=29	$h = 10$, bigram
AG	1h	3h	24m	37m	51m	1s
Sogou	-	-	25m	41m	56m	7s
DBpedia	2h	5h	27m	44m	1h	2s
Yelp P.	-	-	28m	43m	1h09	3s
Yelp F.	-	-	29m	45m	1h12	4s
Yah. A.	8h	1d	1h	1h33	2h	5s
Amz. F.	2d	5d	2h45	4h20	7h	9s
Amz. P.	2d	5d	2h45	4h25	7h	10s

Table 2: Training time for a single epoch on sentiment analysis datasets compared to char-CNN and VDCNN.

- Tag Prediction

Model	prec@1	Running time	
		Train	Test
Freq. baseline	2.2	-	-
Tagspace, $h = 50$	30.1	3h8	6h
Tagspace, $h = 200$	35.6	5h32	15h
fastText, $h = 50$	31.2	6m40	48s
fastText, $h = 50$, bigram	36.7	7m47	50s
fastText, $h = 200$	41.1	10m34	1m29
fastText, $h = 200$, bigram	46.1	13m38	1m37

Table 5: Prec@1 on the test set for tag prediction on YFCC100M. We also report the training time and test time. Test time is reported for a single thread, while training uses 20 threads for both models.

Enriching Word Vectors with Subword Information

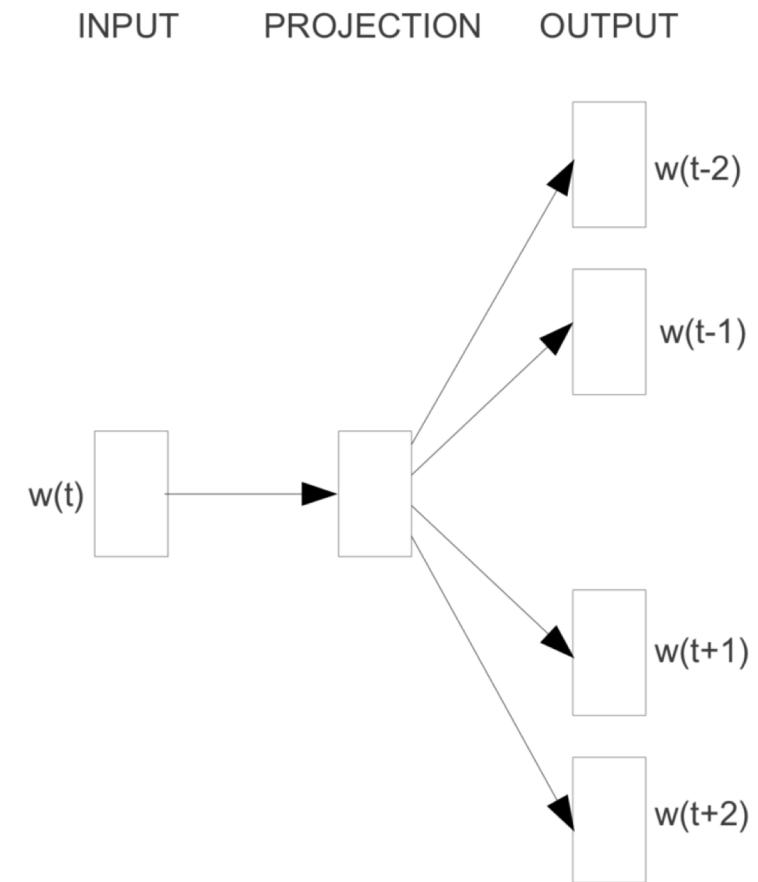
요약

- 대부분의 Word Embedding 방법들은 각 "단어"를 벡터로 할당하기 때문에 언어의 형태 학적인 특성을 무시한다는 한계점 존재
- 이 논문에서는 각 단어를 **Bag of Character n-grams**을 보고 개별 단어가 아닌 character n-gram의 Embedding을 학습하며, Word vector는 n-gram Embedding의 합으로 구함
- 결과적으로는 단순한 모델 구조로 학습이 빠를 뿐만 아니라, 여러 task에서 기존의 방법들 보다 좋은 성능을 냄

Skip-gram (General Model)

- 분산 가설을 이론적 바탕으로 둔 방법
 - "A word is characterized by the company it keeps"
- 주변에 나타나는 단어를 잘 맞추도록
Word Representation을 학습
- log-likelihood

$$\sum_{t=1}^T \sum_{c \in \mathcal{C}_t} \log p(w_c | w_t)$$



Skip-gram

Skip-gram 모델의 한계

- 희귀한 단어들을 잘 학습하지 못하며, **형태론적 특성을 반영하지 못함**
- 단어의 "내부 구조"를 고려하지 않고 각 단어를 별개의 vector로 취급하기 때문
- 특히 단어의 변형이 다양한 언어들에서 이런 점이 문제가 됨
 - ex) French or Spanish : 동사의 40개의 변형이 있음
 - Finnish : 명사의 15개의 변형이 존재
- 또한 Training Dataset에 없는 단어(OOV)의 임베딩은 구할 수 없음
- 대부분의 언어에서 **단어의 변형은 "형태론"적인 규칙을 따름**
- **Character 레벨의 정보**를 사용하면 Vector representation을 개선할 수 있음

Subword Model

- 단어 보다 더 작은 단위인 Subword(Character n-gram)를 임베딩하고 서브 워드 벡터를 합하여 워드 벡터를 계산하면 "내부 구조"를 반영한 representation을 구할 수 있다.

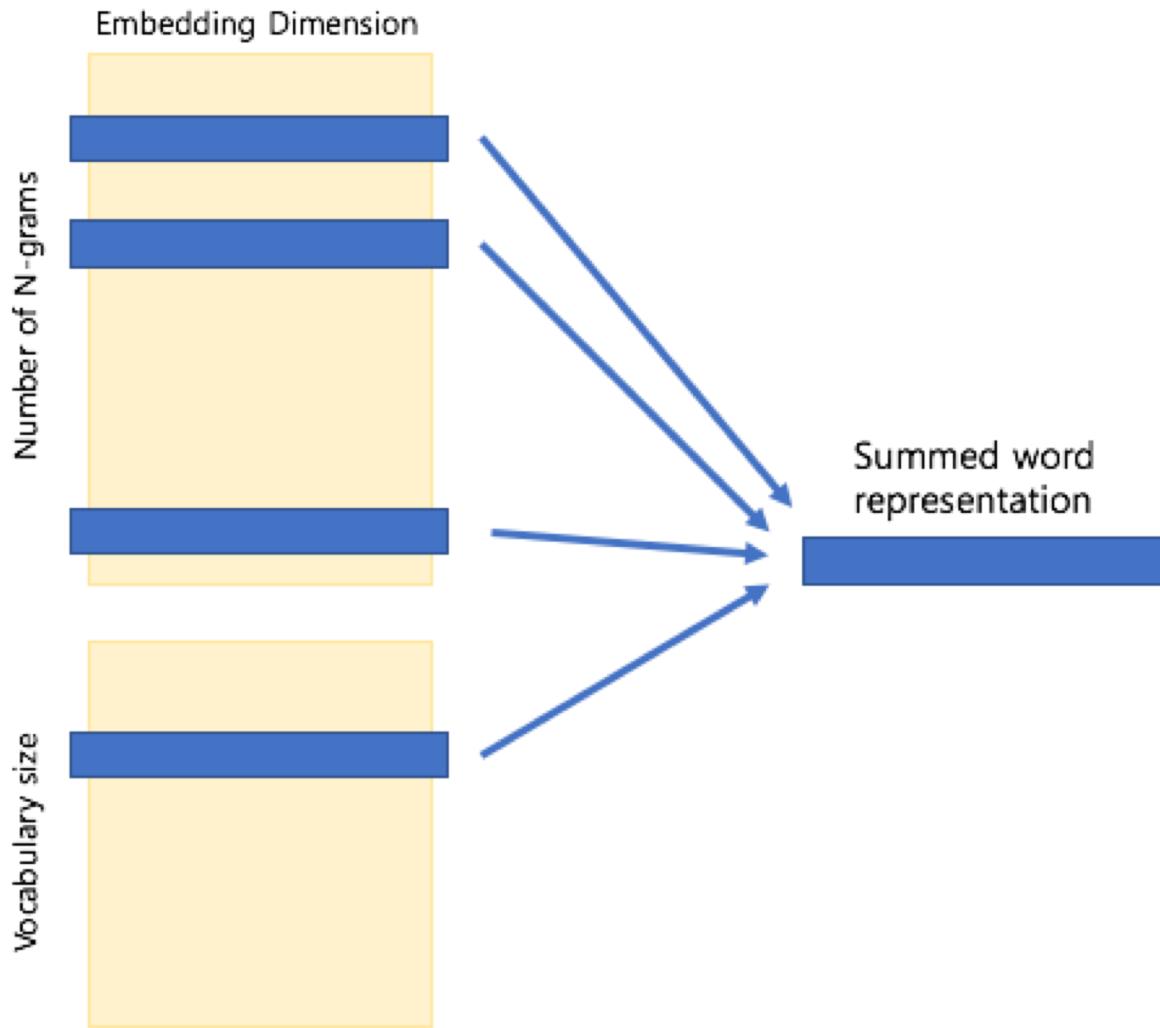
ex) where

- 먼저 단어의 시작과 끝을 구분하기 위해 <, >를 추가함
- 그리고 원래 단어 시퀀스도 추가함

ex) **where -> <wh, whe, her, ere, re>, <where>**

- <, >기호를 덧붙였기 때문에 <her>와 tri-gram her는 다른 토큰으로 취급
- 단어의 vector는 character n-gram vector를 합하여 구함
 $v(\text{where}) = v(<\text{wh}>) + v(\text{whe}) + \dots + v(<\text{where}>)$
- 실제 적용할 때는 3-6 grams 를 모두 추출하여 서브 워드로 사용

Subword Model



Subword Model

- Negative Log-likelihood (Skip-gram with Negative Sampling)

$$\log \left(1 + e^{-s(w_t, w_c)} \right) + \sum_{n \in \mathcal{N}_{t,c}} \log \left(1 + e^{s(w_t, n)} \right)$$

→ Negative examples

$$s(w_t, w_c) = \mathbf{u}_{w_t}^\top \mathbf{v}_{w_c}$$

- Scoring function of Subword Model

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c.$$

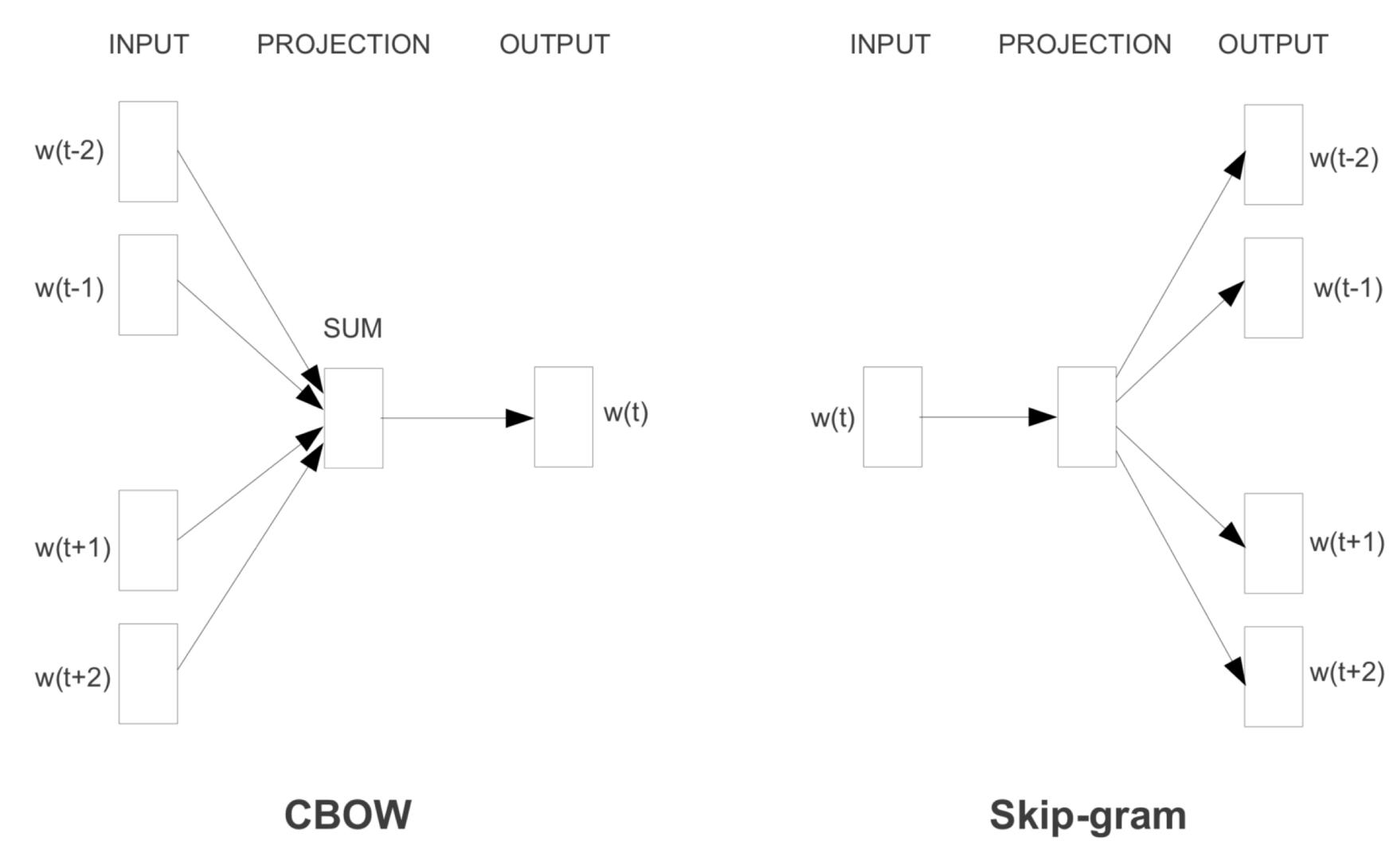
set of n-grams appearing in w.

Experiment

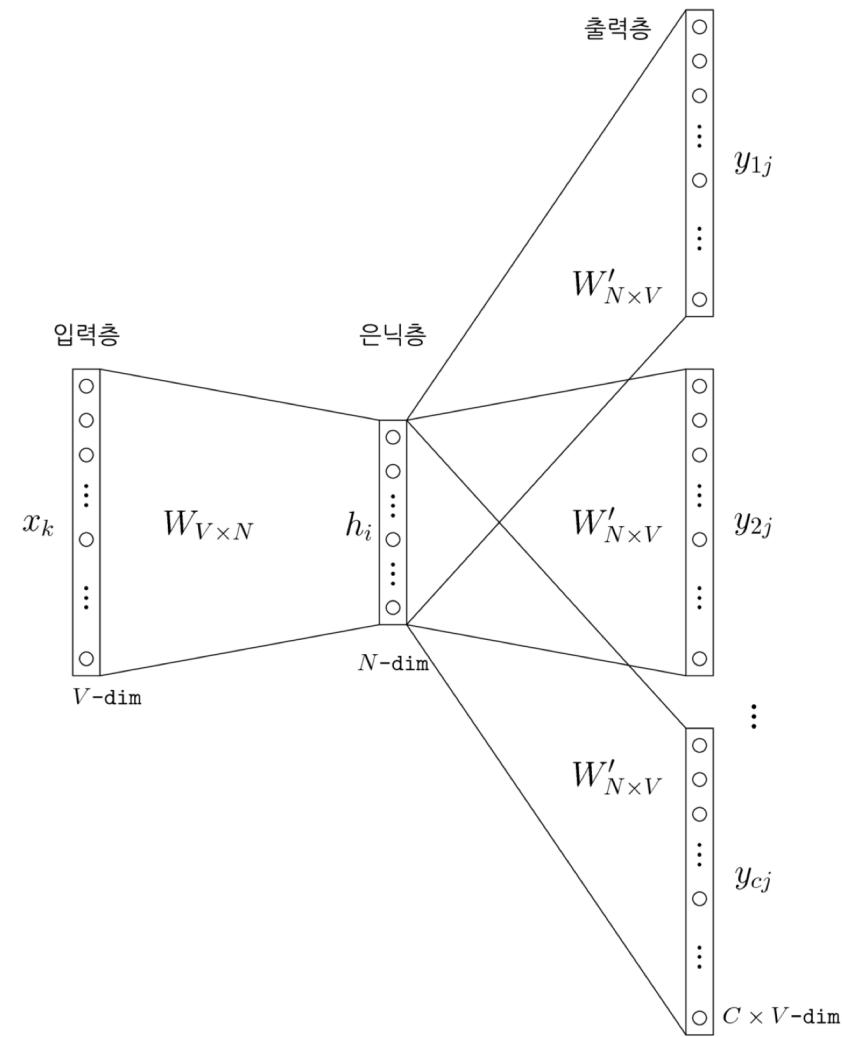
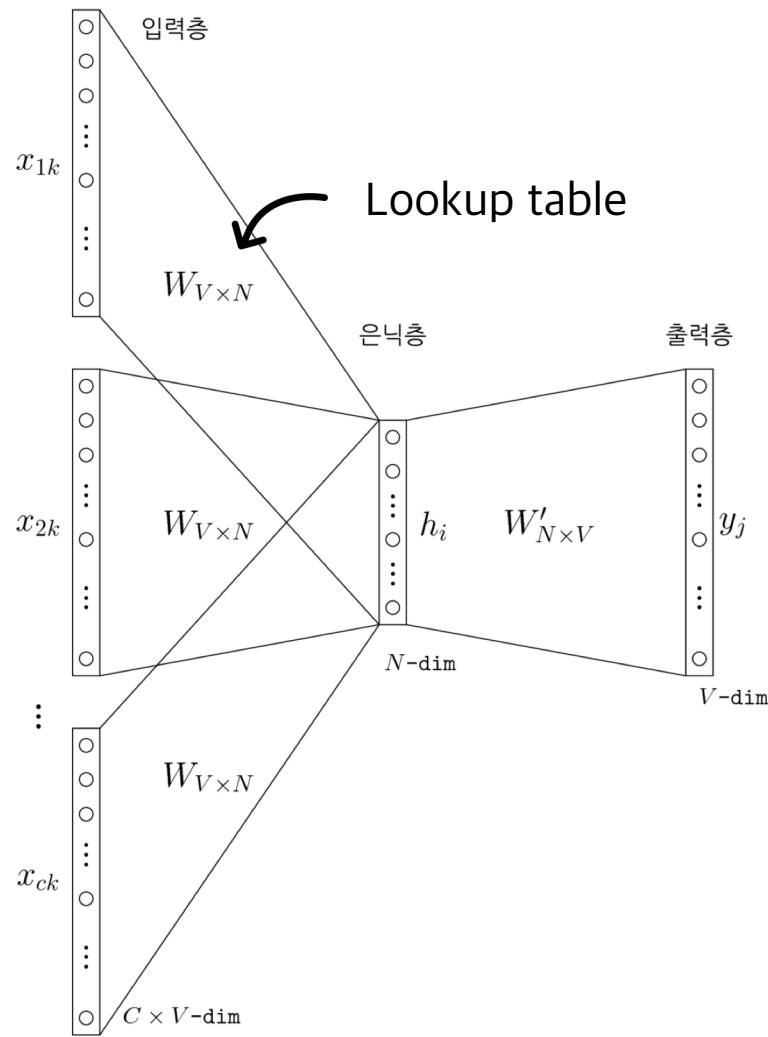
- **Human similarity judgement** (사람이 평가한 단어 유사도와 word vector간의 코사인 유사도의 비교)
- **Word analogy test** ((A is to B as C is to D)로 표현되는 유추 문제에서 예측 정확도 비교)
- **Comparison with morphological representations** (형태학적 특성 반영을 시도한 방법론과의 성능 비교)
- **Effect of the size of the training data** (학습 데이터 수에 따른 성능 비교)
- **Language modeling** (Language model(LSTM)을 pretrain 했을 때의 perplexity 비교)
- **결론**
 - 서브 워드 모델이 기존의 방법론 보다 많은 경우 성능이 개선되었으며, 특히 변형이 많은 언어에서 성능 개선이 두드러짐
 - Rare word가 많은 데이터 셋에서 성능이 좋으며, 부가적으로 OOV의 임베딩도 n-gram vector의 합으로 구할 수 있다는 장점이 있음
 - 형태론적 특성을 반영하기 때문에 적은 수의 데이터만으로도 워드 임베딩이 잘 됨

Appendix.

CBOW & Skip-gram

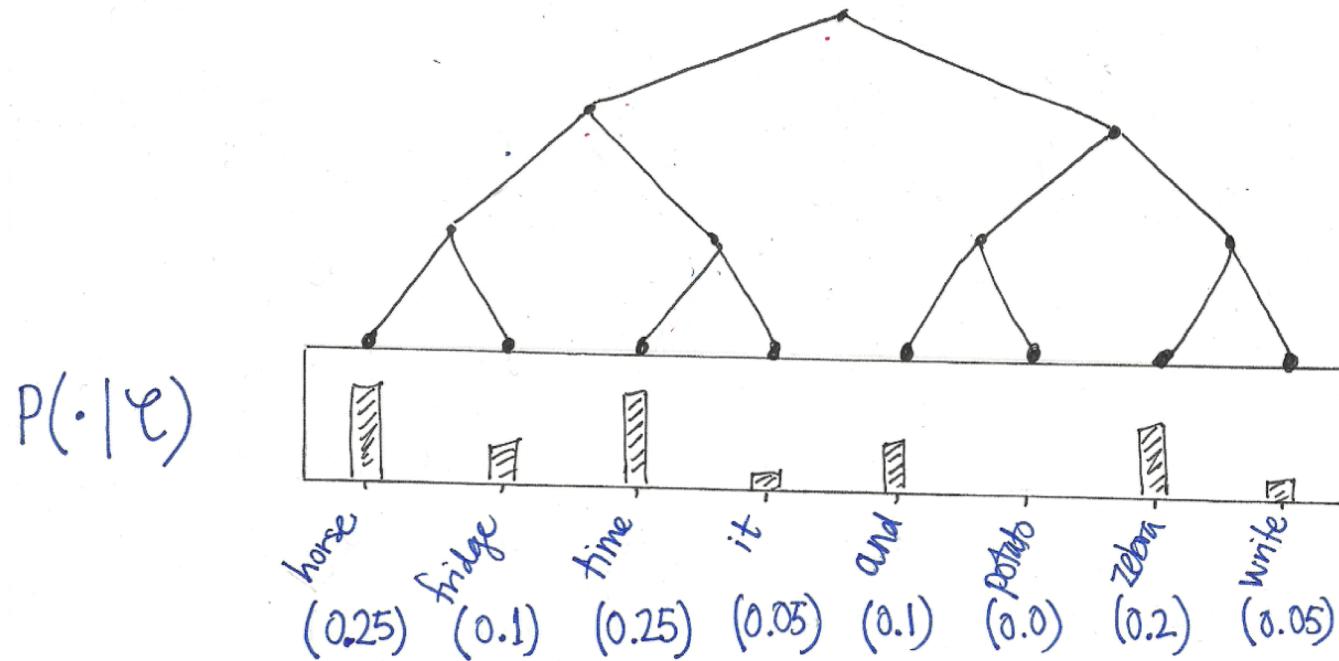


CBOW & Skip-gram



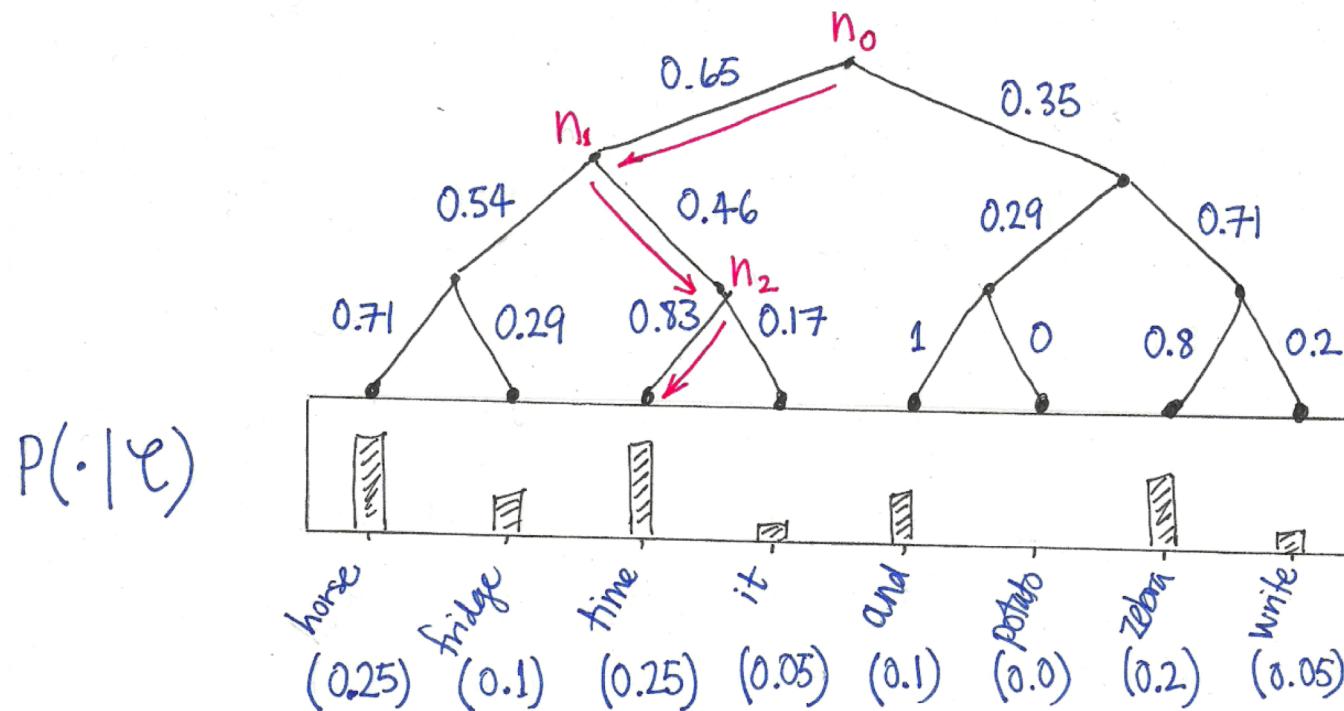
Hierarchical Softmax

- 예시와 단계별 설명으로 Hierarchical Softmax 이해 해보기
(출처 - <http://building-babylon.net/2017/08/01/hierarchical-softmax/>)
- 먼저, 리프 노드가 각 클래스의 확률을 나타내는 이진 트리를 생각해보자.



Hierarchical Softmax

- 이 트리를 루트 노드에서 시작해서 매 노드마다 (왼쪽 혹은 오른쪽) 경로를 선택하는 랜덤 워크 과정이라고 생각해보자.
- 리프 노드에 도달할 확률을 각 클래스의 확률이라고 한다면, 각 노드에서의 Transition Probability는 각각 왼쪽 자식의 서브트리와 확률을 다 더한 것과 오른쪽 자식의 확률을 다 더한 것을 Normalize한 것과 같다.
 - 왼쪽 리프에 도달할 확률의 총합 vs 오른쪽 리프에 도달할 확률의 총합
- **반대로 각 클래스의 확률은 각 리프에 도달하기 까지의 Binary decision 확률의 곱이 된다.**



Hierarchical Softmax

- 예를 들어 $P(\text{"time"}|C)$ 는 다음과 같이 계산된다.

$$P(\text{"time"}|C) = P_{n_0}(\text{left}|C)P_{n_1}(\text{right}|C) P_{n_2}(\text{left}|C)$$

$$\text{where } P_n(\text{right}|C) = 1 - P_n(\text{left}|C)$$

- 각 노드에서의 Transition Probability는 logistic sigmoid를 사용하여 모델링한다.

$$P_n(\text{left}|C) = \sigma(\gamma_n^T \alpha_C)$$

- $P_n(w|\alpha_C)$ 을 구하기 위해서는 리프 노드에 도달하는 경로에 있는 노드의 계산만 하면 된다.
- 각 노드에서는 길이가 h 인 벡터의 내적이 발생하며, 계산은 트리의 높이만큼의 반복 된다.
- 즉, 시간 복잡도는 $O(h \log_2 k)$ 가 된다.