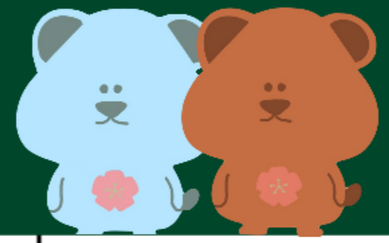
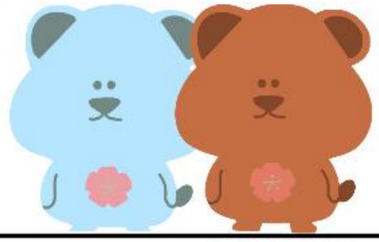


Introduction and Word Vectors

Week1_발표자 : 이승연, 이다현

EWHA EURON

목차



EWHA EURON

How to represent words?

01. How to represent words’ meaning?

01. Denotational Semantics

의미에 대한 가장 일반적인 언어적 사고 방식

signifier (symbol) ⟺ signified (idea or thing)

= denotational semantics

(표시적 의미론)

01. How to represent words' meaning?

02. WordNet : 고전적 NLP 해결책

😀 특징

1. 단어의 synonym과 hypernym을 이용해서 구성
2. 단어에 대한 어느 정도 의미 제공

😒 단점

1. 다양한 뉘앙스를 놓친다.
2. 신조어, slang 등에 취약하므로 지속적인 인력 투입 필요하다.
3. human labor를 통해 구성된 hand-built resources (주관적)
4. 단어 간의 유사도, 관계 파악 어려움

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

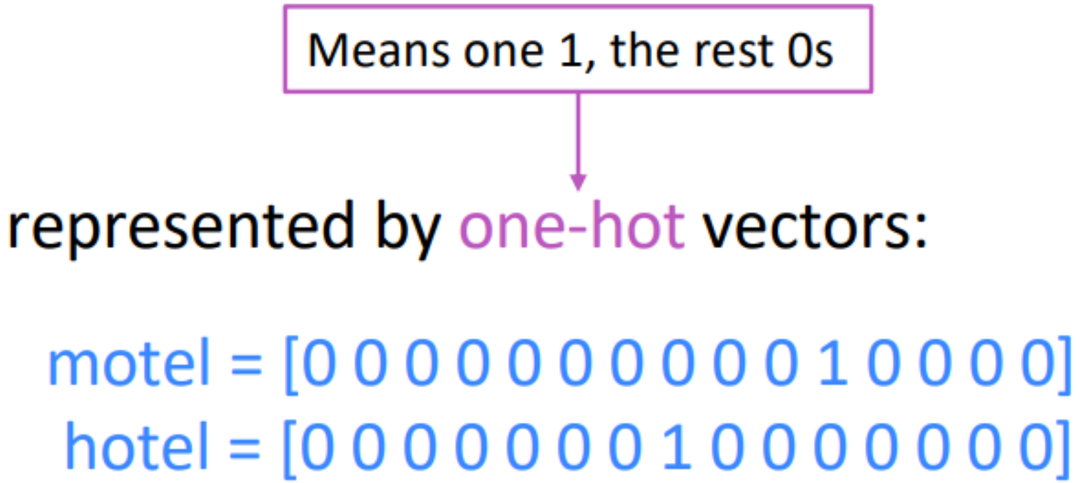
```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

01. How to represent words’ meaning?

03. One-hot encoding vector

😀 정의

단어 집합의 크기를 벡터의 차원으로 하고, 표현하고 싶은 단어의 인덱스에 1의 값을 부여, 다른 인덱스에는 0을 부여하는 벡터 표현 방식



😒 단점

1. 단어 간의 유사성을 정량적으로 표현하는 방법이 없다.
2. 단어 간의 관계를 파악하려면 벡터의 차원이 제곱이 되며 기하급수적으로 커진다.
3. 단어의 개수 = 벡터의 차원이므로 차원이 매우 많이 필요하다.

02. How to represent words?

04. Word vectors

…government debt problems turning into ***banking*** *crises as happened in 2009…*
…saying that Europe needs unified ***banking*** *regulation to replace the hodgepodge…*
…India has just given its ***banking*** *system a shot in the arm…*

These context words will represent ***banking***

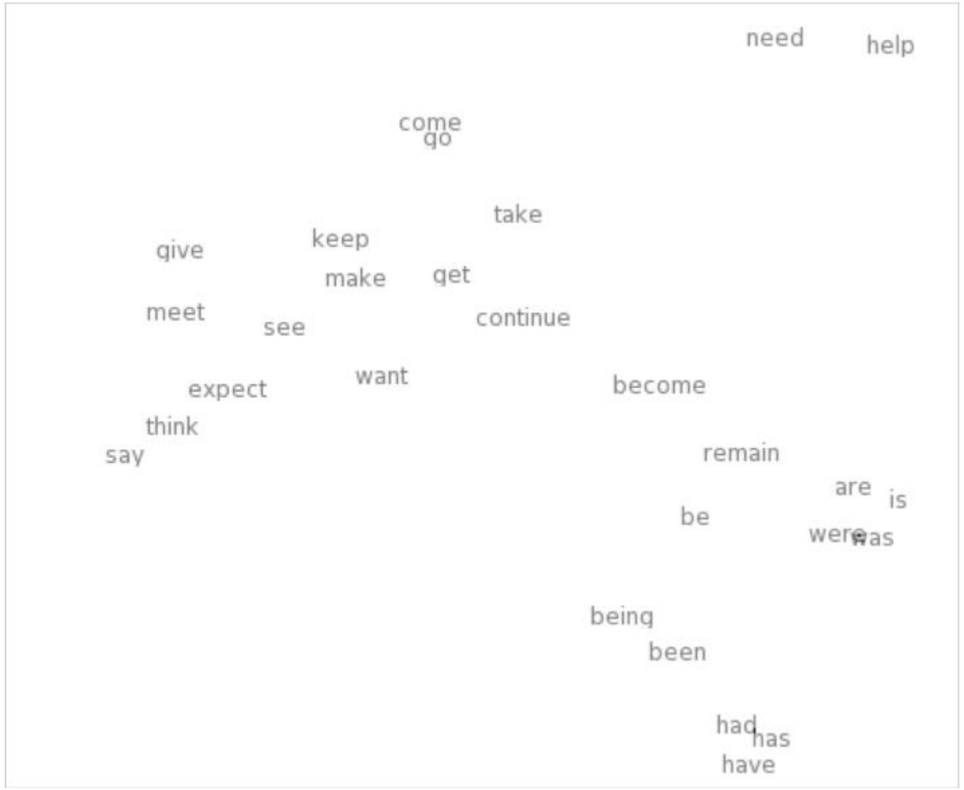
1. Word embedding, word representation이라고도 부른다
2. Fixed-size window로 단어가 나타나는 **context에서 단어의 의미를 예측**한다
3. One-hot Vector 방식이 아닌 **distributed representation**을 사용한다 (앞선 단점을 극복)

02. How to represent words?

모든 단어를 위치시킬 수 있는 vector space를 생각해 보면 n차원의 word vector를 시각화하긴 쉽지 않음

👉 2차원 공간에 벡터 투영

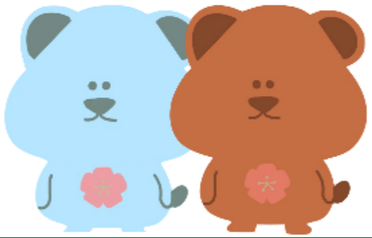
$$expect = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



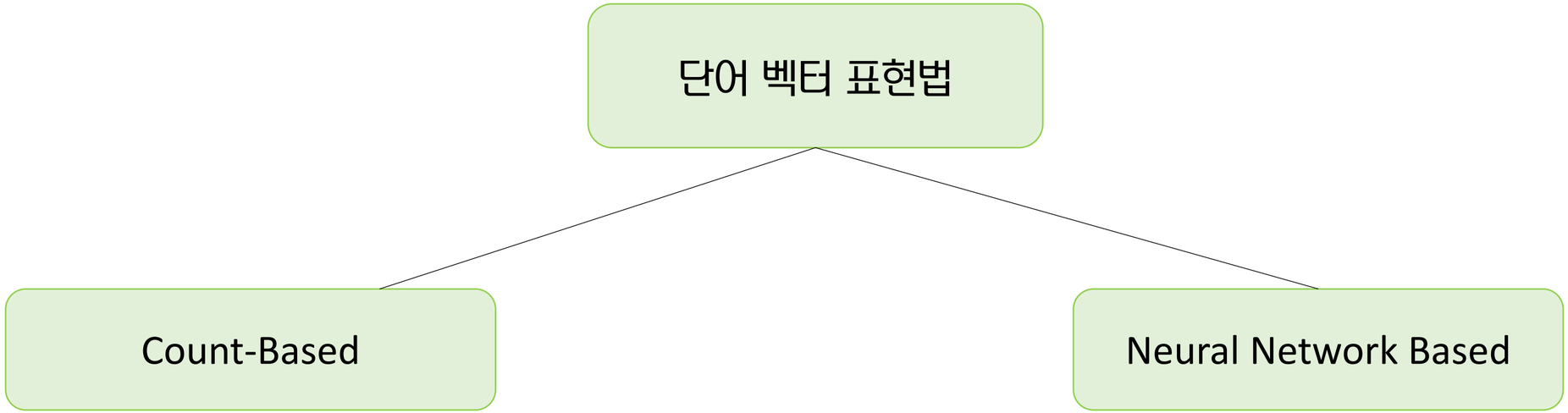
2차원에 투영하며 단어의 일부 특징이 사라질 수 있지만, **단어 간 유사도**를 시각적으로 살펴 보기 좋은 방법

의미적으로 유사한 단어 간에 군집을 이루는 경향이 나타난다.

Word2vec



Word Vector Representations



Bow, TF-IDF, LSA
(→SVD, co-occurrence)

Word2vec, FastText, Glove,
BERT, ELMo, GPT …

Intro_단어를 유의미한 수치를 가지는 벡터로 만들기

👀 기계가 학습하려면, 사람이 쓰는 문자 형태가 아니라 기계가 이해할 수 있는 숫자 형태여야 하는데…

👉 단어(word) 를 벡터(vector) 로 표현하자!

🧐 **문장에서** 특정 **단어의 의미/ 단어간 유사도를 잘 반영**하도록 벡터화 하는 방법 없을까? 수학적으로…

Count-based 벡터화 방법은…
⇨ 원핫 인코딩은 단어의 위치만 반영
⇨ Bow 는 단어의 빈도수만 반영
⇨ TF-IDF 는 단어의 중요도만 반영
"단어의 문맥적 의미" 를 반영하지 못함

👀 **확률**의 개념을 도입해보자

📌 문맥에서 특정 단어가 등장할 확률을 학습하는 **모델**을 만들자!

Word vector 를 파라미터로 갖는 모델

→ objective function (loss function) 정의

→ train the model on certain objective

(1) 단어를 벡터로 변환 (초기값)

(2) 모델 훈련을 통해서 가중치 update를 통해 단어를 가장 잘 표현하는 벡터 찾기

4.1 Language Models

👀 **확률**의 개념을 도입해보자

고양이가 웅덩이를 뛰어넘었다

"The cat jumped over the puddle." > "stock boil fish is toy"

육수 삶는 생선은 장난감이다.

📌 문맥에서 특정 단어가 등장할 확률

$$P(w_1, w_2, \cdots, w_n)$$

단어의 문맥적 의미 = 순서 (sequence) 를 반영

Unigram model:

$$P(w_1, w_2, \cdots, w_n) = \prod_{i=1}^{n} P(w_i)$$

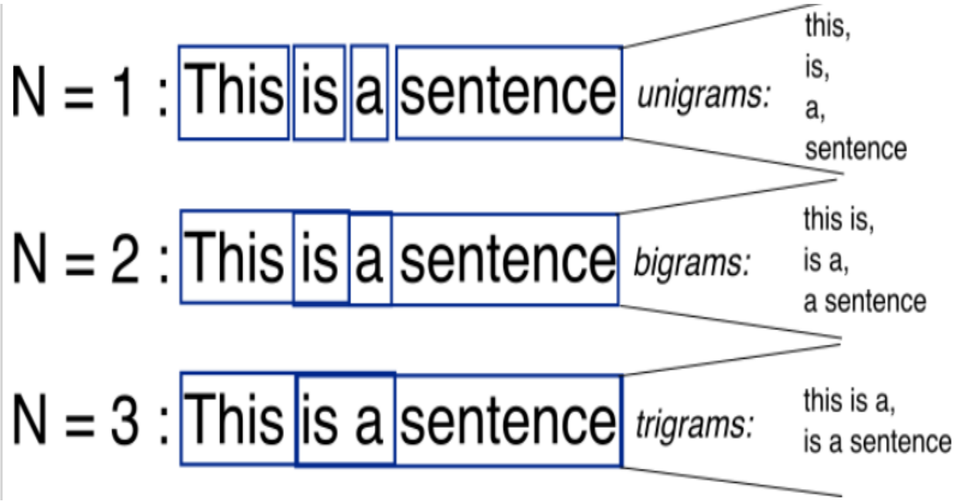
문맥을 이해하려면 문장 안에서 특정 단어는 이전에 등장하는 단어, 이후에 등장하는 단어와 독립성을 가정하기에는 어렵다…

Bigram model:

$$P(w_1, w_2, \cdots, w_n) = \prod_{i=2}^{n} P(w_i | w_{i-1})$$

2개씩 묶어 보는 pairwise 방식을 도입하자 → biagram

✨ N-gram



💡 **핵심** : 문장 내에서 단어가 등장하는 순서를 확률로 표현하게 되면, 단어의 문맥적 (context) 의미를 포착할 수 있다.

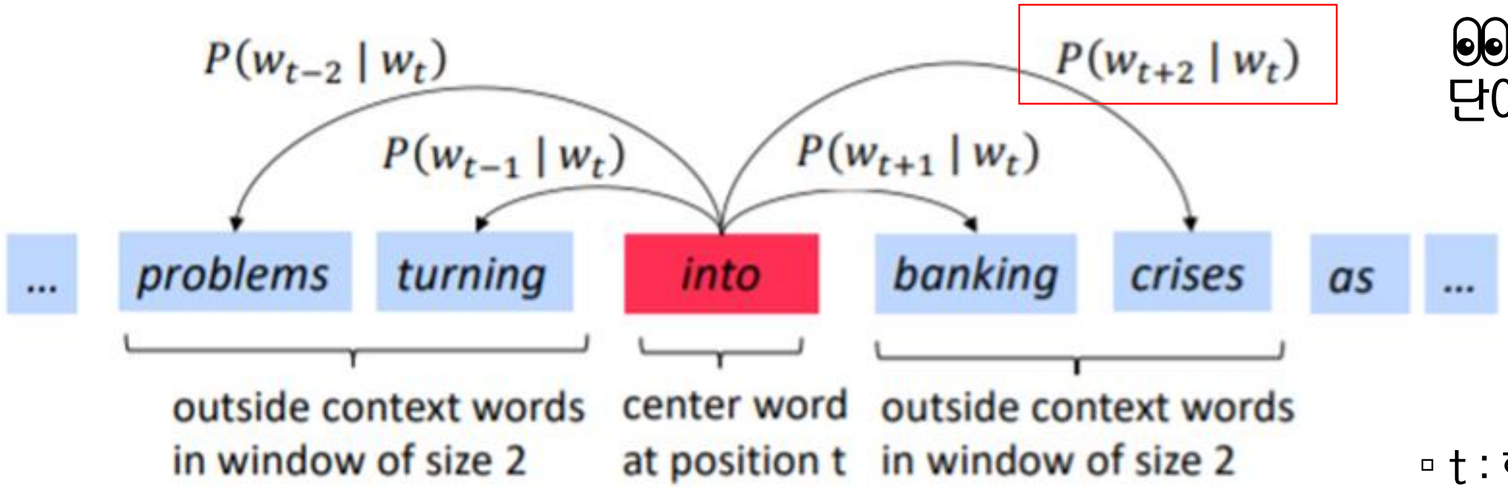
EWHA EURON

Word2vec

개념

- 큰 말뭉치로부터 **단어 관계성 (word vectors) 을 학습하는 신경망 모델을 사용하는 알고리즘**
- 충분한 양의 말뭉치를 바탕으로 random vector 부터 시작하여 각 단어를 잘 표현하는 **vector** 값을 찾는다.
- 단어 벡터간의 유사도를 이용해 **맥락**에서 **특정 단어**가 나타날 **확률**을 계산한다.

학습 방식



👀 확률을 최대화 하는 방향으로 단어 벡터의 값을 결정한다.

- t : 현재 위치
- Wt : 현재 위치의 단어
- W(t+n) : 주변 단어

EWHA EURON

Word2vec

계산방법

1. Likelihood

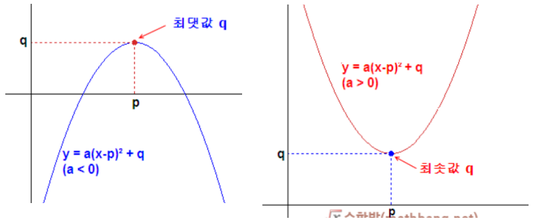
$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

모든 단어에 대해(t) 고정된 크기의 window 만큼(j) 주변의 단어 확률을 곱한다는 의미

⇨ 주변 단어의 발생 확률을 예측

2. Objective function



sometimes called *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

가능도를 최대화 하는 최적의 파라미터를 찾기 위해 음의 로그 가능도 함수를 목적함수 (손실함수) 로 만들어, 이를 최소화하는 방향으로 단어 벡터를 정한다.

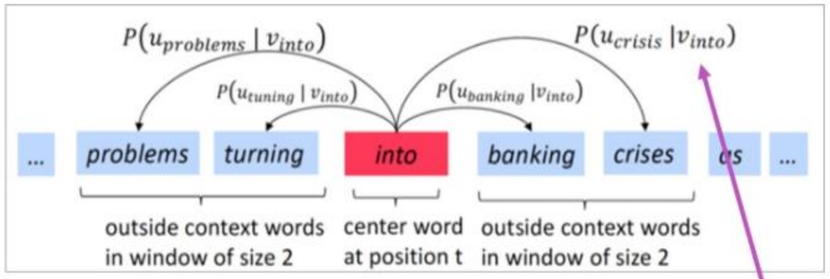
목적함수 최소화

⇔ 다른 단어들의 context 에서 중심 단어를 잘 예측한다.

EWHA EURON

Word2vec

3. P(o|c)



$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Exponential 을 씌워 양의 값을 출력하도록 함

Dot product(내적)으로 o 와 c 의 similarity 를 비교.

$$P(o|c) = \frac{exp(u_o^T \cdot v_c))}{\sum_{w \in V} exp(u_w^T \cdot v_c))}$$

$$u_o^T \cdot v_c = u.v = \sum_{i=1}^{n} u_i v_i$$

큰 값이 나올 수록 확률이 등장할 확률이 높다

확률분포를 제공하기 위해

전체 단어에 대해

Normalize(정규화)한다.

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

😉 P(Wt+j | Wt ; θ) 는 어떻게 계산하지?

👉 특정 단어 w 에 대해 2가지 벡터 개념을 도입한다.

⇨ Vw : center word 중심단어 (c)

⇨ Uw : context word 주변단어 (o)

* Optimization 을 쉽게하기 위해 2개의 벡터 개념을 도입

벡터의 내적 계산을 통해 유사도를 측정하며
이후 소프트맥스 과정을 거쳐 확률을 계산한다.

Word2vec

4. Optimization

👀 최적화 방식을 통해 목적함수를 최소화하는 파라미터 Θ , 즉 u 와 v 를 찾는다.

$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

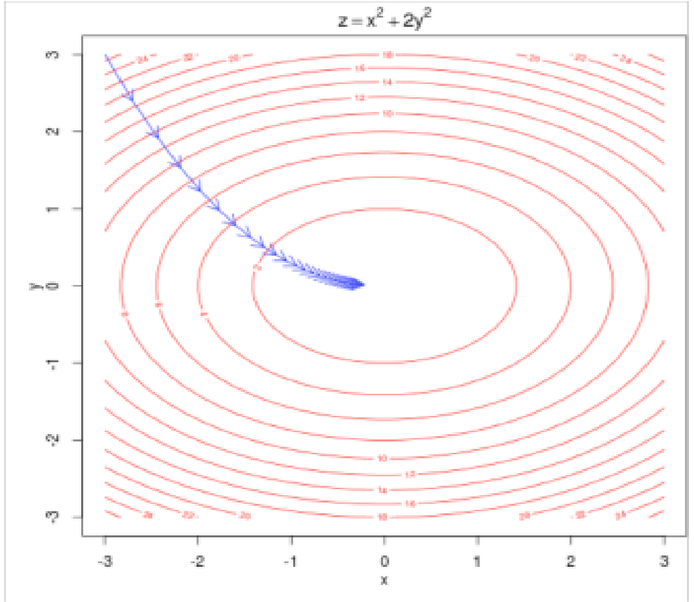
📌 Θ : V개의 단어에 대한 d 차원의 단어 벡터들을 하나의 긴 벡터로 나타낸 것

* 하나의 W 단어에 대해 v 벡터와 u 벡터가 있어야되므로 전체 차원은 2dV 가 됨

초기값으로 random value 로 시작했다가 가중치가 조절되며 문맥을 포함하는 최적의 단어 벡터로 표현된다.

◈ 최적화 방식에는
Gradient descent (강의1), Stochastic Gradient descent (강의2) 등을 적용

📌Gradient descent



초기값을 설정한 후 반복적인 갱신을 통해 최소값에 도달하는 알고리즘으로 경사가 가장 가파른 방향으로 최소값을 찾는다 하여 경사하강법이라고 한다. 이때 갱신되는 속도를 결정할 수 있는게 특징 (learning rate) 이다.

EWHA EURON

Word2vec

Gradient descent

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\neq 0} \log p(w_{t+j}|w_t)$$

$$\log p(o|c) = \log \frac{\exp({u_o}^T v_c)}{\sum_{w=1}^{V}\exp({u_w}^T v_c)}$$

Center word 와 context word 로 각각 미분한다.

* 강의 판서 부분

$$\frac{\partial}{\partial v_c}\log\frac{\exp(u_o^T v_c)}{\sum_{w=1}^{v}\exp(u_o^T v_c)} = \frac{\partial}{\partial v_c}\log\exp(u_o^T v_c) - \frac{\partial}{\partial v_c}\log\sum_{w=1}^{v}\exp(u_o^T v_c)$$

$$\frac{\partial}{\partial v_c}\log\exp(u_o^T v_c) - \frac{\partial}{\partial v_c}\log\sum_{w=1}^{v}\exp(u_o^T v_c)$$

$$= \frac{\partial}{\partial v_c}(u_{o1}v_{c1} + u_{o2}v_{c2} + \dots + u_{o100}v_{c100} + \dots) - \frac{\partial}{\partial v_c}\log\sum_{w=1}^{v}\exp(u_o^T v_c)$$

$$= \frac{\partial}{\partial v_c}(u_o^T v_c) - \frac{\partial}{\partial v_c}\log\sum_{w=1}^{v}\exp(u_o^T v_c)$$

$$= u_o - \frac{\partial}{\partial v_c}\log\sum_{w=1}^{v}\exp(u_o^T v_c)$$

Word2vec

$$u_o - \frac{\partial}{\partial v_c} \log \sum_{w=1}^{v} \exp(u_o^T v_c)$$ f / Z (Vc)

$$= u_o - \frac{1}{\sum_{w=1}^{v} \exp(u_w^T v_c)} \times \frac{\partial}{\partial v_c} \sum_{x=1}^{v} \exp(u_x^T v_c)$$ f / Z (Vc)

$$= u_o - \frac{1}{\sum_{w=1}^{v} \exp(u_w^T v_c)} \times \sum_{x=1}^{v} \exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c$$

$$= u_o - \frac{1}{\sum_{w=1}^{v} \exp(u_w^T v_c)} \times \sum_{x=1}^{v} \exp(u_x^T v_c) \cdot u_x$$

$$= u_o - \frac{\sum_{x=1}^{v} \exp(u_x^T v_c) \cdot u_x}{\sum_{w=1}^{v} \exp(u_w^T v_c)}$$

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = \frac{df(u)}{du}\frac{dg(x)}{dx}$$

$$u_o - \frac{\sum_{x=1}^{v} \exp(u_x^T v_c) \cdot u_x}{\sum_{w=1}^{v} \exp(u_w^T v_c)}$$

$$= u_o - \sum_{x=1}^{v} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{v} \exp(u_w^T v_c)} \cdot u_x$$

$$= u_o - \sum_{x=1}^{v} p(x|c) \cdot u_x$$

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{v} \exp(u_o^T v_c)} = u_o - \sum_{x=1}^{v} p(x|c) \cdot u_x$$

관측된 context 단어

Model 이 예측한 context word

💡 핵심 : 목적함수를 Vc에 대해 편미분 한 것 → (실제 단어와 예측한 단어와의 차이 값)

👉 gradient descent 방식을 통해 실제값과 예측값의 차이를 줄여 나갈 수 있다.

Word2vec

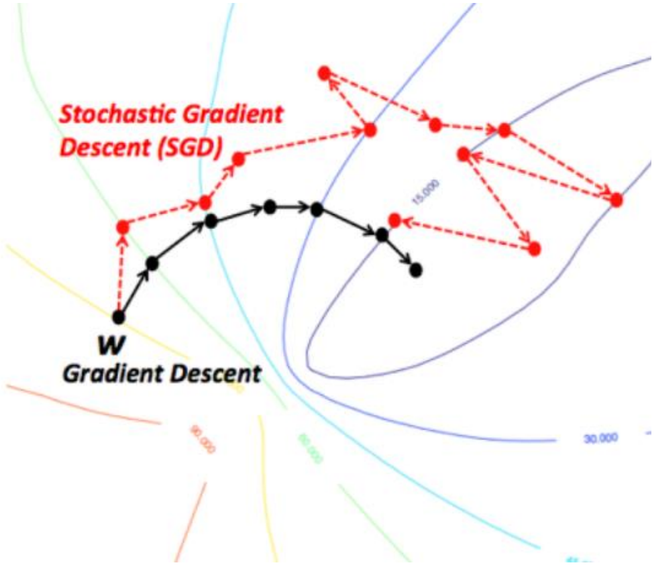
👀 경사하강법은 한번의 갱신마다 모든 데이터들에 대해 계산이 이루어지므로 연산량이 많아!

👉 한 번에 한 개만 Random sampling 을 통하여 일부분을 추출해 해당 포인트에 대한 Gradient 를 계산하자

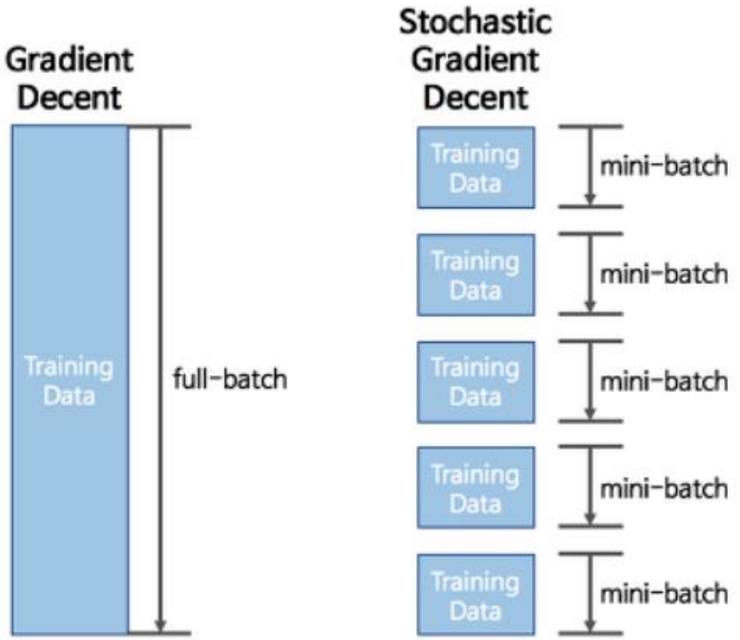
(즉, 조금만 훑어보고 빠르게 최소값으로 가보자는 의미)

✨ Stochastic Gradient Descent

연산 시간 단축 & Shooting 이 일어나므로 local min 에 빠질 위험이 적지만, global optimum 을 찾지 못할 가능성이 있다.

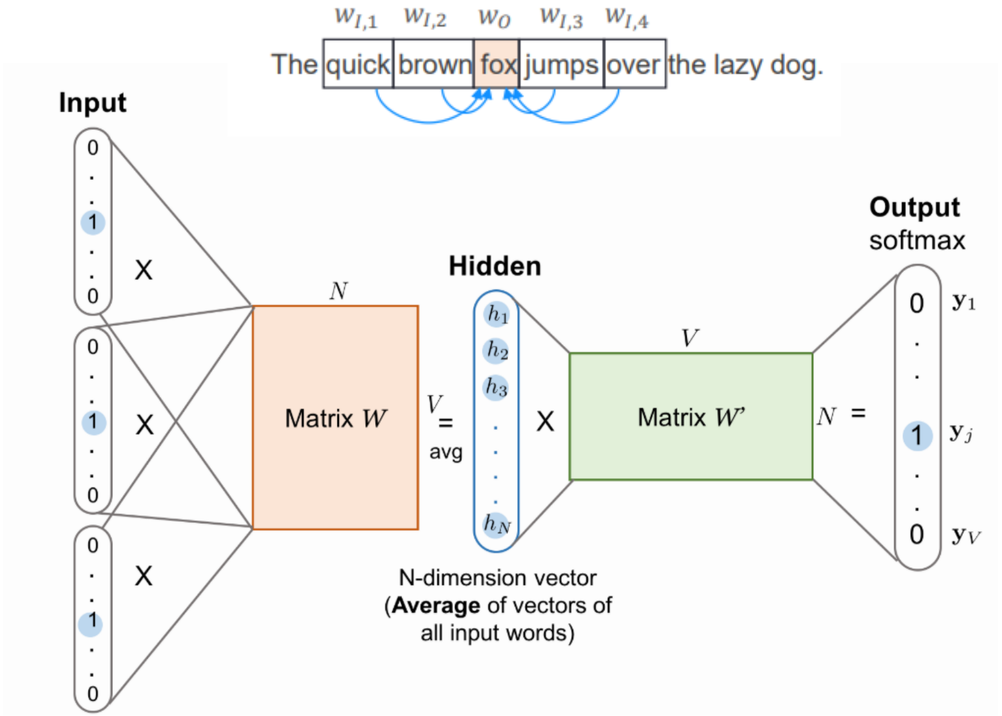


- GD
  - 모든 데이터를 계산한다 => 소요시간 1시간
  - 최적의 한스텝을 나아간다.
  - 6 스텝 * 1시간 = 6시간
  - 확실한데 너무 느리다.
- SGD
  - 일부 데이터만 계산한다 => 소요시간 5분
  - 빠르게 전진한다.
  - 10 스텝 * 5분 => 50분
  - 조금 헤메지만 그래도 빠르게 간다!



4.2 CBOW

Continuous bag of words (CBOW) : 주변 단어로 중심 단어를 예측하는 용도의 신경망



- input : 주변 단어의 원-핫인코딩 벡터
- output : 중앙 단어의 원-핫인코딩 벡터

⇨ 출력벡터를 잘 맞추기 위해 가중치를 갱신하며 학습

$$P(x_c|x_{c-m},\dots x_{c-1},x_{c+1},\dots,x_{c+m})$$

1. Input layer → hidden layer : embedding vector 얻기

input

$$[0\ \ 0\ \ 0\ \ 1\ \ 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10\ \ 12\ \ 19]$$

W

$$(v_{c-m} = \mathbf{W}x^{c-m},\dots,v_{c+m} = \mathbf{W}x^{c+m}) \in \mathbb{R}^n$$

embedding vector

(Word2vec의 최종 결과물)

2. hidden layer

$$\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \cdots + v_{c+m}}{2m} \in \mathbb{R}^n$$

⇨ embedding vector 의 평균이 hidden layer 가 된다.

3. hidden layer → output layer

가중치 파라미터 U를 곱해 score 를 계산

hidden layer ⇨ output layer

$$z = \mathbf{U}\hat{v} \in \mathbb{R}^{|V|}$$

$$\hat{y} = softmax(z) \in \mathbb{R}^{|V|}$$ ⇨ z vector 를 softmax 함수를 취하여 확률값으로 표현

4. Loss ⇨ yhat 을 기준 단어의 one-hot vector 인 y와 오차 측정 ⇨ backpropagating

$$\mathcal{L}(\hat{y},y) = -\sum_{j=1}^{|V|} y_j \log \hat{y}_j = -y_i \log \hat{y}_i$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\hat{y},y)$$

$$\mathcal{L} = -\log \mathcal{P}(w_c|w_{c-m},\cdots,w_{c+m}) = -\log \mathbb{P}(u_c|\hat{v})$$

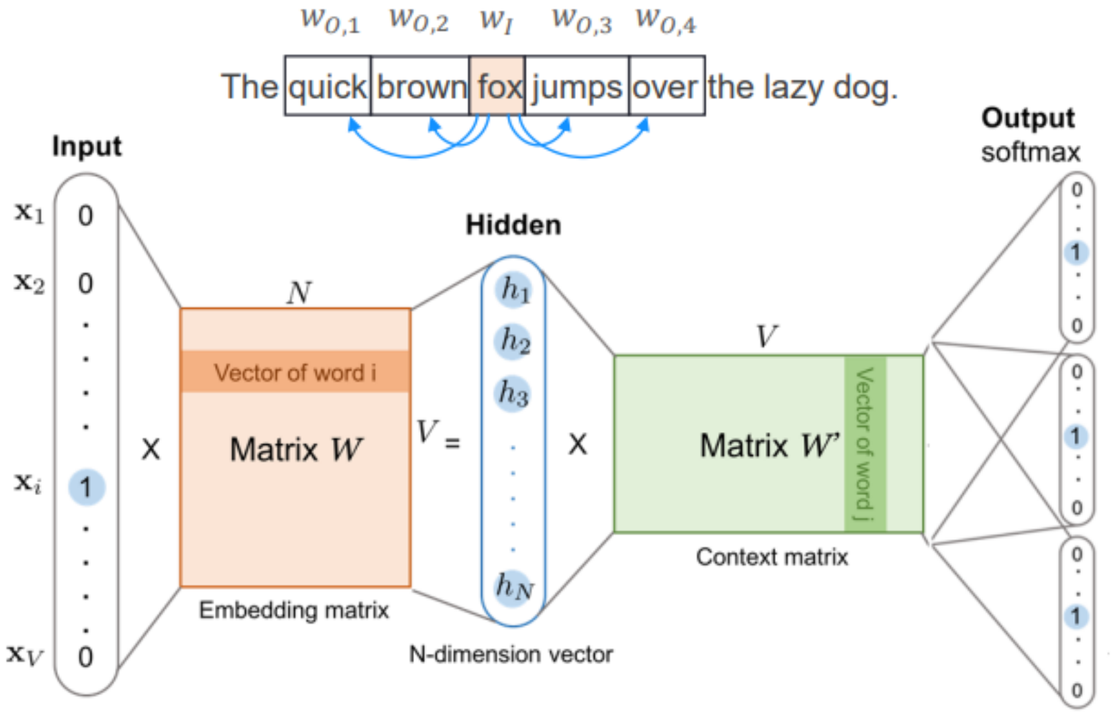
$$= -\log \frac{exp(u_c^\top \hat{v})}{\sum_{j=1}^{|V|} exp(u_j^\top \hat{v})}$$

$$= -u_c^\top \hat{v} + \log \sum_{j=1}^{|V|} exp(u_j^\top \hat{v})$$

4.3 Skip-gram

SkipGram : 중심단어로 주변 단어를 예측하는 용도의 신경망

CBOW 와 마찬가지로 input, ouput 형식은 원핫인코딩 벡터가 되고, hidden layer 를 거쳐 softmax 함수를 사용해 output layer가 나오는 방식으로 학습이 이루어진다.



CBOW 보다 더 어려운 문제에 도전하는 모델이므로 단어의 분산 표현이 더 뛰어날 가능성이 있다. 따라서 성능이 우수해 보통 skipgram 을 더 많이 사용한다.

1. Input layer → hidden layer : embedding vector 얻기

$$x \in \mathbb{R}^{|V|} \qquad v_c = \mathbf{W}x \in \mathbb{R}^n$$

2. hidden layer → output layer

$$z = \mathbf{W}'v_c \qquad \hat{y} = \text{softmax}(z)$$

$$\hat{y}_{c-m}, \ldots, \hat{y}_{c-1}, \hat{y}_{c+1}, \ldots, \hat{y}_{c+m}$$

3. Loss

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

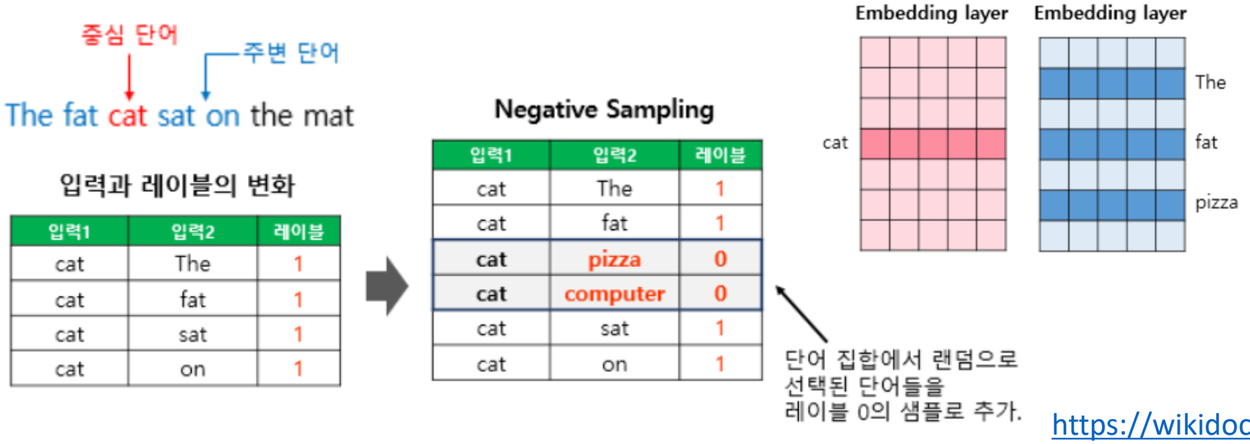
4.4 computational efficiency

기존 학습의 문제점

- objective (loss) function 연산 : 단어 개수 |V| 가 많아지면 계산 복잡도 역시 증가 $O(|V|)$
- 효율적인 학습을 위한 다양한 방법들이 제시됨

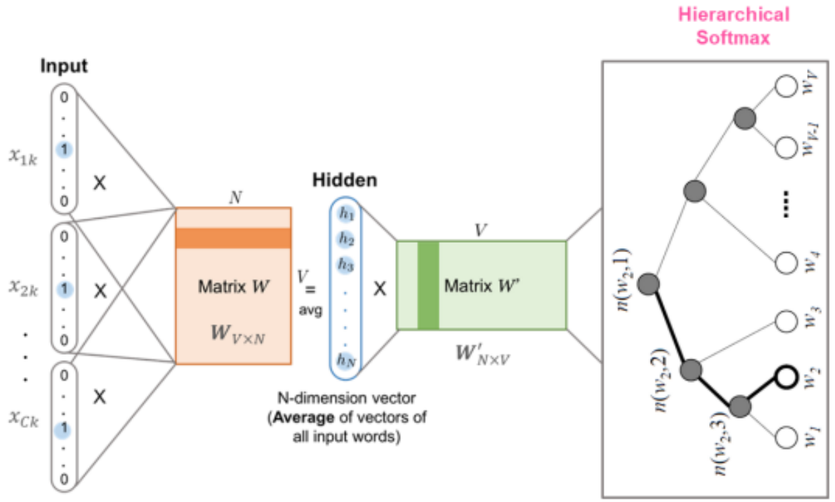
✓ Negative sampling

: 모든 단어의 가중치를 업데이트 시키지 않게 하는 방법으로, 특정 단어에 대한 주변단어와 negative sample (단어 집합에서 무작위로 선택된 주변 단어가 아닌 단어들) 단어 집합을 만들고, 이진 분류 문제로 변환하여 가중치를 업데이트 한다. (기존 방법은 전체 단어 집합 만큼의 크기를 두고 다중 클래스 분류로 접근)



https://wikidocs.net/69141

✓ Hierarchical softmax



모든 단어를 이진 트리 형태로 나타냄
⇨ 이진 분류로 log 2(corpus) 로 계산량이 줄어듦

https://analysisbugs.tistory.com/184

word2vec

```
from gensim.models import Word2Vec
```

https://hoonzi-text.tistory.com/2

하이퍼 파라미터

1. size : 임베딩 된 벡터 차원수를 결정. 너무 작은 값이 아니라면 학습의 경향성은 유지된다. 보통 50~500 사이의 숫자에서 결정한다.
2. window : 훈련시 앞 뒤로 고려하는 단어의 개수로, 값이 커지면 학습량 증가 & 계산량 증가 & 단어의 의미적 정확도 증가
3. sg : 0 으로 설정하면 CBOW, 1 로 설정하면 skip-gram 방식으로 훈련한다.
4. Min_count : 단어 최소 빈도 수 제한 (빈도가 적은 단어들은 학습하지 않는다)
5. Workers : 학습을 위한 프로세스 수

좋은 토큰의
좋은 임베딩 예

```
model_result = model.wv.most_similar("man")
print(model_result)
```

```
[('woman', 0.842622697353363), ('guy', 0.8178728818893433), ('boy', 0.7774451375007629), ('lady', 0.7767927646636963), ('girl', 0.7583760023117065), ('gentleman', 0.7437191009521484), ('soldier', 0.7413754463195801), ('poet', 0.7060446739196777), ('kid', 0.6925194263458252), ('friend', 0.6572611331939697)]
```

```
from gensim.test.utils import common_texts
from gensim.models import Word2Vec


print('CBOW Word2Vec 에 훈련시킬 문장들 : ₩n')
print(common_texts)
Our_CBOW_Word2Vec_Model = Word2Vec(common_texts, size = 10, window =5, min_count =1, workers = 8, sg = 0)

# sg = 0 : CBOW 로 훈련
# sg = 1 : skip-gram 으로 훈련
```

길이가 10인
벡터로 표현해줘

토큰화된 문장
리스트

Cbow 방식

```
CBOW Word2Vec 에 훈련시킬 문장들 :

[['human', 'interface', 'computer'], ['survey', 'user', 'computer', 'system', 'response', 'time'], ['eps', 'user', 'interface', 'system'], ['system',
```

```
[ ]  Our_CBOW_Word2Vec_Model.wv.most_similar('human', topn=5)

[('trees', 0.4093015193939209),
 ('user', 0.3950405418872833),
 ('survey', 0.33694708347320557),
 ('interface', 0.19255289435386658),
 ('computer', 0.17239470779895782)]
```

⇨ 벡터 기반 유사도 측정기준

비슷한 단어 출력

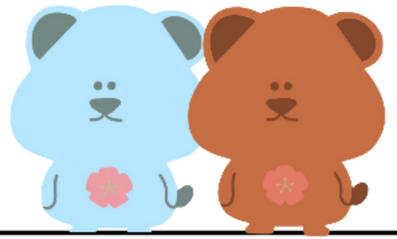
```
[ ]  Our_CBOW_Word2Vec_Model.wv['human']

array([ 0.00997837,  0.01539693, -0.01250029,  0.02434195,  0.04359537,
       -0.0496188 , -0.04765006, -0.03096483, -0.02092905,  0.02508552],
      dtype=float32)
```

⇨ embedding vector

Word2vec의 최종 결과물!

THANK YOU



EWHA EURON