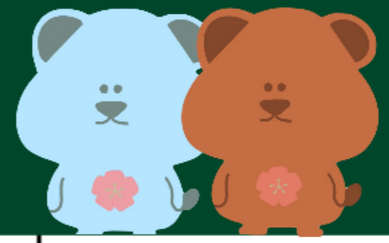
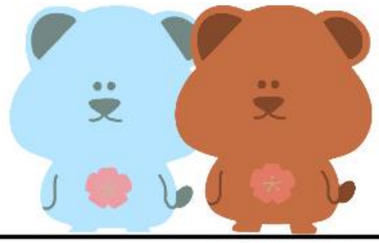


Lecture 1 - Intro & Word Vectors

Week1_발표자: 손소현, 조서영

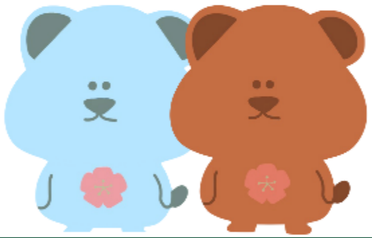
EWHA EURON

목차



EWHA EURON

Human language and word meaning



#01 Human language and word meaning

Human languages and Word meaning

- language
  사람들이 한 말이 어떤 의미인지 '확률 기반의 추론 과정'
  사람들은 '지식 ' 을 'language' 를 통해 -> 사람만이 그렇게 하기 때문에 특별하지만 느리다
- meaning
  사물이 무엇을 나타내는지 (Denotational Semantics)

Commonest linguistic way of thinking of meaning:

signifier (symbol) ⟺ signified (idea or thing)

= denotational semantics

#01 Human language and word meaning

WorNet

- 유의어(Synonym)과 상의어(Hypernym)들의 집합(sets)을 포함하고 있는 백과사전

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))
```

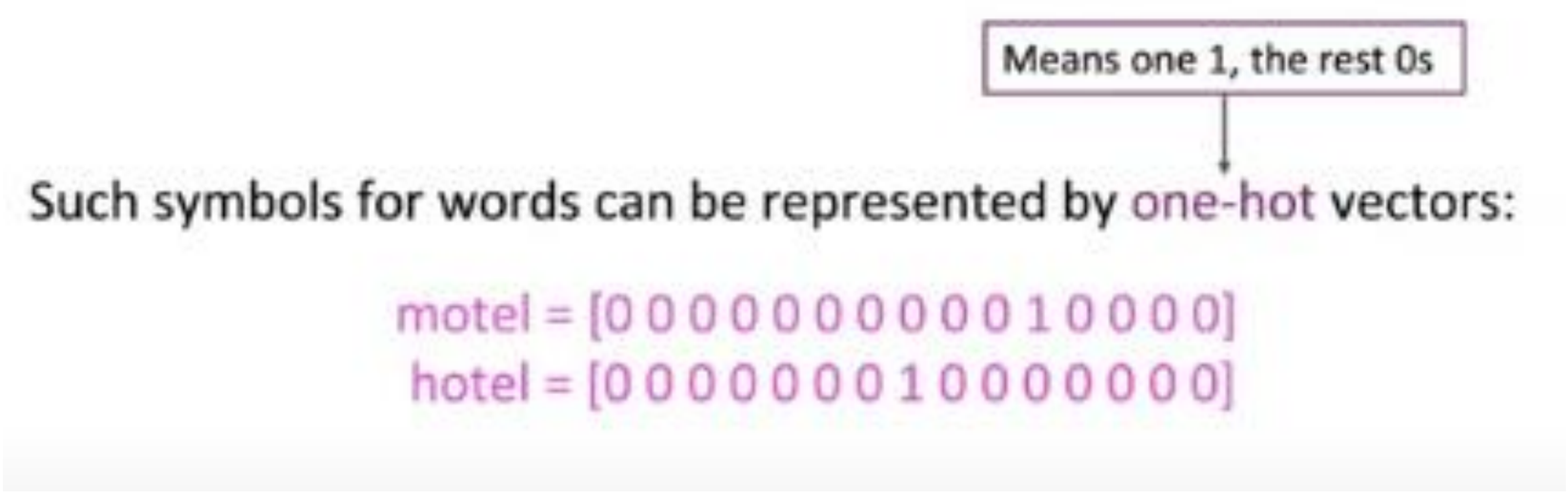
```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),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

- 단어의 뉘앙스를 잃어버림 ex) proficient 가 good의 유의어가 되는 것은 같은 맥락에 있을 때만 가능
- 새로운 의미를 놓침 (매일매일 업데이트하는 것은 불가능) – 주관적임… 등등

#01 Human language and word meaning

One-Hot Vectors

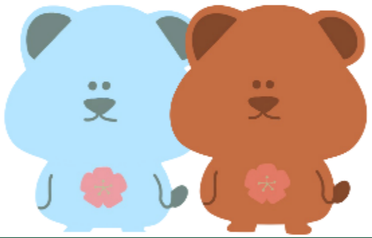
- 벡터의 차원 수 = 단어의 수 (ex 500000)



- language에는 거의 무한대에 가까운 단어가 있음
- 단어간의 관계를 알기 어려움
  ex) motel과 hotel은 아무런 관계가 아님

-> instead : learn to encode similarity in the vectors themselves

Word2vec algorithm introduction

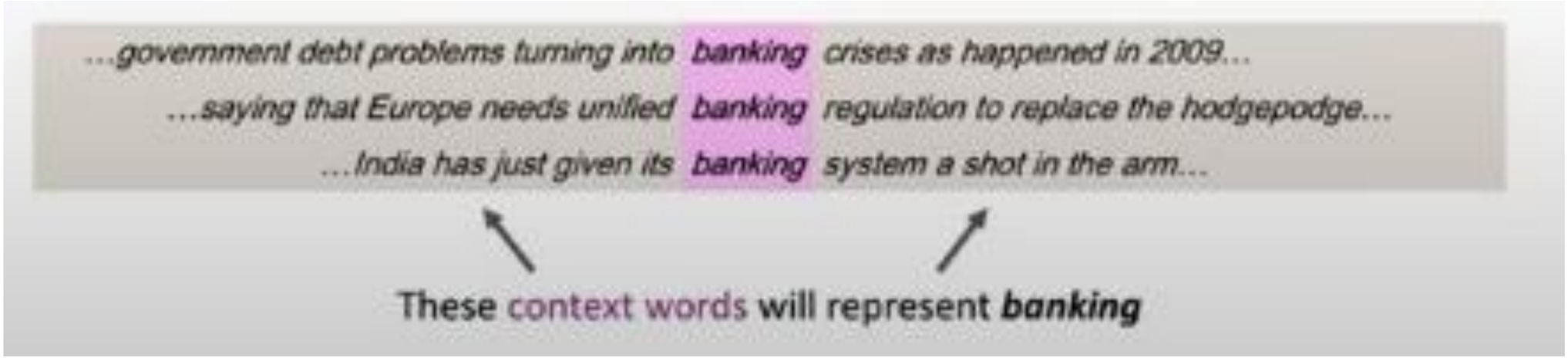


EWHA EURON

#02 Word2vec algorithm introduction

Distributional semantics (해결 방법!!)

- vector 자체에 단어간의 유사도를 인코드함
- 고정된 사이즈의 window를 통해 단어를 표현할 때 주위를 살핌
- 비슷한 문맥에서 나타나는 비슷한 단어들끼리 유사한 벡터를 갖음



#02 Word2vec algorithm introduction

Distributional semantics (해결 방법!!)

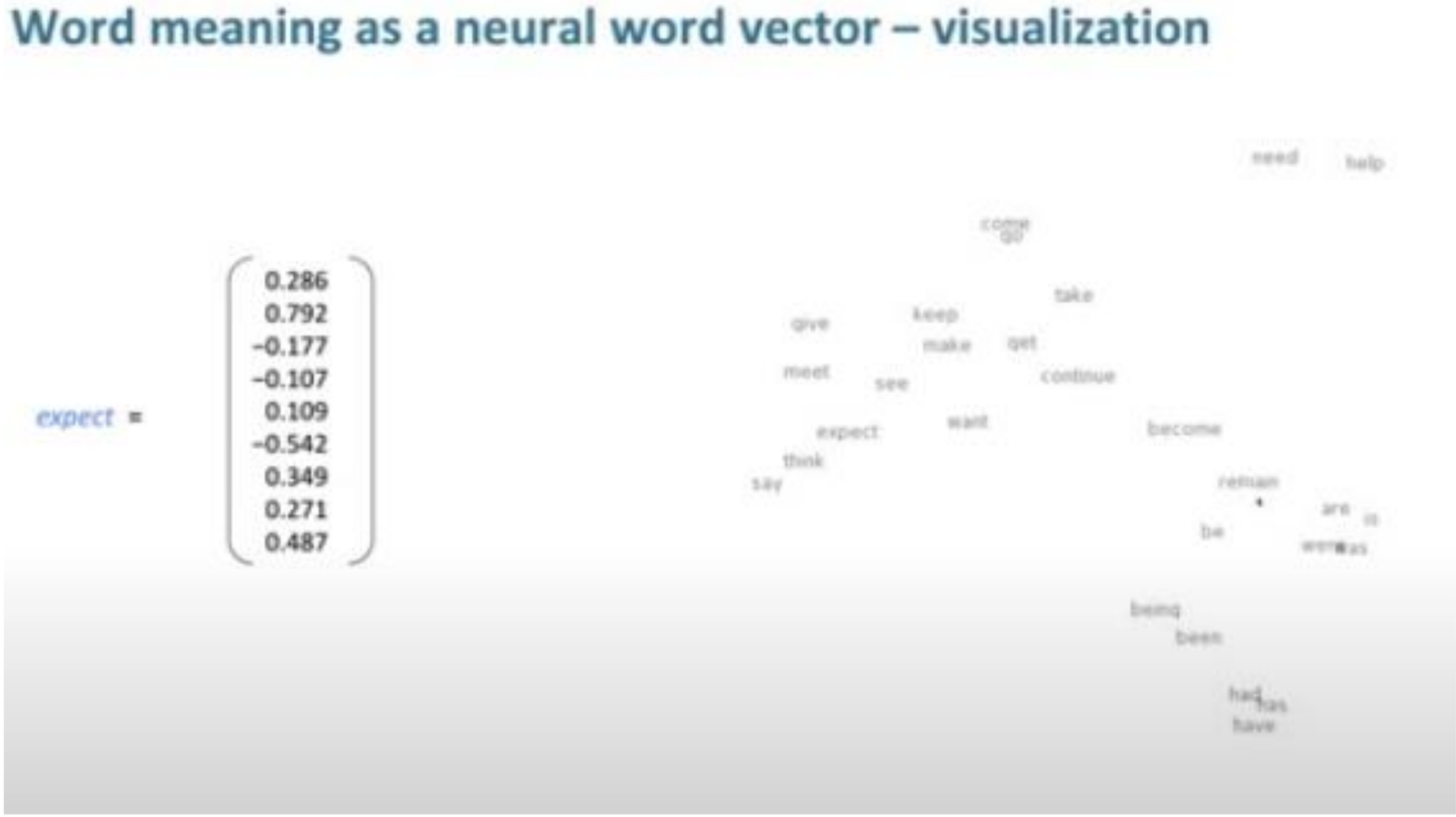
Word Vector = Word Embeddings = Word Representations

단어의 비슷한 벡터는 비슷한 문맥을 가지고 있음

#02 Word2vec algorithm introduction

Distributional semantics (해결 방법!!)

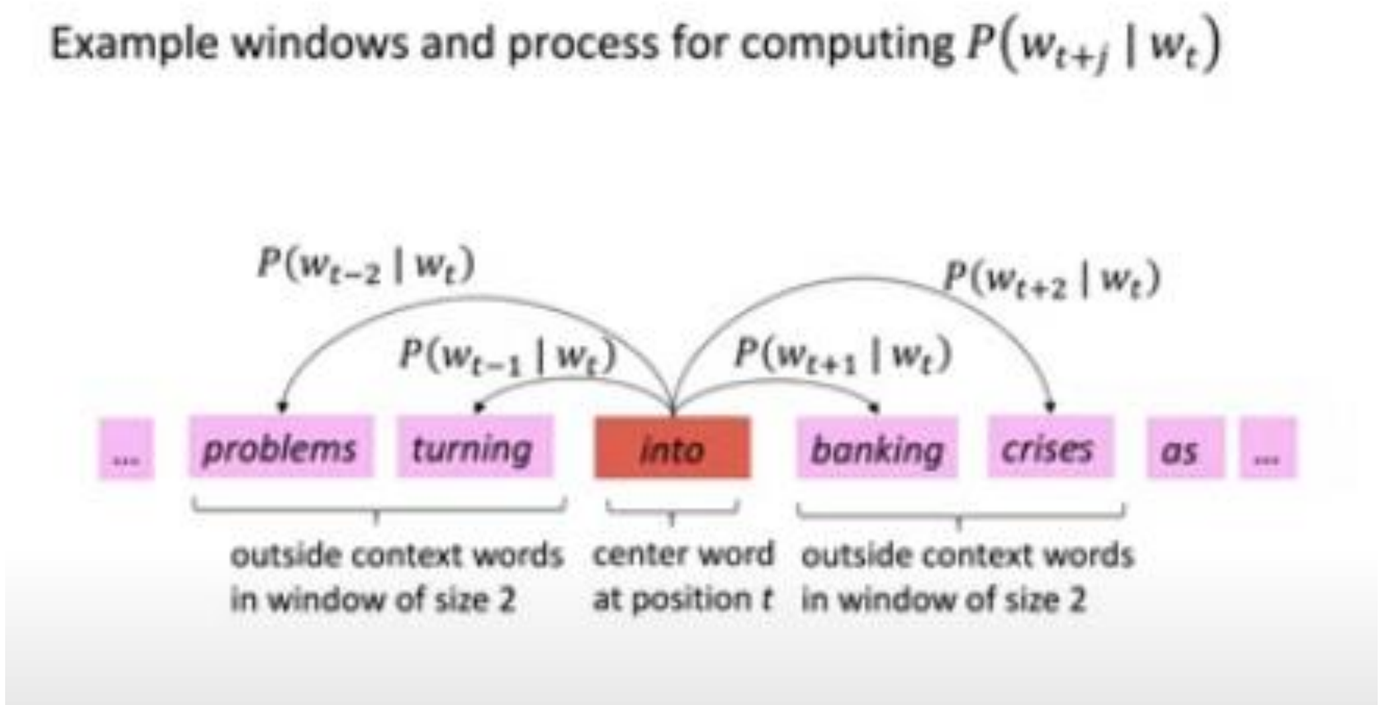
Word Vector = Word Embeddings = Word Representations



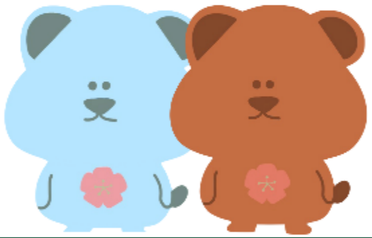
#02 Word2vec algorithm introduction

Word2vec

- 거대한 텍스트의 말뭉치(corpus, body)가 있음
- 각 단어는 벡터에 의해서 표현됨
- corpus내의 모든 단어를 방문하며 학습, 현재위치 t에 위치한 단어는 c, 주변은
- c와 o의 유사도를 이용해 p(o|c) 또는 p(c|o)를 계산
- 추정된 확률 값을 최대화 하는 방향으로 word vector를 변경해 나간다.



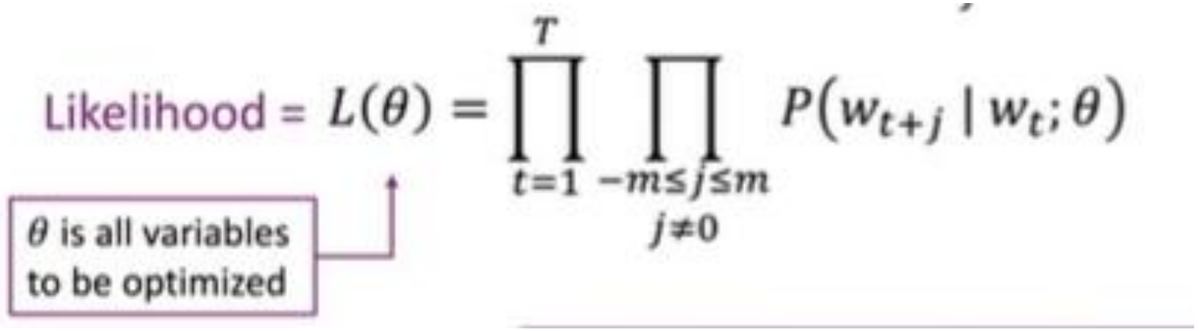
Word2vec objective function and gradients



#03 Word2vec objective function and gradients

계산 방법

1. Likelihood

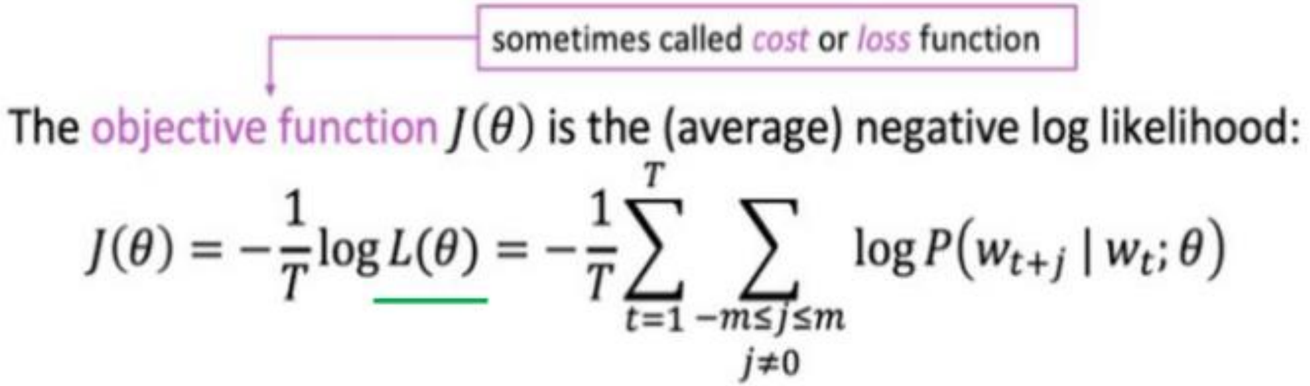


- 모든 단어에 대해 고정된 크기의 window만큼 주변의 단어 확률을 곱한다는 의미
- 주변 단어의 발생 확률을 예측

#03 Word2vec objective function and gradients

계산 방법

2. Objective function



- 가능도를 최대화하는 최적을 파라미터를 찾기 위해 음의 로그 가능도 함수를 목적함수로 만듦
- 목적함수를 최소화하는 방향으로 단어 벡터를 정함
- 목적함수 최소화 = 다른 단어들의 context에서 중심 단어를 잘 예측함

#03 Word2vec objective function and gradients

계산 방법

3. P(o|c)

- We want to minimize the objective function:

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

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

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

Exponential 을 씌워 양의 값을 출력하도록 함

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

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

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

큰 값이 나올 수록 확률이 등장할 확률이 높다

확률분포를 제공하기 위해

전체 단어에 대해

Normalize(정규화)한다.

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

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

EWHA EURON

Optimization basics

#04 Optimization basics

목표: 최적화 방식을 통해 목적함수를 최소화하는 파라미터 θ, 즉 u와 v를 찾는 것

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

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

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

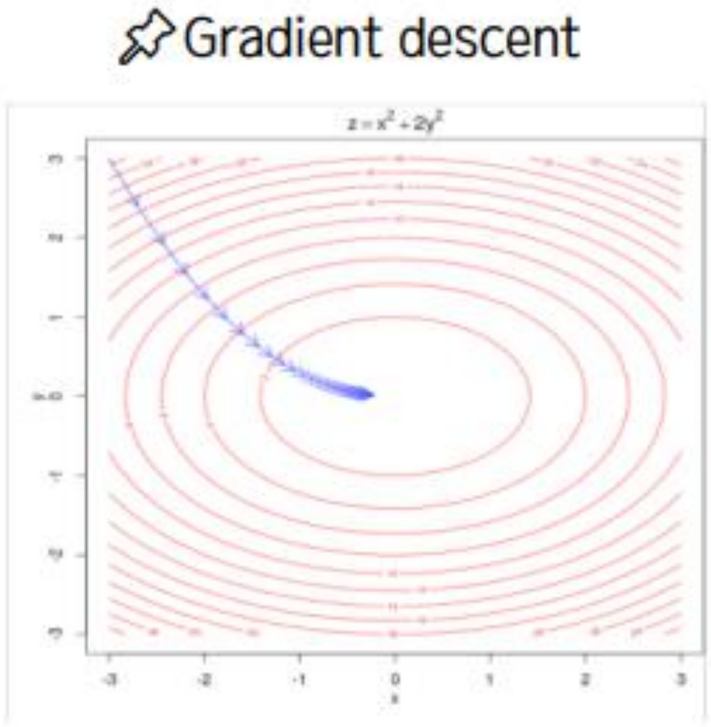
* 하나의 W 단어에 대해 v 벡터와 u 벡터가 있어야되므로 전체 차원은 2dV 가 됨

⬇

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

#04 Optimization basics

목표: 최적화 방식을 통해 목적함수를 최소화하는 파라미터 θ, 즉 u와 v를 찾는 것



- 초기값 설정 후 반복적인 갱신을 통해 최솟값에 도달하는 알고리즘
- 이때 갱신되는 속도를 결정할 수 있는 게 learning rate

#04 Optimization basics

- 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

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

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

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

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

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

#04 Optimization basics

- Gradient descent

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

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

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

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

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

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

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

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

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

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

관측된 context 단어

Model 이 예측한 context word

- 핵심: Objective functio을 Vc에 대해 편미분 한 것 = 실제 단어와 예측한 단어와의 차이 값
- Gradient descent를 통해 실제값과 예측값의 차이를 줄여 나갈 수 있다.

EWHA EURON

Looking at word vectors

#05 Looking at word vectors

- Gensim: 이미 학습되어 배포된 Word vector package
  - 단어를 숫자들로 구성된 벡터로 나타냄

```
In [20]: model['bread']
Out[20]: array([-0.66146  ,  0.94335  , -0.72214  ,  0.17403  , -0.42524  ,
                 0.36303  ,  1.0135   , -0.14802  ,  0.25817  , -0.20326  ,
                -0.64338  ,  0.16632  ,  0.61518  ,  1.397    , -0.094506 ,
                 0.0041843, -0.18976  , -0.55421  , -0.39371  , -0.22501  ,
                -0.34643  ,  0.32076  ,  0.34395  , -0.7034   ,  0.23932  ,
                 0.69951  , -0.16461  , -0.31819  , -0.34034  , -0.44906  ,
                -0.069667 ,  0.35348  ,  0.17498  , -0.95057  , -0.2209   ,
                 1.0647   ,  0.23231  ,  0.32569  ,  0.47662  , -1.1206   ,
                 0.28168  , -0.75172  , -0.54654  , -0.66337  ,  0.34804  ,
                -0.69058  , -0.77092  , -0.40167  , -0.069351 , -0.049238 ,
                -0.39351  ,  0.16735  , -0.14512  ,  1.0083   , -1.0608   ,
                -0.87314  , -0.29339  ,  0.68278  ,  0.61634  , -0.088844 ,
                 0.88094  ,  0.099809 , -0.27161  , -0.58026  ,  0.50364  ,
                -0.93814  ,  0.67576  , -0.43124  , -0.10517  , -1.2404   ,
                -0.74353  ,  0.28637  ,  0.29012  ,  0.89377  ,  0.67406  ,
                 0.86422  , -0.30693  , -0.14718  ,  0.078353 ,  0.74013  ,
                 0.32658  , -0.052579 , -1.1665   ,  0.87079  , -0.69402  ,
                -0.75977  , -0.37164  , -0.11887  ,  0.18551  ,  0.041883 ,
                 0.59352  ,  0.30519  , -0.54819  , -0.29424  , -1.4912   ,
                -1.6548   ,  0.98982  ,  0.27325  ,  1.009    ,  0.94544  ],

In [21]: model['croissant']
Out[21]: array([-0.25144  ,  0.52157  , -0.75452  ,  0.28039  , -0.31388  ,
                 0.274    ,  1.1971   , -0.10519  ,  0.82544  , -0.33398  ,
                -0.21417  ,  0.22216  ,  0.14982  ,  0.47384  ,  0.41984  ,
                 0.69397  , -0.25999  , -0.44414  ,  0.58296  , -0.30851  ,
                -0.076455 ,  0.33468  ,  0.28055  , -0.99012  ,  0.30349  ,
                 0.39128  ,  0.031526 , -0.095395 , -0.004745 , -0.81347  ,
                 0.27869  , -0.1812   ,  0.14632  , -0.42186  ,  0.13857  ,
                 1.139    ,  0.14925  , -0.051459 ,  0.37875  , -0.2613   ,
                 0.011081 , -0.28881  , -0.38662  , -0.3135   , -0.1954   ,
                 0.19248  , -0.52995  , -0.40674  , -0.25159  ,  0.06272  ,
                -0.32724  ,  0.28374  , -0.2155   , -0.061832 , -0.50134  ,
                 0.0093959,  0.30715  ,  0.3873   , -0.74554  , -0.45947  ,
                 0.40032  , -0.1378   , -0.26968  , -0.3946   , -0.64876  ,
                -0.47149  , -0.085536 ,  0.092795 , -0.034018 , -0.61906  ,
                 0.19123  ,  0.20563  ,  0.29056  , -0.010908 ,  0.15313  ,
                 0.33144  ,  0.33806  ,  0.061708 ,  0.20785  ,  0.65348  ,
                -0.053222 ,  0.18589  ,  0.32647  , -0.11923  ,  0.42008  ,
                -0.26931  ,  0.025489 ,  0.0036535,  0.1327   , -0.22763  ,
                 0.07564  ,  0.55773  ,  0.2978   ,  0.28144  ,  0.19775  ,
                -0.23582  ,  0.65303  ,  0.089897 ,  0.35844  ,  0.14304  ],
               dtype=float32)
```

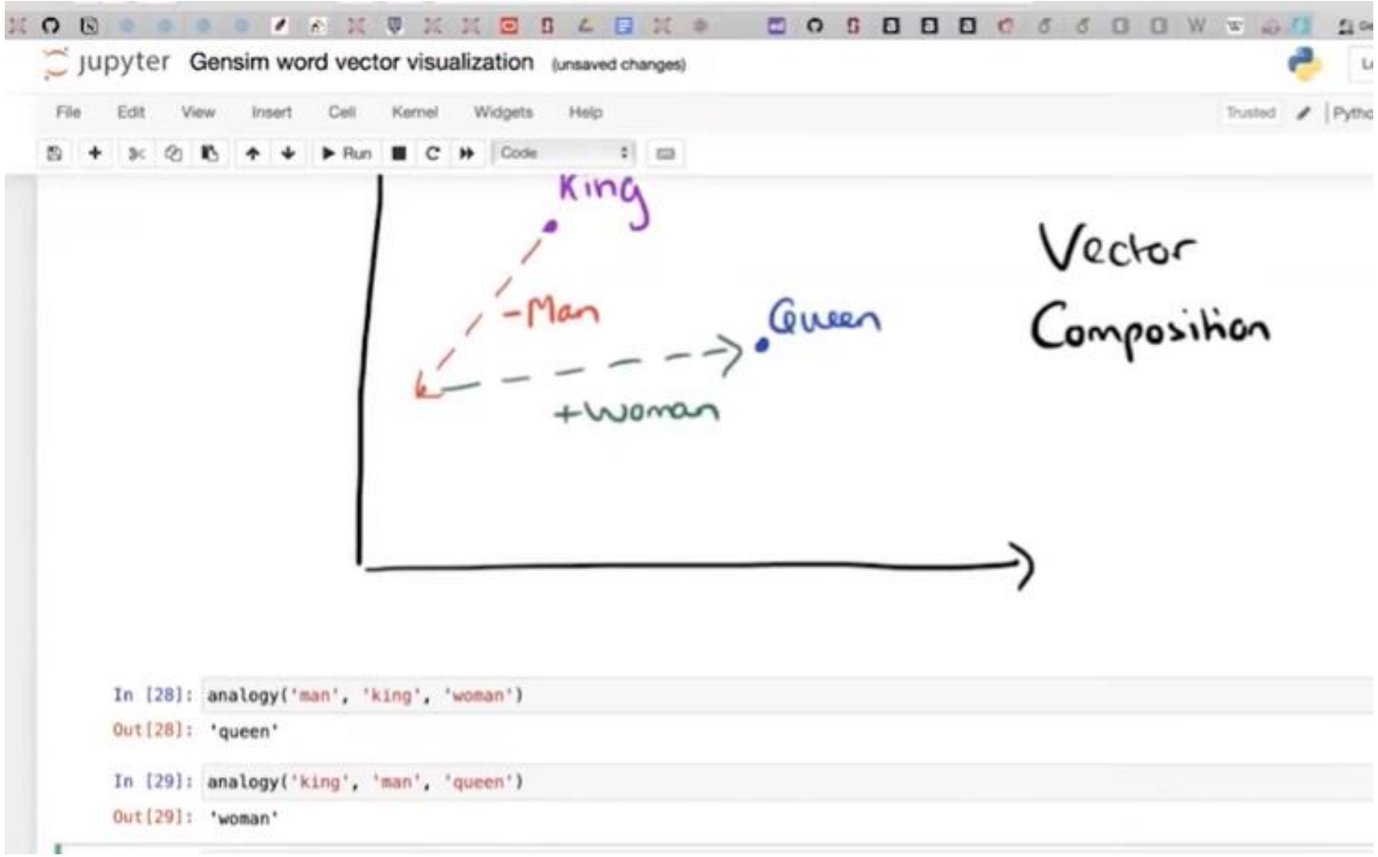
#05 Looking at word vectors

- Gensim: 이미 학습되어 배포된 Word vector package
  - 입력한 단어와 비슷한 뜻을 갖는 단어들 출력

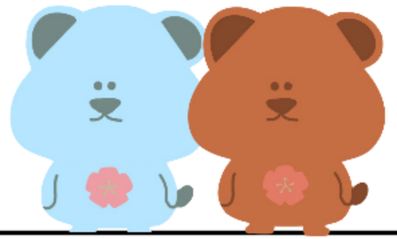
```
In [23]: model.most_similar('usa')
Out[23]: [('canada', 0.6544384360313416),
          ('america', 0.6452244520187378),
          ('u.s.a.', 0.618403434753418),
          ('united', 0.6017189621925354),
          ('states', 0.5970699191093445),
          ('australia', 0.5838716626167297),
          ('world', 0.5590084791183472),
          ('2010', 0.5580702424049377),
          ('2012', 0.5504006147384644),
          ('davis', 0.5464468002319336)]
```

#05 Looking at word vectors

- Gensim: 이미 학습되어 배포된 Word vector package
  - 단어 간의 관계



THANK YOU



EWHA EURON