13기 & 14기
텍스트 세미나

ToBig's 13기 조혜원

Introduction and Word Vectors

Contents

Unit 01 | How to represent Word Meaning

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')]
```

1. WordNet

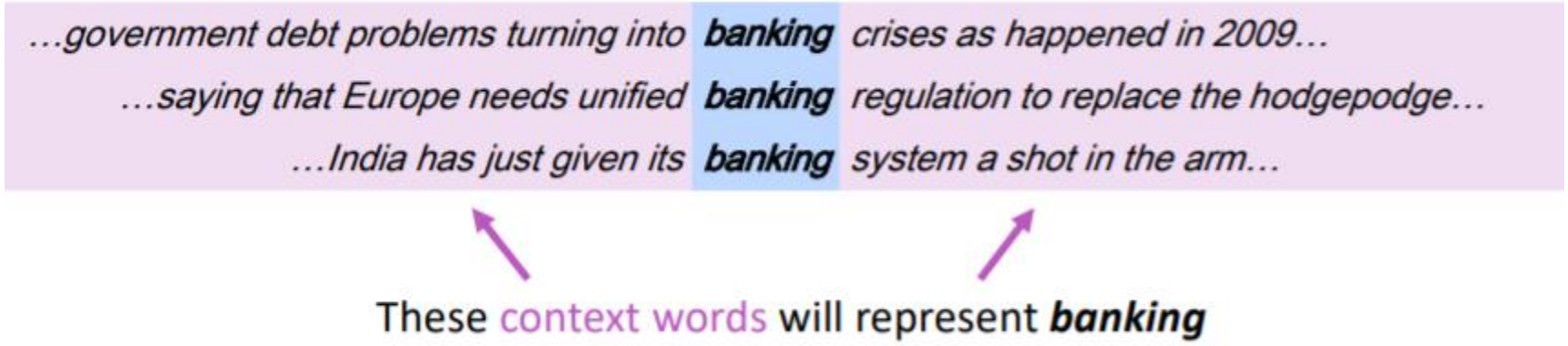
1) 동의어, 상하관계 언어의 집합
2) 단어 의미 간의 유사도와 관계를 얻기 어려움
3) 주관적인 판단 기준, 뉘앙스를 파악하기 어려움
4) 신조어 생성, 관리에 지속적 인력 투입이 필요

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

2. One-Hot Vectors

1) Vector의 차원이 아주 많이 필요

2) 이 역시 단어 간의 관계를 파악하려면 차원이 제곱이 되며 어려워짐



이를 해결한 게 3. Distributional Semantics

1) Vector 자체에 단어 간의 유사도를 인코드 함

2) 고정된 사이즈의 window를 통해 단어를 표현할 때 주위를 살핌

3) 비슷한 문맥에서 나타나는 비슷한 단어들끼리 유사한 벡터를 갖는다

Vector Space

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

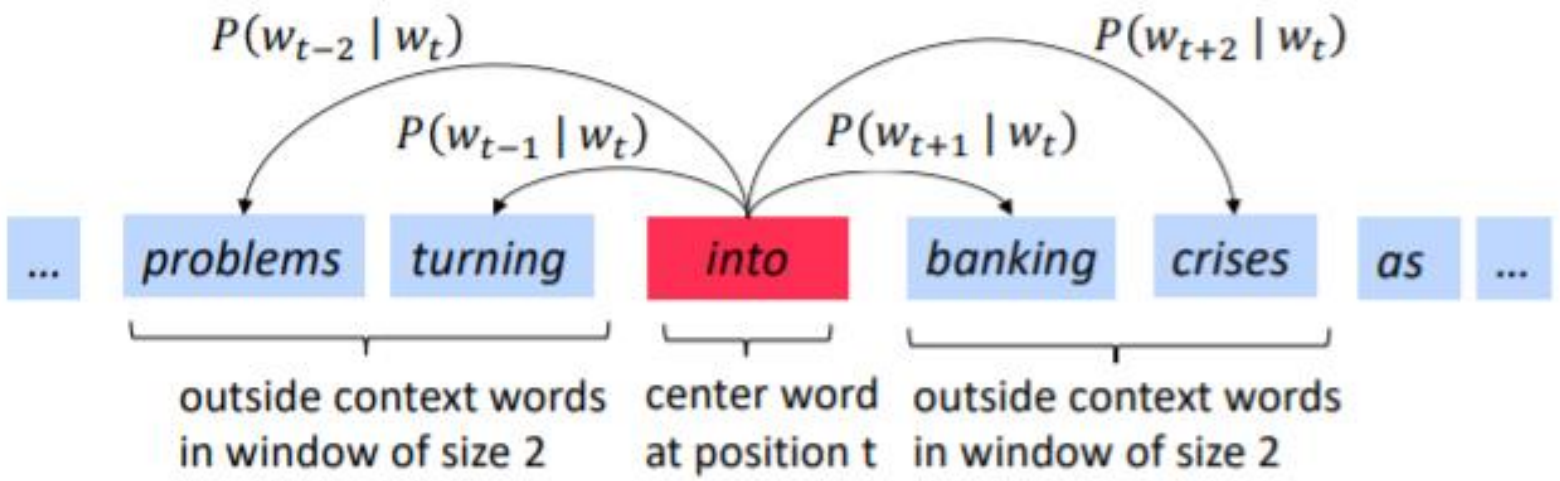
이와 같은 단어의 표현 방법을 Word Vector,
다른 말로 Word Embeddings, Word Representations이 라고 한다.

Word2Vec

1) Word Vector을 학습하는 프레임워크
2) 'Corpus(라틴어로 body란 뜻)'라고 부르는 큰 텍스트 뭉치에서 시작
3) 모든 단어가 벡터로 표현, 단어 벡터간의 유사도를 이용해 맥락에서 특정 단어가 나타날 확률을 계산

4) 가정
: 단어의 뜻을 학습하기에 충분한 corpus(문장의 모음)를 보유하고 있다

5) 목적
: 모든 학습 목표 단어들 각각을 잘 나타내는 vector를 찾는다



1) corpus내의 모든 단어를 방문하며 학습한다. 이때, 현재 위치 *t*에 위치한 단어는 *c*, 주변은 o

2) c와 o의 similarity를 이용해, p(o|c) 또는 p(c|o)를 계산한다

3) **추정된 확률값을 최대화**하는 방향으로 word vector의 값을 변경해나간다

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

- $\boldsymbol{\theta}$ (parameter): 단어를 Vector Space의 Vector로 나타낸 것

→ 각 단어들을 나타내는 Vector로 주변 word의 발생 확률을 예측

모든 position t에 위치한 단어에 대해 다음과 같은 과정을 반복한다.

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

θ is all variables to be optimized

.1) 현재 parameter가 주어졌을 때, 각 단어가 위치 t에서 나타날 때 현재와 같은 context가 t+j 위치에서 나타날 확률들의 곱인 likelihood를 계산한다.

모든 word에 대해(t) fixed size window만큼의 단어(j) 만큼 주변의 단어의 확률을 곱한다는 의미

$$J(\theta) = -\frac{1}{T}\log L(\theta)$$

- minus: maximize 대신 minimize하기 위해
- 1/T: 평균 (Size에 대한 dependency를 줄이면서 값 자체의 크기를 줄이기 위해)
- log: 곱셈을 덧셈으로 바꾸기 위해 (곱셈하는 것 앞에는 log를 붙이는 것이 효과적이다)

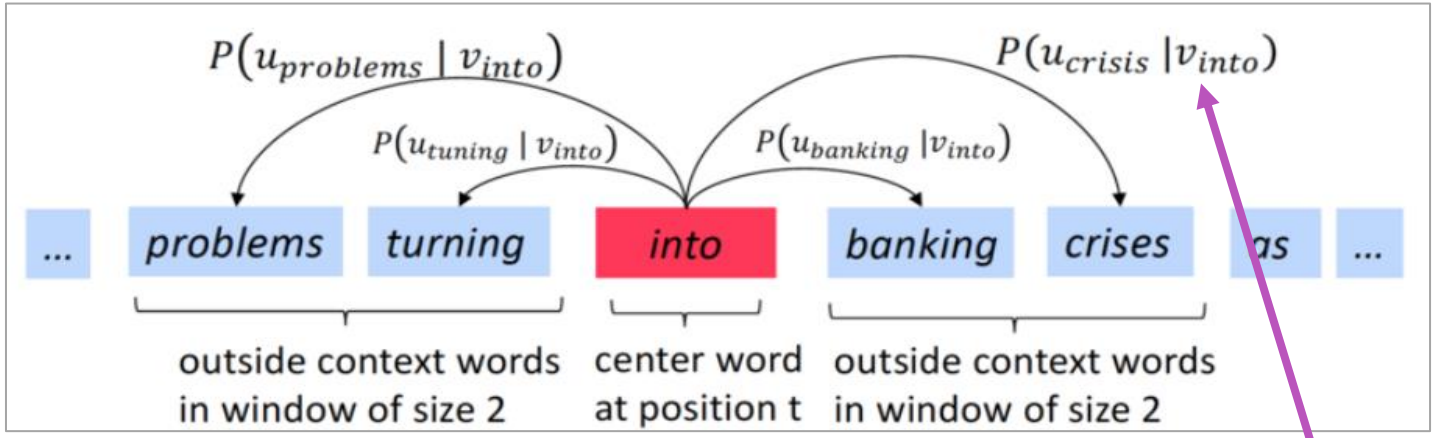
→ Loss Function 최소화 = Predictive Accracy 극대화!

sometimes called *cost* or *loss* function

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

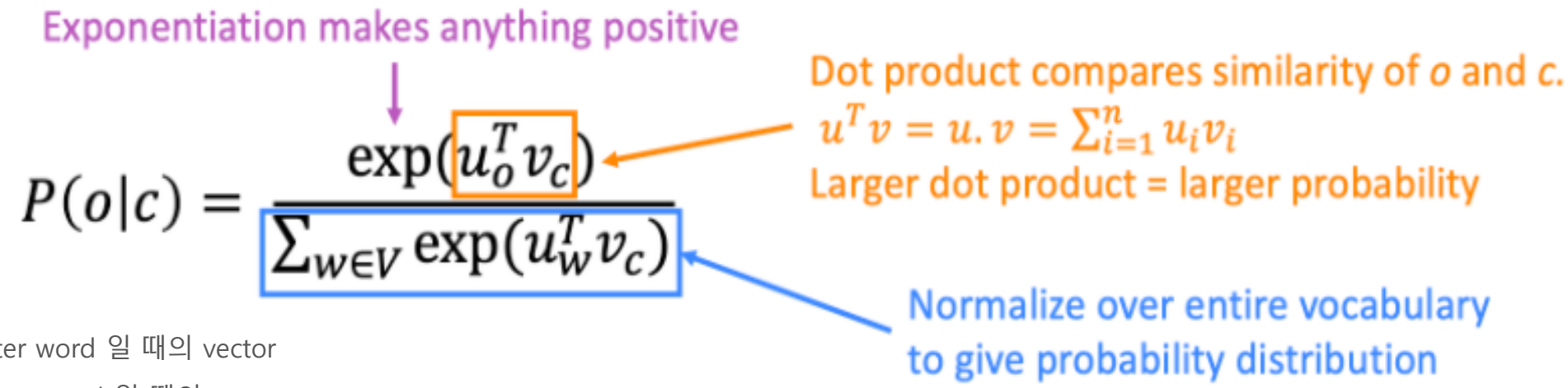
$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

2) 계산한 likelihood를 이용해 negative log likelihood와 일치하는 objective function을 구하고,
해당 목적식을 최소화하는 방향으로 parameter를 업데이트한다



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

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



Vw는 w가 center word 일 때의 vector

Uw는 w가 context word 일 때의 vector

word W에 대해 두 개의 Vector를 사용

- 항상 Positive로 만들기 위해서 exponential로 계산

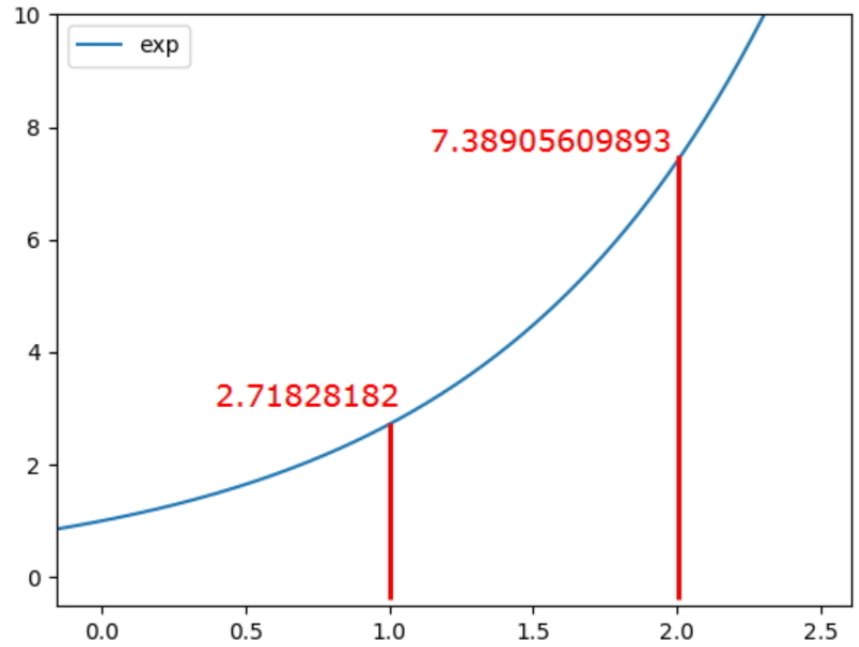
- Vector 간의 내적: 두 Vector 간의 유사도 측정! (값이 클수록 유사도가 높다)

- Softmax 함수와 거의 동일한 꼴 $\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$

+

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

0.25 0.25 0.5 => softmax => 0.2 0.2 0.6



Exp가 soft하지만 max하게 만들어주는 것입니다!

Word2vec의 Two model

1. Skip Gram

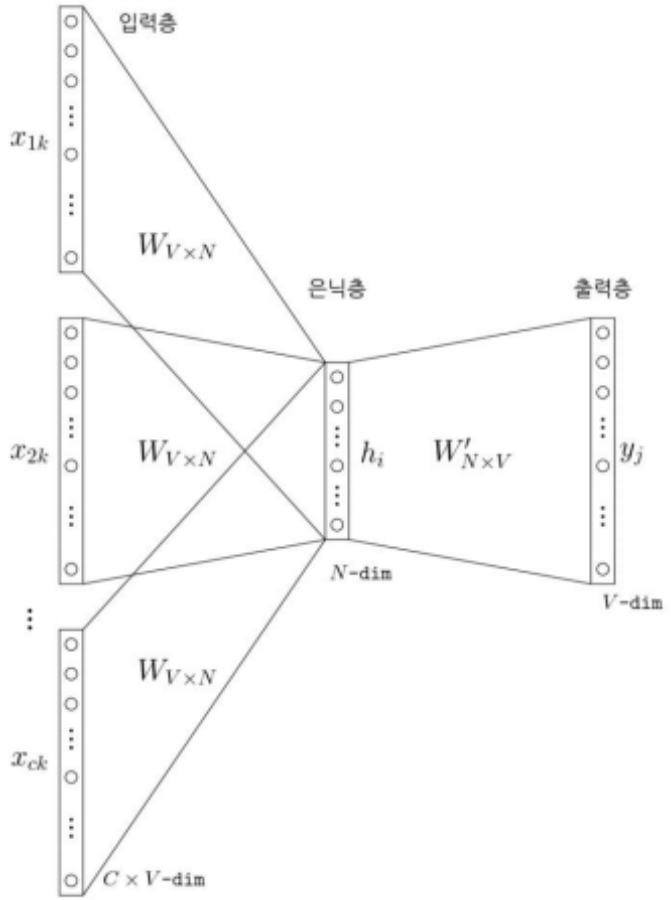
: 주어진 center word로 주변 단어를 예측하는 것 (lec2에서 자세히 다룸)

2. CBOW

: 주어진 주변 단어로 center word를 예측하는 것

CBOW

입력으로
Neighbor Words의
One-hot Vector

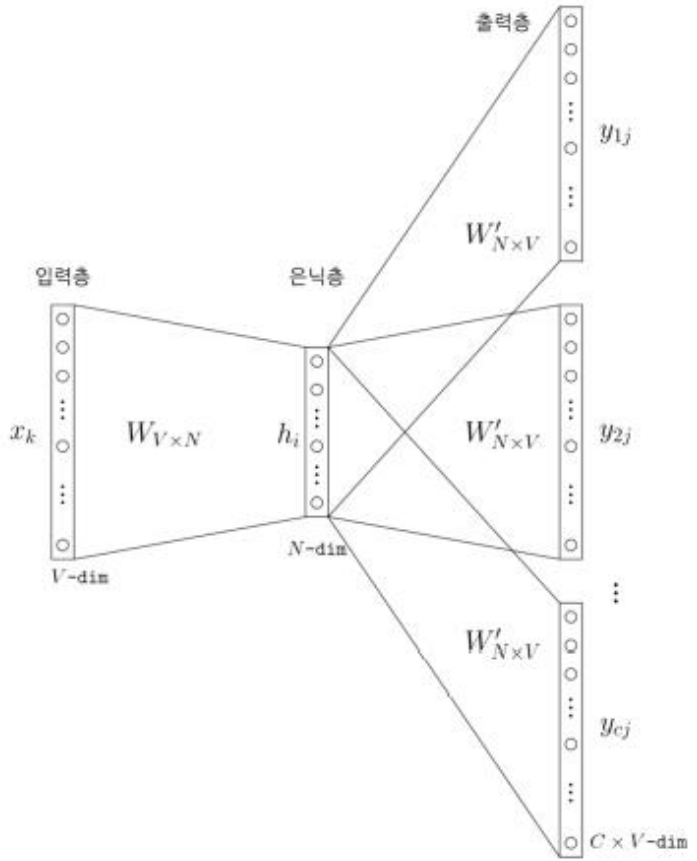


출력으로
Center Word의
One-hot Vector

Skip Gram

입력으로
Target Word의
One-hot Vector

출력으로
Neighbot Words의
One-hot Vector



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

중심단어(c)가 주어졌을 때
주변단어(o)가 나타날 확률을
최대화하는 방향으로 학습

Home

PUBLIC

Stack Overflow

Tags

Users

FIND A JOB

Jobs

Companies

TEAMS What's this?

Free 30 Day Trial

93

Here is my oversimplified and rather naive understanding of the difference:

As we know, **CBOW** is learning to predict the word by the context. Or maximize the probability of the target word by looking at the context. And this happens to be a problem for rare words. For example, given the context `yesterday was a really [...] day` CBOW model will tell you that most probably the word is `beautiful` or `nice`. Words like `delightful` will get much less attention of the model, because it is designed to predict the most probable word. This word will be smoothed over a lot of examples with more frequent words.

On the other hand, the **skip-gram** model is designed to predict the context. Given the word `delightful` it must understand it and tell us that there is a huge probability that the context is `yesterday was really [...] day`, or some other relevant context. With **skip-gram** the word `delightful` will not try to compete with the word `beautiful` but instead, `delightful+context` pairs will be treated as new observations.

UPDATE

Thanks to @0xF for sharing this article

Optimization 하기 전에 잠깐 질문 타임!

Optimization?

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

목적함수를 최소화 하는 파라미터를 찾아내는 것

여기서 파라미터는? 각 단어를 나타내는 두 개의 vector들, u와 v!

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

θ (출처: CS224n)

- 모델의 파라미터인 $\boldsymbol{\theta}$를 Optimize해야 함!

- $\boldsymbol{\theta}$: V개의 word에 대한 d차원의 Word Vector들을 하나의 긴 Vector로 나타낸 것 → 2dV차원!

- 하나의 W에 대해 v 벡터와 u 벡터가 각각 있어야 하기 때문에 각 벡터들은 Random Value로 시작한다.

Optimization

파라미터들 전부를 하나의 vector로 나타내게 되면,
각 vector가 d-dimension이고 vocabulary에 총 V개의 단어가 존재할때,
parameter vector 전체의 dimension은 $2dV$이다.

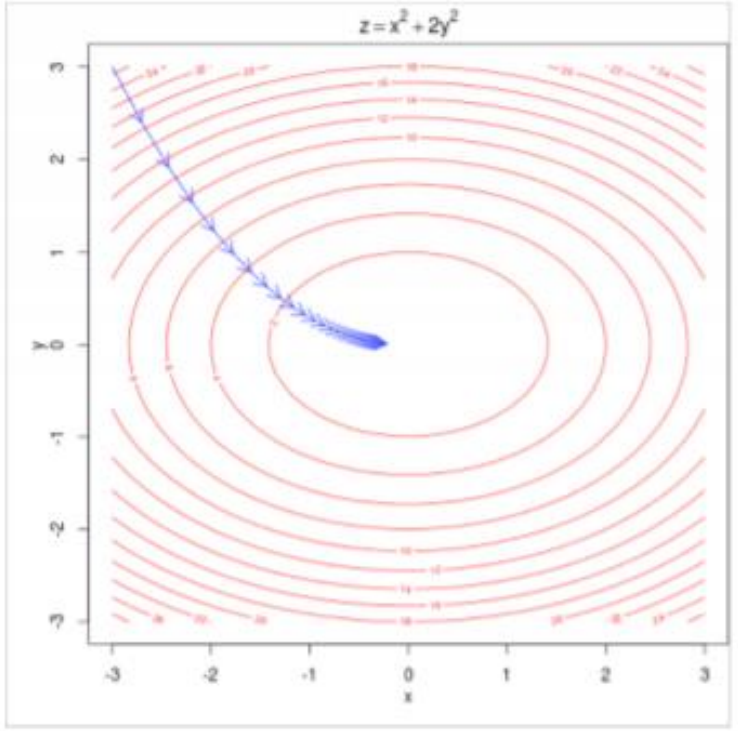
Optimization 방법으로는 우리가 흔히 아는 mini batch Gradient Descent, Stochastic Gradient Descent 등을 적용할 수 있다.

=> lec1에서는 가장 기본적인 Gradient Descent 방법을 알아봅니다

기본적인 목표:

확률(Likelihood)을 최대로 만드는 것!

$\boldsymbol{\theta}$를 조정하면서 J($\boldsymbol{\theta}$)를 minimize하자!



배경이 되는 기초 지식들 (정규세션 optimization 수업 참고)

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

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

$y = f(u), u = g(x),$

$$\{f(g(x))\}' = \frac{d}{dx} f(g(x))$$

$$= \frac{d}{du} f(u) \frac{d}{dx} g(x) \quad \left(= \frac{dy}{du}\frac{du}{dx}\right)$$

$$= f'(u)g'(x)$$

$$= f'(g(x))g'(x)$$

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

$$\log p(o|c) = \log \frac{\exp({u_o}^T v_c)}{\sum_{w=1}^{V}\exp({u_w}^T v_c)}$$ 을 center word 와 주변 단어로 각각 미분한다.

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

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

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

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

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

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

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)

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

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

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

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

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

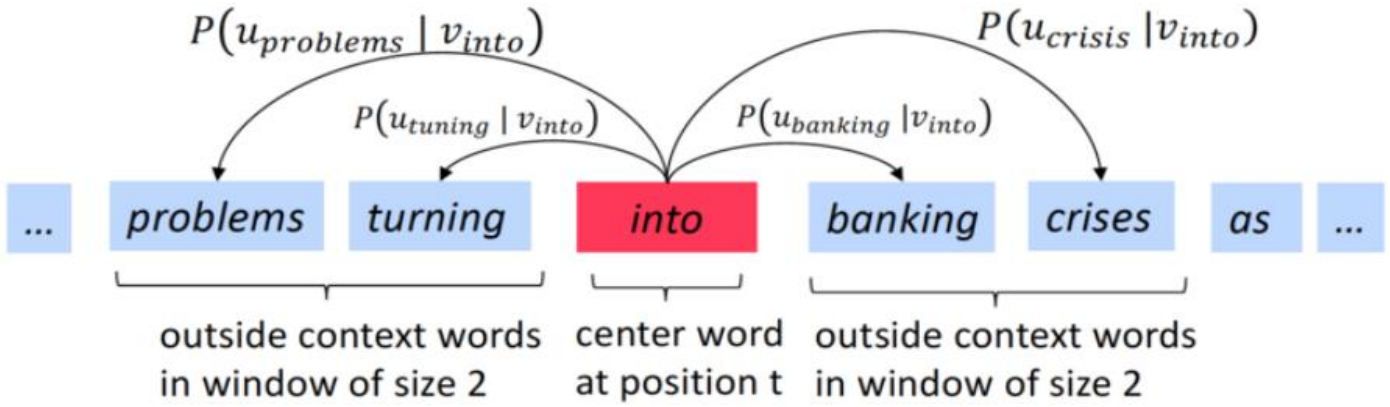
$$= u_o - \sum_{x=1}^{v} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{v} \exp(u_w^T v_c)} \cdot u_x$$

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

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

- $\mathbf{u_o}$: 관측된 context word의 모습
- $\mathbf{\Sigma P(x|c) \cdot u_x}$: model이 생각하는 context의 모습 (Expected Context Word)

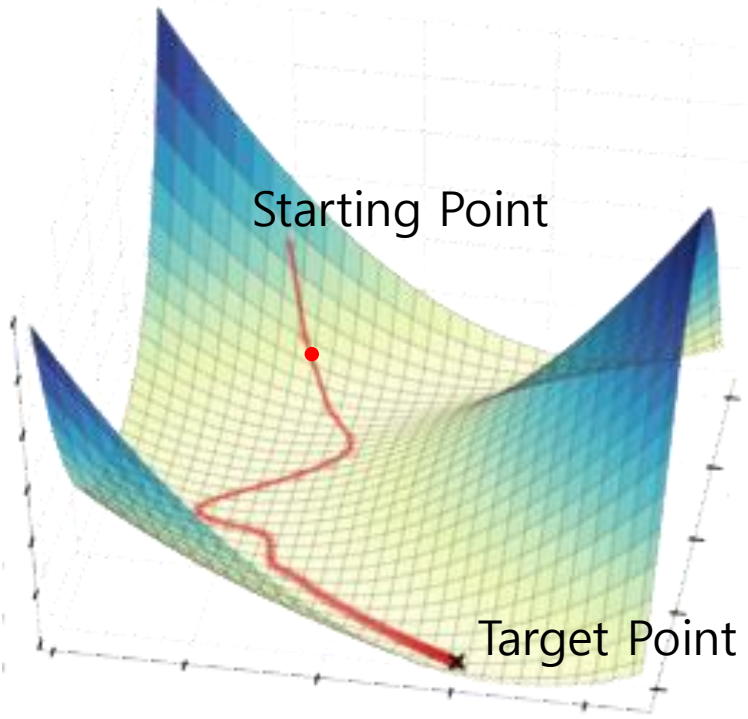
이 식이 나타내는 것은, 목적함수를 V_c에 대해 편미분한 결과가
실제 단어와 word vector 를 통해 예측한 주변 단어 (context word)의 차이를 의미한다는 것이다.
=> 즉, 우리는 slope을 통해 예측과 실제 값의 차이를 줄여 나갈 수 있다는 것을 재확인할 수 있다.

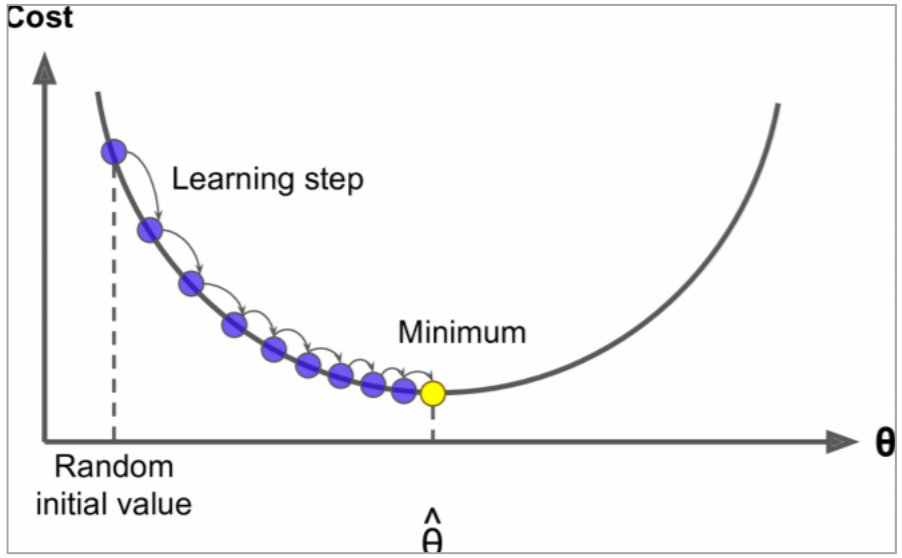


이와 같은 방식을 반복하여 gradients를 계산한다.

Gradient Descent

목적함수 J를 최소화 하는 알고리즘 중 하나로
경사가 가장 가파른 방향으로 이동을 하여 최소점을 찾는 것을
Gradient Descent, 경사하강법이라고 한다.





Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

α = *step size* or *learning rate*

Update equation (for single parameter):

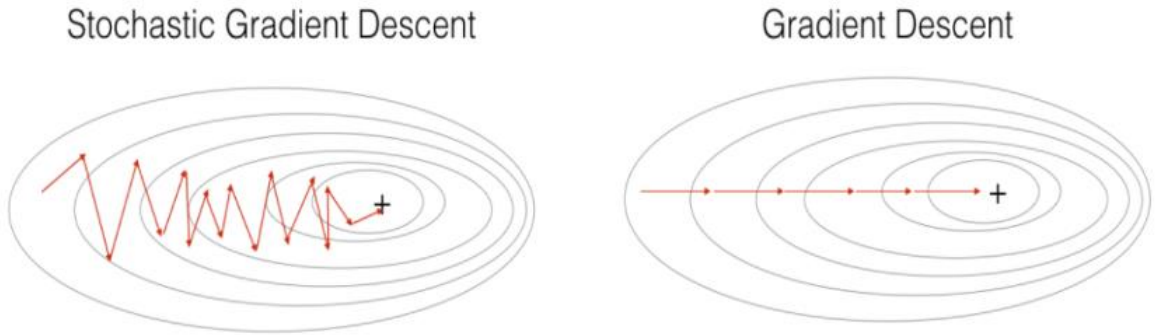
$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

+) Stochastic Gradient Descent Method (SGD)

- 그러나 한 step마다 모든 데이터들에 대한 Gradient를 계산하는 것은 연산량 多
- 따라서 한 번에 한 개만 Random Sampling을 통해서 추출, 해당 포인트에 대한 Gradient를 계산하자! → SGD
- 장점:

1) 실제 Gradient 계산하는 것보다 연산 시간 단축.
2) Shooting이 일어나기 때문에 Local Minima에 빠질 Risk가 적다.

- 단점: Global Optimum을 찾지 못할 가능성 O.



Reference

CS224n 2019 Winter Lecture1: Introduction and Word Vector

투빅스 12기 이유진 님 Logistic Regression 강의

투빅스 13기 이지용님 Optimization 강의

투빅스 13기 정주원님 NLP 강의

https://m.blog.naver.com/wideeyed/221021710286

https://m.blog.naver.com/cbyungjub/221770624548

https://pakalguksu.github.io/2020/02/27/CS224n-1%EA%B0%95-Introduction-and-Word-Vectors/

들어주셔서 감사합니다.