Introduction and Word Vectors

💡 word meaning can be represented rather well by a large vector of real numbers

Index

Human Language and Word meaning

- Human Lagnguage is social system.
- 인간의 언어는 위대하다는 내용
- 다른 종족들에 비해 인간이 지배적일 수 있는 이유가 우리는 language를 사용하기 때문
  ⇒ 시간과 공간을 넘나들며 knowledge를 보존하고 이어갈 수 있다.

⇒ AI나 컴퓨터는 인간의 언어를 어떻게 이해할 수 있을까?

- Through virtuos cycle !

GPT-3 : A first step on the path to universal models

- huge new model released by open AI
- predict one word at a time, following words to complete text

- powerful facility
- couple of exmaples (to question, sql translation ..)
- knows a lot about meaning of human language / sql ⇒ manipulate

Representing meaning of the word

> 📌 meaning : pairing between a signifier (symbol) <=> signified (idea or thing) = denotational semantics

Usable meaning in a computer

- use of resources like dictionary and thesaurus (유의어)

WordNet

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')]
```

- popular NLP solution
- organize words and terms with synonyms and hypernyms (상위어)
- human labor 를 통해서 구성된 hand - built resources

- 단어에 대한 어느정도의 의미를 제공

- deficiency
  1. lacks of nuance

     eg. “proficient” vs “good”

     ⇒ It is synonym but different depend on context
  2. missing new meaning of words, slang
  3. can’t compute accurate word similarity, relation

     eg. “fantastic” vs “great” : 동의어는 아니지만 비슷하다.

  ⇒ Similarity would be useful to make progress on, deep learning models excel

One-hot Vector

- traditional NLP (2012년 이전) : Regard words as discrete symbols
- statistical machine learning method

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]

- vector dimension = number of words in vocab (might be huge at least 500,000)
- two vectors are orthogonal ⇒ no recognition of similarity

- solution : use WordNet synonym lists for similarity but fail badly (incompleteness , etc)

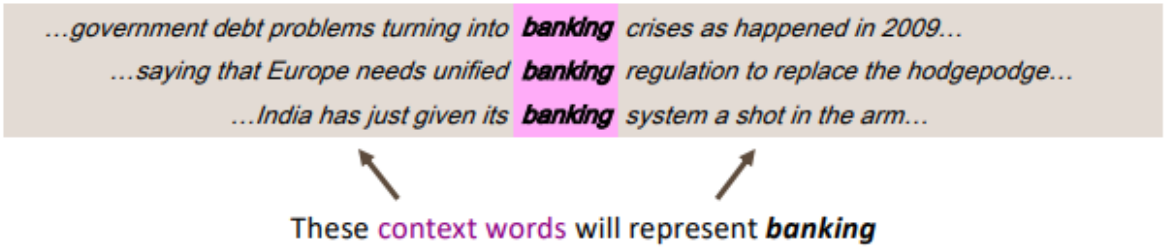
  ⇒ learn to encode similarity in the vectors themselves

Word Vectors

- Representing words by their context
- word embedding or (neural) word representations

- Idea : Distributional semantics

  ⇒ *단어의 의미는 문맥상 주변의 단어들에 의해 형성된다.*



> token & types

> (types 이해안되는 설명..)

> I'm then treating banking as a type which refers to the uses and meaning the word banking has across instances.

> (구글 번역)

> 뱅킹이라는 단어가 인스턴스 전반에 걸쳐 갖는 용도와 의미를 나타내는 types 로 뱅킹을 취급하고 있습니다.

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

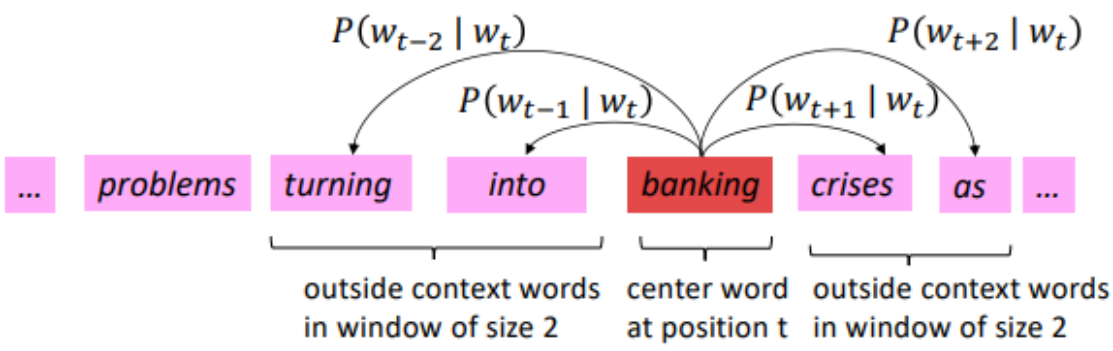
- build dense vector for each word (모든 숫자가 0이 아닌 숫자로 구성)
- this vector will be useful for predicting other words that occur in the context.
- commonly 300 dimension

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

- 단어들의 유사도 시각화
- N 차원의 word vector를 우리가 보기 쉽게 2차원으로 projection 하여 시각화 (일부 특성이 사라질 수 있음)

Word2vec (in 2013)

- framework for learning word vectors
- 충분한 양의 corpus를 바탕으로, Random vector로 시작하여 각 단어를 잘 표현하는 vector 값을 찾는다.
- 단어 벡터간의 유사도를 이용해 맥락에서 특정 단어가 나타날 확률이 maximize 되도록
- Word2Vec is a a bag of words NLP model which doesn't acutally pay any attention to word order or position.



- $p(W_{t+n}|W_t)$: 가운데 단어 W_t가 주어진 경우, 주변 단어 W_{t+n} 이 주어질 확률
- 위 확률을 최대화 하는 vector를 찾고 corpus 안의 모든 단어에 대해 진행

Objective function

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

θ is all variables to be optimized

- position t = 1, ... , T
- (fixed) window size : m
- given center word : W_j

⇒ product of each center word and context wods in window size

sometimes called a *cost* or *loss* function

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

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

📌 Minimizing objective function ↔ Maximizing predictive accuracy

How to calculate $P(W_{t+1}|W_t;\theta)$?

② Exponentiation makes anything positive

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

① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

- v_w : when w is a center word
- u_w : when w is a context word
- an example of softmax function $\mathbb{R}^n \rightarrow (0,1)^n$

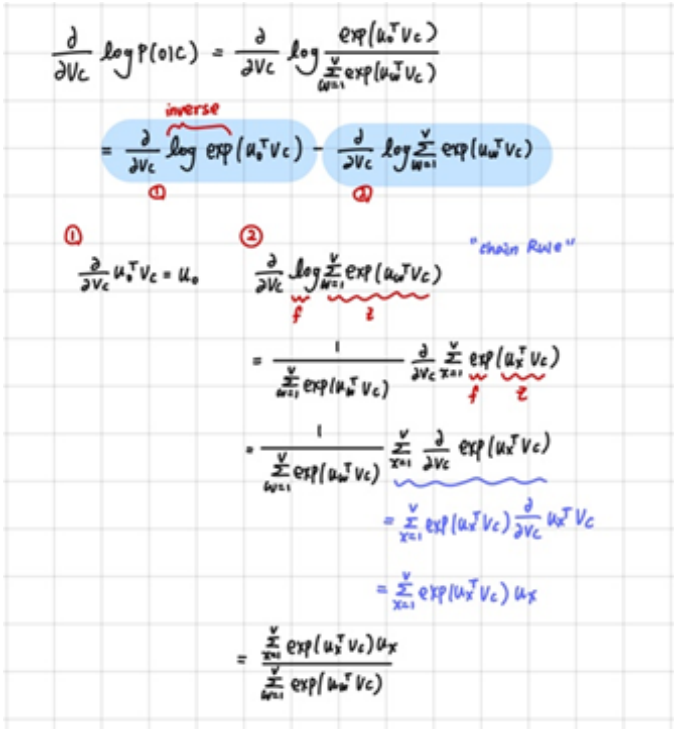
$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

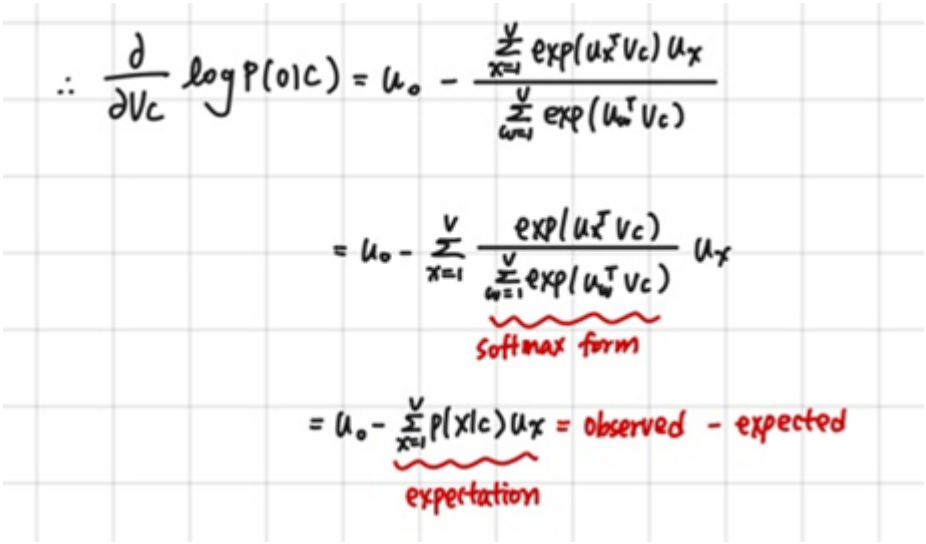
$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

θ (출처: CS224n)

- optimazation을 통해 목적함수를 최소화 하는 파라미터 θ, word를 나타내는 두 verctor u와 v를 찾는다.

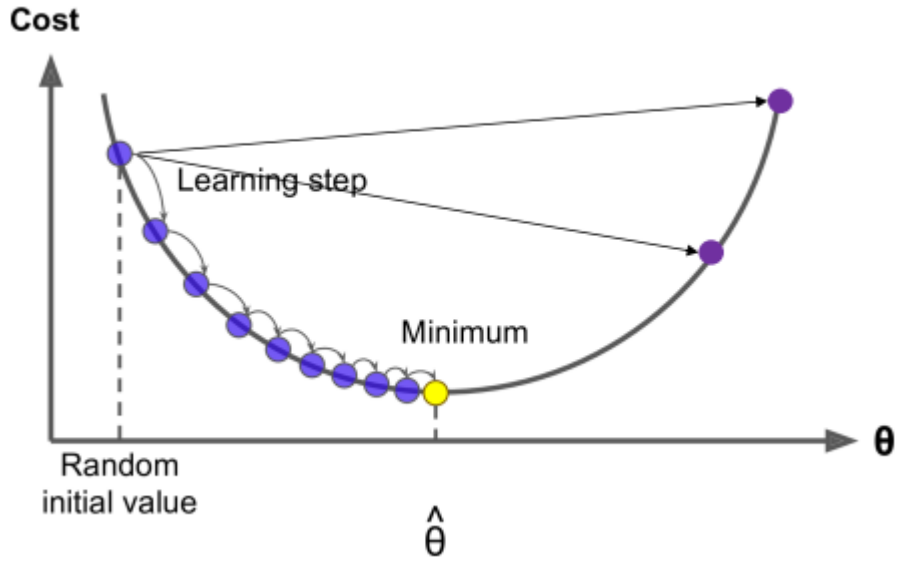
- V개의 단어가 존재하고 d-dimensional vector 일때, word vector는 u,v를 포함하므로 2dV 차원을 갖는다.



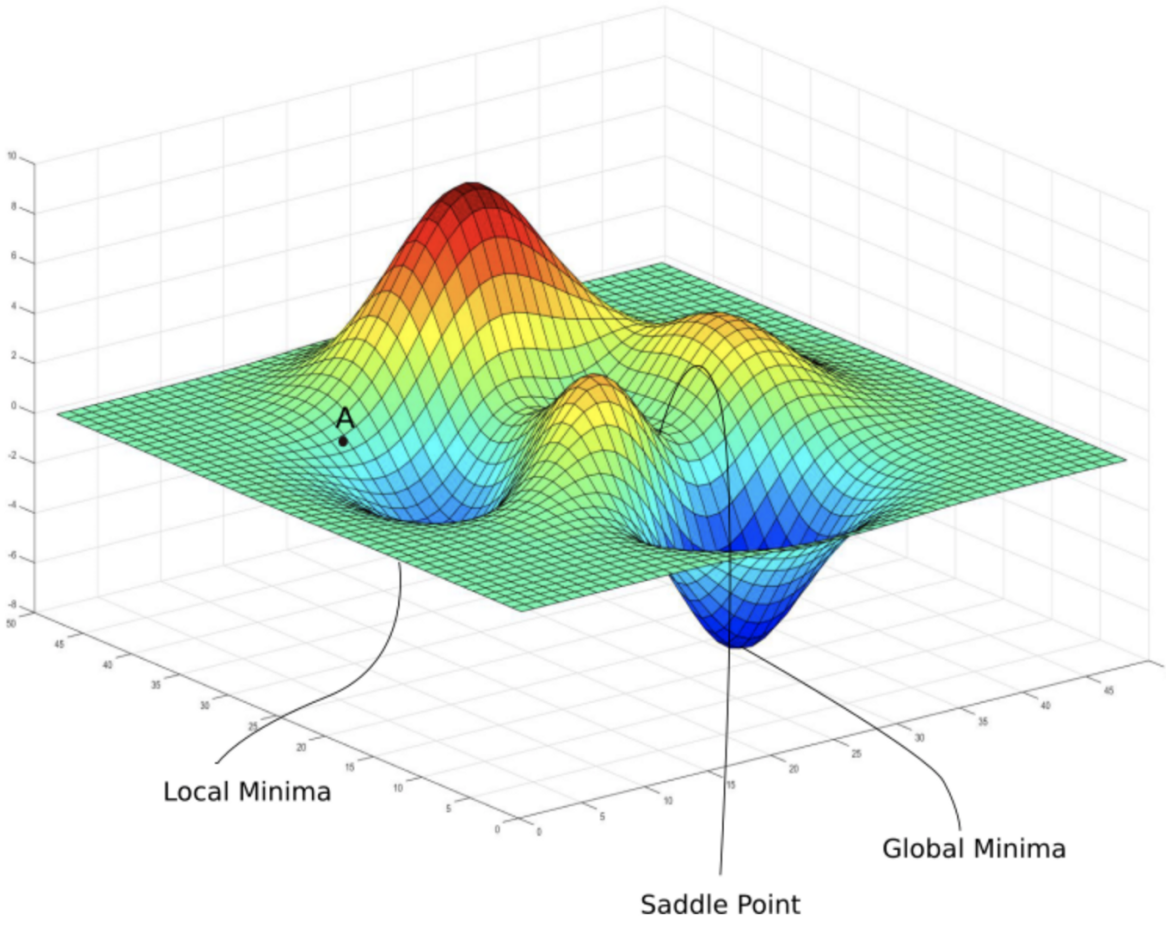


Optimization : Gradient Descent

- gradient descent algorithm - iterative algorithm



- parameter 값 (θ)을 변경해서, 기울기 반대 방향으로 이동하여 결국 최소점으로 이동할 수 있도록 한다.
- step size 가 작으면 시간이 오래걸림, 크면 수렴이상하게 될수도 ⇒ step size 선정이 중요하다



- local minimum에 빠지지 않도록 주의

- Update Equation
  1. matrix notation

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

α = *step size* or *learning rate*

  2. single parameter

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

  3. algorithm

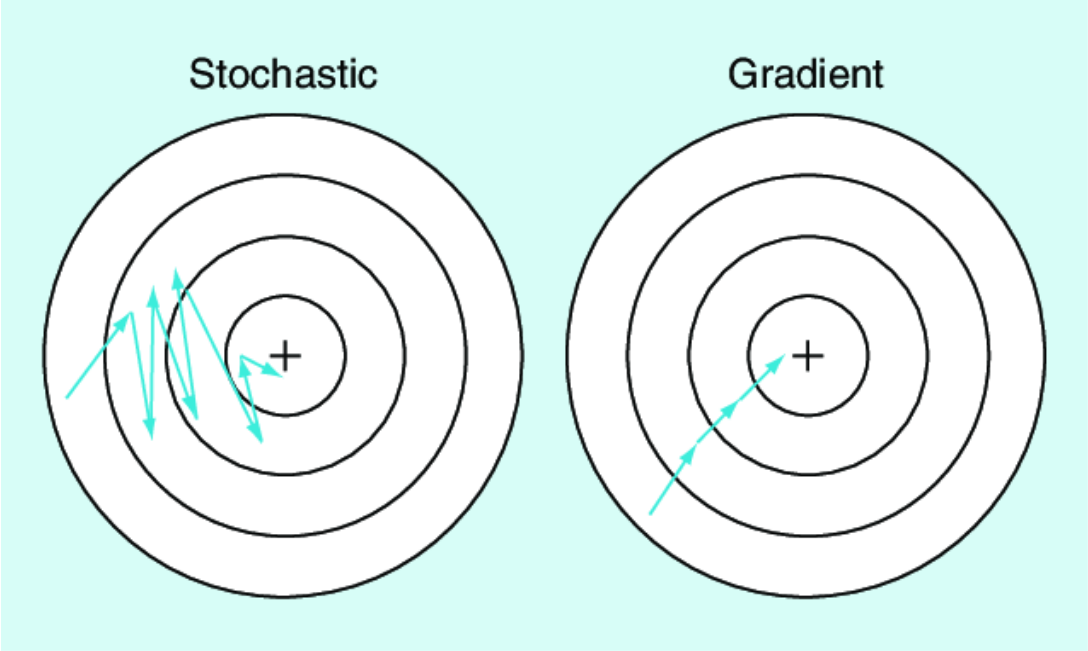
```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

⇒ 하지만 잘 사용하지 않는다

- loss function is of all windows in the corpus ⇒ very expensive to compute

Stochastic Gradient Descent (SGD)

- take one or small batch of center words

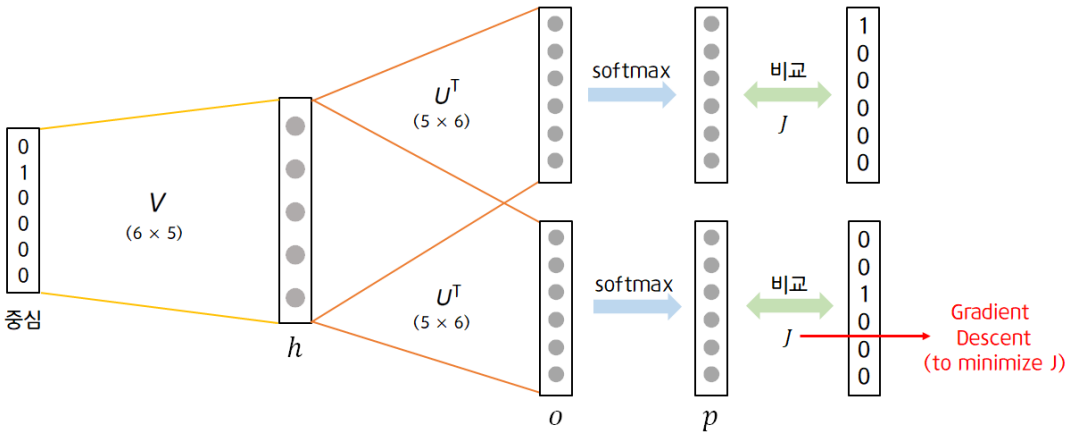



```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

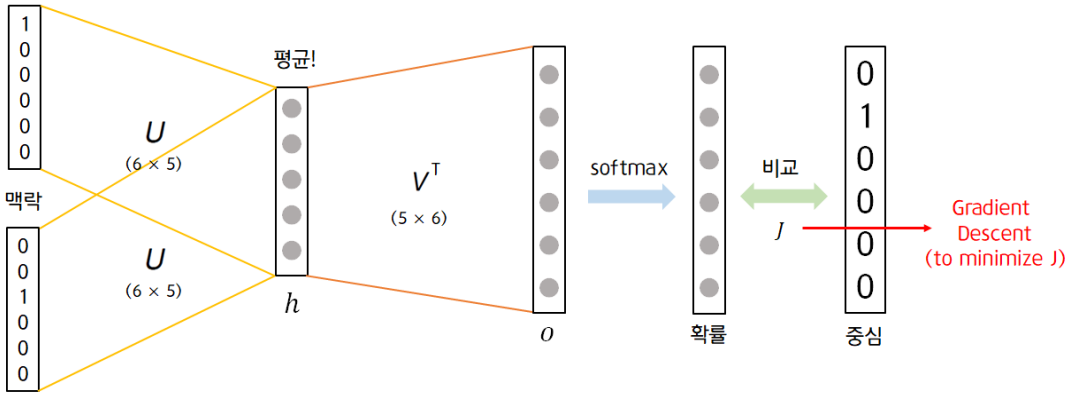
- update graident with few words (sparse vector)
- for each window, at most 2m+1 words
- sparse vector 사용으로 인한 문제점
  - 0에 해당되는 위치에서는 계산이 이루어지더라도 계속 0이기 때문에 실제로 gradient update가 되지 않고 불필요한 계산이 이루어진다.

More details

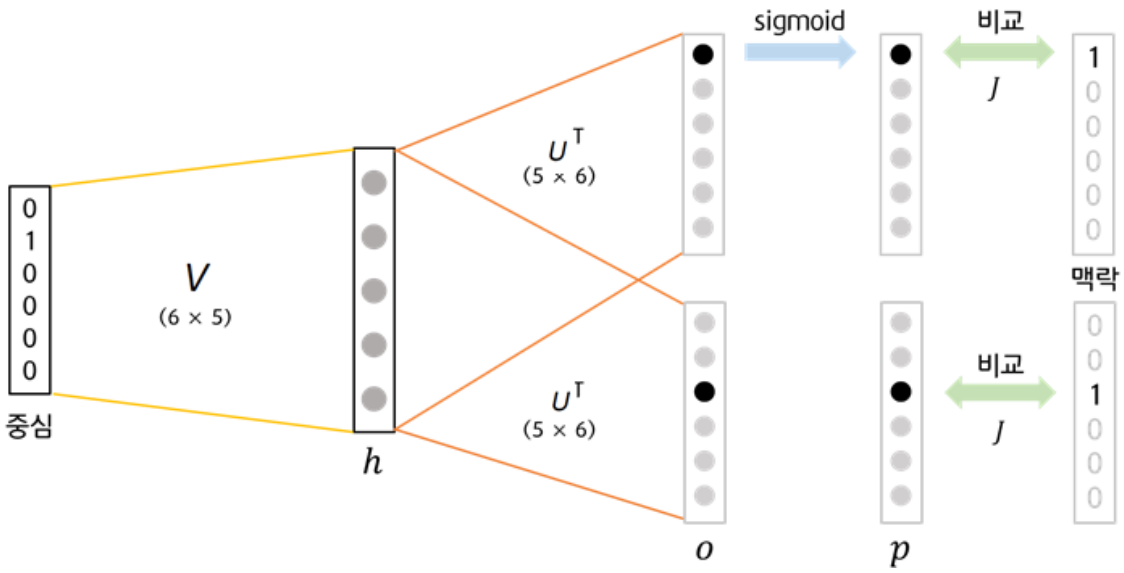
- Two model variants :
  1. Skip-grams(SG) : predict context word (outside) given center word



2. Continuous Bag of Words (CBOW) : predict center word from context word



- Additional efficiency in training: Skip-grams negative sampling (SGNS)



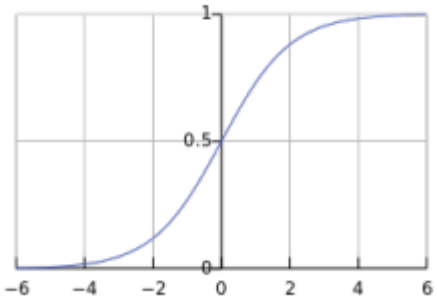
- naive softmax 문제점 : 분모의 계산량이 커짐 (dot product)
- train binary logistic regressions for true pair (center word and a word in its context window) vs several noise paris (center word paired with a random word)
- 기존의 다중분류를 이진분류로 근사시켜 모델을 효율적으로 만드는 데에 기여

- objective function :

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$

  - logistic/sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



- maximizing objective function = minimizing negative log likelihood function

$$\text{minimize} \quad J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\underbrace{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}_{\sigma(\text{양수})}) - \sum_{k=1}^{K} \log(\underbrace{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)}_{\sigma(\text{음수})})$$

- sample starts with unigram dstn of words : how often words actually occur in big corpus

$$P(W) = U(W)^{3/4}/Z$$

Co-occurrence matrix

skip-gram은 중심 단어를 기준으로 맥락 단어가 등장할 확률을 계산하는 것 ⇒ window size를 아무리 증가해도 global co-occurrence는 알 수 없다.

- count-based 의 Co-occurrence matrix :

  1. Window based co-occurrence matrix (단어-문맥 행렬)

     eg) I like deep learning , I like NLP , I enjoy flying

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

     - word2vec과 비슷한 방식으로 한 문장을 기준으로 윈도우에 각 단어가 몇 번 등장하는지 세어 구성한다.
     - symmetric matrix ⇒ row(=column)가 word representation
     - syntactic & semantic 정보를 얻을 수 있다.

  2. Word-Document matrix (단어-문서 행렬)

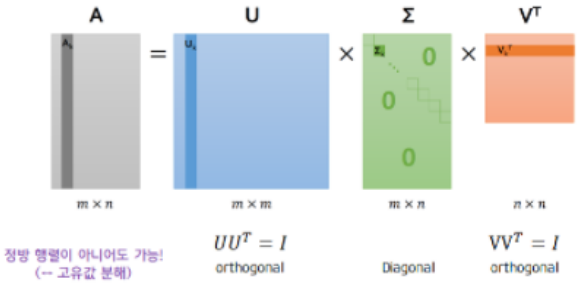
-	doc1	doc2	doc3
나	1	0	0
는	1	1	2
학교	1	1	0
에	1	1	0
가	1	1	0
ㄴ	1	0	0
다	1	0	1
영희	0	1	1
좋	0	0	1

- 한 문서를 기준으로 각 단어가 몇 번 등장하는지 세어 구성
- 문서에 있는 많은 단어들 중 빈번하게 등장하는 특정 단어가 존재한다는 것을 전제
- LSA (Latent Semantic Analysis ; 잠재적 의미 분석)을 가능하게 하는 기법 (eg. 문서 간 유사도 측정 등)

- count-based matrix는 단어의 개수가 증가할수록 차원이 폭발적으로 증가, sparsity issues ⇒ dimension reduction

3. SVD (Singular Value Decomposition; 특이값 분해)

$$A = U\Sigma V^T$$



$\Sigma_k = \sqrt{\lambda_k}$
λ_k : AA^T, A^TA의 kth 고유값 (eigen value)

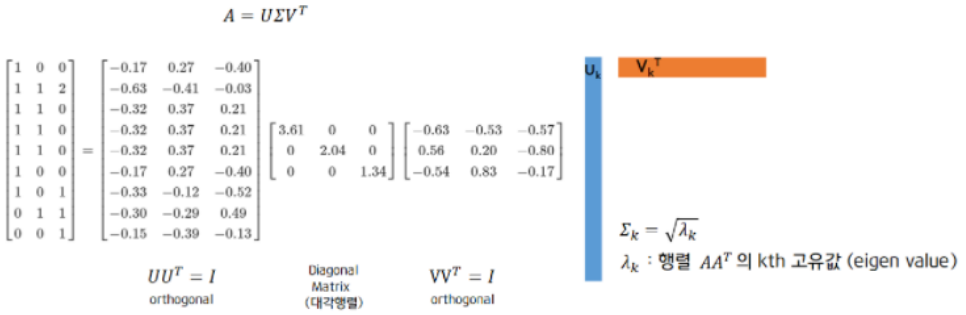
- 특이값 분해를 이용한 행렬 분해
- 고유값 분해와 달리 분해할 행렬이 정방 행렬이 아니여도 가능하다.

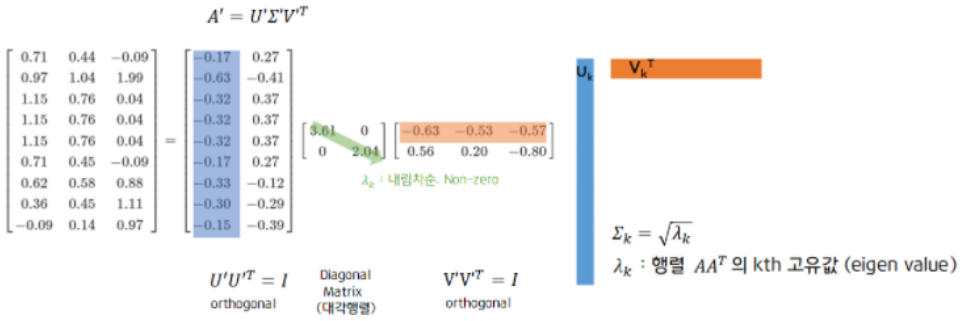
▼ 예제

열 : 문서

-	doc1	doc2	doc3
나	1	0	0
는	1	1	2
학교	1	1	0
에	1	1	0
가	1	1	0
ㄴ	1	0	0
다	1	0	1
영희	0	1	1
좋	0	0	1

행 : 단어





- 특이값 분해를 적용한 후, UU^T에서 상위 2개의 열벡터만을 선택하여 새로운 $U^{'}U^{'T}$를 만들어내고 $A^{'}$를 만들어낸 것을 truncated SVD라고 한다.

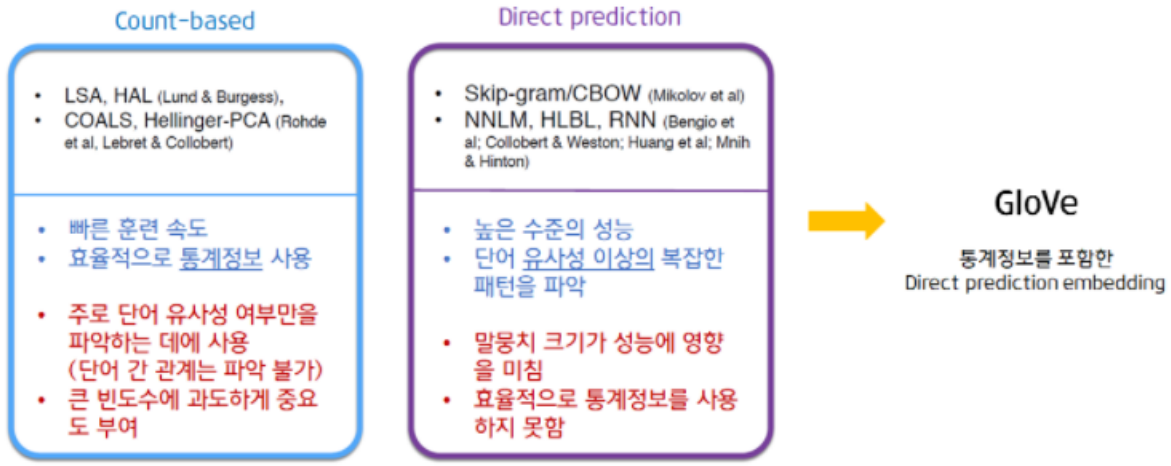
열 : 문서

$$X_1{}' = U'^T A' = U'^T U' \Sigma' V'^T \begin{bmatrix} -2.28 & -1.90 & -2.07 \\ 1.14 & 0.42 & -1.64 \end{bmatrix}$$

$$X_2{}' = A'V' = U'\Sigma' V'^T V' = \begin{bmatrix} -0.63 & 0.56 \\ -2.30 & -0.84 \\ -1.16 & 0.76 \\ -1.16 & 0.76 \\ -1.16 & 0.76 \\ -0.63 & 0.56 \\ -1.20 & -0.24 \\ -1.10 & -0.60 \\ -0.57 & -0.80 \end{bmatrix}$$

행 : 단어

GloVe : Count based vs direct prediction



- Glove의 기본 아이디어 :
  - 임베딩된 단어벡터 간 **유사도** 측정을 수월하게 (word2vec 장점)
  - 말뭉치 전체의 **통계 정보** 반영하 (co-occurrence matrix의 장점)

- 임베딩된 **두 단어벡터의 내적**이 말뭉치 전체에서의 **동시 등장확률 로그값**이 되도록 목적함수 정의

X_{ik} : 전체 말뭉치 중 사용자가 정한 window 내에, i번째 단어와 k번째 단어가 동시에 등장하는 횟수

$X_i = \Sigma_k X_{ik}$; 전체 말뭉치 중 사용자가 정한 window 내에, i번째 단어가 등장하는 횟수

$P_{ik} = p(k|i) = X_{ik}/X_i$; i 번째 단어 (context word) 주변에 k번째 단어가 등장할 조건부 확률

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
	∨	∧	∨	∧
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

$$\frac{P_{ice,solid}}{P_{steam,solid}} = \frac{P(solid|ice)}{P(solid|steam)} = \frac{1.9 \times 10^{-4}}{2.2 \times 10^{-5}} = 8.9$$

- 목적함수 도출하기

Crucial Insight : 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의

F의 input / F의 output

$$F(w_{ice}, w_{steam}, w_{solid}) = \frac{P_{ice,solid}}{P_{steam,solid}} = \frac{P(solid|ice)}{P(solid|steam)} = \frac{1.9 \times 10^{-4}}{2.2 \times 10^{-5}} = 8.9$$

8.9라는 결과를 만들어주는 함수 F를 찾아야 한다!

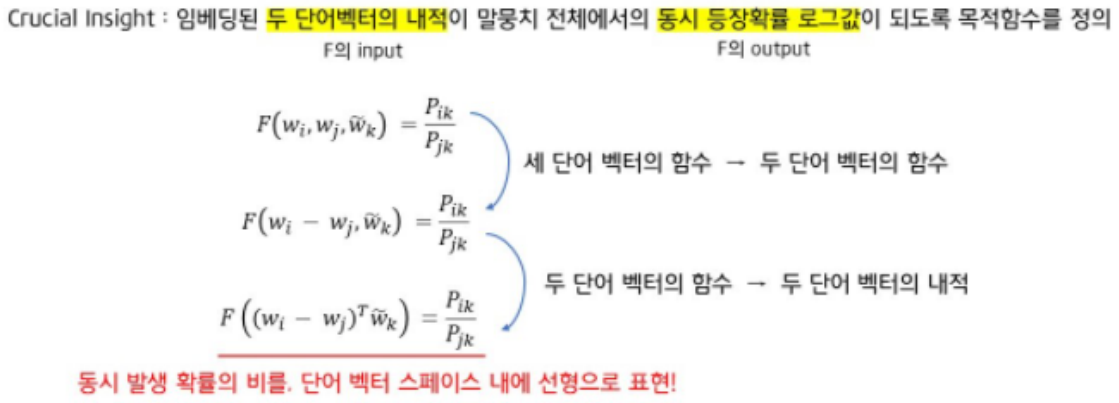
$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

i번째 단어 (맥락 단어)가 있을 때 k번째 단어가 등장할 조건부 확률

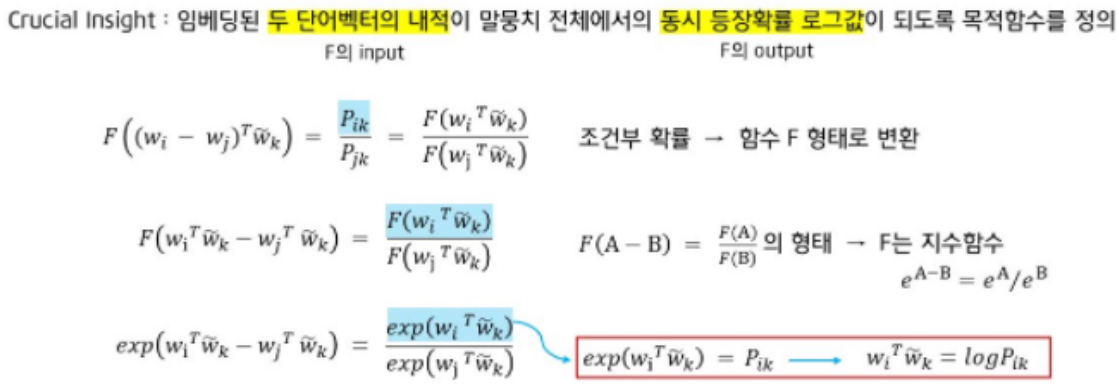
j번째 단어 (맥락 단어)가 있을 때 k번째 단어가 등장할 조건부 확률

두 단어벡터의 내적이 input으로 들어가게 만들어야 한다!

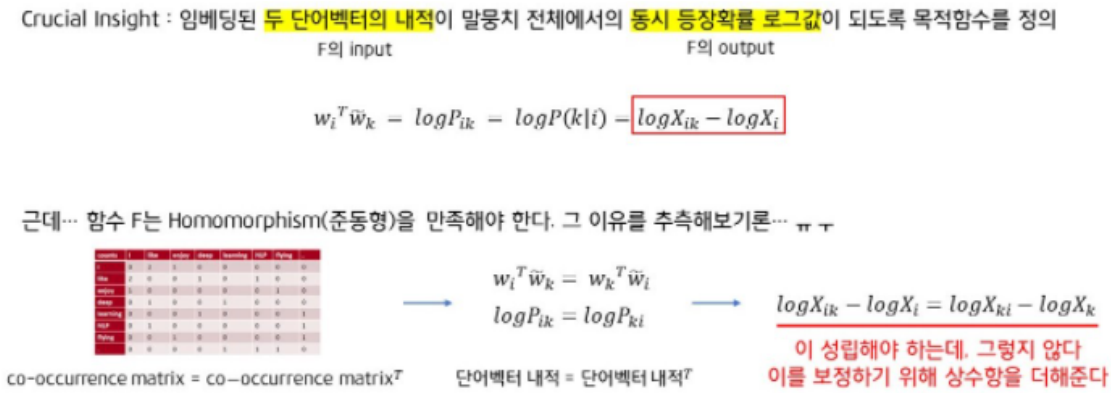
- F라는 함수의 결과값 = 동시 발생 확률의 비



- 두 context vector의 차와 center vector 의 내적값으로 변환하여 단어 벡터 스페이스 내에 선형으로 표현
- input은 두 단어벡터의 내적



- 동시 발생 확률을 F함수에 대한 식으로 다시 정의했을 때 F의 조건식을 도출할 수 있고, 이 조건에 맞는 함수 F는 exp 지수함수



- 좌변을 두 단어벡터의 내적값으로 두면 두변은 $logX_{ik} - logX_i$가 된다.
- input이였던 co-occurrence matirx가 준동형($A = A^T$)을 만족하기 때문에 함수 F의 결과 또한 준동형을 만족해야하는데, 실제 계산해보면 그렇지 않아서 이를 보정하기 위해 상수 b를 더해준다.

$$w_i^T \tilde{w}_k - logX_{ik} \boxed{- logX_i} - logX_{ik} \boxed{- b_i - \tilde{b}_k}$$

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k - logX_{ik}$$

동시 등장 횟수에 대한 정보를 담고 있는 co-occurrence matrix

$$\underset{w_i, \tilde{w}_k, b_i, \tilde{b}_k}{\text{argmin}} \quad J = \sum_{i,j=1}^{V} (w_i^T \tilde{w}_k + b_i + \tilde{b}_k - logX_{ik})^2$$

- Glove의 손실함수
  - 특정 단어가 지나치게 빈도수가 높아 X_{ij}가 튀는 현상을 방지하기 위해 $f(X_{ij})$함수

$$\underset{w_i, \tilde{w}_j, b_i, \tilde{b}_j}{\text{argmin}} \quad J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - logX_{ij})^2 \quad , \quad f(X_{ij}) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha} & if\ x < x_{max} \\ 1, & otherwise \end{cases}$$

특정 단어가 지나치게 빈도수가 높아서 X_{ij} 가 튀는 현상을 방지 하기 위해 추가한 함수

- 빠른 훈련
- 말뭉치 크기가 성능에 미치는 영향을 제한할 수 있음
- 작은 말뭉치, 작은 벡터에도 좋은 성능

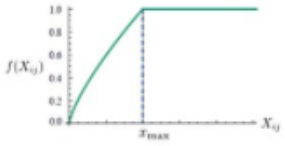
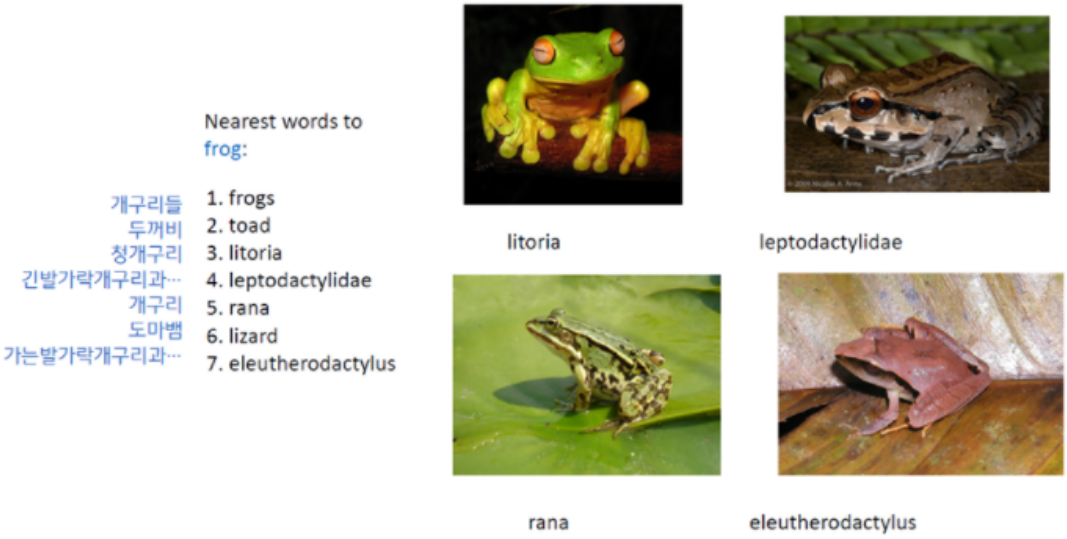


Figure 1: Weighting function f with $\alpha = 3/4$.

Results

Glove 모델을 여러 데이터에 적용한 결과

- Frog



Word embedding Evaluation

단어 임베딩 모델 평가 방식

- Extrinsic evalutaion (외적 평가)

  실제 현실 문제(real task)에 직접 적용하여 성능을 평가하는 방식

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

  GloVe는 외적평가에서 좋은 성능을 보인다.

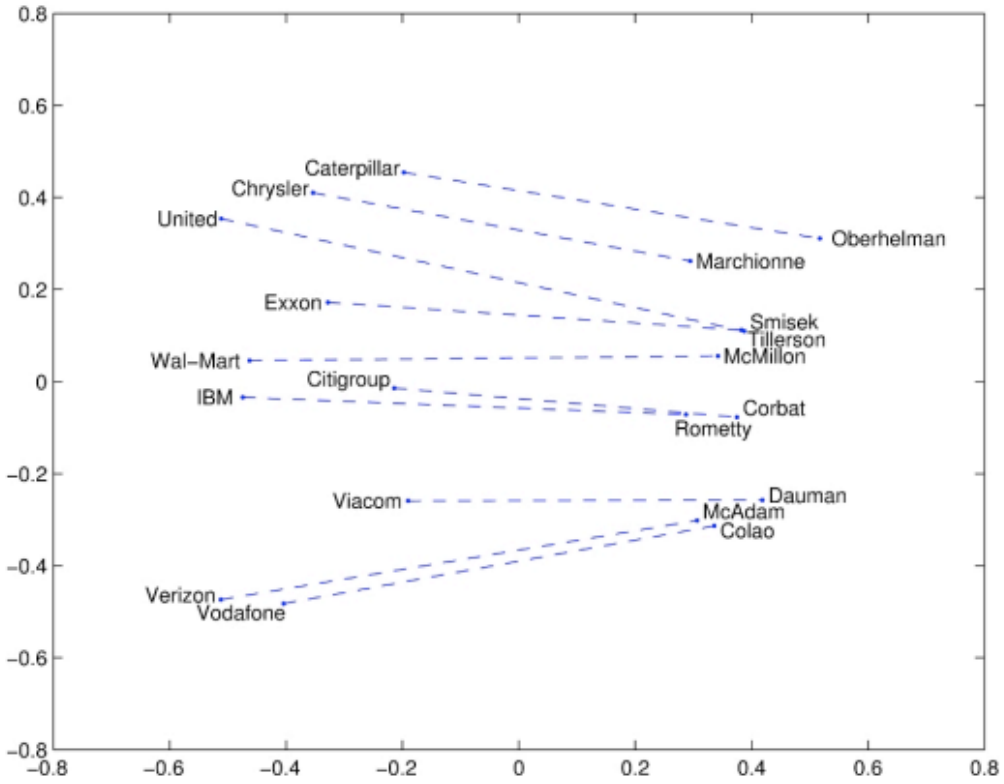
- Intrinsic evaluation

  1. word analogy

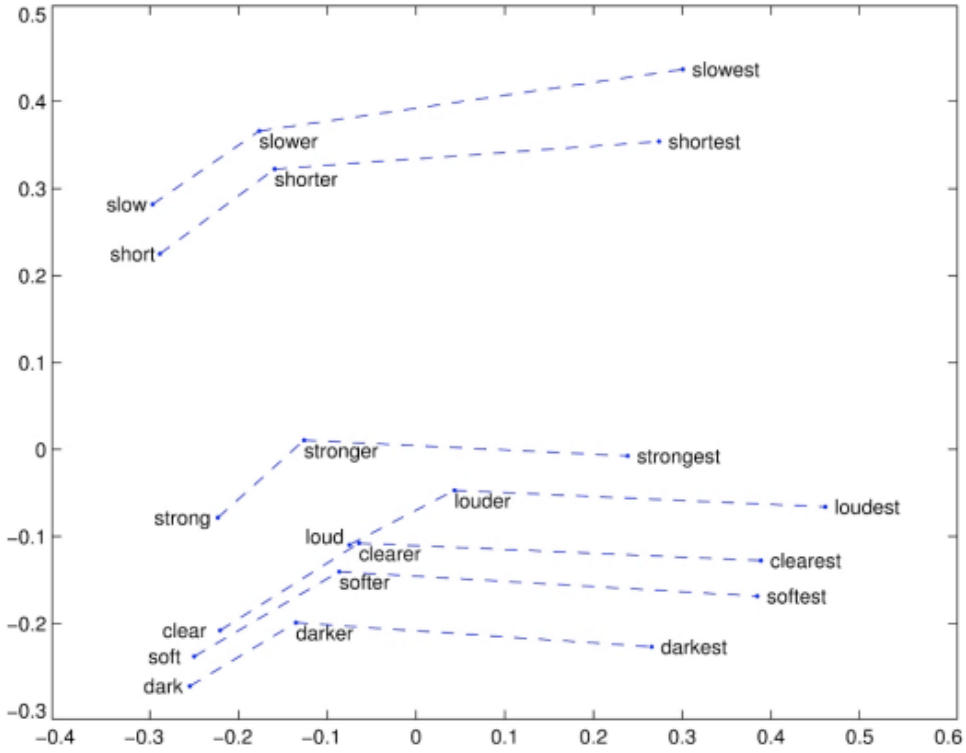
     유사도 ⇒ a:b:: c:?에서 ?에 들어갈 단어 유추

$$d = \underset{i}{argmax} \frac{(w_b - w_a + w_c)^T w_i}{\|w_b - w_a + w_c\|}$$

- 평가 예시
  - Semantic (의미적) example : company - CEO



  - Syntactic (순서적) examples : comparatives and superlatives



- several doels

Model	Dim.	Size	Sem.	Syn.	Tot.
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7

→ GloVe는 좋은 성능을 보인다.

2. Correlation

   일련의 단어 쌍을 미리 구성한 후에 사람이 평가한 점수와, 단어 벡터 간 코사인 유사도 사이의 상관관계를 계산해 단어 임베딩 품질을 평가

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

5 similarity datasets

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

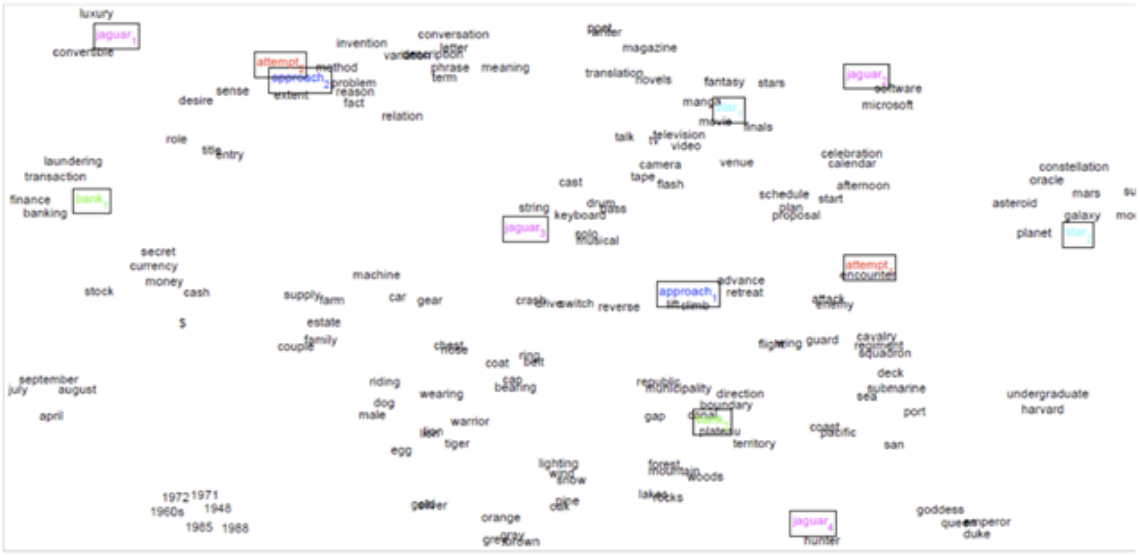
GloVe : good performance!

- 다의어를 해결하는 방법

pike

- A sharp point or staff
- A type of elongated fish
- A railroad line or system
- A type of road
- The future (coming down the pike)
- A type of body position (as in diving)
- To kill or pierce with a pike
- To make one's way (pike along)
- In Australian English, pike means to pull out from doing something: I reckon he could have climbed that cliff, but he piked!

1. **Improving Word Representations Via Global Context And Multiple Word Prototypes (Huang et al. 2012)**



특정 단어의 윈도우들을 클러스터링 한 후, 단어들을 각 클러스터링 중심으로 다시 임베딩

2. **Linear Algebraic Structure of Word Senses, with Applications to Polysemy (Arora, …, Ma, …, TACL 2018)**

$$v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}, \quad \alpha_1 = \frac{f_1}{f_1+f_2+f_3}, \text{ etc., for frequency } f$$

각 의미에 가중치를 부여하고 선형결합을 통해 새로운 단어 벡터를 생성.

이 단어벡터를 가지고 유사 단어들 끼리 clustering 했을 때 상당히 결과가 좋은것으로 보아 내적인 의미까지 잘 파악해서 분류했음을 의미한다.

각 의미에 가중치를 부여하고 선형결합을 통해 새로운 단어 벡터를 생성.

이 단어벡터를 가지고 유사 단어들 끼리 clustering 했을 때 상당히 결과가 좋은것으로 보아 내적인 의미까지 잘 파악해서 분류했음을 의미한다.