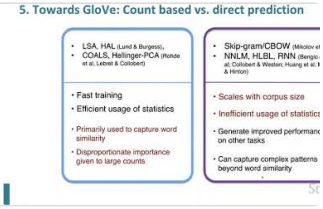
Week 2

발제자 : 유채원

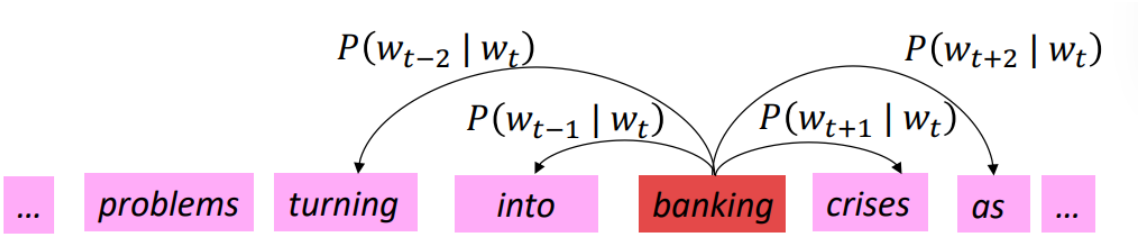
Stanford CS224N NLP with Deep Learning | Winter 2021 | Lecture 2 - Neural Classifiers

For more information about Stanford's Artificial Intelligence professional and graduate programs visit: https://stanford.io/2ZB72nu Lecture 2: Word Vectors, Word Senses, and Neural Network Classifiers 1. Course organization (2 mins) 2. Finish looking at word vectors and word2vec (13

https://www.youtube.com/watch?v=gqaHkPEZAew&list=PLuqhl4iqeAZYU3nRJQ9sgLAqSDTAsuVOX&index=2



1. Word2Vec 복습
   - 모든 주변 단어들에 대한 연산 시행



   - 모든 단어들은 두 가지 vector로 표현한다.
     - V(w) : vector when w is center 중심 단어일 때의 벡터
     - U(w) : vector when w is context 주변 단어일 때의 벡터

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

     - U(o) 와 V(c) 내적 : 유사도를 계산.

$$(u_o^T v_c)$$

       - larger dot product = larger probability
       - 내적이 클수록 확률값도 높다고 할 수 있음
     - 정규화시켜서 0~1의 값으로 표현
   - 이러한 방법은 모든 단어들에 대해 내적을 해야 한다는 것이 단점

     → skip-gram model with negative sampling을 이용한다!

2. SGNS (Skip-gram model with negative sampling)

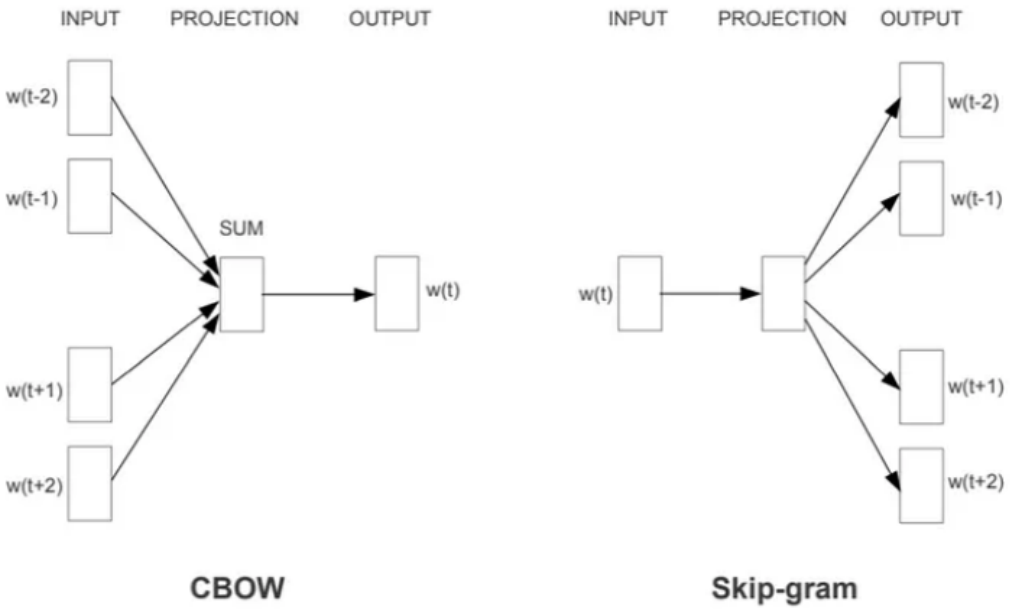
a. 중심 단어와 실제 주변 단어 vs 중심 단어와 아무 단어(noise pairs)를 비교

b. Skip-Gram vs CBOW

   i. Skip-gram : 중심 단어로 주변 단어를 예측하기

   ii. CBOW : 주변 단어로 중심 단어를 예측하기

   - 보통 Skip-Gram의 task가 더 어렵기 때문에 더 성능이 좋음



Source: Exploiting Similarities among Languages for Machine Translation paper.

c. 일반 word2vec은 naive softmax를 이용했음: 전체 단어에 대한 확률을 분모로!

→ 이제는 negative sampling을 이용할 것임

SGNS의 목적 함수

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

우리의 목적은?

: 실제 주변 단어와의 확률은 높이고 Noise pair와의 연관성은 줄이기!

$\left(u_o^T v_c\right)$

: 실제 주변 단어와의 내적 곱

: 최대화 시키는 것이 목표

$\left(-u_j^T v_c\right)$

: 무작위의 단어가 주변 단어로 왔을 때의 내적

: 최소화 시키는 것이 목표

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

: 더 안정적인 결과값을 위하여 합산해준다. (noise pair 하나만 한다면 우연히 값이 높게 나올 수 있음, but 99%의 noise pair은 확률적으로 값이 작을 것!)

여기서 식을 조금만 더 살펴보자면,

P(w)은 다음과 같이 정의한다.

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

U(w) : 단어가 나올 확률에 3/4제곱을 한다. 단어가 많이 나오면 확률을 높이되 또 너무 차이가 나지는 않게 하기 위하여.

d. Co-occurence matrix (window vs full document)

- window : window size = hyper-parameter.
  1. window를 움겨가며 co-occurence matrix를 채운다
  2. ex) window-size = 1 이면, 'I like' → I,like 교차지점에 1 채우기

Example corpus:

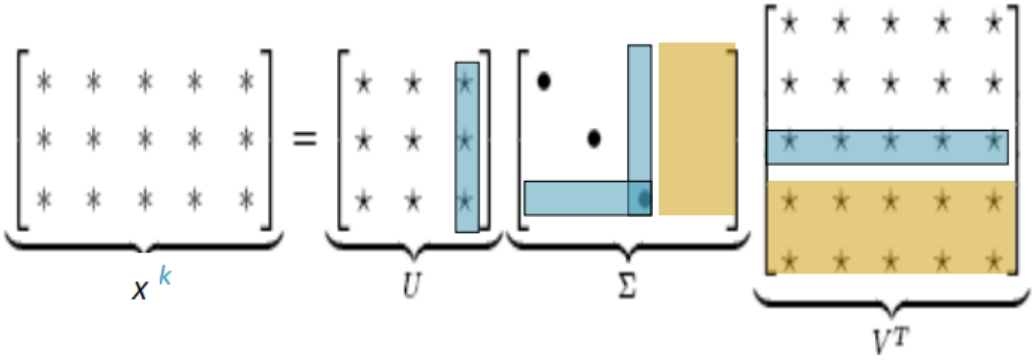
- I like deep learning
- I like NLP
- I enjoy flying

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

근데 이렇게 하면 너무 sparse & 차원이 너무 크다.

차원을 줄이는 방법은? → SVD 이용하기!

- SVD



특이값 분해를 하면 파란색, 노란색으로 칠한 부분은 무시해도 된다. → 차원 축소

근데 그냥 co-occurence matrix에 바로 사용해버리면 성능이 좋지 않음.

(that,he,has) 등의 의미 없는 단어들이 너무 자주 나오는 문제가 생김

- log the frequencies
- min(X, t), with t ≈ 100
- Ignore the function words

다음 중 하나를 시행하고 SVD를 쓰자!

3. GloVe
   a. 기존의 방법론들의 단점들을 개선
      i. LSA: 각 문서에서의 각 단어의 빈도수를 카운트 한 행렬이라는 전체적인 통계 정보를 입력으로 받아 차원을 축소(Truncated SVD)하여 잠재된 의미를 끌어내는 방법론
         1. 전체 코퍼스의 특징은 잘 반영을 하나,
         2. 남자:왕 = 여자:? 등과 같은 analogy는 잘 구하지 못하는 단점이 있음
      ii. Word2Vec : 실제값과 예측값에 대한 오차를 손실 함수를 통해 줄여나가며 학습하는 예측 기반의 방법론
         1. 단어 간 유추 작업은 잘하나,
         2. window size만 고려하기에 전체 코퍼스의 특징은 잘 반영하지 못함

      2가지 단점들을 모두 개선한 것이 GloVe!

   b. P(k|i) = i와 k가 동시에 등장한 횟수 / 특정 단어 i 의 등장 횟수
      - co-occurence matrix에서 k와i의 교차 지점 / i 의 행 전체의 합

- **GloVe 의 아이디어**

  : **임베딩 된 중심 단어와 주변 단어 벡터의 내적이 전체 코퍼스에서의 동시 등장 확률이 되도록 만드는 것 - 아래의 좌변과 우변이 최대한 가까워지도록**

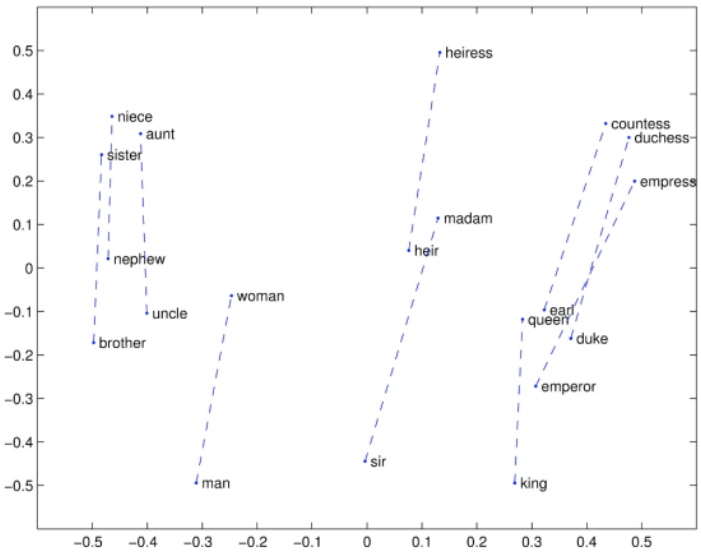
$$w_i \cdot w_j = \log P(i|j)$$

그렇다면 손실함수는 좌변과 우변의 차이를 최소화하는 것.

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

빨간색 밑줄 친 애들이 비슷해질수록 손실함수는 작아짐.

GloVe Visualization



GloVe 이용하면 LSA에서는 어려웠던 Analogy (단어 유추) 도 수월하게 이루어짐을 볼 수 있다.