More details about Word2vec

- 2개의 vector를 쓰는 이유는 계산의 편리성을 위해서이다
- word2vec은 하나의 모델이 아니여 algorithm family를 지칭하는 용어
  - ex
    - Skip-grams (SG) : predict context words given center word (→이번 강의에서 살펴볼 것)
    - Continuous Bag of Words (CBOW) : predict center word from context words
    - ⋮
- naive softmax는 단어 수만큼 dot product를 해야해서 negative sampling을 loss function으로 많이 사용

negative sampling

: train binary logistic regressions to differentiate a true pair (center word and a word in its context window) versus several noise pairs (center word and a random word) (→정은 수로도 충분)

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$$

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

- $J_t(\theta)$: every each word for each window
- $\log \sigma$: sigmoid / logistic function
- $\sigma(x) = \frac{1}{1+e^{-x}}$
- ∵ sigmoid가 symmetric
- u_j: a random word

$$J_{neg\text{-}sample}(u_o, v_c, U) = -\log \sigma(u_o^T v_c) - \sum_{k} \log \sigma(-u_k^T v_c)$$

- $k \in$ (K sampled indices)
- K개의 samples
- $P(w) = U(w)^{3/4}/Z$ 를 이용
- $\frac{3}{4}$ 제곱은 덜 나오는 단어를 더 자주 sample 되게 만든다

Optimization basics

- goal : minimize the cost function $J(\theta)$
- Gradient Descent

현재의 θ에 대해, $J(\theta)$의 미분계수와 반대 방향으로 θ를 움직인다.

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta) \quad \text{in matrix notation}$$

- α: step size / learning rate
  - 너무 작음 : 계산을 많이 해서 시간 오래 ↑
  - 너무 큼 : 잘못된 곳으로 갈 수 있음 or minimum에 가는데 오히려 더 오래 걸릴 수 있음

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta) \quad \text{for single parameter}$$

- 초기값 : 각 dimension 마다 0에 가까운 랜덤한 수로 되어있는 word vector
- 조절 가능한 parameter : step의 크기
- all windows in the corpus를 다 계산해야 해서 시간이 오래 걸림 ↓
- Stochastic Gradient Descent (SGD)
  : Repeatedly sample windows, and update after each one → sparsity

co-occurrence matrix

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

- symmatric
- 윈쪽에서도 window length = 1
- 문제점
  - 단어 수만큼 vector 필요
  - very high dimensional → 메모리 많이 필요
  - sparsity issues
  - ⇒ 가장 중요한 정보를 적은 dimension을 가진 dense vector에 저장해 고정시키자
  - ⇒ $X = USV^T$로 바꾸고 S의 singular value에서 작은 값에 해당되는 것들을 지우자 (Dimensionality Reduction)

+) the, he 같은 function words 가 너무 많이 나와서 영향력이 너무 큼 →
- log the frequencies
- min(X, t) with t≈ 100
- Ignore the function words

The GloVe model of word vectors

- co-occurrence matrix models + neural models = neural model과 유사하지만 co-occurrence를 계산하는 model
- Log-bilinear model
  - 각각 linear한 두개의 w_i, w_j 존재
  - ↳ $w_i \cdot w_j$가 log of a probability와 관련됨

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

$$\downarrow$$

$$w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$$

↳ function word같이 너무 많이 나오는 단어의 영향력을 제한시킴

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + \underbrace{b_i + \tilde{b}_j}_{\text{bias}} - \log \underbrace{X_{ij}}_{\text{co-occurrence}})^2$$

즉, (j가 있을 때 i가 있을 확률) − (j랑 i가 같이 있는 수) = Error

Evaluating word vectors

1) Intrinsic
   - 직접 특정 영역에 대해 평가
   - 빠름
   - 시스템을 이해하는 데 도움이 됨
   - 진짜 도움이 되고 있는지 잘 알 수 없음

   a) analogy를 얼마나 잘 수행하는지 평가

   b) 단어의 유사도에 대해 인간과 얼마나 비슷한 결과를 내리는지 평가

2) Extrinsic
   - 현실에서 잘 작동하는지를 평가
   - 오래 걸림
   - 어디가 문제이고 원인인지 파악하기 힘듦

   a) named entity recognition

Word senses

한 단어에는 보통 여러가지 의미가 담겨있는데 이를 어떻게 처리하지?

→ 단어의 뜻에 따라 한 단어를 여러개로 분류해서 취급 (ex. $bank_1$, $bank_2$) 문제점 − 너무 복잡
− 단어의 뜻의 경계가 애매

→ linear superposition 사용

ex) $v_{pike} = \alpha_1 v_{pike} + \alpha_2 v_{pike} + \alpha_3 v_{pike}$ where $\alpha_1 = \frac{f_1}{f_1+f_2+f_3}$, etc., for frequency f

이런 식으로 평균을 내도 substantial similarity로 인해 결과 좋음