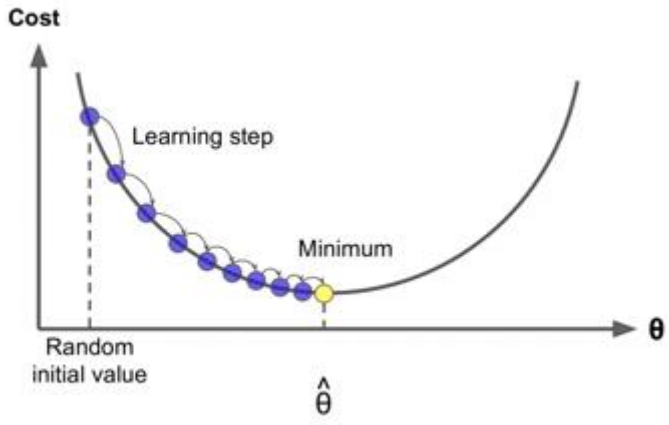
Lecture 02. Word Vectors and Word Senses

Ⅰ. Finish looking at word vectors and word2vec

Word vector를 vector space에 위치시켰을 때, 유사한 의미를 보유하고 있으면 가까이 위치한다. 높은 차원의 vector space에서는 한 단어가 다양한 다른 단어와 서로 다른 방향으로 인접할 수 있다. 2차원 공간에 벡터를 나타낼 경우, 이런 관계를 표현하지 못할 수 있다.

Ⅱ. Optimization basics

1. Gradient Descent



Gradient Descent

-손실함수인 J(Θ)를 최소화해야 함.

-손실함수의 기울기를 계산해서 기울기의 반대 방향으로 조금씩 이동하게 된다. 계속해서 이동하다 보면, 최종적으로 최소점으로 이동하게 된다.

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

α = *step size* or *learning rate*

-전체 parameter θ에 대해 계산하는 것은 비용이 매우 크므로 Stochastic Gradient Descent방식을 사용한다.

2. Stochastic Gradient Descent(SGD)

-전체 데이터 중 일부 sample에 대해서만 gradient를 계산하여 parameter를 업데이트한다. 결국 점진적으로 최소점으로 이동하게 된다.

-mini-batch : 여러 개의 sample을 모은 것. 보통 32개, 64개로 구성. 여러 개의 sample을 사용하기 때문에 noise가 줄어들고 병렬화가 가능하며 빠른 학습이 가능하다.

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

3. Word2vec

-CBOW(Continuous Bag of Words) : 모든 외부 단어로 가운데 단어를 예측

-SG(Sip-grams) : 가운데 1개의 단어로 외부의 모든 단어를 예측하는 방식

-Negative sampling : naïve softmax에 초점을 맞춤. 간단하지만 계산적으로 비싸다.

Ⅲ. Can we capture this essence more effectively by counting?

1. Co-occurrence matrix

-과거에는 window based co-occurrence matrix를 사용함. 특정 크기의 window 내에 나타난 단어들을 모두 count한다.(A라는 단어가 B에 인접하게 몇 번 사용되었는지 확인해서 matrix 구성) 유사한 쓰임/의미를 보유하고 있는 단어들끼리는 비슷한 벡터 구성을 보유하게 됨.

-문제점1 : sparse matrix를 형성하게 됨.

-문제점2 : 단어 수가 많아지면 matrix의 크기가 커진다.

->Solution : 차원을 줄인다.

2. Method of Dimensionality Reduction on X

-co-occurrence matrix X의 singular value decomposition

-U와 V가 orthonormal한 X를 $U\sum V^T$를 factorize한다.

```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```

<파이썬을 이용한 singular value decomposition>

3. Hacks to X

-cells안에서 count를 scaling하는 것은 도움이 많이 된다.

-문제점 : function words(the, he, has)는 너무 자주 언급됨.

-> min(X, t), with $t \approx 100$ 또는 이들을 모두 무시.

-count대신 pearson correlations 사용하고 음수를 0으로 설정한다.

4. Count based vs direct prediction

-Count based : 훈련 속도가 빠르고 통계 정보를 효율적으로 사용한다. 단어의 유사도를 계산하는데만 사용할 수 있어 단어 사이의 관계를 확인할 순 없고 복잡한 패턴을 인식할 수 없다. Ex)LSA, HAL, COALS, PCA 등

-Direct prediction : window 내의 데이터만 사용하기 때문에 전체적인 통계 정보를 활용하지 않는다. 대부분의 영역에서 좋은 성능을 내는 모델을 생성할 수 있고 단어 유사도에 대해 복잡한 패턴도 잡아낼 수 있다.

Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

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

IV. The GloVe model of word vectors

1. GloVe

-word2vec은 window 내의 정보만 사용한다는 점에서 전체적인 문장에 대한 이해가 떨어진다는 단점을 가지고 있다.

-Direct prediction과 count based 방식을 합쳐서 적용한다.(임베딩된 단어벡터 간의 유사도 측정을 가능하게 하면서 corpus 전체의 통계정보도 함께 사용할 수 있음.)

-목적 함수 : $w_i \cdot w_j = \log P(i|j)$

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

-장점 : 작은 규모의 corpus에 대해서도 효과적으로 학습이 가능하다. 자주 등장하지 않는 단어에 대해서도 효과적인 word vector를 생성할 수 있다. 학습이 상대적으로 빠르다.

V. Evaluating word vectors

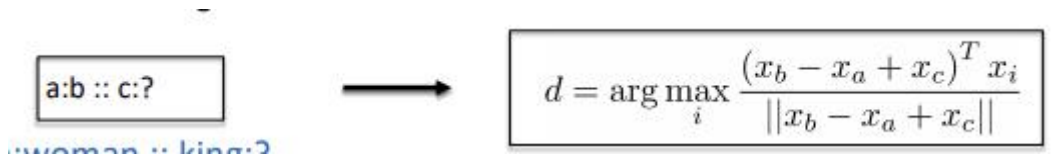
1. Intrinsic evaluation

-올바르게 task를 해결했는지 확인하는 방법

-쉽고 빠르게 검증할 수 있다.

-실제로 서비스에서 사용할 때 그 정도의 성능을 낼 수 있는지에 대한 보장은 못함.

-word vector analogies : captures intuitive semantic과 syntactic analogy questions을 더한 것에 거리를 코사인 한 값이 단어 벡터를 얼마나 잘 평가하는가. 정보가 linear하지 않은 경우는 문제가 생길수도.



-좋은 차원은 대략 300이하. Asymmetric context(단어가 왼쪽에만 있는 경우)는 좋지 않음. 그러나 이것은 downstream tasks에 의해 다를 수 있다. 각각 center wored 주변의

window size 8은 glove vectors에 좋다.

2. Extrinsic evaluation

-실제 시스템에서 사용해서 성능을 확인하는 방법. Ex)NER(Named Entity Recognition)

VI. Word Ambiguity

동일한 단어의 서로 다른 의미를 표현하는 방법은 다음과 같다.

1. Multiple sensors for a word

-하나의 단어가 벡터 공간에서 서로 다른 cluster를 생성하는 경우, 해당 단어를 여러 개로 분류해서 벡터를 생성한다. ex) $apple_1$, $apple_2$,…,$apple_n$

-대부분의 의미가 확장되서 사용되기 때문에 어느정도 연관이 있다. 그렇기 때문에 의미를 구분하는 것이 명확하지 않을 수 있음.

2. Weight Average

-한 단어의 서로 다른 의미를 나타내는 벡터들에 가중치를 부여해서 사용.

ex) pike라는 단어에 대해서 v_{pike1}, v_{pike2},…,v_{piken}의 벡터가 존재한다면, pike에 대한 최종 벡터는 $v_{pike}=\alpha_1 v_{pike1}+\alpha_2 v_{pike2}+\ldots+\alpha_n v_{piken}$, $\alpha_1=\frac{f1}{f1+f2+\cdots+fn}$이다.

-weight average를 사용하면 고차원 영역에서 해당 단어의 각 의미를 나타내는 벡터들이 sparse하게 표현되기 때문에, 전체 벡터의 일부에 단어의 의미를 나타낼 수 있다.