**Simple Word Vector representations:**

**word2vec, GloVe**

Stanford University

CS224d: Deep Learning for NLP

Richard Socher

(Word vectors are super-optimal and great way of representing words.)

First just think background.

Why do we represent the meaning of the word and

How do we represent the meaning of the word???

**How do we represent the meaning of a word?**

단어의 의미를 컴퓨터가 이해할 수 있도록 어떻게 표현할 것인가?

Definition : Meaning (Webster dictionary)

• the idea that is represented by a word, phrase, etc.

• the idea that a person wants to express by using words(여러 개의 단어들), signs (to try to express meaning), etc.

• the idea that is expressed in a work of writing, art, etc

* Meaning is elusive thing we’re trying to capture when we try to have computers understand languages.
* In common, the thing is to be desired to represent to something. 보통 의사소통을 할 때, 남에게 올바른 의미를 전달하려고 애쓴다. We want the representation to be received correctly to other people.
* There is a lot of very interesting linguistic and philosophy, try to understand how to people interact and how to people talk each other and so on
* Word2Vec은 각각의 단어를 vector로 표현하고, 단어의 의미들을 vector space 상에서 linear relationship으로 표현한다. 즉, 단어를 vector space 상의 특정 point에 있다고 생각한다.

A lot of those are very hard to formulate in computation. So, we only go through one example of alternative representation, which is basically trying to represent words and their meanings in terms of its taxonomy.

**How to represent meaning in a computer?**

Common answer: Use a taxonomy (분류체계) like WordNet that has

① hypernyms (is-a) relationships : 해당 단어와 특정 관계가 있는 단어들을 모두 List up한다.

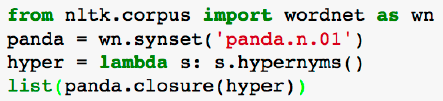
② synonym sets : 해당 단어와 동의어 Set을 List up한다.

taxonomy인 WordNet은 단어의 의미표현을 다른 단어들과 연결성을 이용해서 표현한다 또는 어떤 단어의 의미를 해당 단어의 분류체계 관점으로 표현한다.

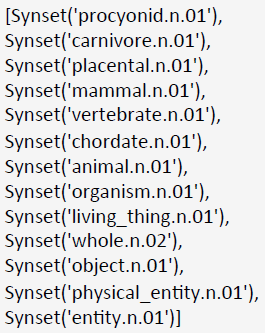
Taxonomy is basically very large graph, and the case of WordNet, which one of the most commonly used taxonomy, that defines a lot of relationships between words. For example, one of those relationships is ‘hypernyms’, which means ‘is-a’ relationship, that defines general terms.

For example, ‘panda’

We want to look at the first meaning of noun panda. And we can ask by python, what are all the words that generalize the ‘panda’?



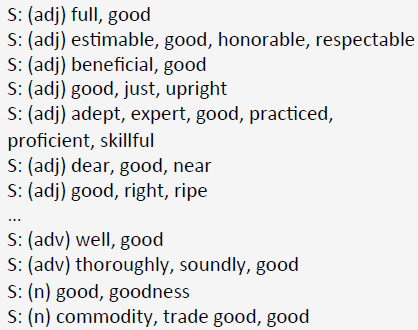
* 여러 개의 relationship 중에서 “is-a” relationship을 선택 → s.hypernyms()
* 즉, panda라는 단어와 is-a관계가 있는 단어들을 모두 List up한다.



* All these words subsume the ‘panda’. That is the ‘panda’ is subset of each of these words.
* Everything in the taxonomy, ends at entity.

That’s one way we could try to have a computer understand or represent a meaning of word. (Basically, connecting it to all other words that we have in the big graph.)

Another information the WordNet gives you is so called synonym(동의어) sets. These are synonyms.



* For example, here are the couple of meanings of word ‘good’. WordNet assumes the meaning here is completely identical (same meaning) (good에는 여러 개의 동의어가 있는데, 각 동의어에 해당되는 또 다른 단어들끼리는 모두 똑같다고 생각한다.).

하지만, 이처럼 어떤 단어의 의미를 표현하기 위해서 관련된 다른 단어들을 List up을 해야 되는데, 이는 binary 표현법 (List up or not), 즉, discrete 표현법이 된다.

So, what are some problems with those kinds of discrete representation and trying to formulate all of that in the taxonomy?

**Problems with this discrete representation**

• Great as resource but missing nuances (늬앙스, 미묘한 차이), **synonyms**: adept, expert, good, practiced, proficient, skillful?

* What we really use the term ‘expert’ interchangeably with ‘good’ probably is not.
* There missing a lot of nuances.
* 예를 들면, 비록 동의어라 하더라도 expert가 사용되는 문맥에 good을 대체할 순 없다. 즉, 의미가 서로 비슷하지만, 사용되는 문맥은 완전 다르다.

• Missing new words (impossible to keep up to date (최신상태가 항상 되게 하는 것): wicked, badass, nifty, crack, ace, wizard, genius, ninjia (올해 새로 생긴 단어 List)

* 언어는 항상 (빠르게) 진화하므로 일일이 손으로 업데이트하기란 쉽지 않다. 반면, 나중에 소개될 vector 표현법은 데이터로부터 학습하기 때문에 새로운 단어 습득에 용이하다.
* These days, those are obviously not in WordNet anymore
* It’s very hard to keep it up to date
* Ideally, we’d learn our vector representations and our word representations from data. We just give it more data and read continuously news and it starts learning about new words.

• Subjective

* Learning the vector representation and word representation from data, will be less subjective
* 단어들의 관계를 정의하는데 있어서 사람이 직접 작업하기 때문에 주관성이 있다.

• Requires human labor to create and adapt.

* This problem tends to be found in these kinds of discrete representation that we have.
* For example, WordNet for English. But WordNet does not exist for every other language.

• Hard to compute accurate word similarity

* Taxonomy graph에서 나온 Synonyms set을 가지고 있을 때, similarity를 측정하는 것은 단지 1bit로 similar하다/안하다 라고 밖에 표현하지 못한다. 예를 들어, list(panda.closure(hyper))를 실행하면 단지 list up밖에 되지 않는다. (list up된다/안된다)

제한적이다. → 이러한 것이 discrete representation의 문제점이다.

* The vast majority of rule-based and statistical NLP work regards words as atomic symbols
* In vector space terms, this is a vector with one 1 and a lot of zeros. This is very large and very sparse vector. We call this vector an “one-hot vector”. (여기서 벡터 크기는 Vocabulary 사이즈와 같다.)
* Dimensionality: 20K (speech) – 50K(PTB) – 500K(Big vocab) – 13M (Google 1T)
* The main problem is that, when you represent the word as index, and when you do a logistic regression or something, you don’t get any notion of similarity from these kinds of representation.

For example, model and hotel is quite similar, but we don’t get similarity.

When we have dot-product, we get 0 → don’t get similarity between two words. 이처럼 discrete 표현법의 일종인 one-hot vector 표현법을 사용하면 단어간의 유사성을 측정하기 어렵다. 모델과 호텔은 비슷한 단어이지만 0이 나오므로 유사성을 측정할 수 없다.

**Distributional similarity based representations**

Discrete 표현법이 단어간의 유사성을 표현하는데 제한이 되는 것을 극복하기 위해 이웃 단어들을 고려하는 방법 또는 표현법 등장.

The main idea of today lecture and of almost all word vector representation, is to use neighbors of the word, to represent that word.

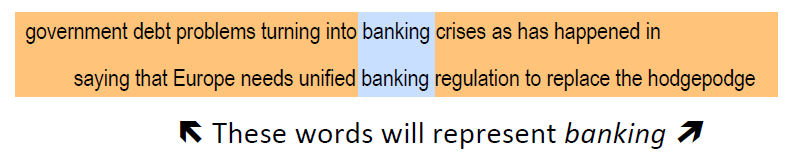
- You can get a lot of value by representing a word by means of its neighbors.

- “You shall know a word by the company it keeps” – (J. R. Firth 1957: 11)

- One of the most successful ideas of modern statistical NLP.

- It does permeate into deep learning

For example, instead of thinking ‘banking’ is just discrete thing, we are going to try to represent ‘banking’ in terms of left-right neighbors.



Bank might both have debt problem and crises in their context. Now, when represent that one word instead of single index, with bunch of neighbors of the word, you can gain some information.

So, now, we have a natural question, how do we actually try to represent the word in terms of its neighbors? The answer is with ‘co-occurrence matrix’.

**How to make neighbors represent words?**

Answer: With a co-occurrence matrix X

There are two options that we usually have, to create the co-occurrence matrix X. 즉, 내 이웃을 인정하는 범위가 어디까지냐에 따라 크게 2가지로 나뉠 수 있다.

• 2 options: full document vs windows

1. **Full document**: (document 전체를 context라고 생각)

we won’t go too much into. It just describes the co-occurrence in terms of words appearing in the same document. This kinds of representation that just looks at whether the words appear in the entire of the document, will not have any notion of part-of-speech tags. So, you might say all sports terms, will create roughly similar kinds of vectors (if you don’t look at local context)

Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”.

This word-document co-occurrence matrix is what will gain essentially is general topics if you just ignore all the order or words. But, we will basically lose any notion of syntax, part of speech when we just look at entire document as a context.

Word-document co-occurrence matrix를 SVD하고, document기준으로 vector로 나타내면 다음과 같다. doc1 = [ topic 1, topic 2, … , topic n ] 이고 물론 topic1~n은 모두 숫자이다. 숫자가 의미하는 것이 해당 topic의 가중치라고 생각할 수 있다. LDA 모델에서는 확률적인 접근방법으로 가중치를 확률로 표현할 수 있다.

[내 생각] Document 전체를 context라고 생각하면, 데이터 locality성을 잃어버린 것과 마찬가지로 된다. 이미지 데이터와 비교해보면 이미지에서 locality는 critical한 부분을 가진다. Text 데이터 역시 locality는 POS의 의미를 지닌다.

1. **Windows around words**: (window size만 context라고 생각 - 이 강의에서 많이 사용)

Instead of taking the entire of the document as the context, we only look at the words like 5-left words & 5-right words.

Window around each word → captures both syntactic (POS) and semantic information

POS를 획득할 수 있는 이유는 context 크기가 제한되기 때문이다. 즉, 어떤 단어 주변에 자주 붙는 단어들을 통해 해당 단어의 POS를 추출할 수 있다. (물론 제한된 dimension의 vector space상에 표현해서 linear relationship으로부터 추출될 것이다.) 하지만, 주변의 이웃 범위가 커지면, 즉 보통 문장 크기를 벗어난 것이면 POS를 추출하기가 어려울 것이다. 다시 말해, POS를 추출하기 위해서는 최소 문장 크기만큼의 window size를 설정해야 되지 않나 생각이 든다.

**Window based co-occurrence matrix**

Let’s go through example to give you some intuition of what actually looks like.

Let’s assume like this:

• Window length 1 (more common: 5-10)

* For every word, we only look at the left nearest neighbor and the right.

• Symmetric window (irrelevant whether left or right context)

* We don’t really care of the neighboring word to the left or to the right (이웃 word가 왼쪽에 있든 오른쪽에 있든 상관 안 한다.)
* Sometimes some people look at only left or right

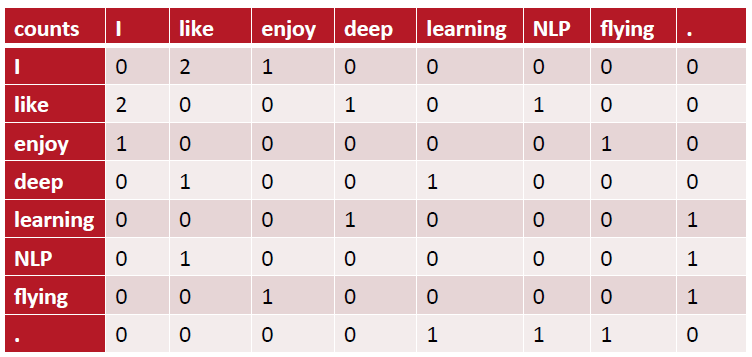
• Example corpus (simple corpus):

- I like deep learning

- I like NLP

- I enjoy flying

Let’s look at what in 1-length window, co-occurrence matrix (대칭행렬) look like.



* START\_TOKEN은 없다고 간주한다. 여기서 마침표가 END\_TOKEN 역할을 해준다.
* Like와 enjoy가 서로 similar하다는 것을 알 수 있다. overlap이 있으므로. (dot-product) → this is improvement over one-hot representation. (단어간의 similarity를 측정할 수 있는 것이 co-occurrence matrix로 표현하는 것이 one-hot 표현법보다 더 좋은 장점이다.)
* 이처럼 word를 하나의 vector로 표현할 수 있고(가로/세로 모두 사용가능하나 여기서는 가로로 기준을 잡자), 다른 word들과 dot-product를 하면서 similarity를 측정할 수 있다.
* This is just one way of representing words as vectors

**Problems with simple co-occurrence vectors**

However, these kinds of representation have some problem:

• Increase in size with vocabulary

* If we have subsequent model that we trained on the top of it, that change all the time
* New word가 생기면, update the whole matrix를 해야 된다. → 비효율

• Very high dimensional: require a lot of storage

* If we have 300million neighboring words, that representation can be very hard to store.

• Subsequent classification models sparsity issues

* We try to eventually estimate statistical parameters on the top of these words (300million x 300million). It’s going to be very hard to train for any kinds of models on the top of that kind of co-occurrence matrix.
* Every classification algorithm learn all the weight (따라서 sparse matrix처럼 정보가 많이 없는 matrix의 모든 weight를 training하는 것은 시간적으로 비효율적. -> 즉, 거대한 softmax weight를 가지게 될 우려가 있다. )

→ These lead to less robust model

**Solution: Low dimensional vectors**

The solution is to use low dimensional vectors from the very large sparse matrix.

• Idea : store “most” of the important information in a fixed, small number of dimensions: a dense vector representation

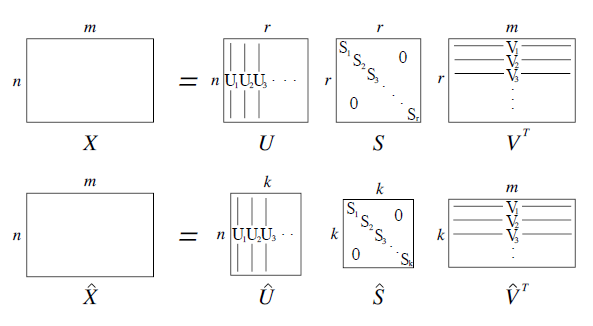
• Usually around 25 – 1000 dimensions (broad range) (but, it depends on your task)

* For simple task like finding name entity, locations, or people, 25 dim might be applied, whereas if you build machine translation, you’d better have 1000 dim (1000 dim까지 가면 training data가 정말 많이 있어야 한다.)

• That leads us to the question, how should we reduce the dimensionality from having this large co-occurrence matrix, to smaller dimensional vectors.

**Method 1: Dimensionality Reduction on X**

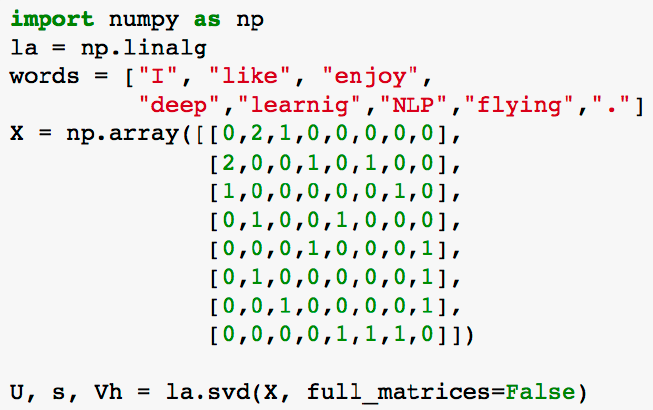
Singular Value Decomposition of co-occurrence matrix X.



* We can represent the original co-occurrence matrix and find to best-length k approximation to that, in terms of Least Squares by using SVD.
* Orthonomal columns in left singular vectors and orthonomal rows in right singular vectors. And along diagonal, singular values, which are sorted.
* Each of these columns basically describes principle component.
* Actually, diagonal matrix is automatically ordered and the first singular value is the most important axis variation in your data in the first column of
* Instead of representing X, all of the orthonomal columns and rows and we can just take top-k. That allow us to reduce noise.
* 마지막 부분의 eigen-value들은 노이즈로 취급될 수 있다. (덜 중요한 패턴)

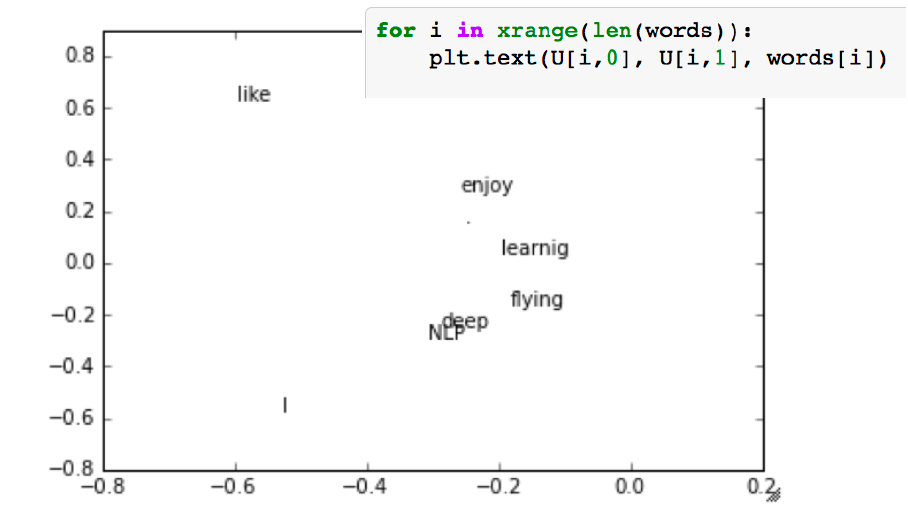
**Simple SVD word vectors in Python**

Corpus: I like deep learning. I like NLP. I enjoy flying.



* Words represent the vocabulary
* The matrix represents the co-occurrence matrix.

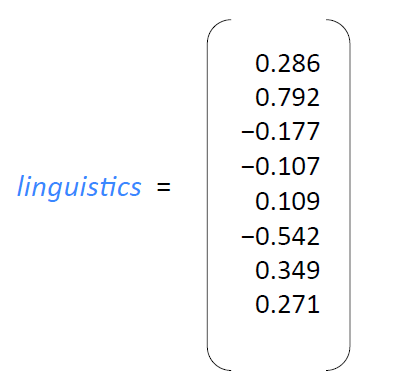
Printing first two columns of U corresponding to the 2 biggest singular values (2D Visualization)



* This is simplest type of word vector representation as low dimensional
* For each word, we’re going to use the k-first elements of the left singular vectors
* We can visualize this in 2D (SVD로부터 top-2추출), by taking first 2 elements (the biggest axis variation inside our low dimensional space)
* What we observe in this vector space, is that we actually capture intuitively a lot of similarity.
* For instance like and I, they’re probably two most frequent words in the corpus. And so some notion here captures a frequency in one of these axises
* We also know that like is most similar to enjoy in some ways, because enjoy is nearest neighbor to like
* You can say, enjoy and learning are similar because they both roughly in similar places.
* The most similar to learning, is flying. Both happen at the end of sentences hence they’re very similar.
* Like와 I는 공통된 세로 축을 가지므로 비슷한 linear relationship을 가지는데 이는 Like와 I가 frequency가 높아서 일수도 있다.
* 여기서 생기는 궁금증. 왜 U 매트릭스에서 추출하는 것일까? 원래 매트릭스가 대칭행렬이기 때문에 그런 것일까?

**Word meaning is defined in terms of vectors**

• In all subsequent models, including deep learning models, a word is represented as a dense vector.



* Linguistics 단어는 SVD를 통해 top-8를 거친 후 결과로 나온 8개의 element가 있는 하나의 벡터로 표현된다.
* Mostly, 2D vectors is only for visualizing the vectors. Almost all cases, there will be higher than 2D. (25 dim – 1000 dim)
* How do we make decision to use the number of dimension? The answer is it’s to be determined by the overall tasks. (Name entity, sentiment analysis, machine translation, …)

**Hacks to X**

• Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:

① min(X,t), with t~100 : ex) the maximum count of ‘the’ is 100.

② Ignore them all

* Rare words actually captures a lot more semantics.
* Instead of allowing taking the raw co-occurrence matrix, you actually have to do some hacks (①, ②).
* ① : this way is to do not allow the function words to determine the meaning of all the other words. That leads more syntactic representations. (명사는 the 나 a 다음에 잘 나타나게 되는데, 이런 경우를 같이 묶으면, loss some semantic information 할 수 있다.)

• Ramped windows that count closer words more

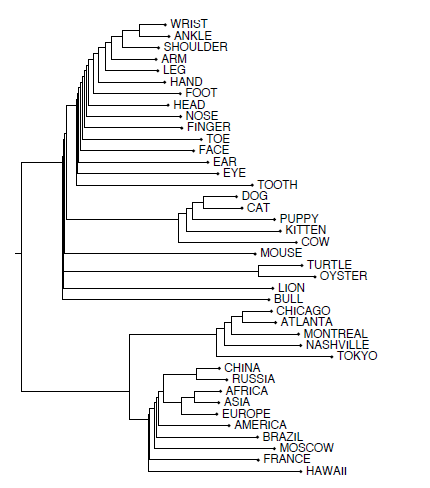
* In co-occurrence matrix, counts with weight of them. 즉, 기존의 co-occurrence matrix와 같이 count만 고려하는게 아니고, center word와의 거리에 따라 weight도 부여해준다. 만약 window-size가 1이면 weight는 언제나 동일할 것이다.

• Use Pearson correlations instead of raw counts, then set negative values to 0

* It’s one of the most powerful thing to do transformation to matrix
* Pearson correlation will be negative for most words. That is, the most words don’t appear in the context of other words. (팬더는 뉴클리어밤이랑 왠만하면 같이 나오지 않는다.)
* Pearson correlation의 결과는 [-1, 1]을 가지는 대칭행렬이 완성된다.
* Co-occurrence matrix를 그냥 count로 채우는 것보다는 pearson correlation 후의 결과값으로 채우면, 어떤 의미적인 정보가 더 추가되지 않을까 싶다.

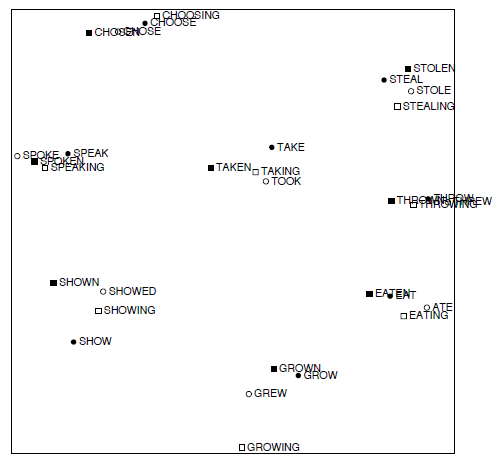
**Interesting semantic patterns emerge in the vectors**

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence Rohde et al. 2005



* Here is interesting first results when we use this kinds of SVD based methods, and try to find the nearest neighbors of specific words and just do hierarchical clustering in top-k vector space.
* WRIST, ANDKLE, SHOULDER are most similar in the resulting vector space
* But, again, there is no magic here. But, still there is a semantic similarity. For example, WRIST and ANKLE are probably appearing in the same context.
* 이렇게 계층적 구조로 단어들을 표현함으로써 단어들간의 semantic similarity를 표현할 수 있다.

**Interesting syntactic patterns emerge in the vectors**



* Again, project down word vectors using PCA, orthonomal columns (SVD) (top-k선택)
* Since we have these windows, we’re actually capturing syntactic information, automatic information
* We didn’t have to tell the model about different tenses of the verves, yet it learns that these are go together.
* For example, TAKE, TAKEN, TAKING are very close to one another in the vector space in TOOK and SHOW, SHOWN, SHOWING are very close to each other in the vector space in SHOWED.
* This gives us some hint – maybe we can get a way by using vectors to even capture automatical information.
* Window size가 적절하게 설정되어 있으므로 위와 같이 Syntactic analysis를 할 수 있다. 따로 모델에 동사 시제에 관한 정보를 주지 않았는데도 grouping을 할 수 있다.
* 이처럼 root가 같은 동사들끼리 grouping되는 이유는 이들의 이웃단어들이 비슷하게 등장할 것이기 때문이다. 따라서, root가 같은 동사들은 비슷한 word vector를 가질 것이다.

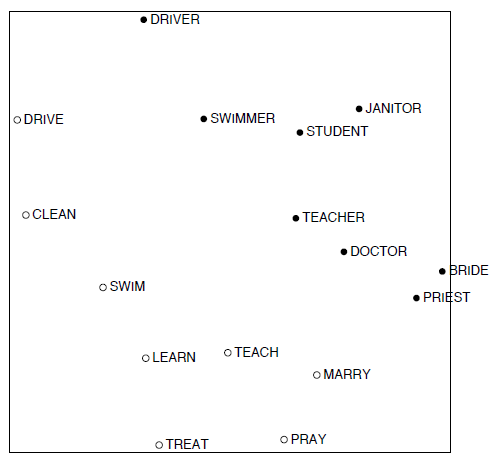
※ TALK

Similar words appear in similar context.

Similar geometric structures

Shown, showed, showing, show의 이웃단어들은 비슷할 것이다. 따라서 이들은 비슷한 linear relationship을 가질 것이다.

**Interesting semantic patterns emerge in the vectors**



* 다른 의미의 단어들이라도 동사와 동사의 profession 사이에는 similar structure(=similar direction, 어떤 패턴)를 공유하고 있다. 이러한 패턴은 linear relationship을 통해 확인된다.
* We didn’t tell the model anything about the word and that certain profession. It just fell out of this statistics in the model. (정보의 출발점은 횟수(=statistics) 그리고 여러 단계의 가공 (pearson correlation, SVD)을 통해서 linear relationship을 얻는다.)

※ TALK

When we project down word vectors, we can have nice sort of linear relationship. (의미적인 정보)

**Problems with SVD**

공통적으로 SVD-based 방법론들은 train을 one-shot으로 끝내버린다.

• Computational cost scales quadratically for n x m matrix (word-doc matrix or word-word matrix):

→ Bad for millions of words or documents

• Hard to incorporate new words or documents

* Since SVD Operation is global operation in the entire of matrix, it’s not easy to add new words.
* Even there is some way to iterative method, it’s non-trivial either.

• Different learning regime than other DL models

* Deep Learning method will look at the specific example, and try to learn something from that example, and then move on the next one.
* SVD는 단 한번에 train을 끝내는 반면, Deep learning 모델은 점진적으로 weight를 update해가면서 모델을 발전시켜나가는 방향이다.

**Idea: Directly learn low-dimensional word vectors** (based on just single window)

• Old idea. Relevant for this lecture & deep learning:

- Leaning representations by back-propagating errors. (Rumelhart et al. 1986)

- A neural probabilistic language model (Bengio et al. 2003)

- NLP (almost) from Scratch (Collobert & Weston, 2008)

- A recent, even simpler and faster model : word2vec (Mikolov et al. 2013) → intro now

**Main Idea of word2vec**

• Instead of capturing all co-occurrence counts directly (globally for the entire corpus),

• Predict surrounding words of every word

* Instead of collecting a large co-occurrence matrix, we are just looking at each example one at the time and we’re going to say can I predict left or right words based on center word?
* 이렇게 왼쪽-오른쪽 단어들을 예측하기만 해도 우리는 We’ll eventually also capturing a co-occurrence statistics. (just like previous method) But, it does it in very online way. So, it’s easy to add new words. (easy to update the model)

SVD-based 방법론들은 모든 단어들의 횟수가 미리 정해져 있어야 하는 반면, Word2Vec 방법론은 횟수를 전부 알지는 못해도 sampling을 통해서 횟수를 확률로 표현한다. 그리고 점진적으로 모델을 train해서 정확도를 높여가는 방식이다. (sampling이 거의 population을 흉내 낼 때는 SVD와 같이 모든 횟수를 추정할 수 있을 것이다. 그리고 좋은 sample 몇 개만 가져도 정확한 확률을 가질 수 있을 것이다.)

* Instead of keeping all the counts at the first, for specific context ( \_ \_ \_ □ \_ \_ \_ ), given that I know this word(=□) in my context, could I predict toward to left words or right words?
* It turns out when you are trying to learn a vector representation of □, that predicts other vectors (=other words).
* It’s very interesting and deep mathematical that your connection to trying to capturing counts. In some ways, they’re actually quite similar.

• Both are quite similar, see “Glove: Global Vectors for Word Representation” by Pennington et al. (2014) and Levy and Goldberg (2014) … more later

* It shows the connection

• Faster and can easily incorporate a new sentence/document or add a word to the vocabulary

**Details of Word2Vec**

• Predict surrounding words in a window of length m of every word.

• Objective function: Maximize the log probability of any context word given the current center word:

* Where θ represents all variables we optimize and T is a time step. (sequence)
* j≠0 : means we ignore the center word
* Often, the window size(c) is from 5 to 10.
* We have a very large corpus of T tokens (840 billion) → it takes very long time
* 여기서는 거리에 따라 weight가 부여되는 ramp window는 사용하지 않는다.

Now, the next question is what is actual representation of (=the probability of context words given the center word of window)

• Predict surrounding words in a window of length c of every word

• For , the simplest first formulation is:

* Context words = = = outside words = output words
* Center word = = = inside word = input word
* : outside word의 output vector
* : inside word의 input vector
* where v and are “input” and “output” vector representations of w (so every word has two vectors!)
* For input word, we can update output words
* Once we move to the next window, outside word가 inside word가 된다. 따라서, 단어마다 항상 2개의 vector를 가지고 있어야 한다. → 이렇게 2개로 나누면 목적함수를 optimize하기 더 쉽다.
* 조금 특이하게 Word2vec는 한 단어를 표현하는 dense vector를 2개로 표현한다. (최적화에 도움) 그리고, 어차피 같은 Context 내에 있으므로 dense vector들인 과 가 같아 지도록 하는 게 목표이다. 즉, cosine similarity를 사용해서 두 개의 벡터의 유사성을 표현한다.
* 우리가 weight 최적화해야될 파라미터는 dense vector들인 과 파라미터이다.
* This will the objective function for each single window
* Where o is the outside (or output) word id, c is the center word id, u and v are “center” and “outside” vectors of o and c
* exp 함수 : 항상 positive number
* Every word has two vectors. One vector is when we represent as outside word and Another vector is as you’re trying to predict the outside word.
* To be honest, you don’t need to separate the word vectors, and (원래, 하나의 vector에서 quadratic expression을 만들면 된다.) 그러나 2개 vector를 가짐으로써 optimization을 좀 더 쉽게 할 수 있다. This is not very beautiful, but in the end in order to capture as much as you can all of statistics, you are just going to average or concatenate these two vectors to represent each word. So, at the very end of the optimization after learning two vectors, for each word, we’re just going to average two.
* If you are familiar with logistic regression, you can see this as a dynamic logistic regression. (를 weight라고 생각) (w개의 class중에서 o class일 확률)
* 와 는 어차피 같은 context에 있는 vector들이다. 같은 context (window-size에 의해 결정)에 있는 단어들은 비슷한 모습의 word vector를 가져야 한다. (목표)

• This is essentially “dynamic” logistic regression

※ 나의 생각

• 주어진 정보는 center word와 붙어있는 word들의 횟수

• 25 dimension에 해당하는 단어는 모두 같다.

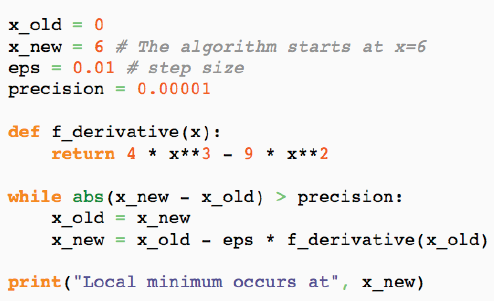
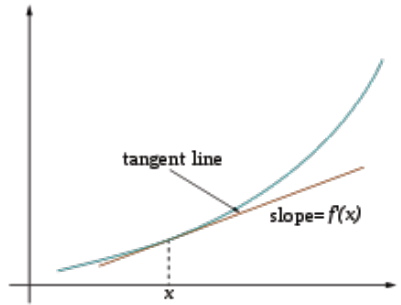
• 는 그냥 cosine similarity 측정하기 위한 과정

**Cost/Objective functions**

• We will optimize (maximize or minimize) our objective/cost functions.

• For now: minimize → gradient descent

• Refresher with trivial example: (from Wikipedia) Find a local minimum of the function with derivative



**Derivations of gradient**

• The basic Lego piece

• Useful basics :

• Chain rule! If *y = f(u)* and *u=g(x)*, i.e*. y=f(g(x))*, then:

**Interactive Whiteboard Session!**

Assuming we want to update the center word vector,

* Log function will make all math more simpler and actually optimization result is the same. (it’s monotonously increasing function)
* 보통, 와 들은 처음에 random small number로 초기화된다.
* Too expensive to compute → we will actually come up with much more efficient ways to avoid having to have the entire normalization and what we might do is just have a very simple binary problem of saying improve the probability of these to being on.
* Actually, this is the simplest way. Nobody would actually implement this equations.
* Our objective function is not scalable. So, what we’ll do is to approximate the normalization constant or to define the negative predictions, where you only basically sample few of the words that do not appear the context.

**Approximations: PSet1** (To solve very expensive computation)

• With large vocabularies this objective function is not scalable and would train too slowly! → Why?

- a million dot-product → very inefficient

• Idea: approximate the normalization or

- Instead of having normalization constant over the entire sum, we actually sub-sample here

- Instead of introducing softmax, we just have very simple binary logistic regression

• Define negative prediction that only samples a few words that do not appear in the context

• Similar to focusing on mostly positive correlations

* If you have millions of words, most of the words don’t appear in any given other words context
* So, what you will do is to focus on most of the positive correlations and just randomly sub-sample a couple of the words that don’t appear

• You will derive and implement this in PSet1

**Linear Relationships in word2vec**

Once we are able to optimize the function very efficiently, we can have amazing things.

• These representations are very good at encoding dimensions of similarity!

• Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space syntactically

* Similarly for verb and adjective morphological forms semantically (Semeval 2012 task 2)

**Count based vs direct prediction**

**1. Count based method list (traditional PCA and SVD method)**

LSA, HAL (Lund & Burgess),

COALS (Rohde et al),

Hellinger-PCA (Lebret & Collobert)

* Fast training (장점)  
  - they require huge size of ramp but, they quite fast
* Efficient usage of statistics (장점)

- Instead of collecting statistics and optimizing function one element at a time, you just can collect at once and then you can run merely on statistics

* Primarily used to capture word similarity (단점)

- they didn’t have any of the beautiful linear relationship

* Disproportionate importance given to large counts (단점)

- fixed counts

**2. Direct prediction method list**

NNLM, HLBL, RNN, Skip-gram/CBOW,

(Bengio et al; Collobert & Weston; Huang el al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)

* Scales with corpus size (단점)

- If you add large corpus, you need to go over many more windows

* Inefficient usage of statistics (단점)

- Only after learning it long enough, you can capture overall the statistics of the dataset

* Generate improved performance (장점)
* Can capture complex patterns beyond word similarity (장점)

**Word2vec 추가 노트**

<https://www.youtube.com/watch?v=vkfXBGnDplQ>

Word2vec : learn word vector from its surrounding context

• SVD, LSA, LSI

- compress co-occurrence matrix in some way using SVD

• N-gram

- transition probability matrix

• Word2vec

- learn the vector directly

- we don’t have the big matrices

- randomly initialization and gradient descent

**Conceptual Level of word2vec**

“The fox jumped **over** the lazy dog”

Maximize the likelihood of seeing this context given the word over.

Q. 어떻게 P( fox | over )를 정의할 것인가?

일단 어떤 방식이든 다음과 같이 word vector로 나타내긴 할 것이다.

하지만, word2vec에서는 word vector를 조금 다르게 표현하고 있다.

→ we have two vectors for every word. It should depend on whether it’s the input or the output. And there is a context window around every input word.

조건부 확률을 나타낼 때, 한 단어가 들어갈 자리는 2개 있으므로, 한 단어에 대해서 두 개의 벡터, 과 으로 표현될 수 있다.

High-level로 보면 2개의 for문으로 알고리즘이 돌아간다. 모든 단어들의 개수와 window size만큼.

Q. How should we define ?

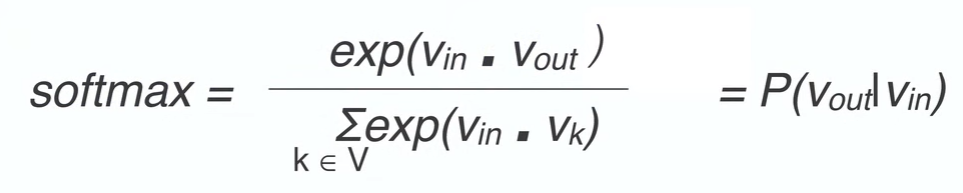
We need to measure loss function between and .

과 이 동시에 발생한다고 생각하자. 우리는 과 이 최대한 같아질 수 있도록 하는 게 목표이다. 그러기 위해서는 과 의 similarity를 계산해야 되는데, (즉, 2개의 벡터를 비교해야된다. 두 벡터를 비교하는 방법에는 Euclidian distance 또는 subtract하고 norm을 구하는 방법들이 있다.) 여기서는 Cosine similarity를 사용한다.

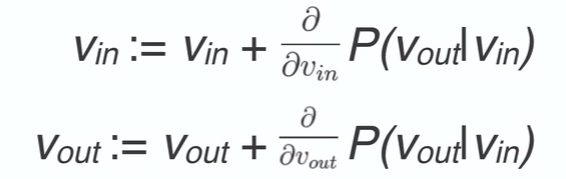
하지만 Cosine similarity를 사용하면 범위가 [-1, 1]이 되는데, 우리는 확률적 의미의 출력을 원하기 때문에 softmax를 통해서 [0, 1]이 되도록 해준다.

softmax를 개념적으로 이해하면 다음과 같다. Probability of choosing 1 of N discrete items. Mapping from vector space to a multinomial over words. 즉, vocabulary에 있는 word들 중에서 1개 word를 뽑을 확률이다.

**Algebric Level of word2vec**



We learn the parameters by using gradient descent on the softmax prob. For example, we see update :



* 지속적으로 update하면서 과 이 되도록 같아지도록 해준다.