CSC 439/539
Statistical Natural Language Processing
Lecture 3: Distributional Similarity and Word Embeddings

Mihai Surdeanu
Fall 2017

Take-away

- Distributional similarity
- Word embeddings

Overview

- ”Traditional” distributional similarity
- Latent semantic indexing
- Word embeddings

Overview

- ”Traditional” distributional similarity
- Latent semantic indexing
- Word embeddings

Distributional hypothesis

- Distributional hypothesis:
  - By looking at a word’s context, one can infer its meaning (Harris, 1954)
  - You shall know a word by the company it keeps (Firth, 1957)

Example

- tasty *tnassiorc*
- *tnassiorc* with butter
- *tnassiorc* and coffee
- greasy *tnassiorc*

Example

- tasty *tnassiorc*
- *tnassiorc* with butter
- *tnassiorc* and coffee
- greasy *tnassiorc*

Co-occurrence matrix

- Counts computed over a context window (say +/- 10 words) around each occurrence of a word
  - Context may be generated across syntactic relations as well
- Example!

From words to vectors

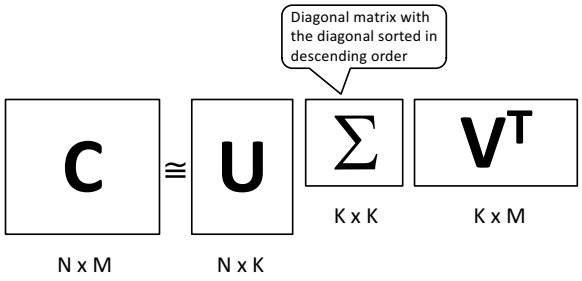
- We now have a context vector for each word in the vocabulary!
- We can use these vectors for any similarity task, e.g., finding the top 10 most-related words (your homework 1!)
- What are the advantages and disadvantages of this approach?

Overview

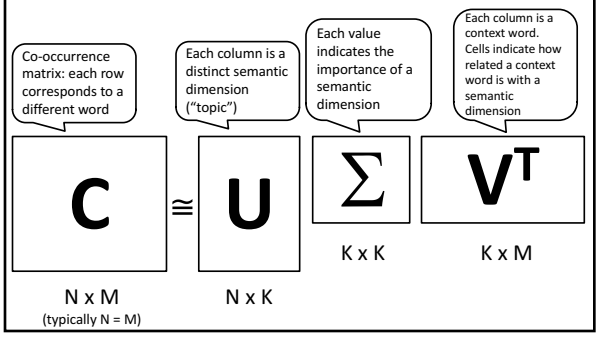
- ”Traditional” distributional similarity
- Latent semantic indexing
- Word embeddings

Singular value decomposition

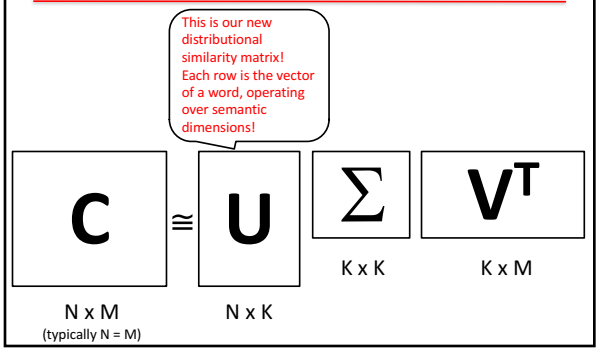
This produces the best rank K approximation of C



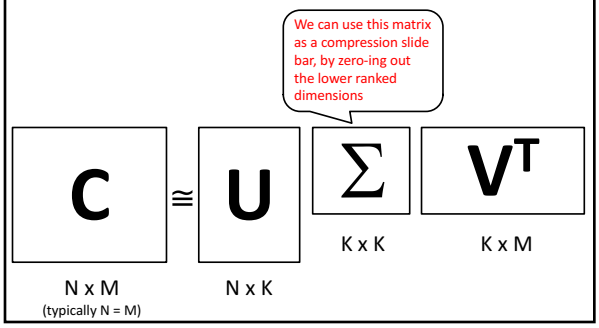
Singular value decomposition for distributional similarity



Singular value decomposition for distributional similarity



Singular value decomposition for distributional similarity



The problem

- The runtime of SVD is cubic in the size of the vocabulary…
- Word embeddings do the same things, but much quicker (approximately linear time in the size of the vocabulary)

Overview

- ”Traditional” distributional similarity
- Latent semantic indexing
- Word embeddings

Intuition

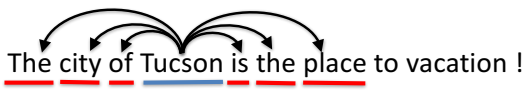
You shall know a word by the company it keeps.

- Firth, 1957

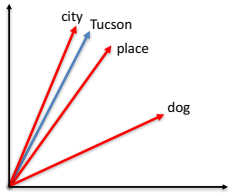
You shall know the company by the word it keeps.

- Word2vec, skip-gram

Intuition



P(city|Tucson) ++
P(place|Tucson) ++
…
P(dog|Tucson) - -
…



18

Word2vec definition

- (Neural) language model that predicts the surrounding words of every word
- For the learning part, I prefer the name "dynamic logistic regression"

Objective function

- Predict surrounding words in a window of length *m* (on each side), for every word
- Objective: maximize the log probability of any context word given the current center:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{-m \le j \le m, j \neq 0} \log p(w_{t+j}|w_t)$$

- This is similar to your objective function for LR!
  - But we operate over context words rather than labels

Content from Stanford's CS224d, lecture 2

Objective function

- We implement $p(w_{t+j}|w_t)$ using a softmax around two vectors, a "center" vector (c) and an "outside" vector (o):

$$p(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W} \exp\left(u_w^T v_c\right)}$$

Content from Stanford's CS224d, lecture 2

Details

- Thus each word gets 2 vectors: one for when the word serves as "center", another for when it serves as "outside" context
  - The "center" vector is what word2vec outputs
  - But other research showed that $v = v_o + v_c$ performs better in practice
- The algorithm learns the weights for these vectors that maximize p(o|c) according to J
  - These are the values of your "features"
  - The algorithm dynamically learns which features are the best for this task
    - Hence the "dynamic" logistic regression name

The problem

- What is the problem with this objective function? What happens with all these vectors during training?

The problem

- What is the problem with this objective function? What happens with all these vectors during training?
- So, how do we avoid this Kumbaya outcome?

The problem

- What is the problem with this objective function? What happens with all these vectors during training?
- Solution: add negative examples!

Negative examples

- New objective function:

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

- How do we pick negative examples?
  - P(w)=U(w)$^{3/4}$/Z
  - Less frequent words are sampled more often
  - Words in the actual context are excluded

Content from Stanford's CS224d, lecture 3

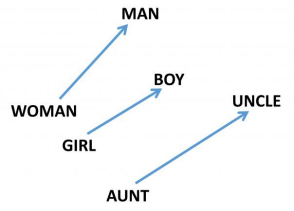
Variants

- The model we just saw is called skip-gram
  - Empirically, this works well
- Other possibilities: the continuous bag of words (CBOW) model
  - Predict the center word given its outside (context) words
  - Typically generates worse vectors. Why?

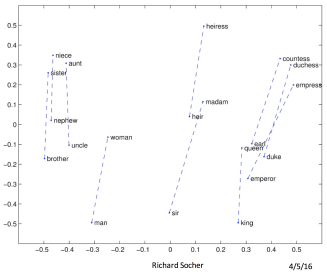
Important hyper parameters

- Size of vectors
  - Usually between 50 and 500
- Size of window around each word
  - Usually between 10 and 20
- Number of negative examples
  - Usually up to 10 times more than positive ones

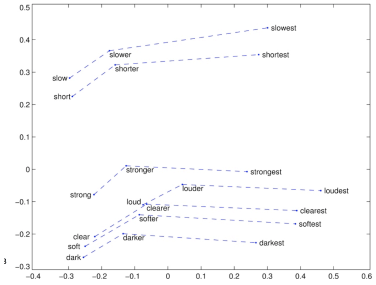
What do word embeddings learn?

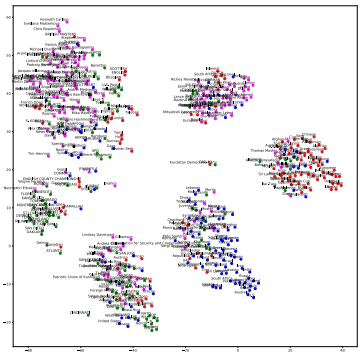


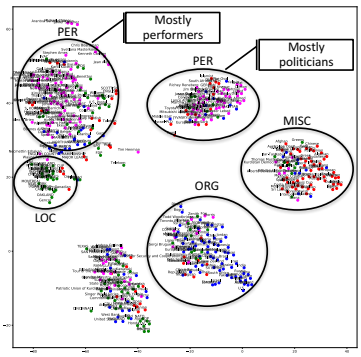
What do word embeddings learn?



What do word embeddings learn?







Readings

- informationretrieval.org, chapter 18 (for LSI)
- CS224d, lectures 2 and 3 at Stanford

Take-away

- Distributional similarity
- Word embeddings