Vector Semantics

Dense Vectors

(based on Jurafsky, TensorFlow)

Three Perspectives on Meaning

1. **Lexical Semantics**
   - The meanings of individual words
2. **Formal Semantics** (or Compositional Semantics or Sentential Semantics)
   - How those meanings combine to make meanings for individual sentences or utterances (see skipped chapters)
3. **Discourse or Pragmatics**
   - How those meanings combine with each other and with other facts about various kinds of context to make meanings for a text or discourse
   - **Dialog or Conversation** is often lumped together with Discourse

Lexical Semantics

- Focus on word meanings:
  - Relations of meaning among words
  - Internal meaning structure of words (current chapters)

The old days: words were represented as discrete, atomic symbols
- Prof. Litman = prof1
- Prof. Hwa = prof2
- Teach(prof1,NLP17)
- Teach(prof2,NLP16)
- ids are arbitrary, atomic, led to data sparsity…

Last class: sparse vectors
- still had data sparsity problems

Today's class: dense vectors
- Count then reduce
- Parameters of prediction model

4

Review: Context vector

- Consider a target word *w*
- One binary feature f_i for each of the *N* words in the lexicon v_i
- Which means "word v_i occurs in the neighborhood of *w*"
- w=(f_1,f_2,f_3,…,f_N)

Intuition

- Define two words by these sparse features vectors
- Apply a vector distance metric
- Say that two words are similar if two vectors are similar

	arts	boil	data	function	large	sugar	summarized	water
apricot	0	1	0	0	1	1	0	1
pineapple	0	1	0	0	1	1	0	1
digital	0	0	1	1	1	0	1	0
information	0	0	1	1	1	0	1	0

Distributional similarity

- So just need to specify 3 things
  1. How the co-occurrence terms are defined
  2. How terms are weighted
     - (Frequency? Logs? Mutual information?)
  3. What vector distance metric to use?
     - Cosine? Euclidean distance?

Defining co-occurrence vectors

- Could have windows of neighboring words
  - Bag-of-words
  - Generally remove **stopwords**
- But the vectors are still very sparse
- So instead of using ALL the words in the neighborhood could use the words occurring in particular relations

Co-occurrence vectors based on dependencies

- For the word "cell": vector of NxR features
  - R is

	subj-of, absorb	subj-of, adapt	subj-of, behave	...	pobj-of, inside	pobj-of, into	...	nmod-of, abnormality	nmod-of, anemia	nmod-of, architecture	...	obj-of, attack	obj-of, call	obj-of, come from	obj-of, decorate	...	nmod, bacteria	nmod, body	nmod, bone marrow
cell	1	1	1		16	30		3	8	1		6	11	3	2		3	2	2

2. Weighting the counts

("Measures of association with context")

- Use the frequency of some feature as its weight or value
- But could use any function of this frequency

Intuition: why not frequency

Object	Count	PMI assoc	Object	Count	PMI assoc
bunch beer	2	12.34	wine	2	9.34
tea	2	11.75	water	7	7.65
Pepsi	2	11.75	anything	3	5.15
champagne	4	11.75	much	3	5.15
liquid	2	10.53	it	3	1.25
beer	5	10.20	<SOME AMOUNT>	2	1.22

- "drink it" is more common than "drink wine"
- But "wine" is a better "drinkable" thing than "it"
- Idea:
  - We need to control for chance (expected frequency)
  - We do this by normalizing by the expected frequency we would get assuming independence

Dan Jurafsky

Sparse versus dense vectors

- PPMI vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)
- Alternative: learn vectors which are
  - **short** (length 200-1000)
  - **dense** (most elements are non-zero)

12

Dan Jurafsky

Sparse versus dense vectors

- Why dense vectors?
  - Short vectors may be easier to use as features in machine learning (less weights to tune)
  - Dense vectors may generalize better than storing explicit counts
  - They may do better at capturing synonymy:
    - *car* and *automobile* are synonyms; but are represented as distinct dimensions; this fails to capture similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor

13

Dan Jurafsky

Three methods for getting short dense vectors

- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- "Neural Language Model"-inspired predictive models
  - skip-grams and CBOW
- Brown clustering

14

Vector Semantics

Dense Vectors via SVD

Dan Jurafsky

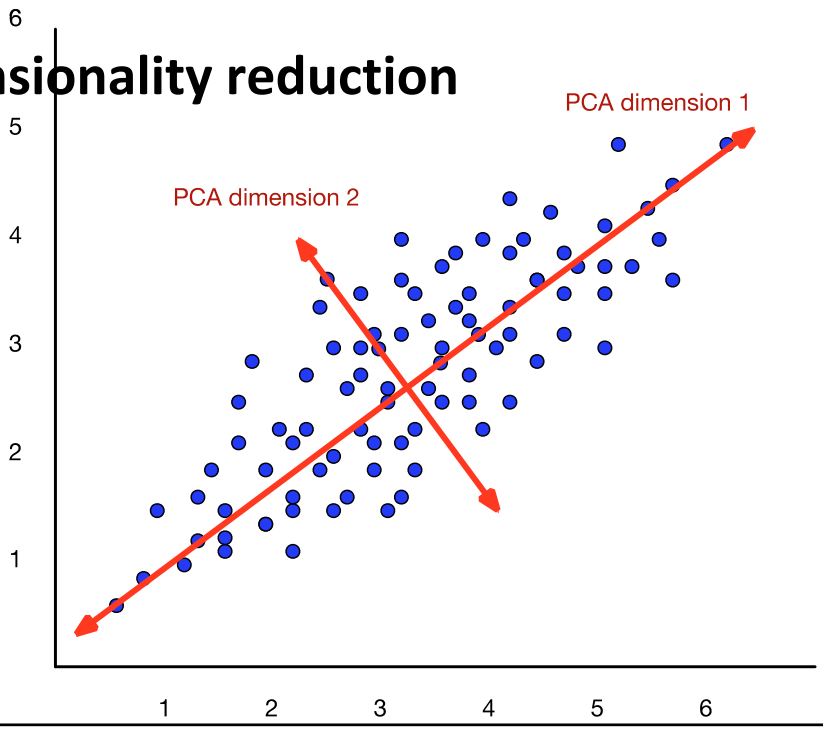
Intuition

- Approximate an N-dimensional dataset using fewer dimensions
- By first rotating the axes into a new space
- In which the highest order dimension captures the most variance in the original dataset
- And the next dimension captures the next most variance, etc.
- Many such (related) methods:
  - PCA – principle components analysis
  - Factor Analysis
  - SVD

16

Dan Jurafsky

Dimensionality reduction



17

Dan Jurafsky

Singular Value Decomposition

Any rectangular w x c matrix X equals the product of 3 matrices:

W: rows corresponding to original but m columns represents a dimension in a new latent space, such that

- M column vectors are orthogonal to each other
- Columns are ordered by the amount of variance in the dataset each new dimension accounts for

S: diagonal *m* x *m* matrix of **singular values** expressing the importance of each dimension.

C: columns corresponding to original but m rows corresponding to singular values

18

Dan Jurafsky

SVD applied to term-document matrix: Latent Semantic Analysis

Deerwester et al (1988)

- If instead of keeping all m dimensions, we just keep the top k singular values. Let's say 300.
- But instead of multiplying, we'll just make use of W.
- Each row of W:
  - A k-dimensional vector
  - Representing word W

19

Dan Jurafsky

LSA more details

- 300 dimensions are commonly used
- The cells are commonly weighted by a product of two weights
  - Local weight: Log term frequency
  - Global weight: either idf or an entropy measure

20

Dan Jurafsky

Let's return to PPMI word-word matrices

- Can we apply to SVD to them?

21

Dan Jurafsky

SVD applied to term-term matrix

$$\underset{|V|\times|V|}{\begin{bmatrix} X \end{bmatrix}} = \underset{|V|\times|V|}{\begin{bmatrix} W \end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_V \end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix} C \end{bmatrix}}$$

22

(simplifying by assuming the matrix has rank |V|)

Dan Jurafsky

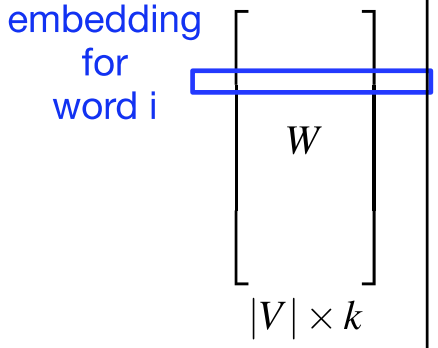
Truncated SVD on term-term matrix

$$\underset{|V|\times|V|}{\begin{bmatrix} X \end{bmatrix}} = \underset{|V|\times k}{\begin{bmatrix} W \end{bmatrix}} \underset{k\times k}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}} \underset{k\times|V|}{\begin{bmatrix} C \end{bmatrix}}$$

Dan Jurafsky

Truncated SVD produces embeddings

- Each row of W matrix is a k-dimensional representation of each word *w*
- K might range from 50 to 1000
- Generally we keep the top k dimensions, but some experiments suggest that getting rid of the top 1 dimension or even the top 50 dimensions is helpful (Lapesa and Evert 2014).



Dan Jurafsky

Embeddings versus sparse vectors

- Dense SVD embeddings sometimes work better than sparse PPMI matrices at tasks like word similarity
  - Denoising: low-order dimensions may represent unimportant information
  - Truncation may help the models generalize better to unseen data.
  - Having a smaller number of dimensions may make it easier for classifiers to properly weight the dimensions for the task.
  - Dense models may do better at capturing higher order co-occurrence.

25

Vector Semantics

Embeddings inspired by neural language models: skip-grams and CBOW

Dan Jurafsky

Prediction-based models: An alternative way to get dense vectors

- **Skip-gram** (Mikolov et al. 2013a) **CBOW** (Mikolov et al. 2013b)
- Learn embeddings as part of the process of word prediction.
- Train a neural network to predict neighboring words
  - Inspired by **neural net language models**.
  - In so doing, learn dense embeddings for the words in the training corpus.
- Advantages:
  - Fast, easy to train (much faster than SVD)
  - Available online in the `word2vec` package
  - Including sets of pretrained embeddings!

27

- CBOW
  - Predict "mat" from "the cat sits on the"
  - Better for smaller corpora
- Skip-gram
  - Prediction goes the other way around

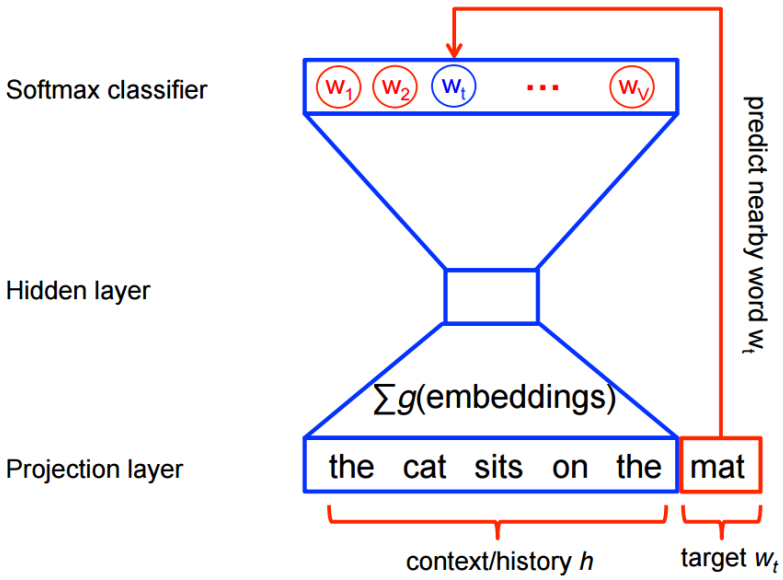
28

Neural Probabilistic Language Models

- Maximize the probability of the word w_t (for "target") given the previous words *h* (for "history") in terms of a *softmax function*

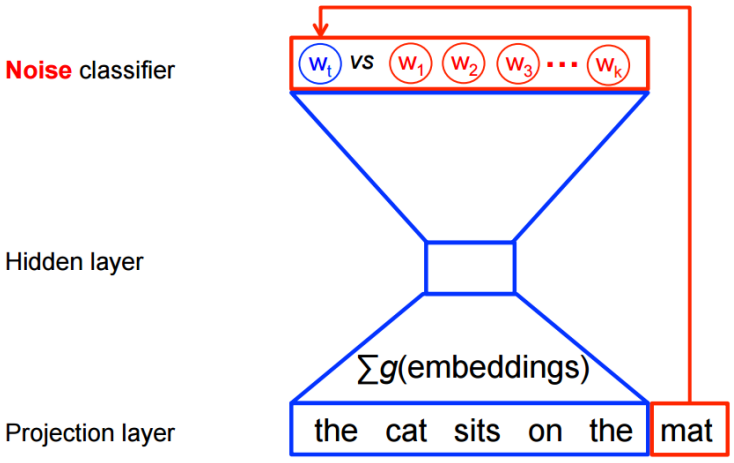
  $P(w_t | h)$ = softmax(score(w_t, h))

- Score: computes word/history compatibility (typically dot-product)
- Softmax: squashes a vector of real values to a vector of real values in the range (0,1) that add up to 1



Very expensive, because need to compute and normalize each probability using the score for all other words in the current context , *at every training step*.

30



The CBOW and skip-gram models are instead trained using a binary classification objective (logistic regression) to discriminate the real target words from imaginary (noise) words , in the same context

The Skip-gram Model: An Example

- Dataset: the quick brown fox jumped over the lazy dog
- Dataset of (context, target) pairs with window size of 1
  - ([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), …
- Since skip-gram tries to predict each context word from its target word, dataset becomes (input, output) pairs
  - (quick, the), (quick, brown), (brown, quick), (brown, fox), ….
- Training step to predict the from quick, need noisy examples (e.g., sheep) to optimize model discriminating real from noise words

32

Dan Jurafsky

Skip-grams

- Predict each neighboring word
  - in a context window of 2*C* words
  - from the current word.
- So for C=2, we are given word w_t and predicting these 4 words:

$$[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$$

33

Dan Jurafsky

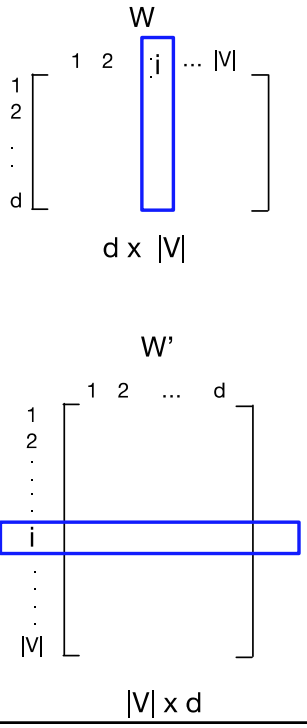
Skip-grams learn 2 embeddings for each w

Input embedding *v,* in the matrix *W*

- Column *i* of the matrix *W* is the 1 × *d* embedding v_i for word *i* in the vocabulary.

Output embedding *v'*, in matrix W'

- Row *i* of the matrix *W'* is a *d* × 1 vector embedding v'_i for word *i* in the vocabulary.



34

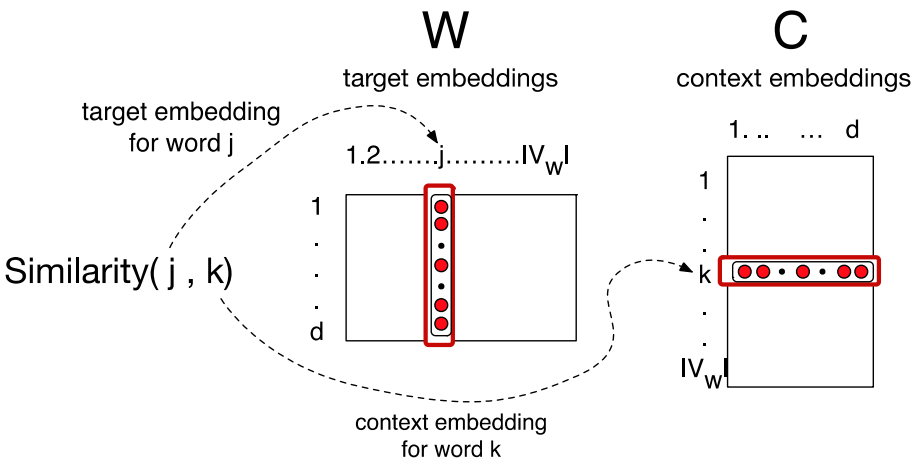
Dan Jurafsky

Setup

- Walking through corpus pointing at word $w(t)$, whose index in the vocabulary is j, so we'll call it w_j ($1 < j < |V|$).
- Let's predict $w(t+1)$, whose index in the vocabulary is k ($1 < k < |V|$). Hence our task is to compute $P(w_k|w_j)$.
  - The heart is computing the dot product between the context vector for w_k and the target vector for w_j

Dan Jurafsky

Intuition: similarity as dot-product between a target vector and context vector



Dan Jurafsky

Similarity is computed from dot product

- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product
- We'll need to normalize to get a probability
  - softmax

37

Dan Jurafsky

Embeddings from W and W'

- Since we have two embeddings, v_j and c_j for each word w_j
- We can either:
  - Just use v_j
  - Sum them
  - Concatenate them to make a double-length embedding

38

Dan Jurafsky

Learning

- Start with some initial embeddings (e.g., random)
- iteratively make the embeddings for a word
  - more like the embeddings of its neighbors
  - less like the embeddings of other words.

39

Dan Jurafsky

Problem with the softamx

- Have to compute over every word in vocab

- Instead: just sample a few of those negative words

40

Dan Jurafsky

Goal in learning

- Make the word like the context words

lemon, a [tablespoon of apricot preserves or] jam
c1 c2 w c3 c4

$$\sigma(x) = \frac{1}{1+e^{x}}$$

- We want this to be high:

$$\sigma(c1 \cdot w) + \sigma(c2 \cdot w) + \sigma(c3 \cdot w) + \sigma(c4 \cdot w)$$

- And not like *k* randomly selected "noise words"

[cement metaphysical dear coaxial apricot attendant whence forever puddle]
n1 n2 n3 n4 n5 n6 n7 n8

- We want this to be low:

$$\sigma(n1 \cdot w) + \sigma(n2 \cdot w) + \ldots + \sigma(n8 \cdot w)$$

41

Dan Jurafsky

Properties of embeddings

- Nearest words to some embeddings (Mikolov et al. 2013)

target:	Redmond	Havel	ninjutsu	graffiti	capitulate
	Redmond Wash.	Vaclav Havel	ninja	spray paint	capitulation
	Redmond Washington	president Vaclav Havel	martial arts	grafitti	capitulated
	Microsoft	Velvet Revolution	swordsmanship	taggers	capitulating

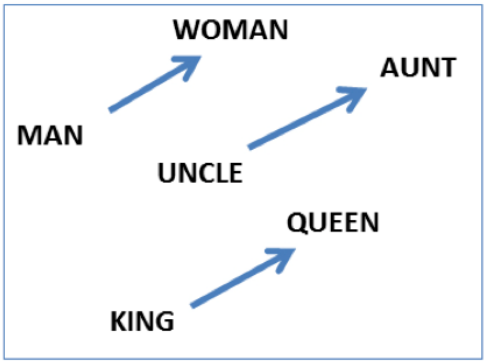
42

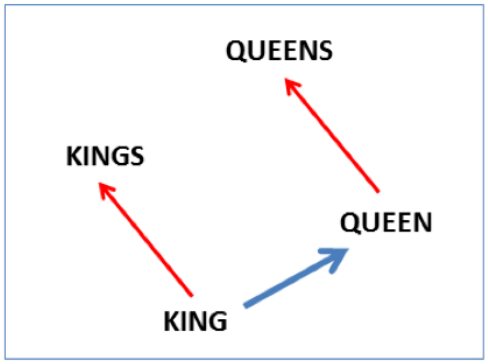
Dan Jurafsky

Embeddings capture relational meaning!

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





43

Vector Semantics

Brown clustering

Dan Jurafsky

Brown clustering

- A clustering algorithm that clusters words based on which words precede or follow them
- These word clusters can be turned into a kind of vector

45

Dan Jurafsky

Brown clustering algorithm

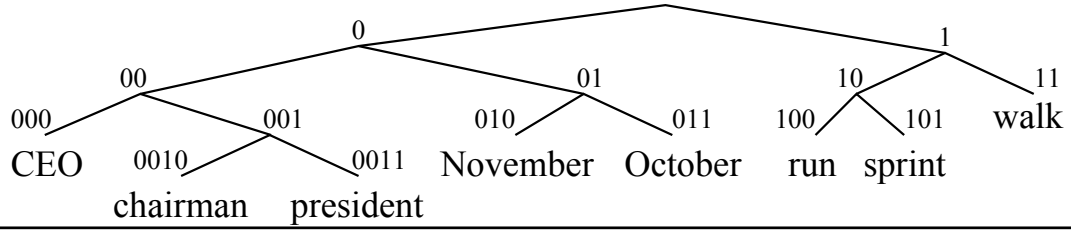
- Each word is initially assigned to its own cluster.
- We now consider consider merging each pair of clusters. Highest quality merge is chosen.
  - Quality = merges two words that have similar probabilities of preceding and following words
  - (More technically quality = smallest decrease in the likelihood of the corpus according to a class-based language model)
- Clustering proceeds until all words are in one big cluster.

46

Dan Jurafsky

Brown Clusters as vectors

- By tracing the order in which clusters are merged, the model builds a binary tree from bottom to top.
- Each word represented by binary string = path from root to leaf
- Each intermediate node is a cluster
- Chairman is 0010, "months" = 01, and verbs = 1



47

Dan Jurafsky

Brown cluster examples

Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays
June March July April January December October November September August
pressure temperature permeability density porosity stress velocity viscosity gravity tension
anyone someone anybody somebody
had hadn't hath would've could've should've must've might've
asking telling wondering instructing informing kidding reminding bothering thanking deposing
mother wife father son husband brother daughter sister boss uncle
great big vast sudden mere sheer gigantic lifelong scant colossal
down backwards ashore sideways southward northward overboard aloft downwards adrift

48

Dan Jurafsky

Class-based language model

- Suppose each word was in some class $c_{i:}$

$$P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$$

$$P(\text{corpus}|C) = \prod_{i-1}^{n} P(c_i|c_{i-1})P(w_i|c_i)$$

49

Summary

- Singular Value Decomposition (SVD)
  - Create lower dimensional embeddings from
    - a full term-term matrix
    - a term-document matrix (e.g., Latent Semantic Analysis)
- Algorithms inspired by neural language models (word prediction)
  - Skip-gram, CBOW
  - Find embeddings that have high dot-product with neighboring words and low dot-product with noise words
- Brown Clustering
  - Word grouping based on relationship with neighboring words, then create vectors

50