Word meaning as classification

CMSC 723 / LING 723 / INST 725

Hal Daumé III [he/him]

17 Sep 2019

(Many slides c/o Dan Jurafsky & James Martin)

Announcements, logistics

- HW2 (ddl Sept 26 before class):
  - Written portion was posted last week
  - You should have received a login for Jupyter Hub server via email
    if not please contact us ASAP (either after class to talk to Amr)
  - Programming portion posted in next 24 hours (hopefully today)!

- Readings for next few weeks are now posted

R 19 Sep	Data collection and annotation	DataInNLP, and AnnCaseStudy	
T 24 Sep	Measurement and validity	Measurement, and MeasurementCaseStudy, Sec "Reliability, Validity, ..."	
R 26 Sep	Crowdsourcing annotations	CrowdsourcingNLP, and AnnMyths	HW2
T 01 Oct	Multilinguality and linguistic variety	TheBenderRule, and Elicitation, Sec 3, and optional: ActiveElicitation	

Tf-idf and PPMI are sparse representations

- tf-idf and PPMI vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)

Alternative: dense vectors

- vectors which are
  - **short** (length 50-1000)
  - **dense** (most elements are non-zero)

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 distinct dimensions
      - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
  - **In practice, they work better**

Dense embeddings you can download!

- **word2vec**
  - https://code.google.com/archive/p/word2vec/
- **Fasttext**
  - http://www.fasttext.cc/
- **Glove**
  - http://nlp.stanford.edu/projects/glove/

Word2vec

- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: **predict** rather than **count**

Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
- Train a classifier on a binary **prediction** task:
  - Is *w* likely to show up near "*apricot*"?
- We don’t actually care about this task
  - But we'll take the learned classifier weights as the word embeddings

Use running text as implicitly supervised training data!

- A word *s* near *apricot*
  - Acts as gold ‘correct answer’ to the question
  - “Is word *w* likely to show up near *apricot*?”
- No need for hand-labeled supervision

Word2Vec: **Skip-Gram** Task

- Word2vec provides a variety of options
  - Let's do "skip-gram with negative sampling" (SGNS)

Skip-gram algorithm

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

Skip-Gram Training Data

- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 target c3 c4

Assume context words are those in +/- 2 word window

Skip-Gram Goal

- Given a tuple (t,c) = target, context
  - (*apricot*, *jam*)
  - (*apricot*, *aardvark*)
- Return probability that c is a real context word:

- $P(+|t,c)$
- $P(-|t,c) = 1 - P(+|t,c)$

How to compute p(+|t,c)?

- Intuition:
  - Words are likely to appear near similar words
  - Model similarity with dot-product!
- Classification model
  - $P(+ \mid t, c) = 1/(1+\exp(-e_t \cdot e_c))$

Skip-Gram Training Data

- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 t c3 c4

- Training data: input/output pairs centering on *apricot*
- Assume a +/- 2 word window

Skip-Gram Training

- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

- For each positive example, we'll create *k* negative examples.
- Using *noise* words
- Any random word that isn't *t*

Skip-Gram Training

- Training sentence:
- ... lemon, a tablespoon of **apricot** jam a pinch ...
- c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples - k=2

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Choosing noise words

- Could pick w according to their unigram frequency P(w)
- More common to chosen then according to $p_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

- α= ¾ works well because it gives rare noise words slightly higher probability
- To show this, imagine two events p(a)=.99 and p(b) = .01:

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

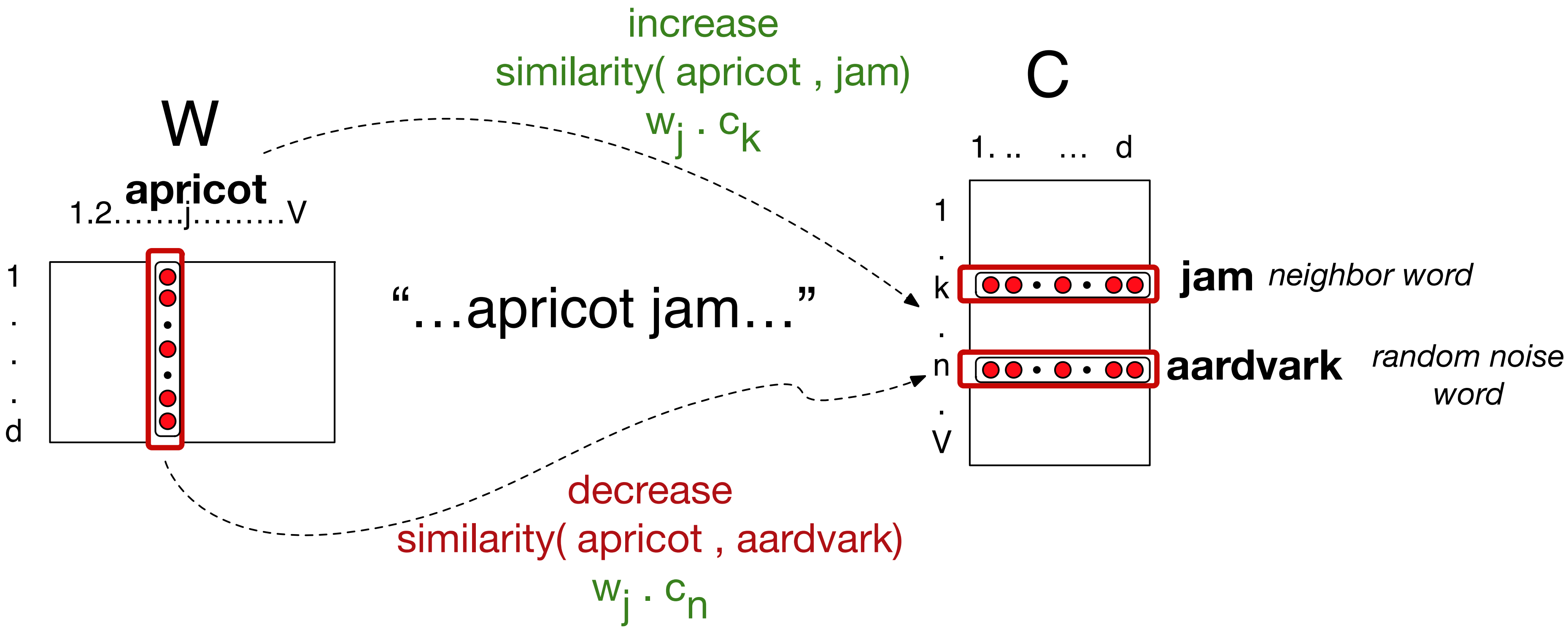
$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

Setup

- Let's represent words as vectors of some length (say 300), randomly initialized.
- So we start with 300 * V random parameters
- Over the entire training set, we'd like to adjust those word vectors such that we
  - Maximize the similarity of the target word, context word pairs (t,c) drawn from the positive data
  - Minimize the similarity of the (t,c) pairs drawn from the negative data.

Learning the classifier

- Iterative process.
- We’ll start with 0 or random weights
- Then adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely
- over the entire training set:


increase
similarity(apricot , jam)
wj . ck
W
apricot
1.2…….j………V
1
.
.
.
d
C
1. ..   … d
1
.
k
.
n
.
V
"…apricot jam…"
jam neighbor word
aardvark
random noise word
decrease
similarity(apricot , aardvark)
wj . cn

Train using gradient descent

- Actually learns two separate embedding matrices W and C
- Can use W and throw away C, or merge them somehow

Summary: How to learn word2vec (skip-gram) embeddings

- Start with V random 300-dimensional vectors as initial embeddings
- Use logistic regression, the second most basic classifier used in machine learning after naïve bayes
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

Evaluating embeddings

- Compare to human scores on word similarity-type tasks:
  - WordSim-353 (Finkelstein et al., 2002)
  - SimLex-999 (Hill et al., 2015)
  - Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
  - TOEFL dataset: *Levied is closest in meaning to: imposed, believed, requested, correlated*

Properties of embeddings

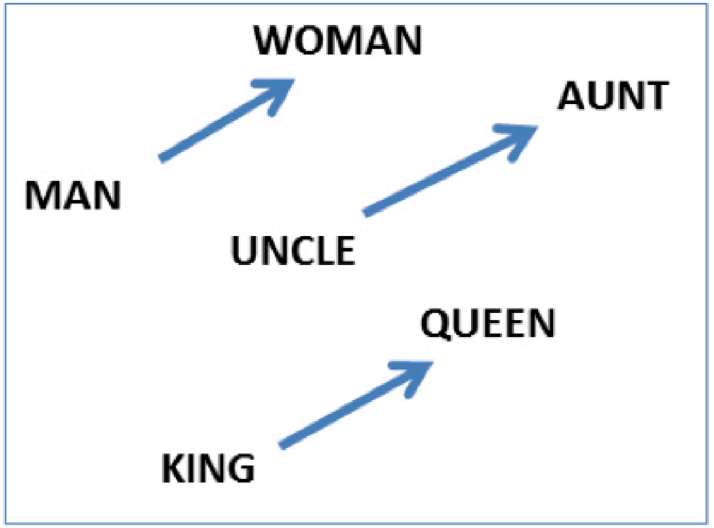
Similarity depends on window size C

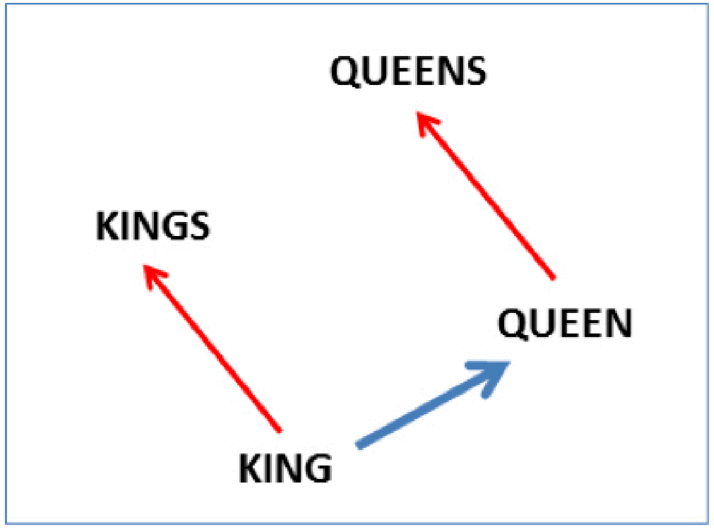
- C = ±2 The nearest words to *Hogwarts:*
  - *Sunnydale*
  - *Evernight*
- C = ±5 The nearest words to *Hogwarts:*
  - *Dumbledore*
  - *Malfoy*
  - *halfblood*

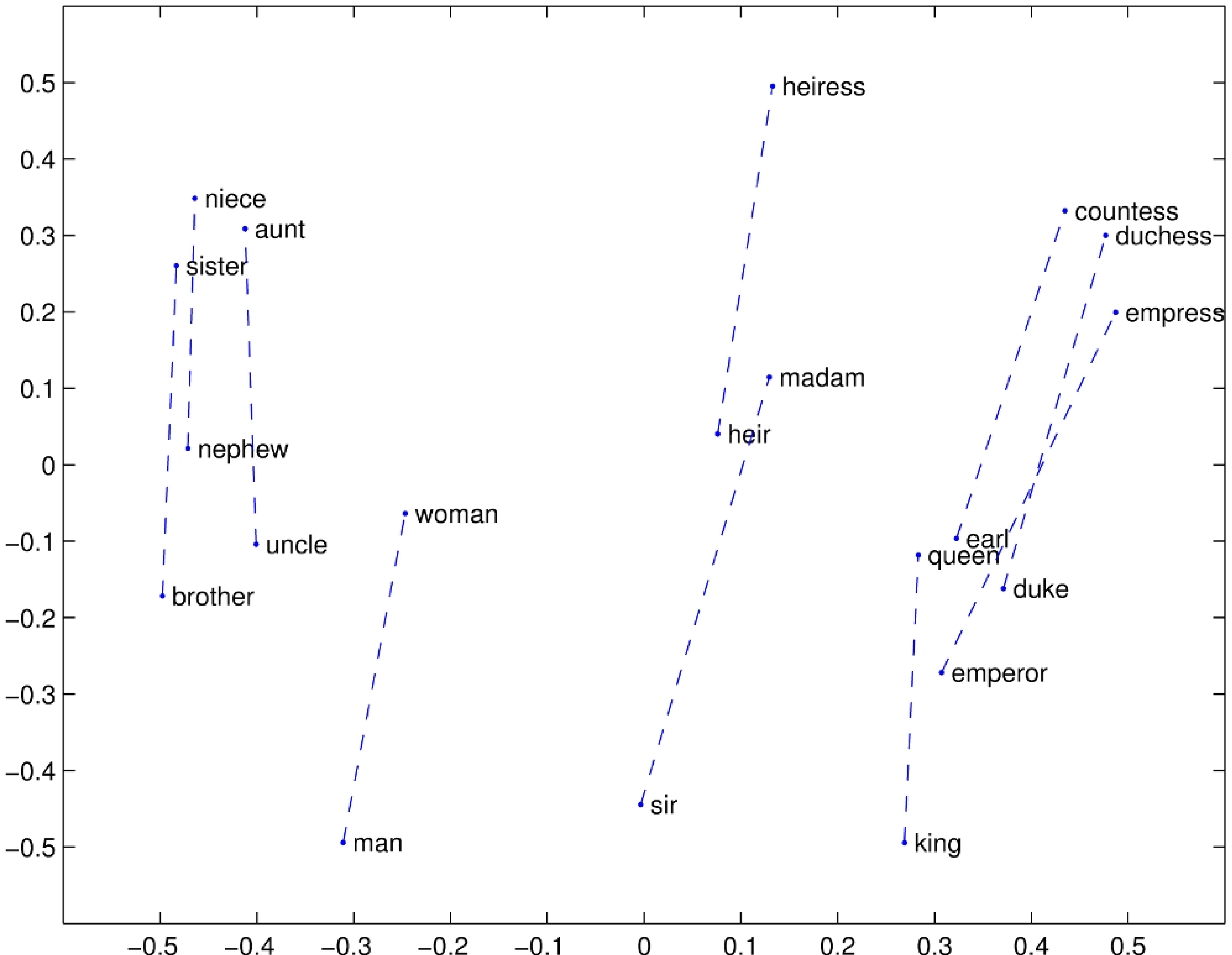
Analogy: Embeddings capture relational meaning!

vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) ≈ vector(‘queen’)

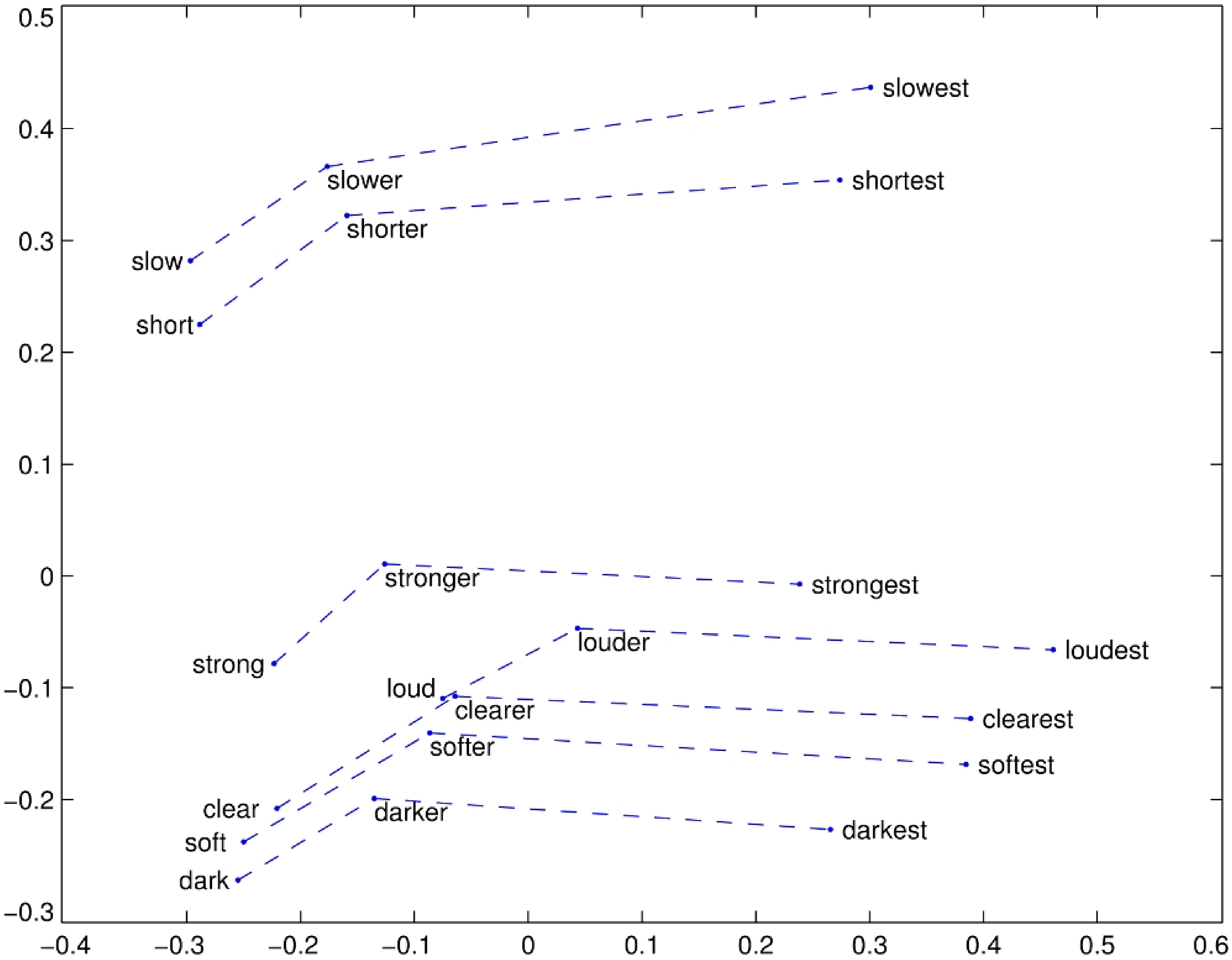
vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) ≈ vector(‘Rome’)





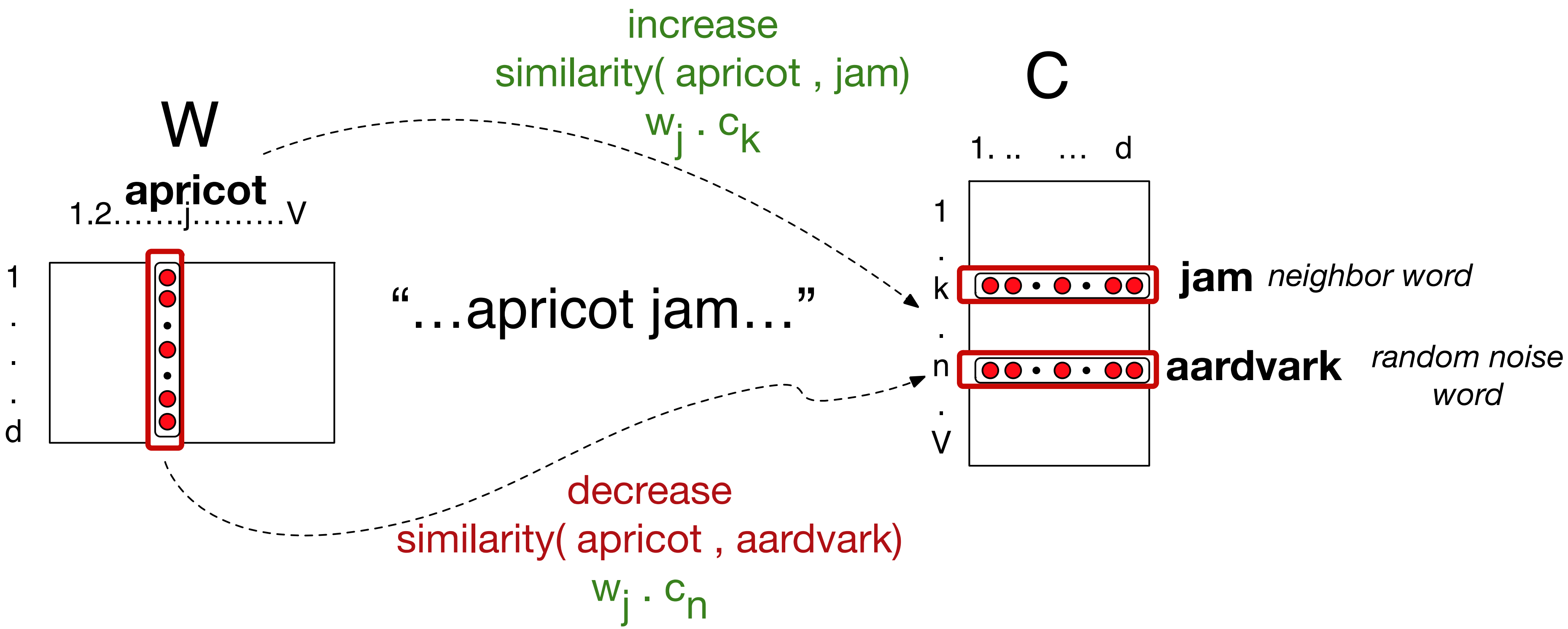


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
heiress
niece
aunt
sister
nephew
uncle
brother
woman
man
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king


slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
soft
darker
darkest
dark
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6

Are embeddings so magical?

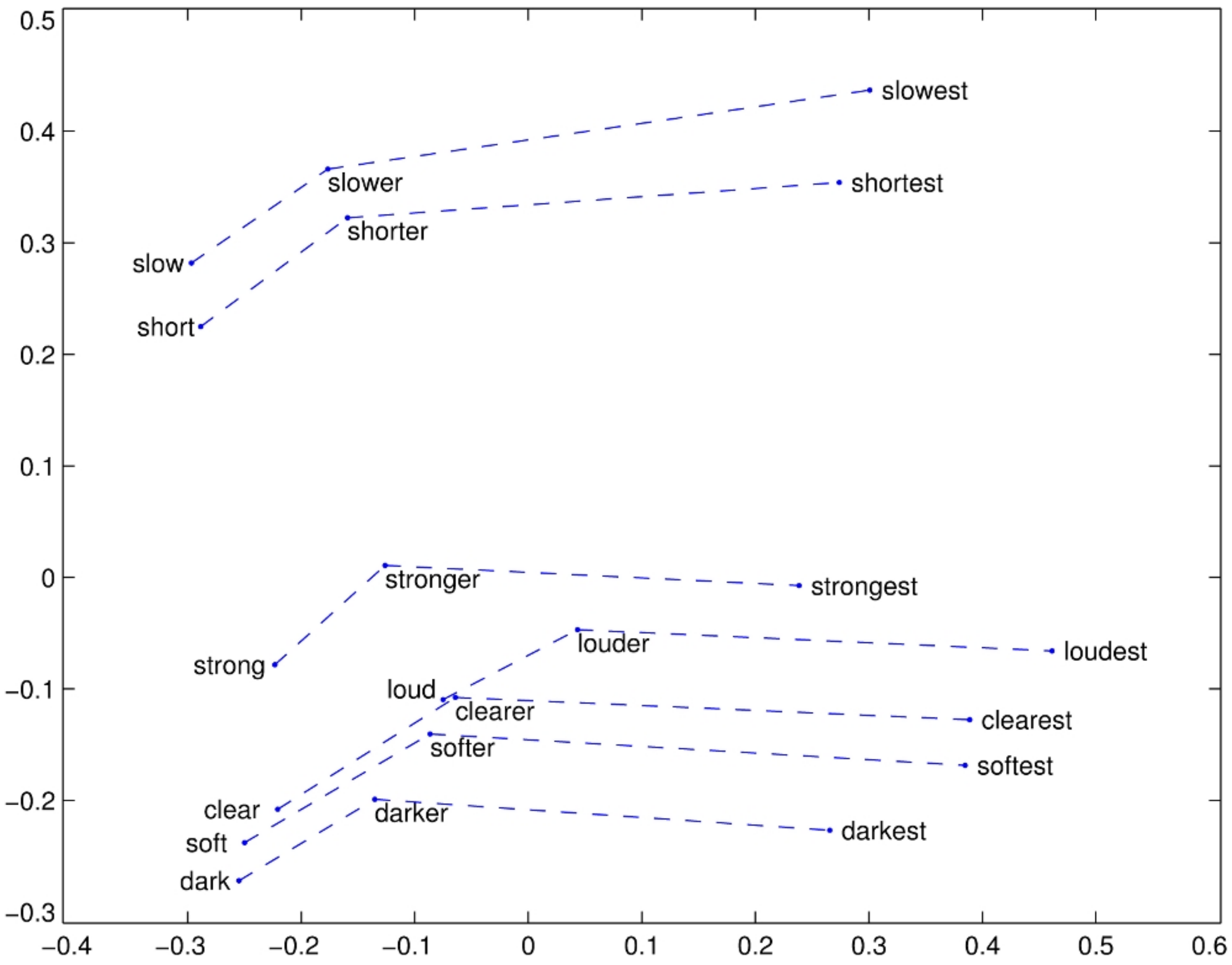
Word2vec as matrix factorization



M = WC, but… what is M? (Levy & Goldberg, NeurIPS 2014)

$$M = \log\left(\frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}\right) - \log k = PMI(w_i, c_j) - \log k$$

How does this help us understand these “semantic” embedding dimensions?



Does this matter empirically?

WS353 (WordSim) [13]		MEN (WordSim) [4]		Mixed Analogies [20]		Synt. Analogies [22]	
Representation	Corr.	Representation	Corr.	Representation	Acc.	Representation	Acc.
SVD (k=5)	0.691	SVD (k=1)	0.735	SPPMI (k=1)	0.655	SGNS (k=15)	0.627
SPPMI (k=15)	0.687	SVD (k=5)	0.734	SPPMI (k=5)	0.644	SGNS (k=5)	0.619
SPPMI (k=5)	0.670	SPPMI (k=5)	0.721	SGNS (k=15)	0.619	SGNS (k=1)	0.59
SGNS (k=15)	0.666	SPPMI (k=15)	0.719	SGNS (k=5)	0.616	SPPMI (k=5)	0.466
SVD (k=15)	0.661	SGNS (k=15)	0.716	SPPMI (k=15)	0.571	SVD (k=1)	0.448
SVD (k=1)	0.652	SGNS (k=5)	0.708	SVD (k=1)	0.567	SPPMI (k=1)	0.445
SGNS (k=5)	0.644	SVD (k=15)	0.694	SGNS (k=1)	0.540	SPPMI (k=15)	0.353
SGNS (k=1)	0.633	SGNS (k=1)	0.690	SVD (k=5)	0.472	SVD (k=5)	0.337
SPPMI (k=1)	0.605	SPPMI (k=1)	0.688	SVD (k=15)	0.341	SVD (k=15)	0.208

Where did things go from here…

- Training objectives (fasttext, glove, etc.)
- Multilingual embeddings (also fasttext, others)
- Lots of “bias in embeddings” stuff
- Trying to tease out what embeddings are actually learning

- Using embeddings as an input representation is a very good default for dense models. But bag of n-grams still often wins

Side-note on biases

- There are >64 papers on biases and “debiasing”
- I have many thoughts on this, we’ll talk about it later

- For now:
  - Embeddings reflect co-occurance in data
  - Co-occurences reflect social structure as reflected in text
  - Text != the real world

Where did things go from here…

- Training objectives (fasttext, glove, etc.)
- Multilingual embeddings (also fasttext, others)
- Lots of “bias in embeddings” stuff
- Trying to tease out what embeddings are actually learning

- Using embeddings as an input representation is a very good default for dense models. But bag of n-grams still often wins

Today

- Embeddings
- Word2vec
- Word2vec isn’t magic