Dan Jurafsky and James Martin

Speech and Language Processing

Chapter 6:
Vector Semantics, Part II

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 (Mikolov et al.)
https://code.google.com/archive/p/word2vec/

Fasttext http://www.fasttext.cc/

Glove (Pennington, Socher, Manning)
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

Brilliant insight: 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
- The idea comes from **neural language modeling**
  - Bengio et al. (2003)
  - Collobert et al. (2011)

Examples

Word-in-context exam questions

1. The twin girls were ____. They look exactly the same.

2. You are too ____! I wish you would be quiet.

3. She got good _____ from her doctor, so she is much better now.

In fact, this is the CBOW model, not the SG model

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

Technically, the SG model predicts the context words given a target word. SG+NG is an approximate scheme as inference on SG is hard.

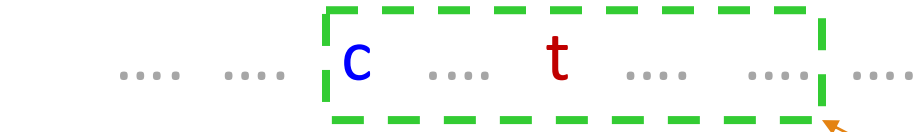
Skip-Gram Training Data

Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 target c3 c4

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



model parameters

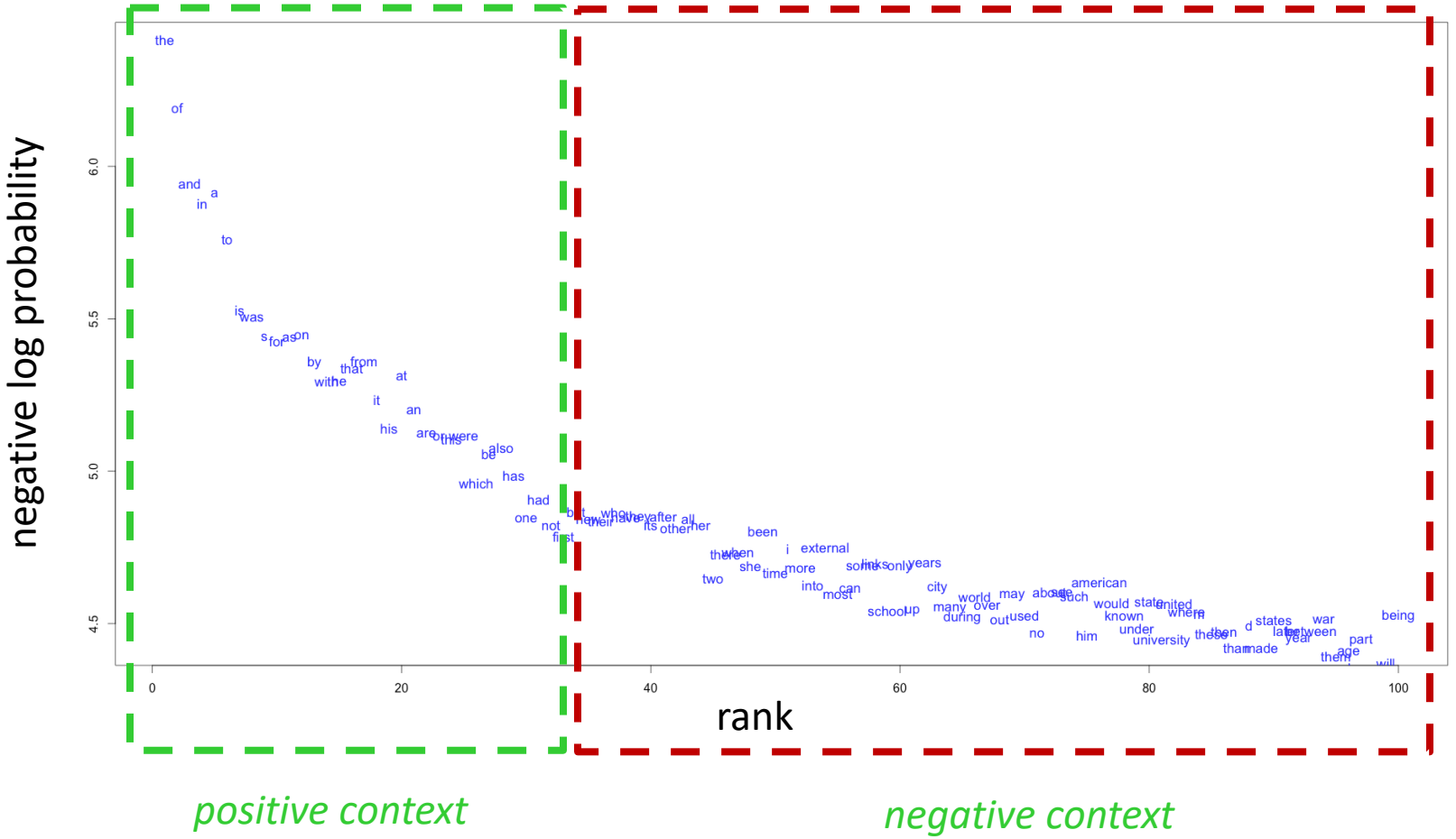
context(t)

$$f(w_i) = P(c = w_i \mid t)$$

$$l(t) = \prod_{w_i \in \text{context(t)}} f(w_i)$$

$$l(\mathbf{T}, \mathbf{C}) = \prod_{t \in \text{training}} l(t)$$

Regression to Binary Classification



Skip-Gram's New Goal

Given a tuple (t,c) = target, context

- (*apricot*, *jam*)
- (*apricot*, *aardvark*)

Decide whether a word (e.g., *jam* or *aardvark*) is a plausible context word for a target word (e.g., *apricot*)

Use logistic regression to model the 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!
- Similarity(t,c) $\propto t \cdot c$

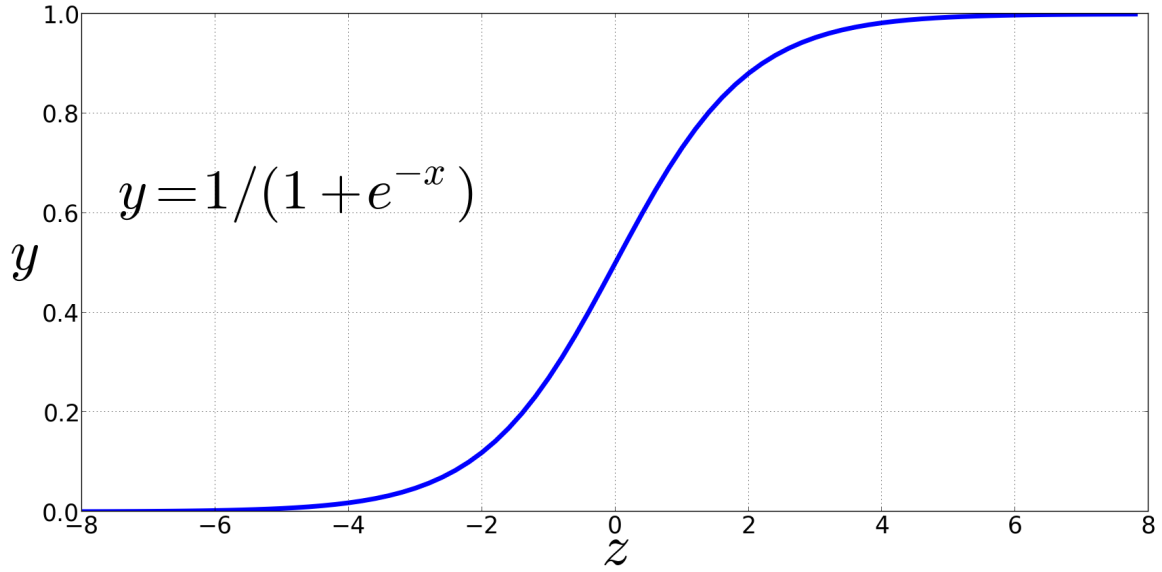
Problem:

- *Dot product is not a probability!*
  - *(Neither is cosine)*

Turning dot product into a probability

The sigmoid lies between 0 and 1:

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



Turning dot product into a probability

$$P(+ \mid t, c) = \frac{1}{1 + e^{-t \cdot c}} = \sigma(t \cdot c)$$

$$P(- \mid t, c) = 1 - P(+ \mid t, c)$$

$$= \frac{e^{-t \cdot c}}{1 + e^{-t \cdot c}}$$

$$= \frac{1}{e^{t \cdot c} + 1} = \sigma(-t \cdot c)$$

For all the context words:

Assume all context words are independent

$$P(+|t, c_{1:k}) = \prod_{i=1}^{\kappa} \frac{1}{1 + e^{-t \cdot c_i}}$$

$$\log P(+|t, c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1 + e^{-t \cdot c_i}}$$

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*

Asssume 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

k=2

positive examples +	
t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -			
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:

Objective Criteria

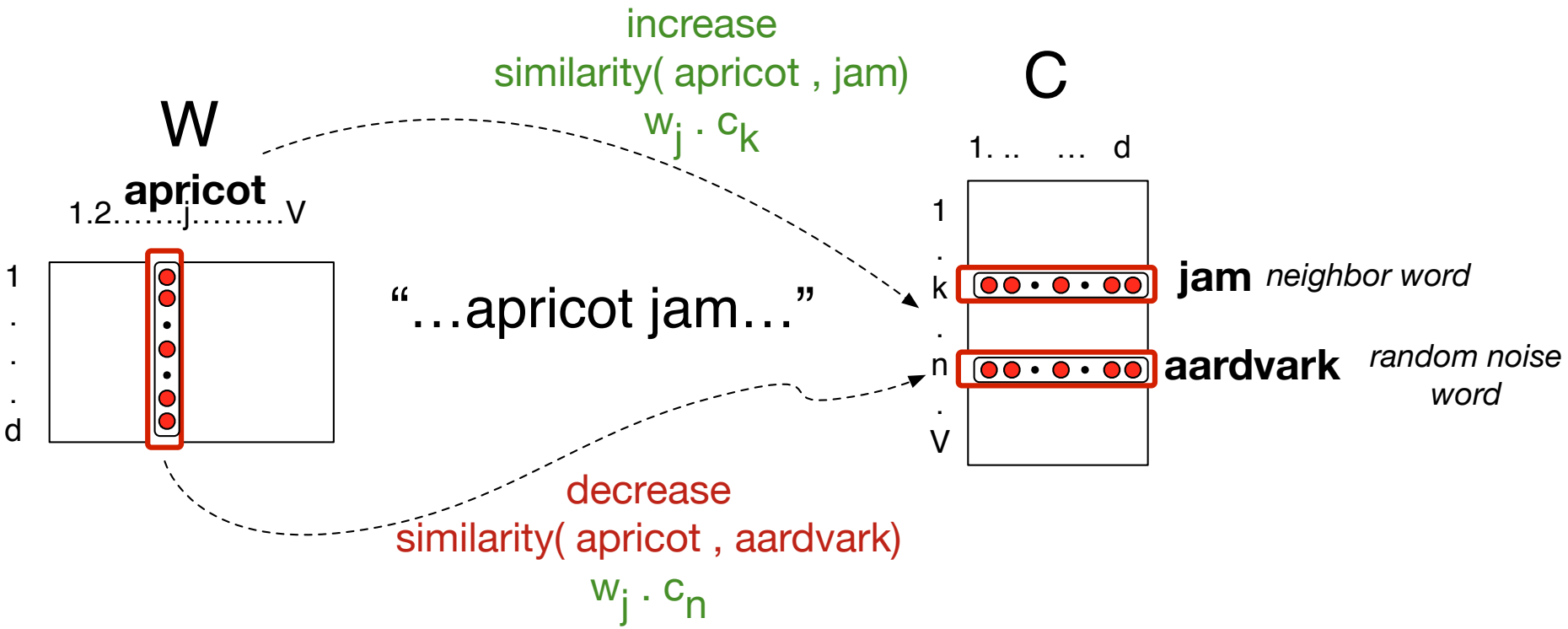
We want to maximize…

$$\sum_{(t,c)\in+} logP(+|t,c) + \sum_{(t,c)\in-} logP(-|t,c)$$

Maximize the + label for the pairs from the positive training data, and the – label for the pairs sample from the negative data.

Focusing on one target word t:

$$
\begin{aligned}
L(\theta) &= \log P(+|t,c) + \sum_{i=1} \log P(-|t,n_i) \\
&= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t) \\
&= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}
\end{aligned}
$$


increase
similarity(apricot , jam)
$w_j \cdot c_k$
W
apricot
1.2.....j.....V
1
.
.
.
d
"…apricot jam…"
C
1. ..
…
d
1
.
k
.
n
.
V
jam
neighbor word
aardvark
random noise word
decrease
similarity(apricot , aardvark)
$w_j \cdot c_n$

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,

- 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*

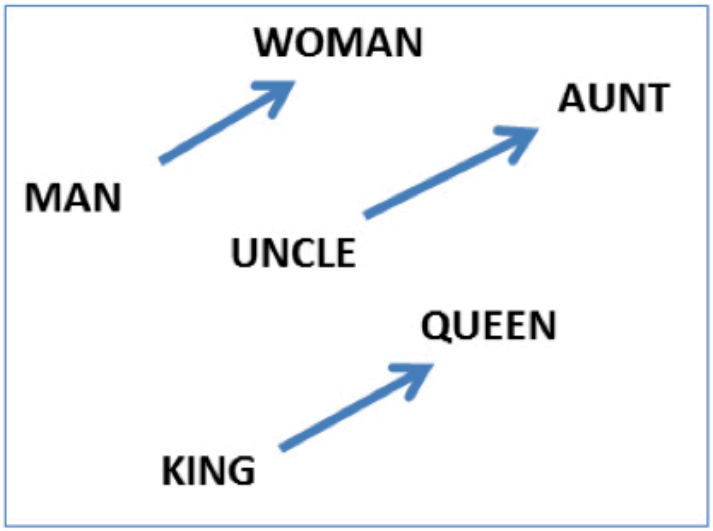
C = ±5 The nearest words to *Hogwarts:*

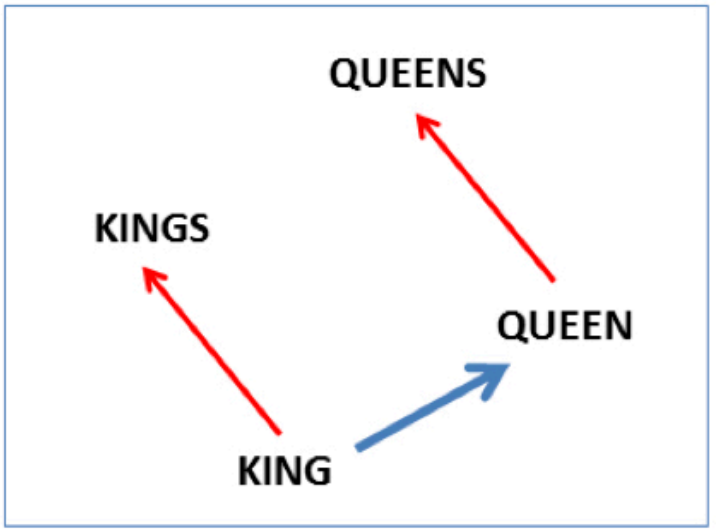
- *Dumbledore*
- *Malfoy*
- *halfblood*

Analogy: Embeddings capture relational meaning!

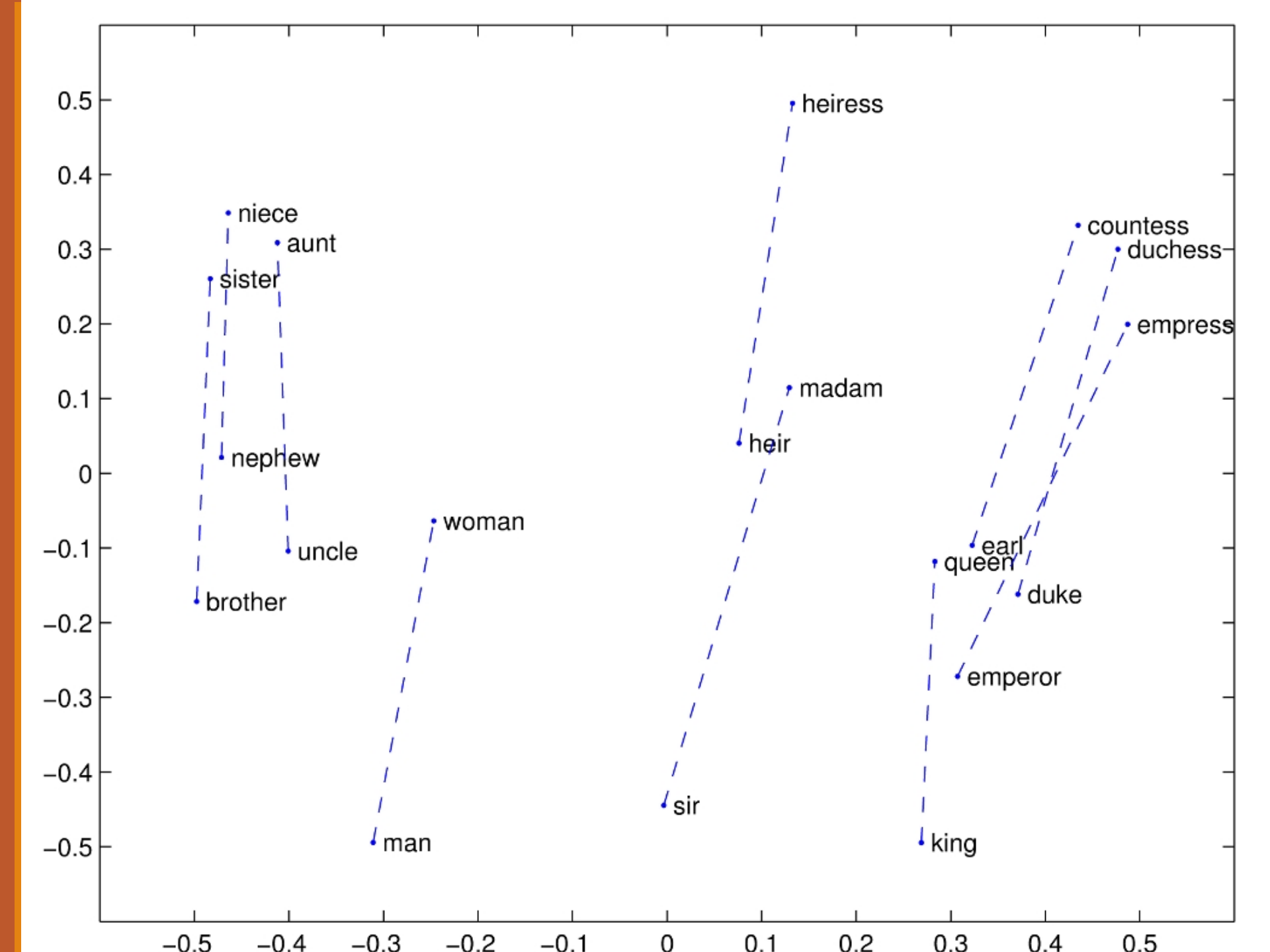
vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

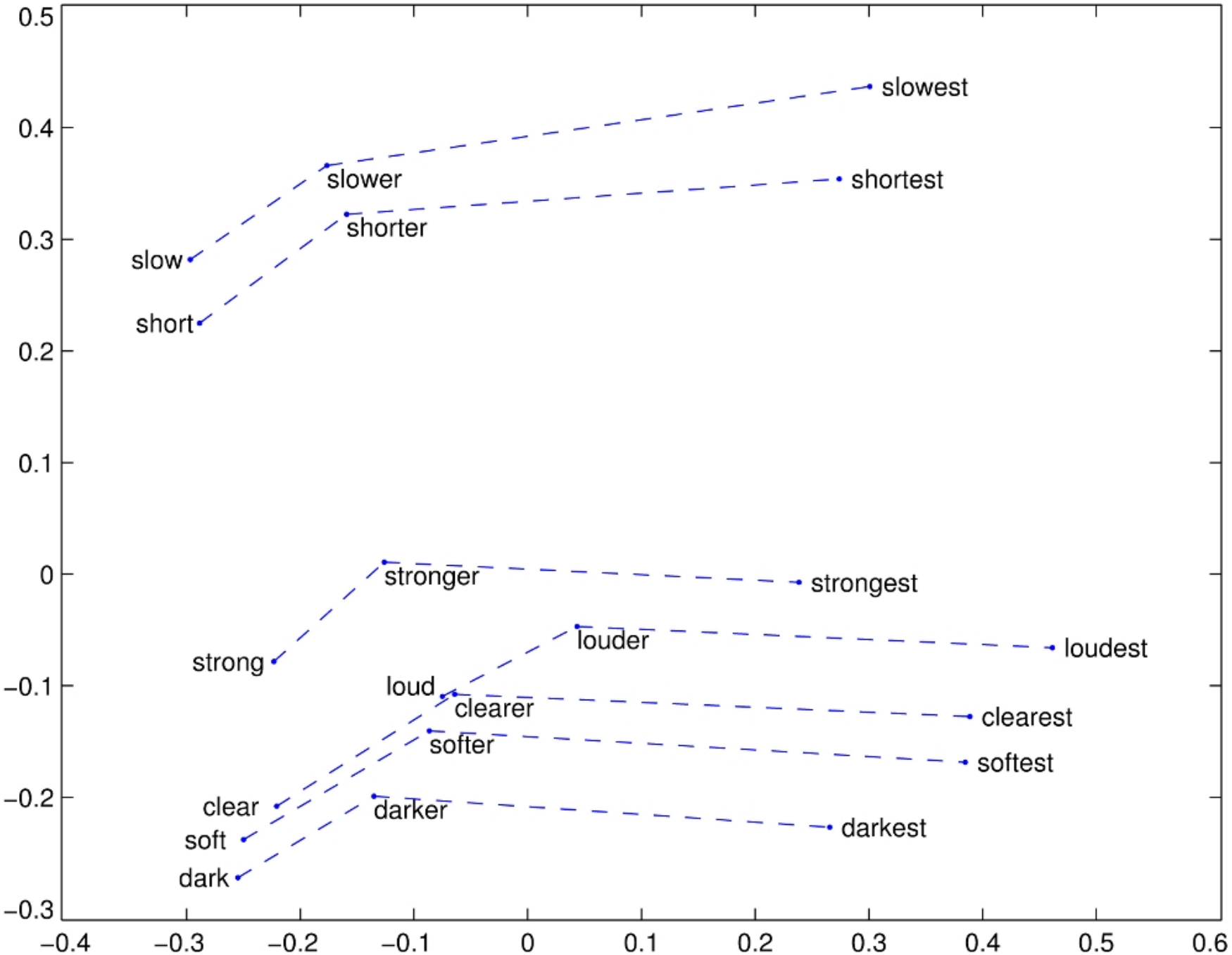
vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





Hinton already came across this "dumb" idea:
https://www.cs.toronto.edu/~hinton/charniak.txt

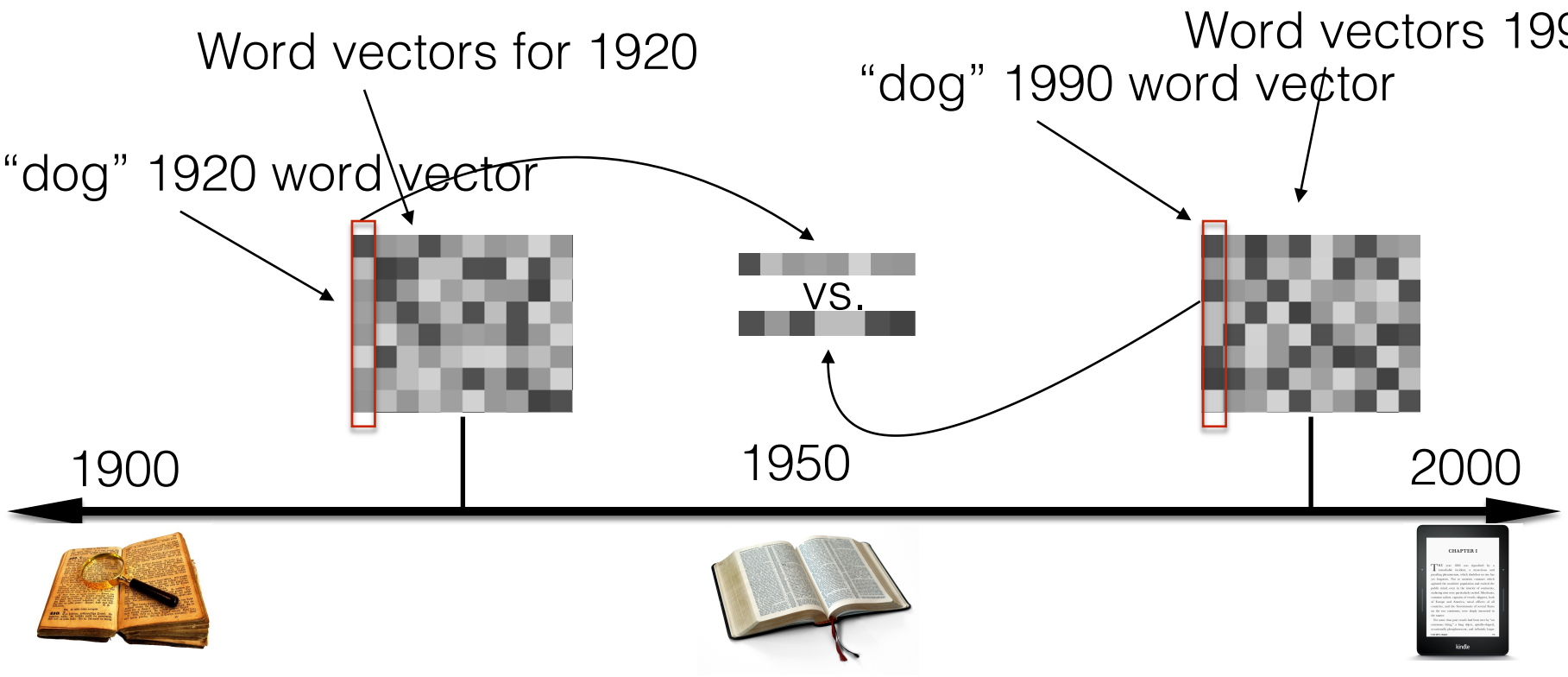

heiress
niece
aunt
sister
countess
duchess
empress
madam
heir
nephew
woman
uncle
earl
queen
duke
brother
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5


slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
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

Embeddings can help study word history!

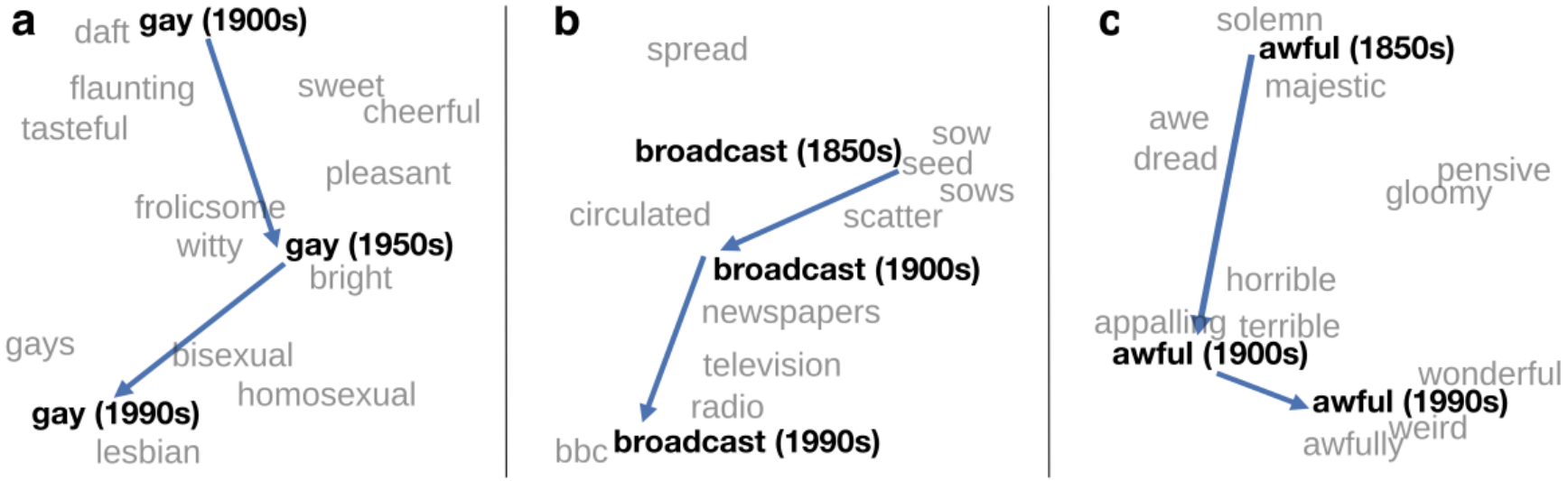
Train embeddings on old books to study changes in word meaning!!

Diachronic word embeddings for studying language change!



Visualizing changes

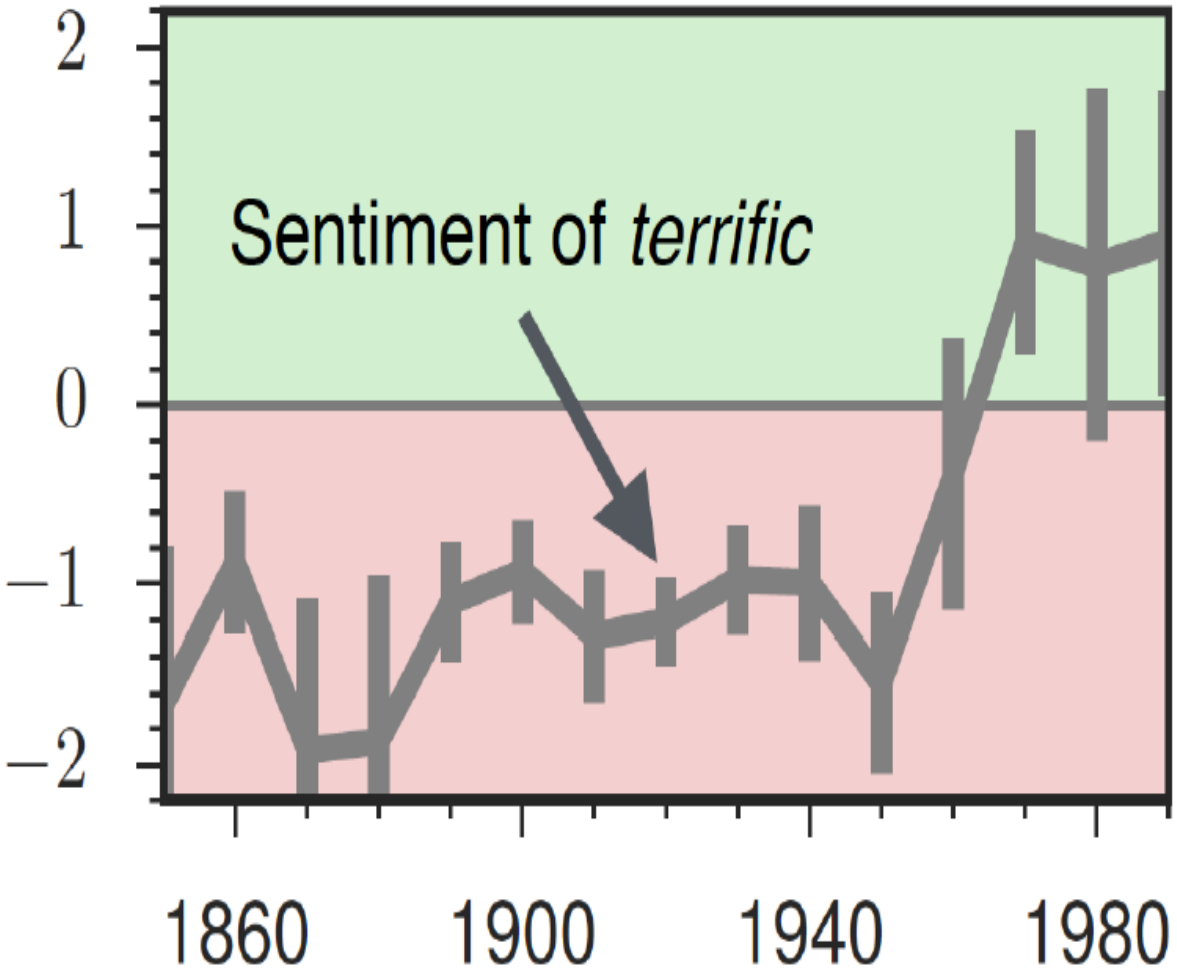
Project 300 dimensions down into 2



~30 million books, 1850-1990, Google Books data

The evolution of sentiment words

Negative words change faster than positive words



Embeddings and bias

Embeddings reflect cultural bias

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

Ask “Paris : France :: Tokyo : x”

- x = Japan

Ask “father : doctor :: mother : x”

- x = nurse

Ask “man : computer programmer :: woman : x”

- x = homemaker

[OPTIONAL]

More Technical Details /1

1. Approximate simplification

$$P(c = w_i \mid t) \quad \Longrightarrow \quad P(c \in C_+ \mid t)$$

$$P(c = w_i \mid t_j) \propto \mathbf{c}_i^\top \mathbf{t}_j \qquad P(+ \mid c_i, t_j) \propto \mathbf{c}_i^\top \mathbf{t}_j$$

learns a proper generative model (LM)

learns an effective discriminative model

2. We need to learn two embeddings for each word ($\mathbf{w}_i$): $\mathbf{c}_i$ and $\mathbf{t}_i$;
   - only $\mathbf{t}_i$ is used in word2vec

3. Most articles on word embeddings are non-rigorous (including this set of slides). For a rigorous treatment, see

[OPTIONAL]

More Technical Details /2

3. Most articles on word embeddings are non-rigorous (including this set of slides). For a rigorous treatment, see:
   - https://ruder.io/word-embeddings-1/index.html (also Parts 2 & 3)
   - word2vec Explained- Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method + word2vec Parameter Learning Explained

4. Discourse model
   - Random Walks on Context Spaces: Towards an Explanation of the Mysteries of Semantic Word Embeddings

5. Individual dimensions for (word) embeddings have little meaning

Conclusion

Concepts or word senses

- Have a complex many-to-many association with **words** (homonymy, multiple senses)
- Have relations with each other
  - Synonymy, Antonymy, Superordinate
- But are hard to define formally (necessary & sufficient conditions)

Embeddings = vector models of meaning

- More fine-grained than just a string or index
- Especially good at modeling similarity/analogy
  - Just download them and use cosines!!
- Can use sparse models (tf-idf) or dense models (word2vec, GLoVE)
- Useful in practice but know they encode cultural stereotypes