COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

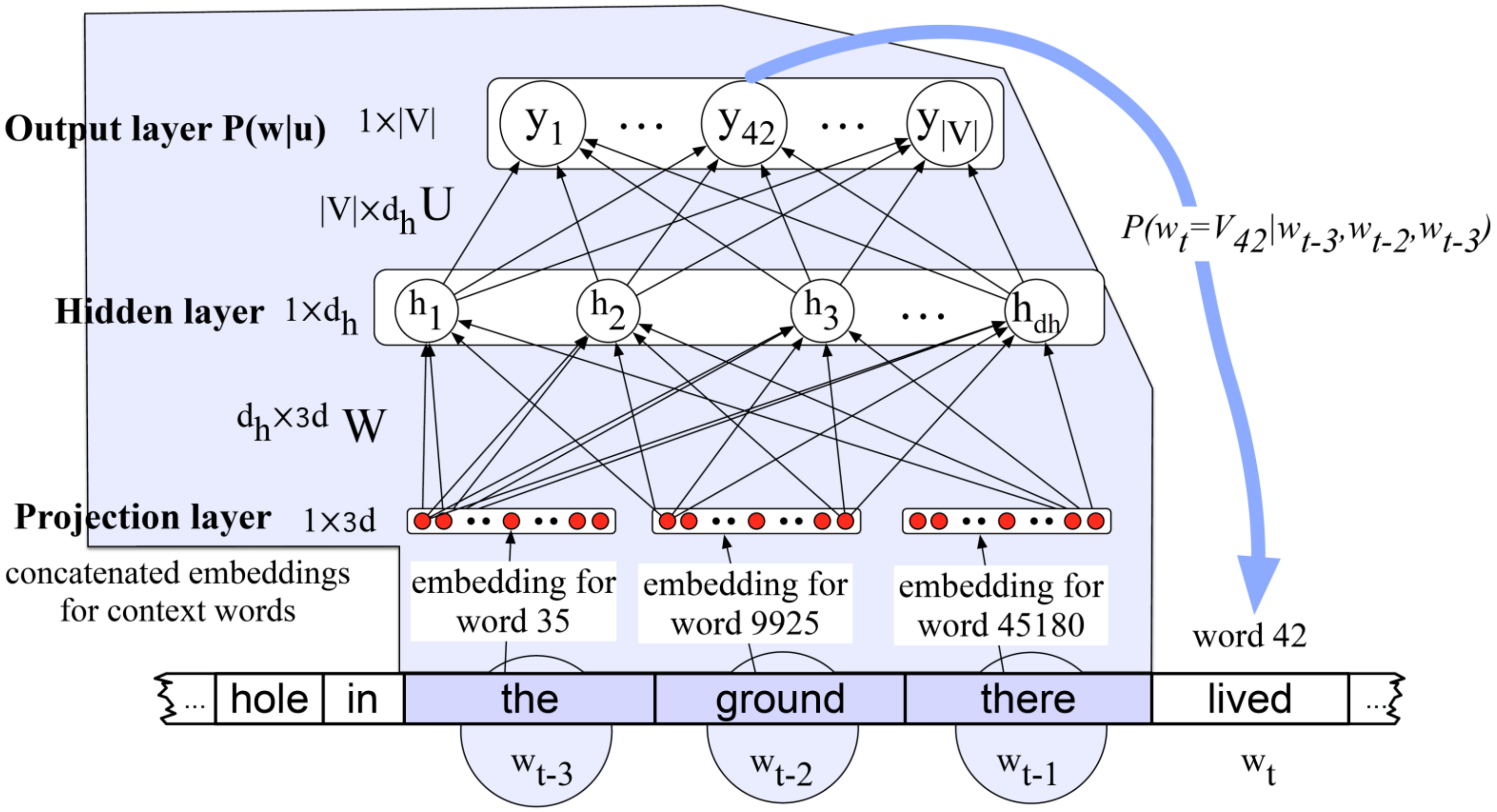
Dense Word Embeddings

CMSC 470

Marine Carpuat

Slides credit: Jurasky & Martin

How to generate vector embeddings?
One approach: feedforward neural language models



Training a neural language model just to get word embeddings is expensive!

Is there a faster/cheaper way to get word embeddings if we don't need the language model?

Roadmap

- Dense vs. sparse word embeddings

- Generating word embeddings with Word2vec
  - Skip-gram model
  - Training

- Evaluating word embeddings
  - Word similarity
  - Word relations
  - Analysis of biases

Word embedding methods we've seen so far yield 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)

Why short dense vectors?

- Short vectors may be easier to use as **features** in machine learning (fewer 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
- Key idea: **predict** rather than **count**

Word2vec

Approach:

- 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"*?

Note: we don’t actually care about this task!

But we'll take the learned classifier weights as the word embeddings

Insight: running text provides 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)

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

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

- Training sentence:

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

c1 c2 target c3 c4

Skip-Gram Model

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

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

Return probability that c is a real context word:

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

We model probability of positive/negative examples using a logistic regression inspired model

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

Dot product between vector representation of t and vector represention of c
Motivation: words are likely to appear near similar words

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

Skip-gram model for all context words:

Assumption: 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

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
  - imagine two events p(a)=.99 and p(b) = .01: $$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$ $$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

Skip-gram: training set-up

- Let's represent words as vectors of some length (say 300), randomly initialized.
- So we start with 300 * V random parameters and use gradient descent to update these parameters
- We need to define a loss function / training objective

Skip-gram: training algorithm

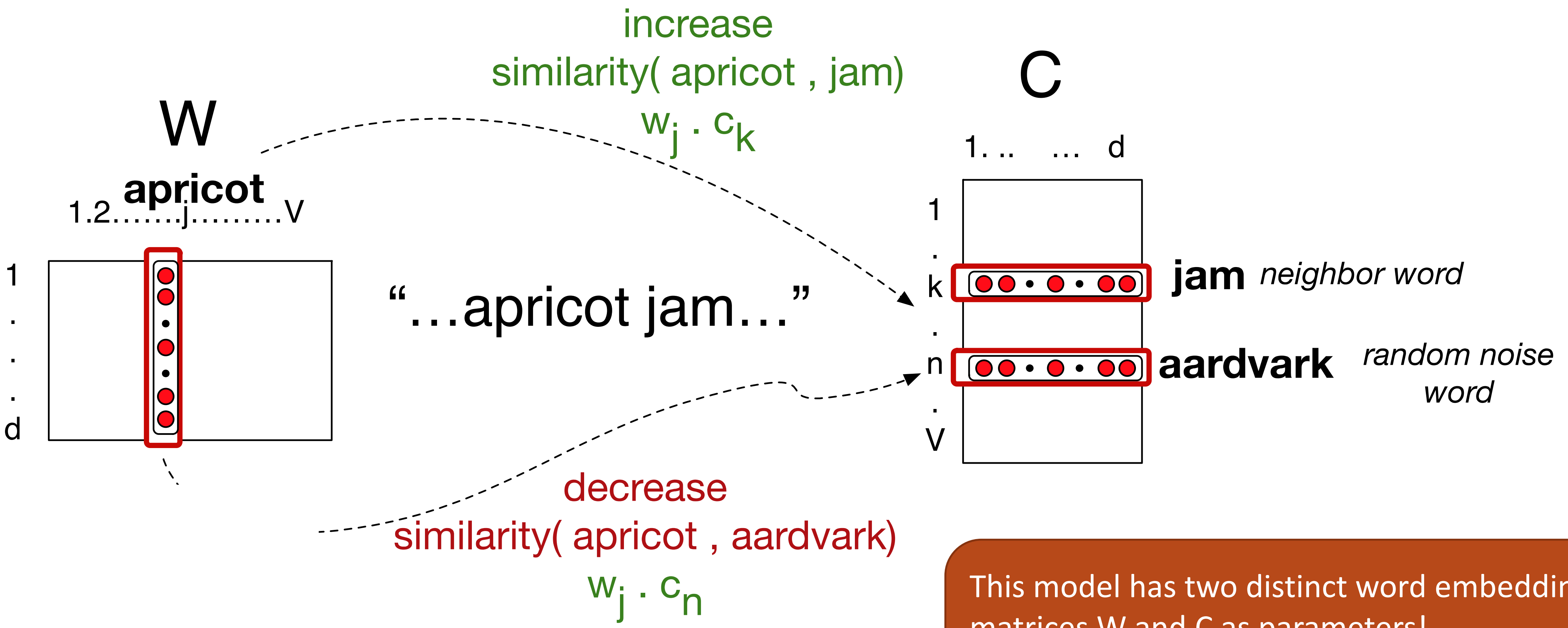
- Objective: we want to maximize

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

- Intuition: over the entire training set, we'd like to adjust word vector parameters such that
  - the similarity of the positive target word, context word pairs (t,c) is maximized
  - the similarity of the negative (t,c) pairs is minimized

- Optimization algorithm: stochastic gradient descent
  - Iteratively updating t parameters, and c parameters

Skip-gram illustrated



This model has two distinct word embedding matrices W and C as parameters!
We can use W and throw away C, or merge them (by addition or concatenation)

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

- Start with V random d-dimensional vectors as initial embeddings
- Take a corpus and take pairs of words that co-occur within window L as positive examples
- Construct negative examples
- Train a logistic regression classifier to distinguish positive from negative examples
- Throw away the classifier and keep the embeddings!

Evaluating embeddings

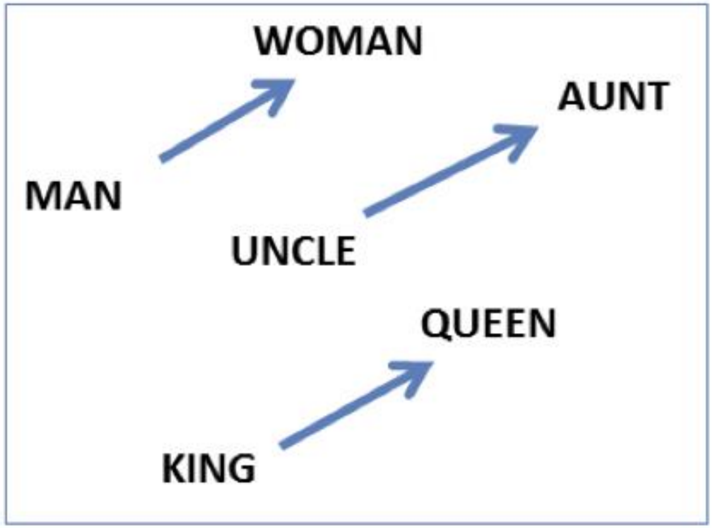
- We can use the same evaluations as for other distributional semantic models (see lecture 2)

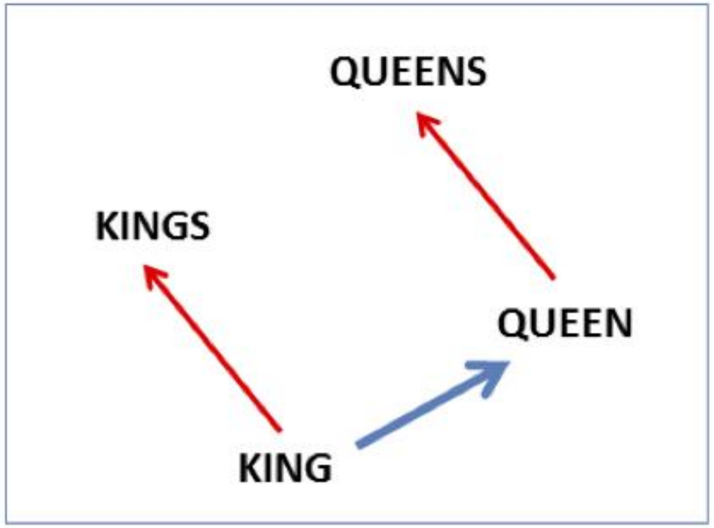
- Compare to human scores on word similarity-type tasks:
  - WordSim-353 (Finkelstein et al., 2002)
  - SimLex-999 (Hill et al., 2015)
  - Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
  - TOEFL dataset: *Levied is closest in meaning to: imposed, believed, requested, correlated*

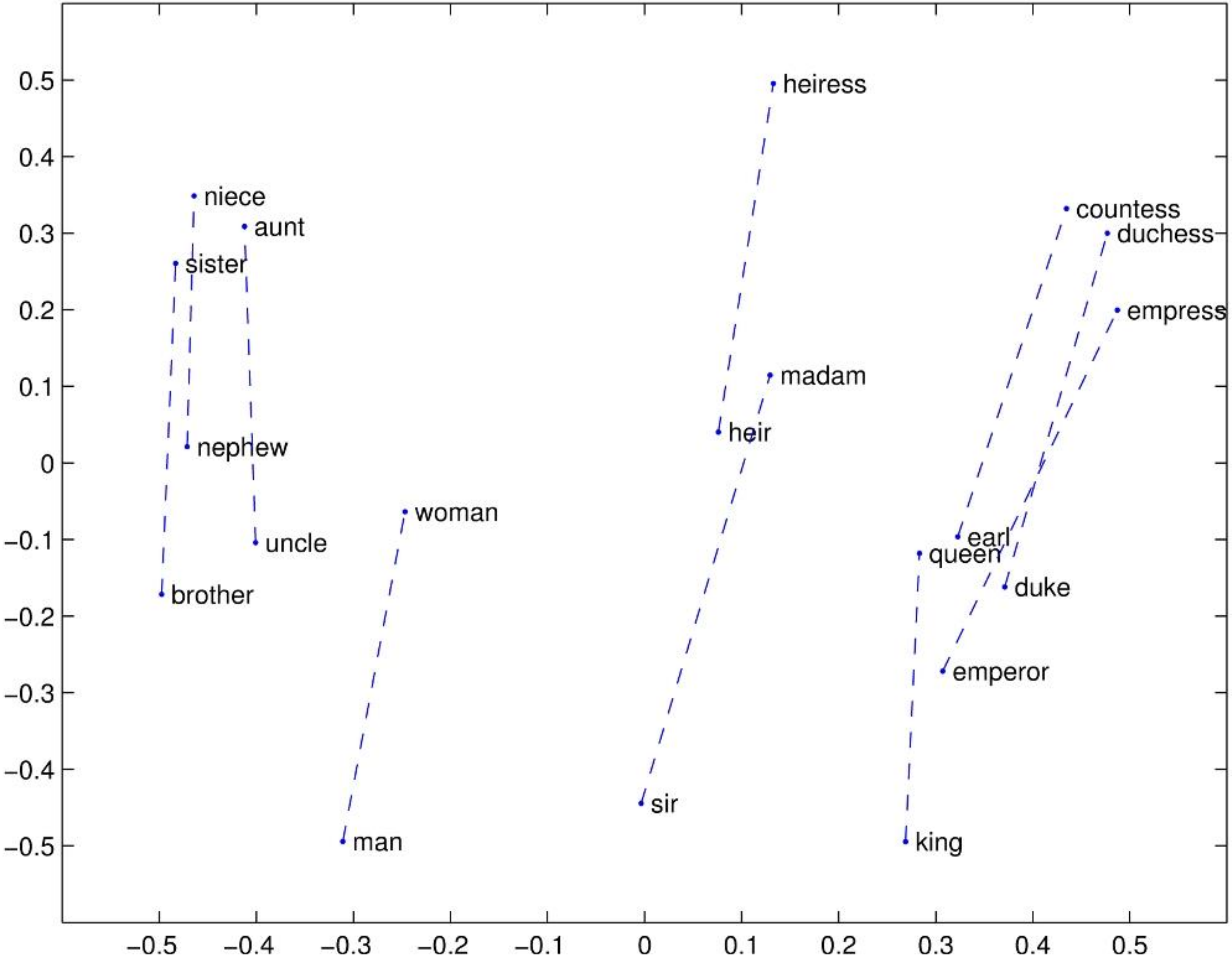
Analogy: geometry of embedding space can capture relational meaning

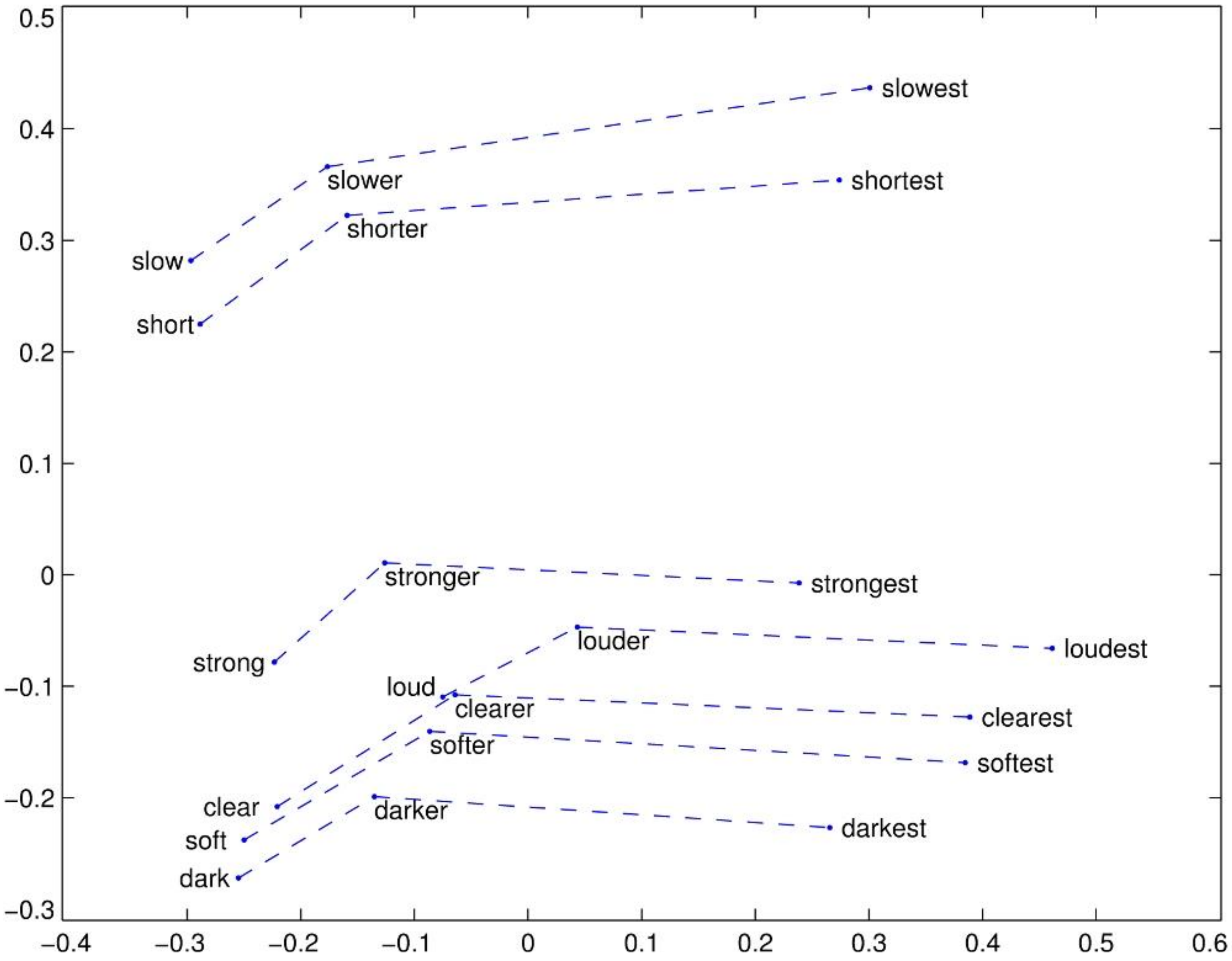
vector(*‘king’*) - vector(*‘man’*) + vector(*‘woman’*) ≈ vector(‘queen’)

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





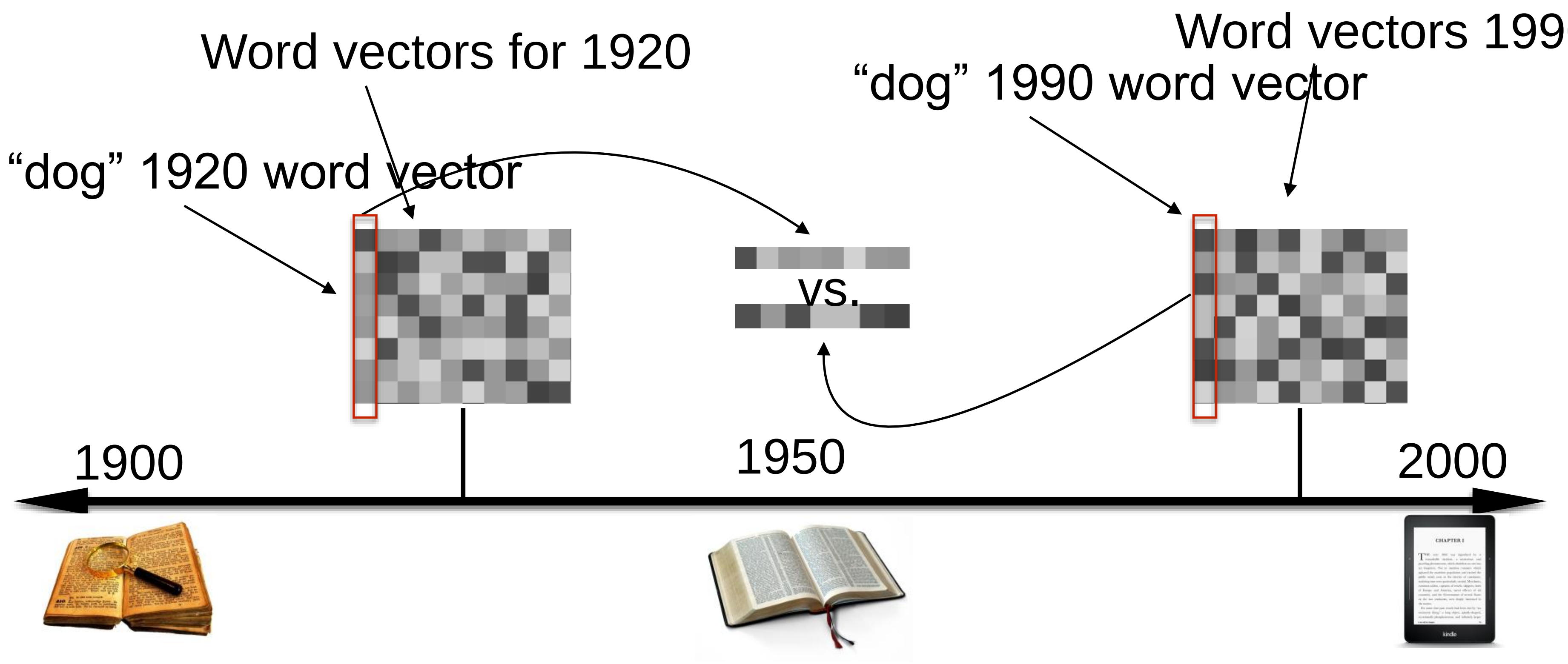

heiress
niece
aunt
sister
countess
duchess
empress
madam
heir
nephew
woman
uncle
earl
queen
brother
duke
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5



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

Word embeddings are a very useful tool

- Can be used as features in classifiers
  - Capture generalizations across word types
- Can be used to analyze language usage patterns in large corpora
  - E.g., to study change in word meaning



Word vectors for 1920
"dog" 1920 word vector
Word vectors 1990
"dog" 1990 word vector
vs.
1900
1950
2000

Yet word embeddings are not perfect models of word meaning

- Limitations include
  - One vector per word (even if the word has multiple senses)
  - Cosine similarity not sufficient to distinguish antonyms from synonyms
  - Embeddings reflect biases and stereotypes implicit in training text

Embeddings reflect human biases and stereotypes

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

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.

Embeddings reflect human biases and stereotypes

- Implicit Association test (Greenwald et al 1998): How associated are
  - concepts (*flowers*, *insects*) & attributes (*pleasantness*, *unpleasantness*)?
  - Studied by measuring timing latencies for categorization.
- Psychological findings on US participants:
  - African-American names are associated with unpleasant words (more than European-American names)
  - Male names associated more with math, female names with arts
  - Old people's names with unpleasant words, young people with pleasant words.
- Caliskan et al. replication with embeddings:
  - African-American names had a higher cosine with unpleasant words
  - European American names had a higher cosine with pleasant words

- Embeddings reflect and replicate all sorts of pernicious biases.

Caliskan, Aylin, Joanna J. Bryson and Arvind Narayanan. 2017. Semantics derived automatically from language corpora contain human-like biases. Science 356:6334, 183-186.

So what can we do about bias?

- Use embeddings as a historical tool to study bias
  - Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

- Do not download and use word embeddings blindly: know what is the underlying model, how they were trained, on what data

- Also: ongoing research on attempting to mitigate bias

Roadmap

- Dense vs. sparse word embeddings
- Generating word embeddings with Word2vec
  - Skip-gram model
  - Training
- Evaluating word embeddings
  - Word similarity
  - Word relations
  - Analysis of biases