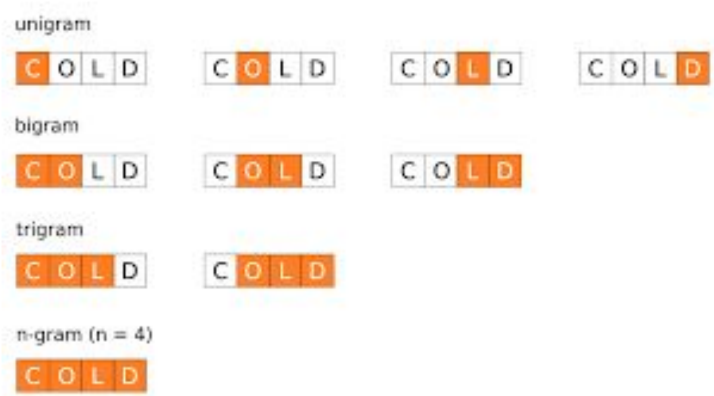
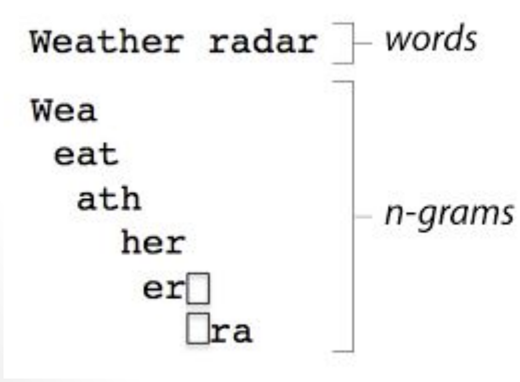
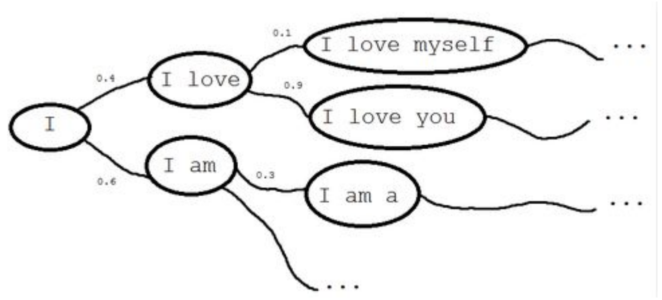
Topic 5 (Pt 3) :
Language Model & Word Embeddings

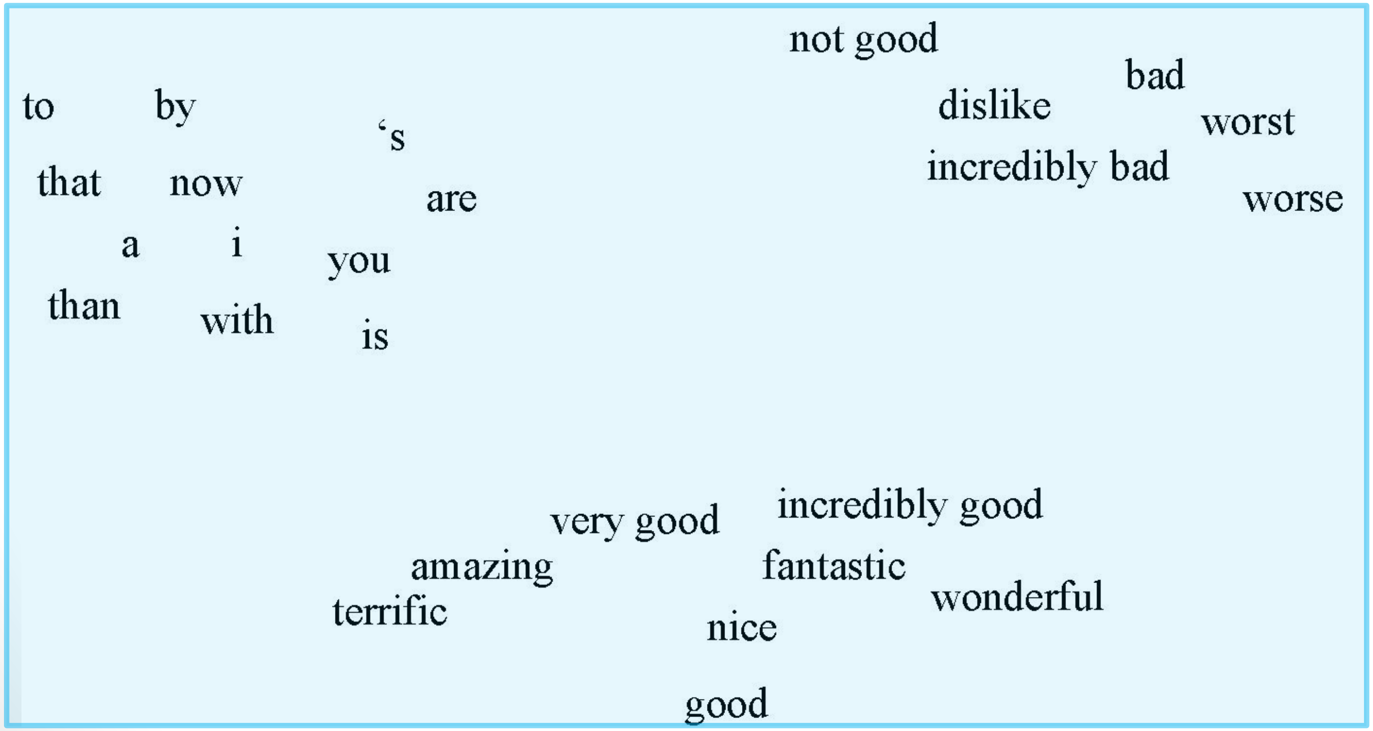






Building a Model of Meaning (Semantics)

- Represent each word as a vector.
- Similar words are nearby in space



Defining a word as a vector

- Known as an "embedding" because it's embedded into a space
- The standard way to represent meaning in NLP
- Fine-grained model of meaning for similarity
  - NLP tasks like sentiment analysis
    - With words, requires **same** word to be in training and test
    - With embeddings: ok if **similar** words occurred!!!
  - Question answering, conversational agents, etc

Why Word Embeddings?

- Image and audio processing systems work with rich, high-dimensional datasets encoded as vectors representing individual raw pixel-intensities for image data
- Natural language processing systems traditionally treat words as discrete atomic symbols
- Encodings for natural language can be arbitrary: provide no useful information on relationship between words/symbols
- Little is learned about possible similarity between words

Why Word Embeddings?

- Vector space models (VSMs) represent (embed) words in a continuous vector space where semantically similar words are mapped to nearby points ('are embedded nearby each other’)
- Words that appear in the same contexts share semantic meaning
- Count-based vs predictive methods

Example of types of embeddings:

- Tf-idf (will be discussed in Topic 6: NLP ML)
  - A common baseline model
  - Sparse vectors
  - Words are represented by a simple function of the counts of nearby words
- Word2vec
  - Dense vectors
  - Representation is created by training a classifier to distinguish nearby and far-away words

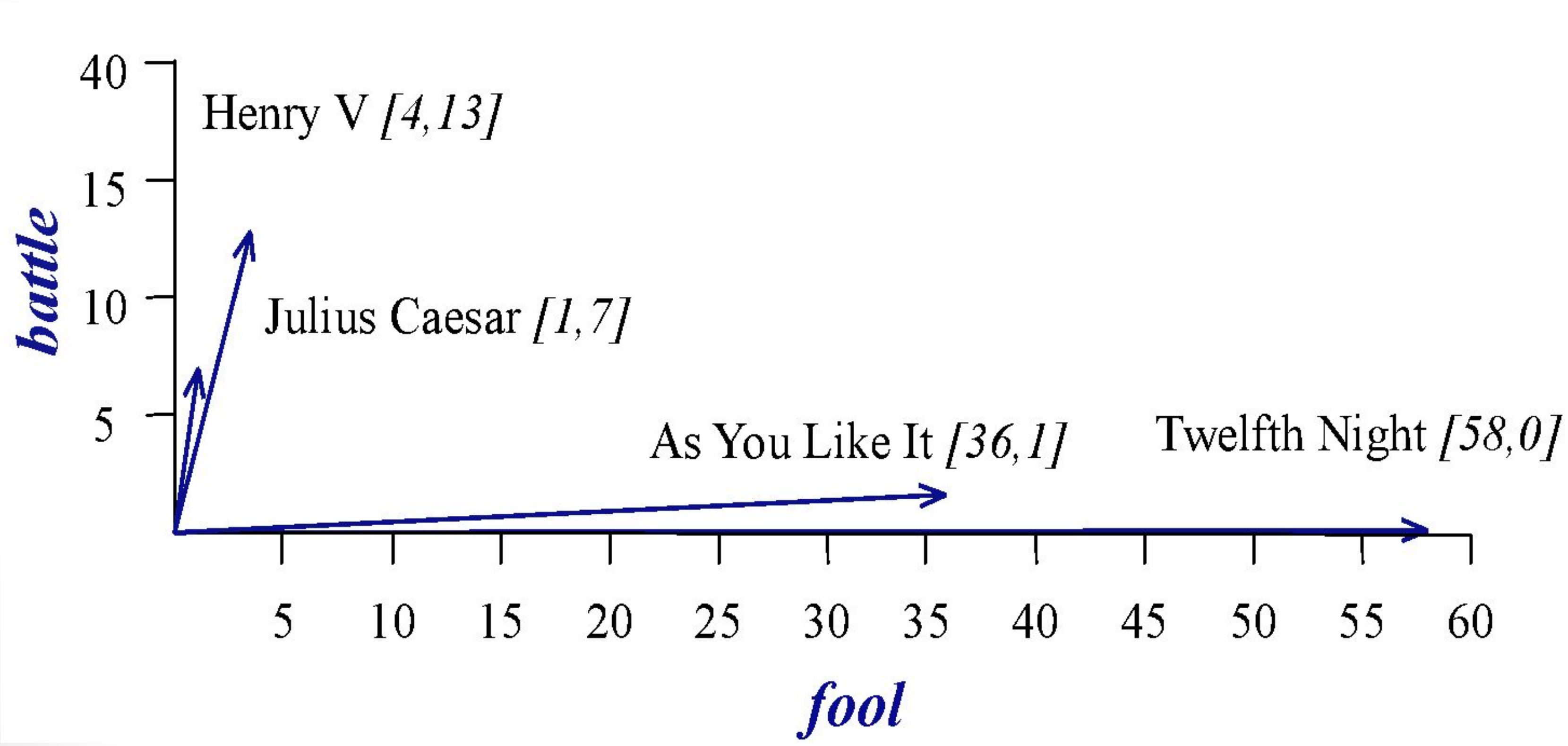
Term-document matrix

Each document is represented by a vector of words

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

frequency/counts

Visualizing document vectors



Vectors are the basis of information retrieval

comedy history

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- Vectors are similar for the two comedies
- Different than the history
- Comedies have more fools and wit and fewer battles.

Words can be vectors too

[redacted]	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

battle is "the kind of word that occurs in Julius Caesar and Henry V"

fool is "the kind of word that occurs in comedies, especially Twelfth Night"

More common: word-word matrix (or "term-context matrix")

- Two **words** are similar in meaning if their context vectors are similar

sugar, a sliced lemon, a tablespoonful of	**apricot**	jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

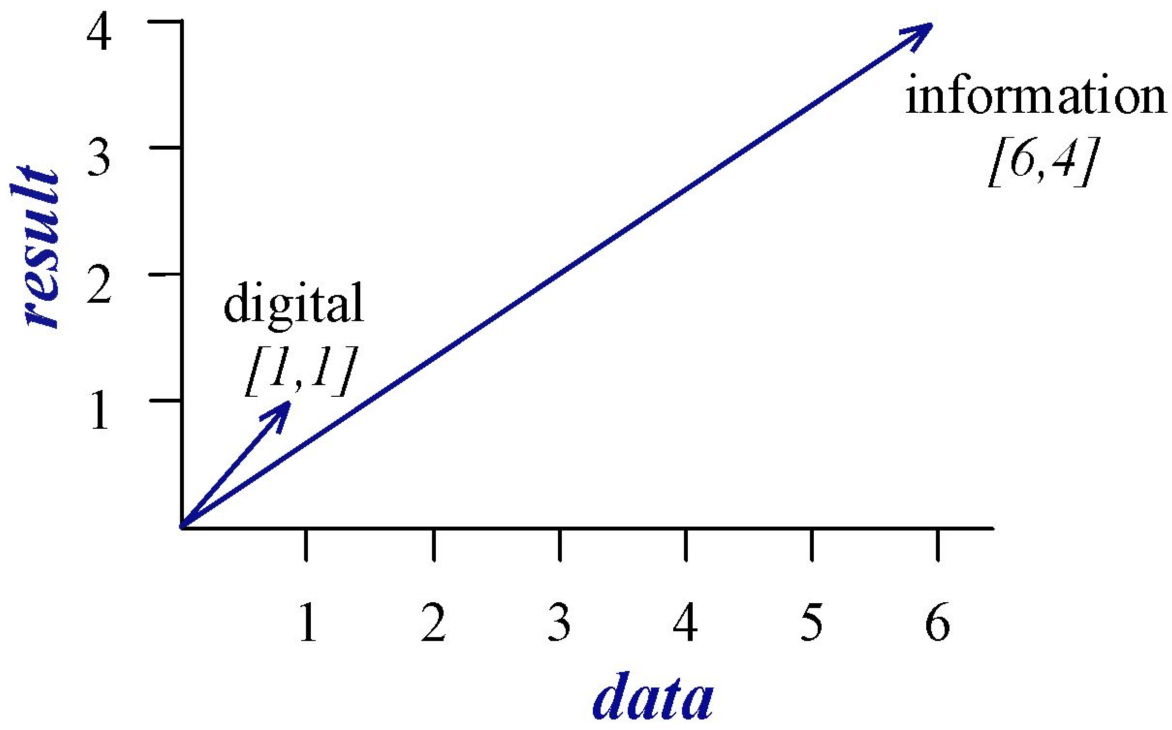
More common: word-word matrix (or "term-context matrix")

- Two **words** are similar in meaning if their context vectors are similar

sugar, a sliced lemon, a tablespoonful of	**apricot**	jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Visualizing document vectors



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)

Word2Vec: **Skip-Gram** Task

- Word2vec provides a variety of options. One example is :
  - "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:

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

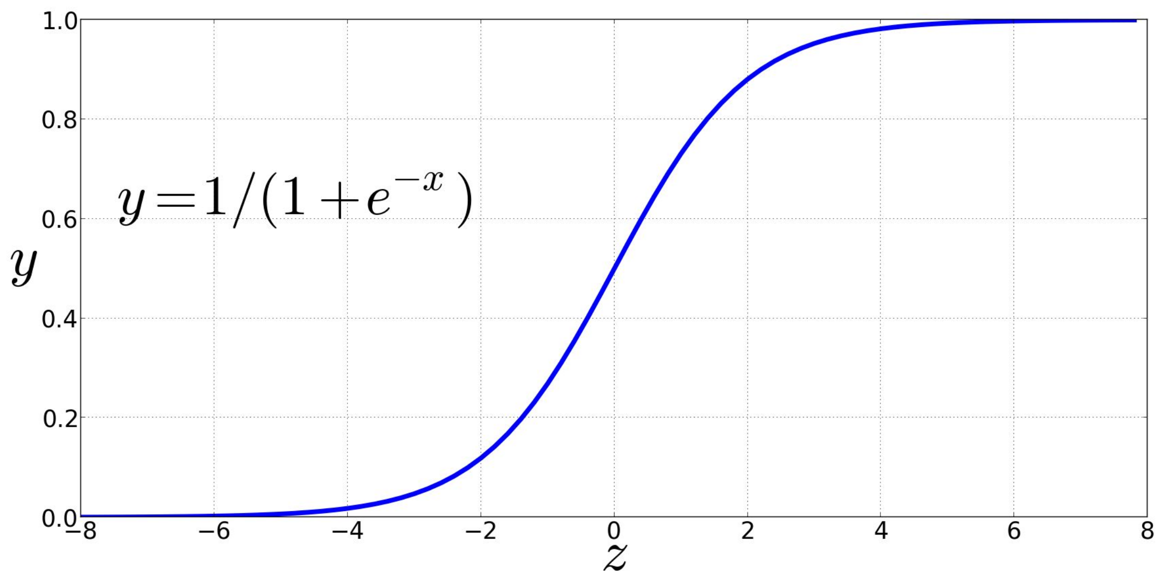
How to compute p(+|t,c)?

- Intuition:
  - Words are likely to appear near similar words
  - Model similarity with dot-product!
  - Similarity(t,c) $\propto$ (proportional to) $t \cdot c$
- *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

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

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

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

For all the context words:

- Assume all context words are independent

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

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

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*
- 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

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 :
  - We maximize the similarity of the target word, context word pairs (t,c) drawn from the positive data
  - We 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

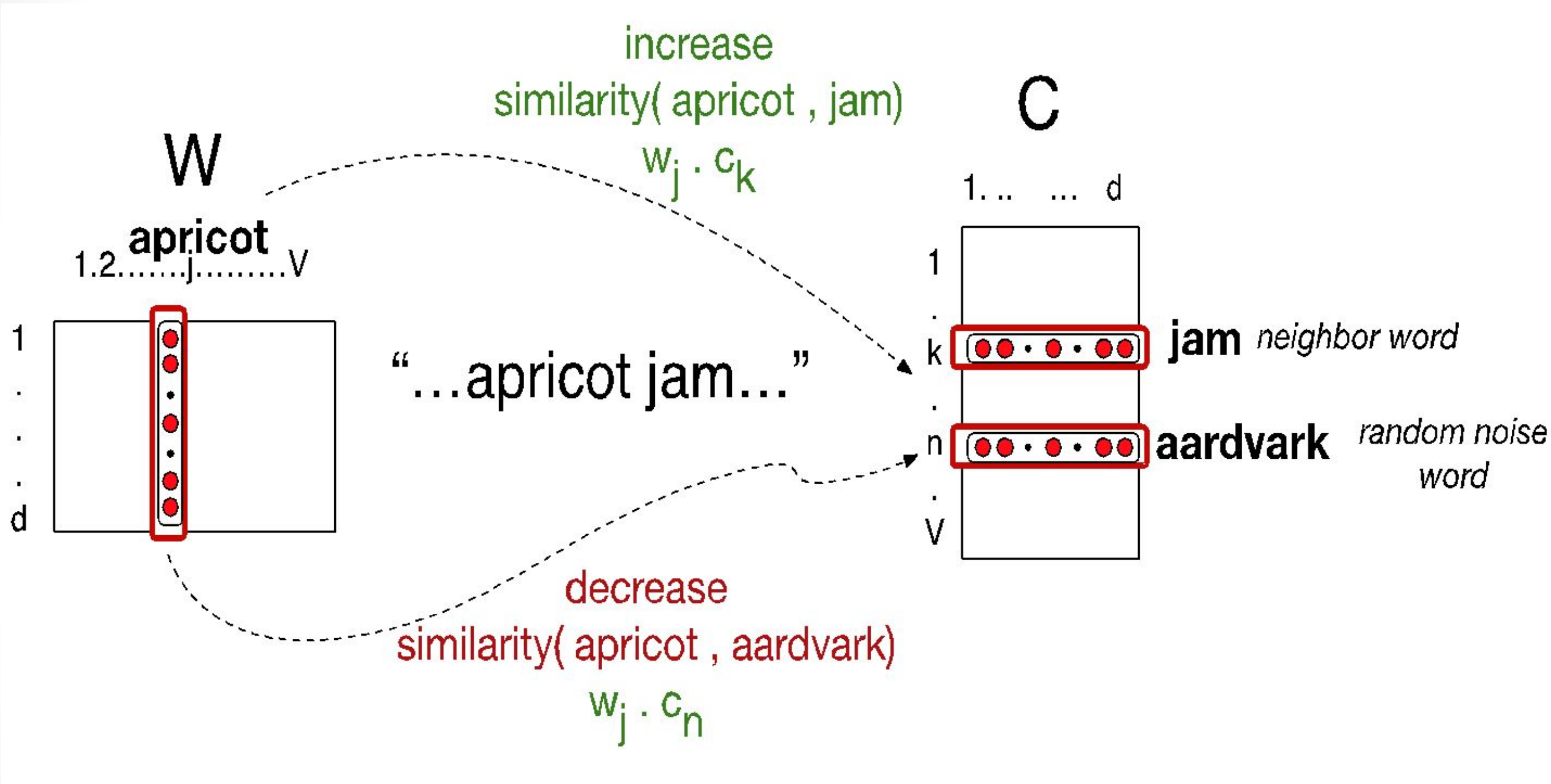
- We want to maximize…

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

- 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, 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

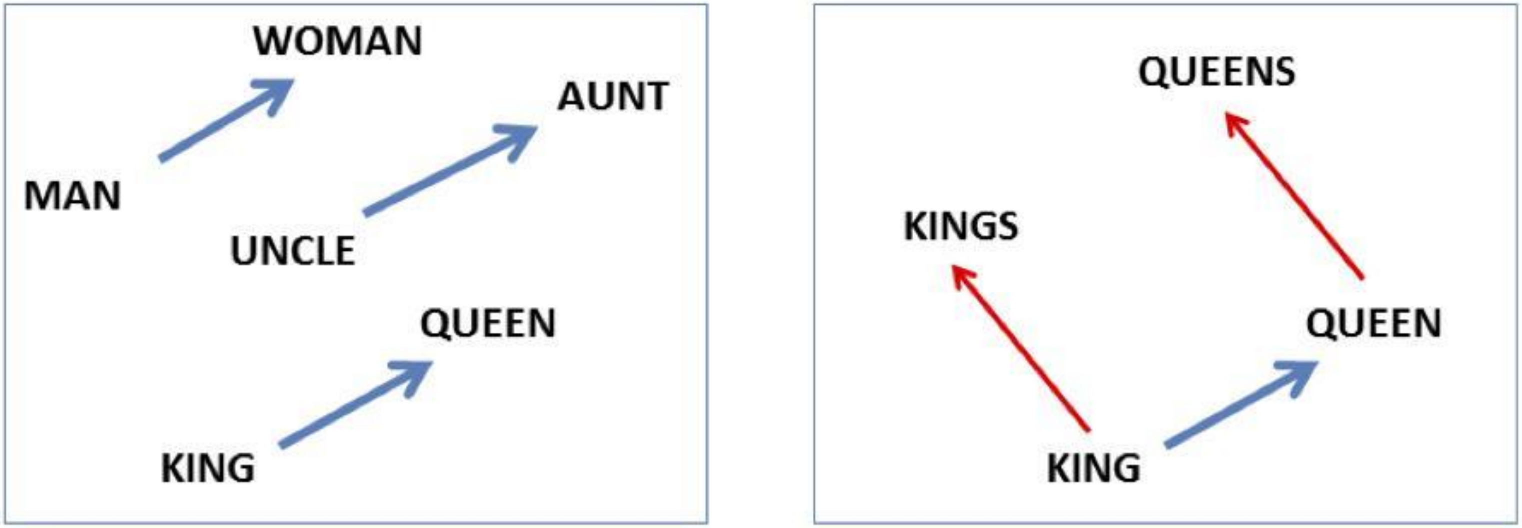
Similarity depends on window size C

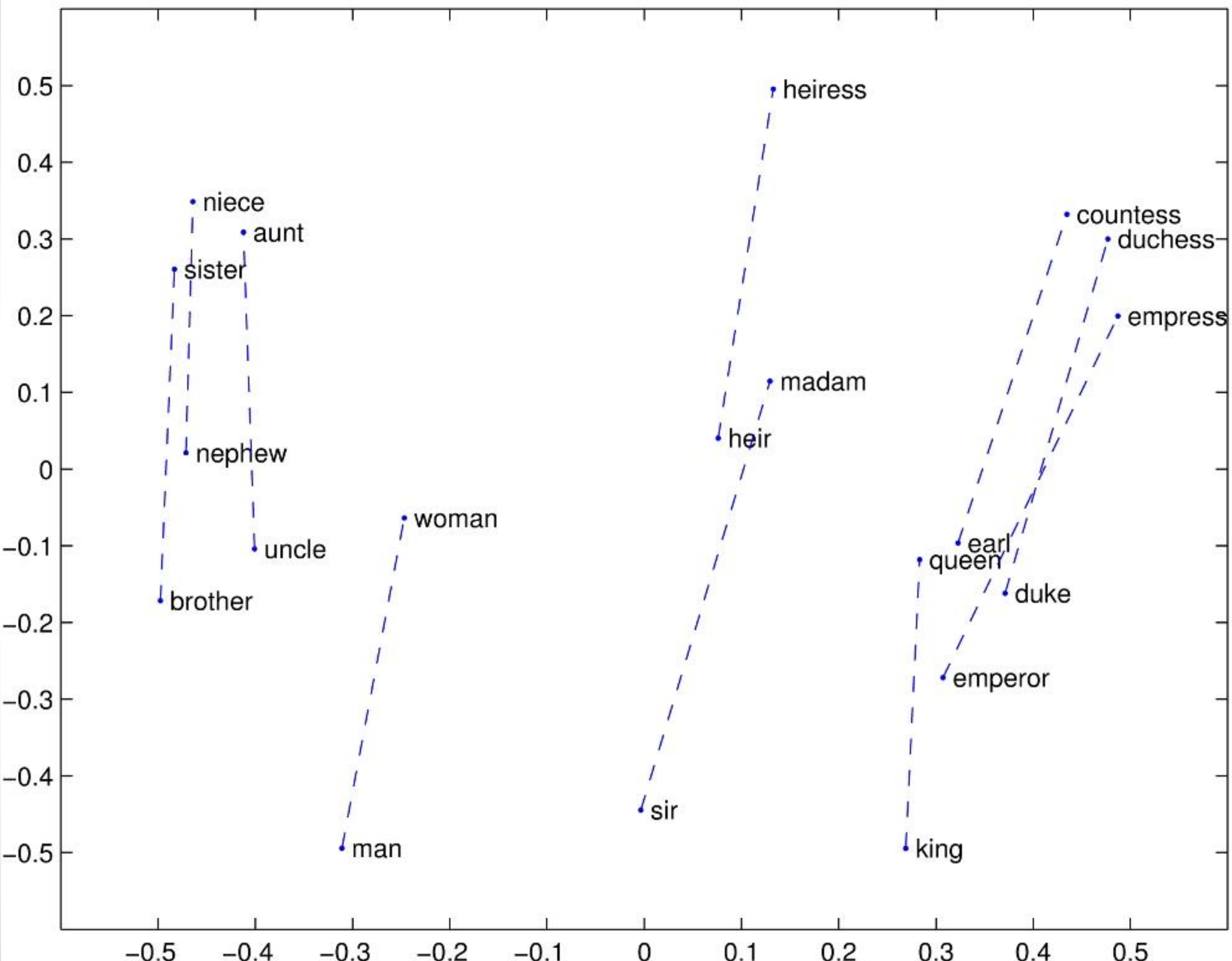
- C = ±2 The nearest words to *Hogwarts:*
  - *Sunnydale*
  - *Evernight*
- C = ±5 The nearest words to *Hogwarts:*
  - *Dumbledore*
  - *Malfoy*
  - *halfblood*

Analogy: Embeddings capture relational meaning!

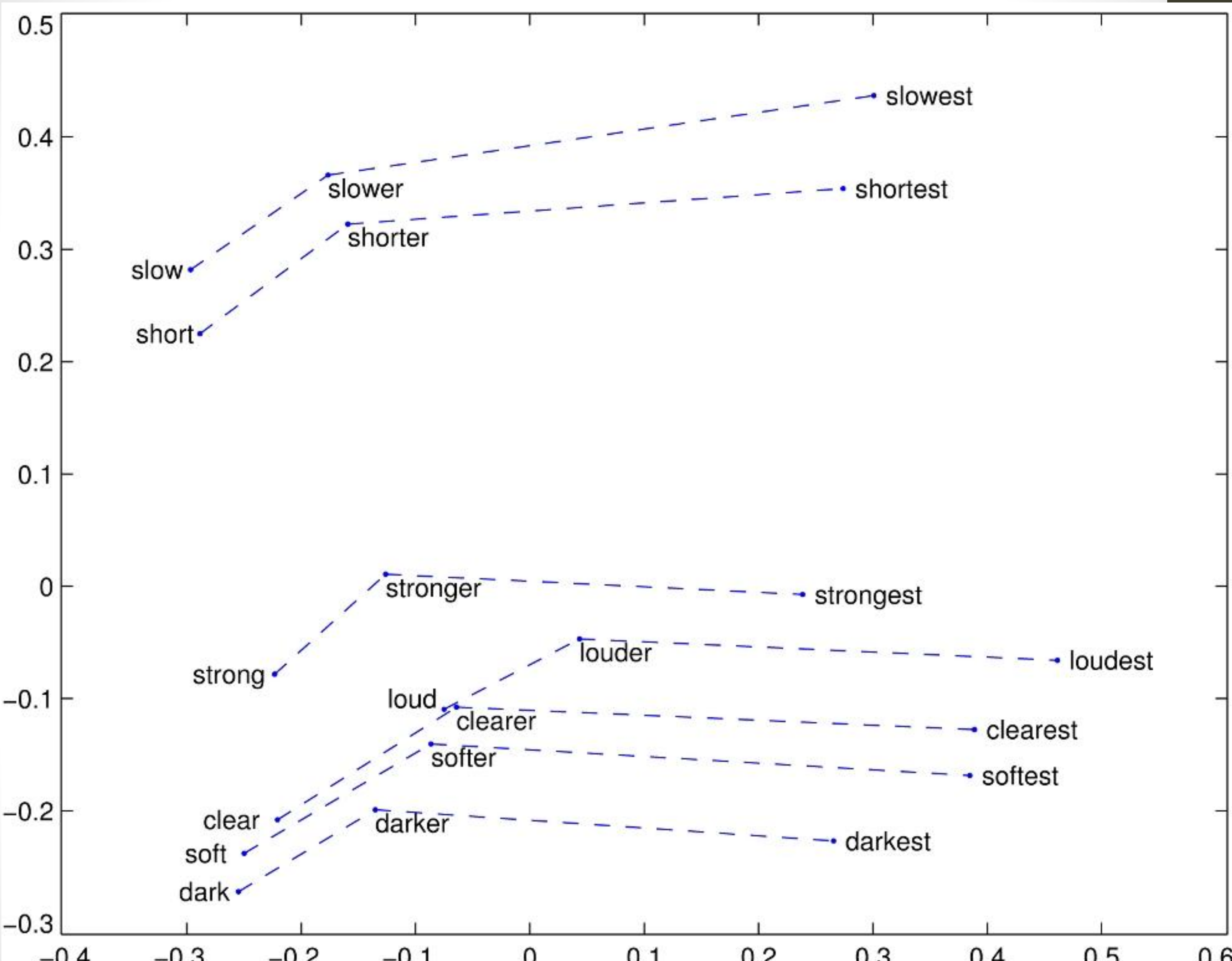
- vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

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





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



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

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

Using Word2Vec with Tensorflow

- Several ways:
  - Pip install Tensorflow from your Python pip directory (from Windows command prompt)

  C:\python\Lib\site-packages > py -m pip install --upgrade https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-1.12.0-py3-none-any.whl

  - In Python interpreter or editor to check installation

  >> import tensorflow as tf

  - Use Colaboratory (recommended):
    - colab.research.google.com
    - https://colab.research.google.com/notebooks/welcome.ipynb
  - https://www.tensorflow.org/tutorials/representation/word2vec