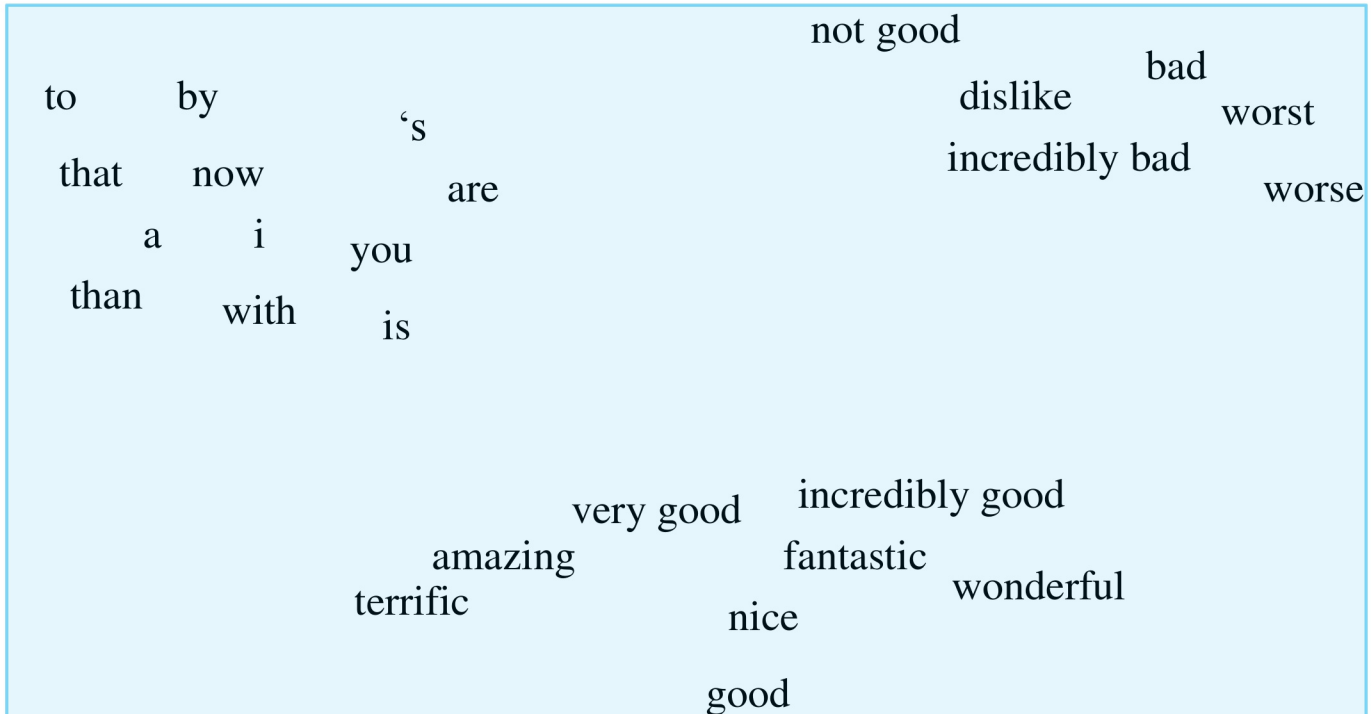
We'll build a new model of meaning focusing on similarity

Each word = a vector

- Not just "word" or word45.

Similar words are "nearby in space"



We define a word as a vector

Called 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!!!

2 kinds of embeddings

tf-idf

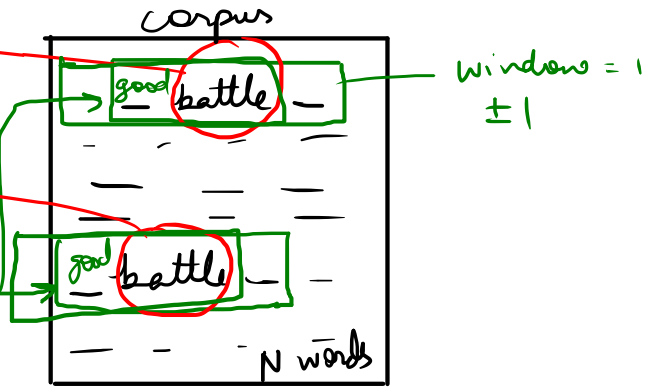
- The workhorse of Information Retrieval!
- A common baseline model
- **Sparse** vectors
- Words are represented by a simple function of the counts of nearby words

Word2vec

- **Dense** vectors
- Representation is created by training a classifier to distinguish nearby and far-away words

Review: words, vectors, and co-occurrence matrices

Term-document matrix



Each document is represented by a vector of words

D1 battle D2 good D3 fool D4 wit → content → N contents

	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

N words

Document vectors

N content

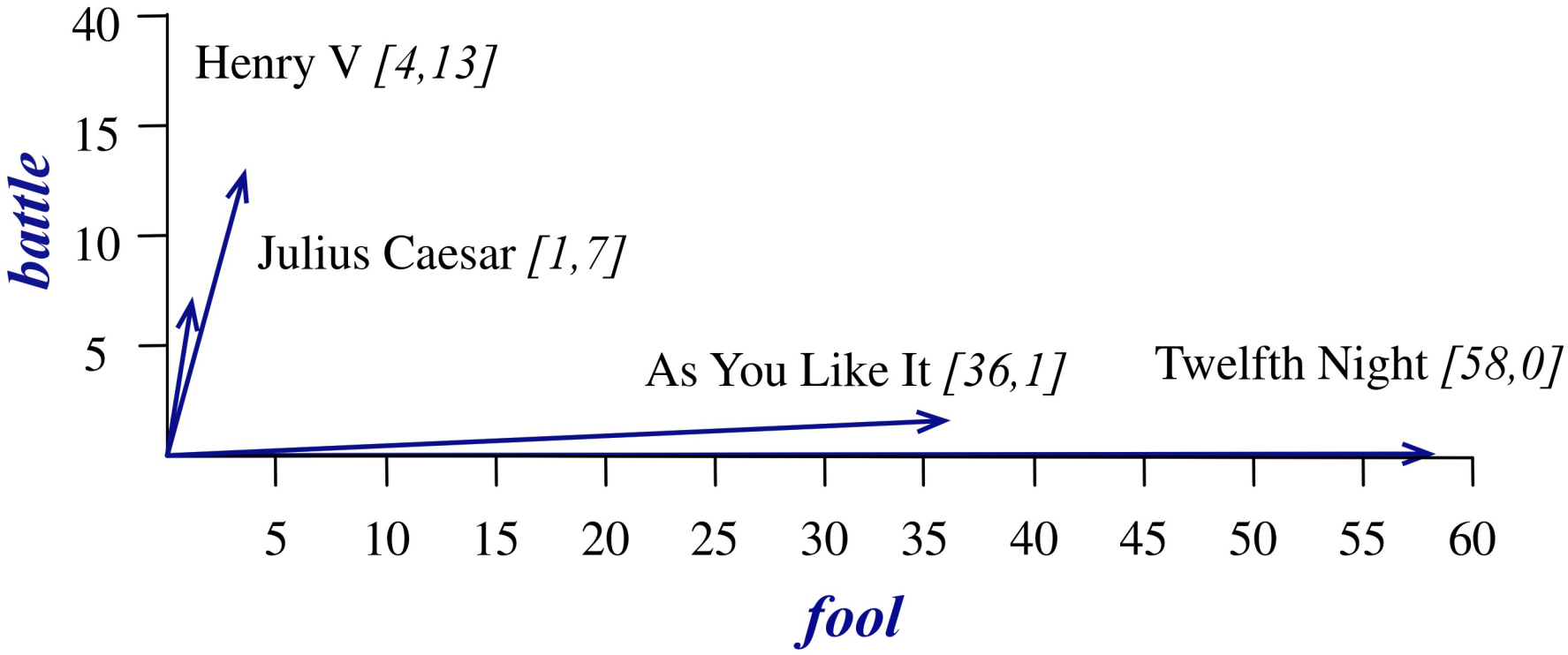
N words

Word coccurance Matrix

Word-word matrix

word-content mat

Visualizing document vectors



Vectors are the basis of information retrieval

	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*.

New idea for word meaning: Words can be vectors too!!!

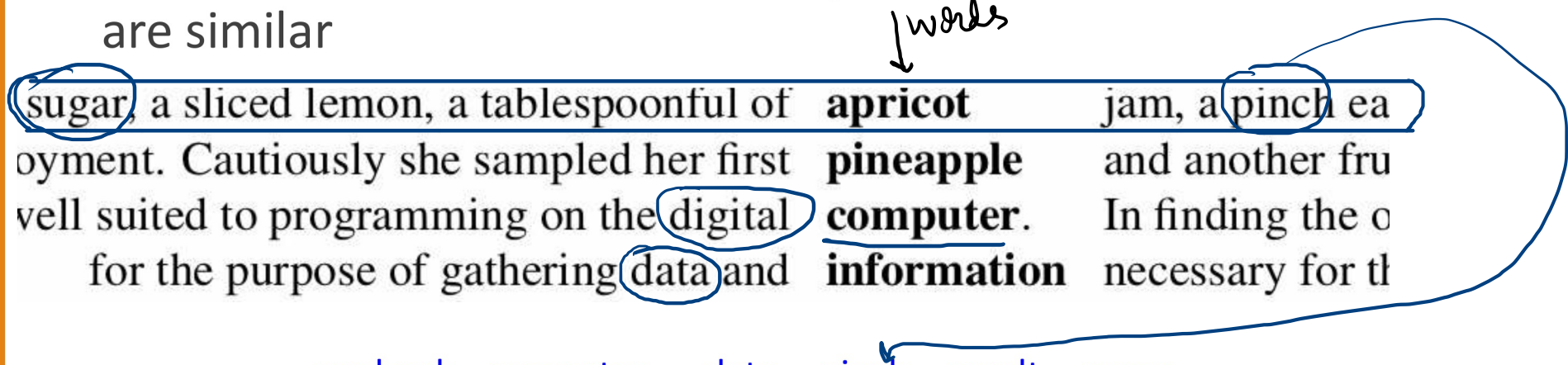
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

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

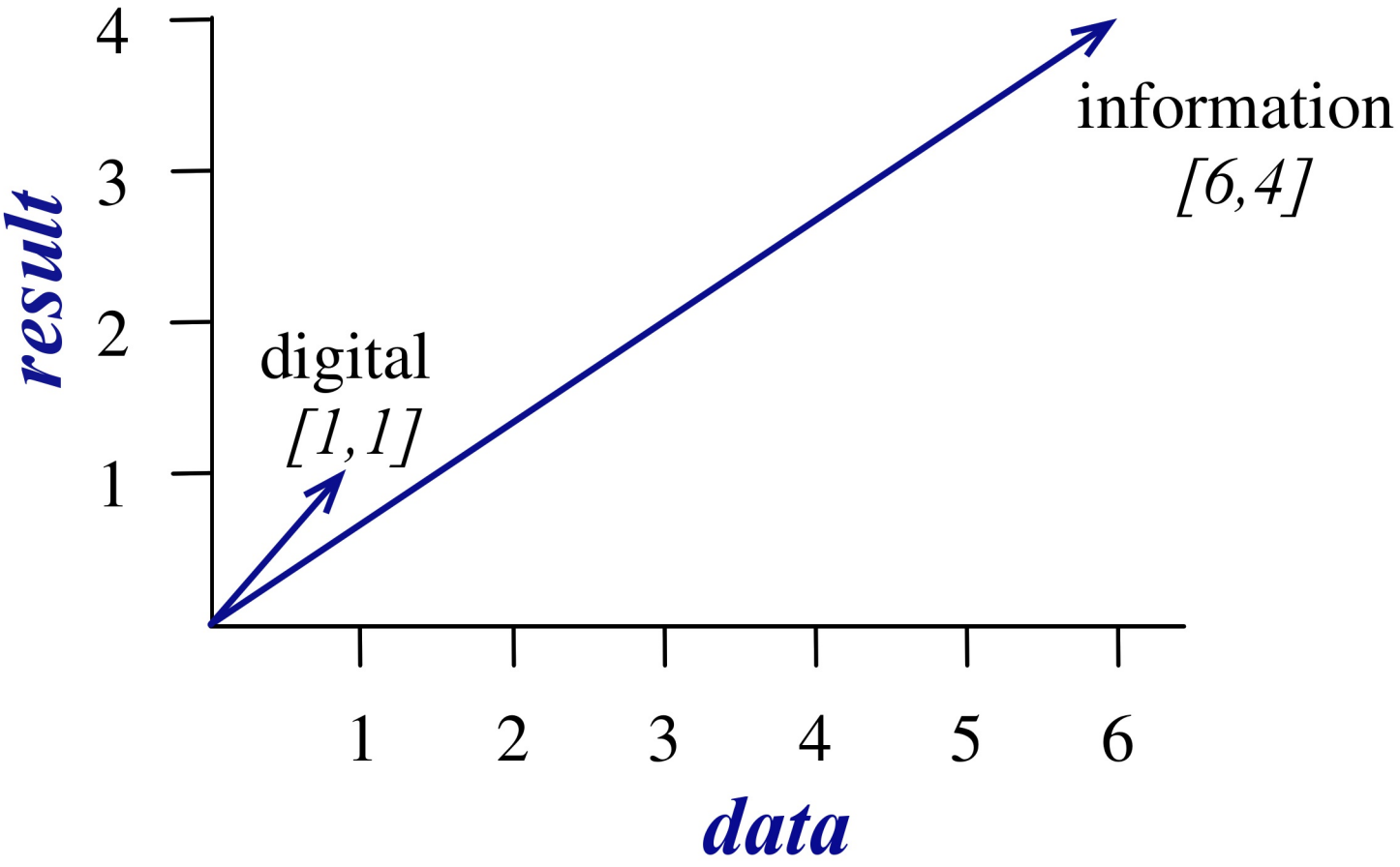
fool is "the kind of word that occurs in comedies, especially Twelfth Night"

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

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



	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	


4
3
2
1
result
digital
[1,1]
information
[6,4]
1
2
3
4
5
6
data

Cosine for computing word similarity

$$\text{cosine}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

v_i is the count for word v in context i

w_i is the count for word w in context i.

$$\vec{a}\cdot\vec{b} = |\vec{a}||\vec{b}|\cos\theta$$

$$\frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|} = \cos\theta$$

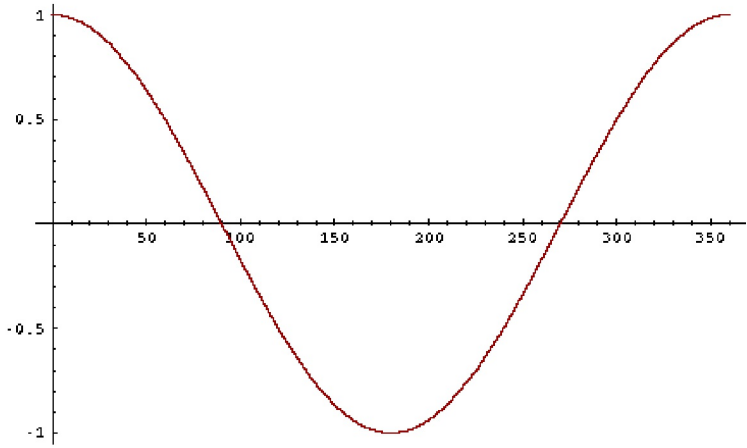
Cos($\vec{v}$,$\vec{w}$) is the cosine similarity of $\vec{v}$ and $\vec{w}$

Cosine as a similarity metric

-1: vectors point in opposite directions

+1: vectors point in same directions

0: vectors are orthogonal



$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\ \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\ \frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

	large	data	computer
apricot ←	1	0	0
digital	0	1	2
information ←	1	6	1

Which pair of words is more similar?

cosine(apricot,information) =

$$\frac{1+0+0}{\sqrt{1+0+0}\ \sqrt{1+36+1}} = \frac{1}{\sqrt{38}} = .16$$

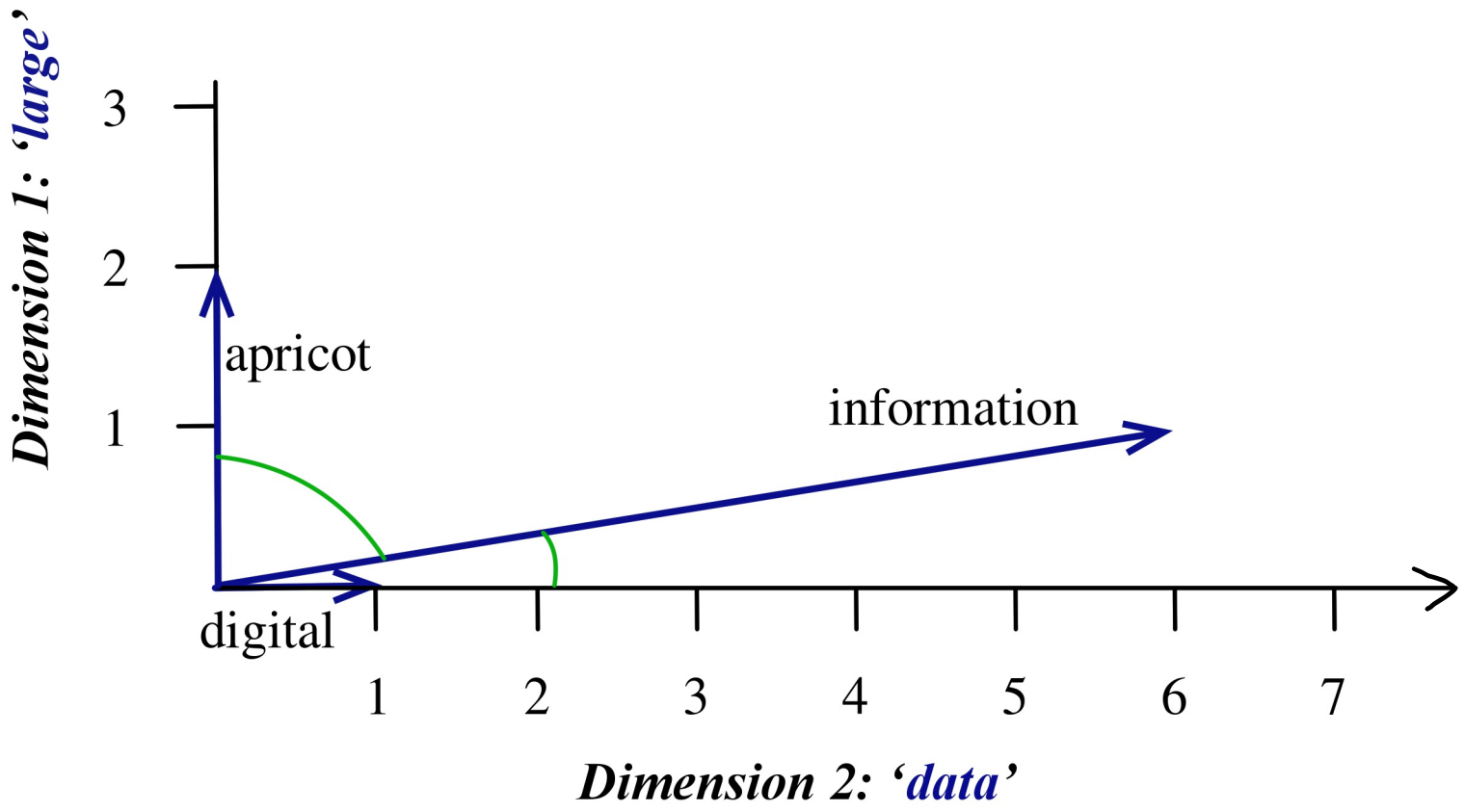
cosine(digital,information) =

$$\frac{0+6+2}{\sqrt{0+1+4}\ \sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58$$

cosine(apricot,digital) =

$$\frac{0+0+0}{\sqrt{1+0+0}\ \sqrt{0+1+4}} = 0$$

Visualizing cosines (well, angles)



But raw frequency is a bad representation

- Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.
- But overly frequent words like *the*, *it,* or *they* are not very informative about the context
- Need a function that resolves this frequency paradox!

Many functions can help in reweighting the counts

tf-idf:

tf-idf value for word t in document d:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Words like "the" or "good" have very low idf

Pointwise mutual information

- $\mathrm{PMI}(w_1, w_2) = log \frac{p(w_1,w_2)}{p(w_1)p(w_2)}$ = $\frac{\text{Probability of Occuring together}}{\text{Probability of occuring independently}}$

See if words like "good" appear more often with "great" than we would expect by chance

Vector Semantics

Dense Vectors

Sparse versus dense vectors

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

Three methods for getting short dense vectors

- "Neural Language Model"-inspired models
  - Word2vec (skip-grams, CBOW), Glove
- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- Brown clustering

Vector Semantics

Embeddings inspired by neural language models: word2vec

Embeddings you can download!

Word2vec (Mikolov et al)

https://code.google.com/archive/p/word2vec/

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

Word2Vec: Skip-Gram Task

Word2vec provides a variety of options. Let's do

"skip-gram with negative sampling" (SGNS)

Approach: predict if candidate word *c* is a "neighbor"

- Treat the target word *t* and a true neighboring context word *c* as **positive examples**.
- Randomly sample other words in the lexicon to get negative examples
- Use logistic regression to train a classifier to distinguish those two cases
- Use the weights as the embeddings

Skip-Gram Training Data

Assume a +/- 2 word window, given training sentence:

- *I would like to go someplace nearby for lunch*

  `[target]`

The training data:

- input/output pairs (centering on *go)*

Skip-Gram Training data

I would like to go someplace nearby for lunch

Positive

{target, context}

- *{go, like}*
- *{go, to}*
- *{go, someplace}*
- *{go, nearby}*

Skip-Gram Training data

I would like to go someplace nearby for lunch

Positive

{target, context}

- *{go, like}*
- *{go, to}*
- *{go, someplace}*
- *{go, nearby}*

That's fine for positive data. But for training a binary classifier we need negative examples.

Let's sample other words from the lexicon (that don't occur with the target word in this context).

Skip-Gram Training data

I would like to go someplace nearby for lunch

Positive

{+ target, context}

{go, like}

{go, to}

{go, someplace}

{go, nearby}

Negative

{- target, context}

{go, aardvark}

{go, incubate}

{go, twelve}

{go, therefore}

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.

The classifier's goal

- Compute the probability that c is a real context word and not a fake noise word
- P(real|t,c)
- P(fake|t,c)

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
- So:
  - Similarity(t,c) $\propto t \cdot c$
- We'll need to normalize to get a probability
  - (cosine isn't a probability either)

Turning dot products into probabilities

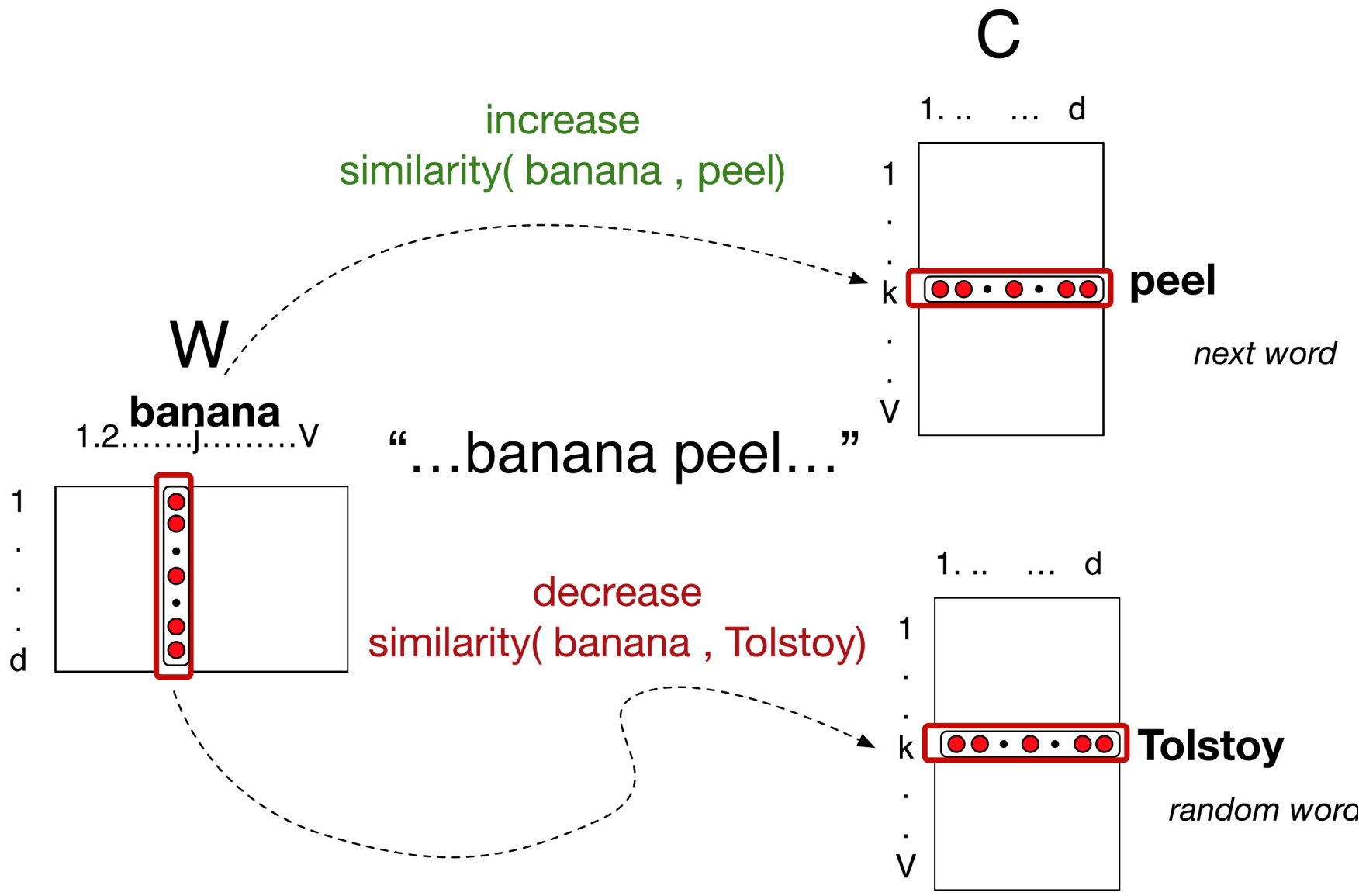
- $\text{Sim(t,c)} = t \cdot c$
- To turn this into a probability.
- We'll use the sigmoid from logistic regression:

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

Learning the classifier

- How to learn?
  - Stochastic gradient descent!

- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.


C
1. .. ... d
1
k
V
peel
next word
increase
similarity(banana , peel)
W
banana
1.2.....j.....V
1
d
"…banana peel…"
decrease
similarity(banana , Tolstoy)
1. .. ... d
1
k
V
Tolstoy
random word

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
- the – label for the pairs sample from the negative data.

Train using stochastic gradient descent

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.

Or in other words

- 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.

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 count models (tf-idf/PPMI) or dense predict models (word2vec, GLoVE)
  - Useful in practice but also encode cultural stereotypes
    - Can **debias** embeddings, and use them to **study bias**