

Vector Semantics & Embeddings

Word Meaning

What do words mean?

N-gram or text classification methods we've seen so far

- Words are just strings (or indices w_i in a vocabulary list)
- That's not very satisfactory!

Introductory logic classes:

- The meaning of "dog" is DOG; cat is CAT
$$\forall x \text{ DOG}(x) \rightarrow \text{MAMMAL}(x)$$

Old linguistics joke by Barbara Partee in 1967:

- Q: What's the meaning of life?
- A: LIFE

That seems hardly better!

Desiderata

What should a theory of word meaning do for us?

Let's look at some desiderata

From **lexical semantics**, the linguistic study of word meaning

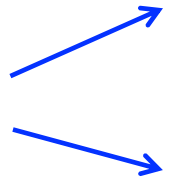
Lemmas and senses

lemma



mouse (N)

sense



1. any of numerous small rodents...

2. a hand-operated device that controls
a cursor...

Modified from the online thesaurus WordNet

A **sense** or “**concept**” is the meaning component of a word
Lemmas can be **polysemous** (have multiple senses)

Relations between senses: Synonymy

Synonyms have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- water / H₂O

Relations between senses: Synonymy

Note that there are probably no examples of perfect synonymy.

- Even if many aspects of meaning are identical
- Still may differ based on politeness, slang, register, genre, etc.

Relation: **Synonymy**?

water/H₂O

"H₂O" in a surfing guide?

big/large

my big sister != my large sister

The Linguistic Principle of Contrast

Difference in form → difference in meaning

Abbé Gabriel Girard 1718

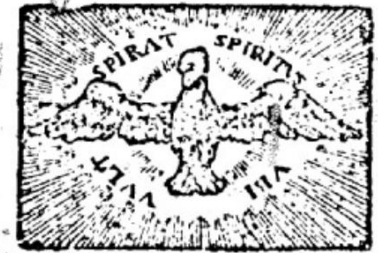
Re: "exact" synonyms

"je ne crois pas qu'il y ait de
mot synonyme dans aucune
Langue."

[I do not believe that there
is a synonymous word in any
language]

LA JUSTESSE
DE LA
LANGUE FRANÇOISE,
OU
LES DIFFERENTES SIGNIFICATIONS
DES MOTS QUI PASSENT
POUR
SYNONIMES.

Par M. l'Abbé GIRARD C. D. M. D. D. F.



A PARIS,
Chez LAURENT D'HOURY, Imprimeur-
Libraire, au bas de la rue de la Harpe, vis-
à vis la rue S. Severin, au Saint Esprit.

M. DCC. XVIII.

Avec Approbation & Privilège du Roy.

Relation: Similarity

Words with similar meanings. Not synonyms, but sharing some element of meaning

car, bicycle

cow, horse

Ask humans how similar 2 words are

| word1 | word2 | similarity |
|--------|------------|------------|
| vanish | disappear | 9.8 |
| behave | obey | 7.3 |
| belief | impression | 5.95 |
| muscle | bone | 3.65 |
| modest | flexible | 0.98 |
| hole | agreement | 0.3 |

Relation: Word relatedness

Also called "word association"

Words can be related in any way, perhaps via a semantic frame or field

- coffee, tea: **similar**
- coffee, cup: **related**, not similar

Semantic field

Words that

- cover a particular semantic domain
- bear structured relations with each other.

hospitals

surgeon, scalpel, nurse, anaesthetic, hospital

restaurants

waiter, menu, plate, food, menu, chef

houses

door, roof, kitchen, family, bed

Relation: Antonymy

Senses that are opposites with respect to only one feature of meaning

Otherwise, they are very similar!

| | | | |
|------------|------------|-----------|-----------|
| dark/light | short/long | fast/slow | rise/fall |
| hot/cold | up/down | in/out | |

More formally: antonyms can

- define a binary opposition or be at opposite ends of a scale
 - long/short, fast/slow
- Be *reversives*:
 - rise/fall, up/down

Connotation (sentiment)

- Words have **affective** meanings
 - Positive connotations (*happy*)
 - Negative connotations (*sad*)
- Connotations can be subtle:
 - Positive connotation: *copy, replica, reproduction*
 - Negative connotation: *fake, knockoff, forgery*
- Evaluation (sentiment!)
 - Positive evaluation (*great, love*)
 - Negative evaluation (*terrible, hate*)

Connotation

Osgood et al. (1957)

Words seem to vary along 3 affective dimensions:

- **valence**: the pleasantness of the stimulus
- **arousal**: the intensity of emotion provoked by the stimulus
- **dominance**: the degree of control exerted by the stimulus

| | Word | Score | | Word | Score |
|------------------|------------|-------|--|-----------|-------|
| Valence | love | 1.000 | | toxic | 0.008 |
| | happy | 1.000 | | nightmare | 0.005 |
| Arousal | elated | 0.960 | | mellow | 0.069 |
| | frenzy | 0.965 | | napping | 0.046 |
| Dominance | powerful | 0.991 | | weak | 0.045 |
| | leadership | 0.983 | | empty | 0.081 |

Values from NRC VAD Lexicon (Mohammad 2018)

So far

Concepts or word senses

- Have a complex many-to-many association with **words** (homonymy, multiple senses)

Have relations with each other

- Synonymy
- Antonymy
- Similarity
- Relatedness
- Connotation

Vector Semantics & Embeddings

Word Meaning

Vector Semantics & Embeddings

Vector Semantics

Computational models of word meaning

Can we build a theory of how to represent word meaning, that accounts for at least some of the desiderata?

We'll introduce **vector semantics**

The standard model in language processing!

Handles many of our goals!

Ludwig Wittgenstein

PI #43:

"The meaning of a word is its use in the language"

Let's define words by their usages

One way to define "usage":

words are defined by their environments (the words around them)

Zellig Harris (1954):

If A and B have almost identical environments we say that they are synonyms.

What does recent English borrowing *ongchoi* mean?

Suppose you see these sentences:

- Ong choi is delicious **sautéed with garlic**.
- Ong choi is superb **over rice**
- Ong choi **leaves** with salty sauces

And you've also seen these:

- ...spinach **sautéed with garlic over rice**
- Chard stems and **leaves** are **delicious**
- Collard greens and other **salty** leafy greens

Conclusion:

- Ongchoi is a leafy green like spinach, chard, or collard greens
 - We could conclude this based on words like "leaves" and "delicious" and "sauteed"

Ongchoi: *Ipomoea aquatica* "Water Spinach"

空心菜
kangkong
rau muống
...



Idea 1: Defining meaning by linguistic distribution

Let's define the meaning of a word by its distribution in language use, meaning its neighboring words or grammatical environments.

Idea 2: Meaning as a point in space (Osgood et al. 1957)

3 affective dimensions for a word

- **valence**: pleasantness
- **arousal**: intensity of emotion
- **dominance**: the degree of control exerted

| | Word | Score | | Word | Score |
|------------------|------------|-------|--|-----------|-------|
| Valence | love | 1.000 | | toxic | 0.008 |
| | happy | 1.000 | | nightmare | 0.005 |
| Arousal | elated | 0.960 | | mellow | 0.069 |
| | frenzy | 0.965 | | napping | 0.046 |
| Dominance | powerful | 0.991 | | weak | 0.045 |
| | leadership | 0.983 | | empty | 0.081 |

NRC VAD Lexicon
(Mohammad 2018)

Hence the connotation of a word is a vector in 3-space

Idea 1: Defining meaning by linguistic distribution

Idea 2: Meaning as a point in multidimensional space

Defining meaning as a point in space based on distribution

Each word = a vector (not just "good" or " w_{45} ")

Similar words are "**nearby in semantic space**"

We build this space automatically by seeing which words are **nearby in text**



We define meaning of a word as a vector

Called an "embedding" because it's embedded into a space (see textbook)

The standard way to represent meaning in NLP

Every modern NLP algorithm uses embeddings as the representation of word meaning

Fine-grained model of meaning for similarity

Intuition: why vectors?

Consider sentiment analysis:

- With **words**, a feature is a word identity
 - Feature 5: 'The previous word was "terrible"'
 - requires **exact same word** to be in training and test
- With **embeddings**:
 - Feature is a word vector
 - 'The previous word was vector [35,22,17...]'
 - Now in the test set we might see a similar vector [34,21,14]
 - We can generalize to **similar but unseen** words!!!

We'll discuss 2 kinds of embeddings

tf-idf

- Information Retrieval workhorse!
- 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 **predict** whether a word is likely to appear nearby
- Later we'll discuss extensions called **contextual embeddings**

Vector Semantics & Embeddings

Vector Semantics

Vector
Semantics &
Embeddings

Words and Vectors

Words and Vectors

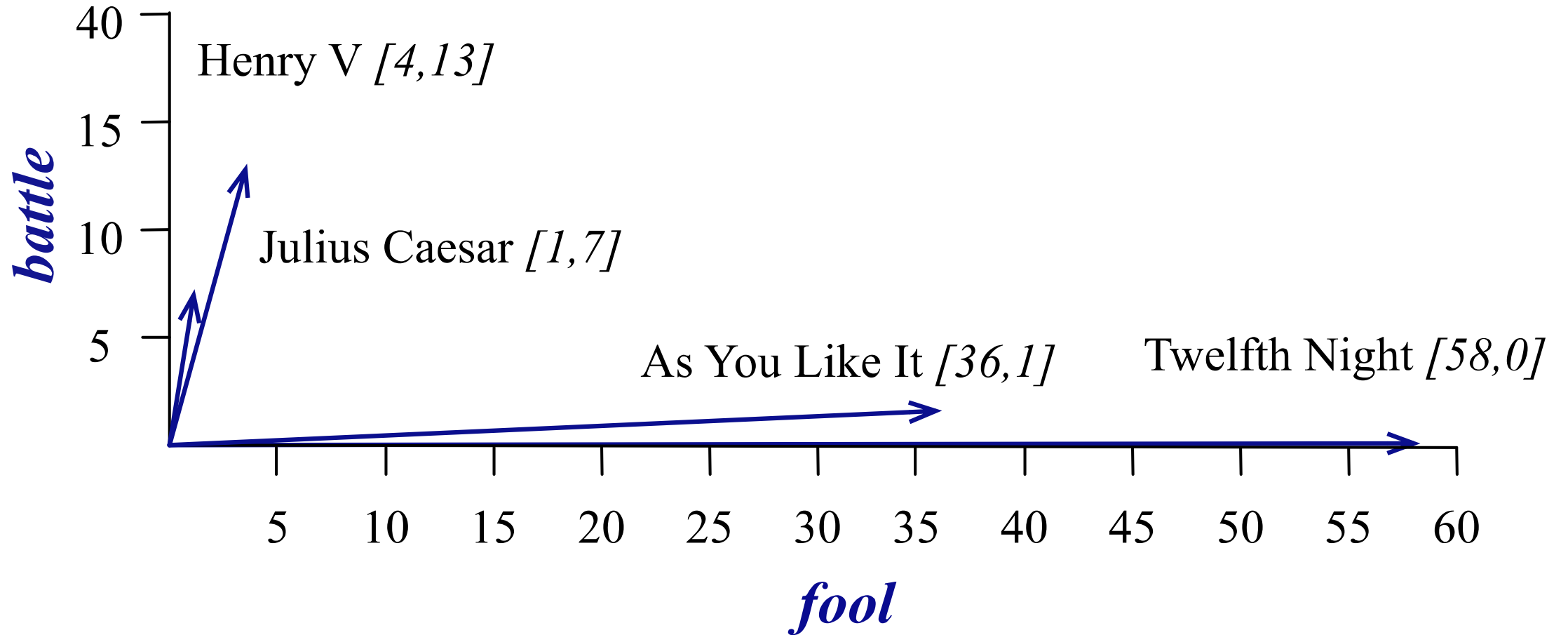
“The most important attributes of a vector in 3-space are {Location, Location, Location}”
Randall Munroe, <https://xkcd.com/2358/>

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 |

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

But comedies are different than the other two

Comedies have more *fools* and *wit* and fewer *battles*.

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"

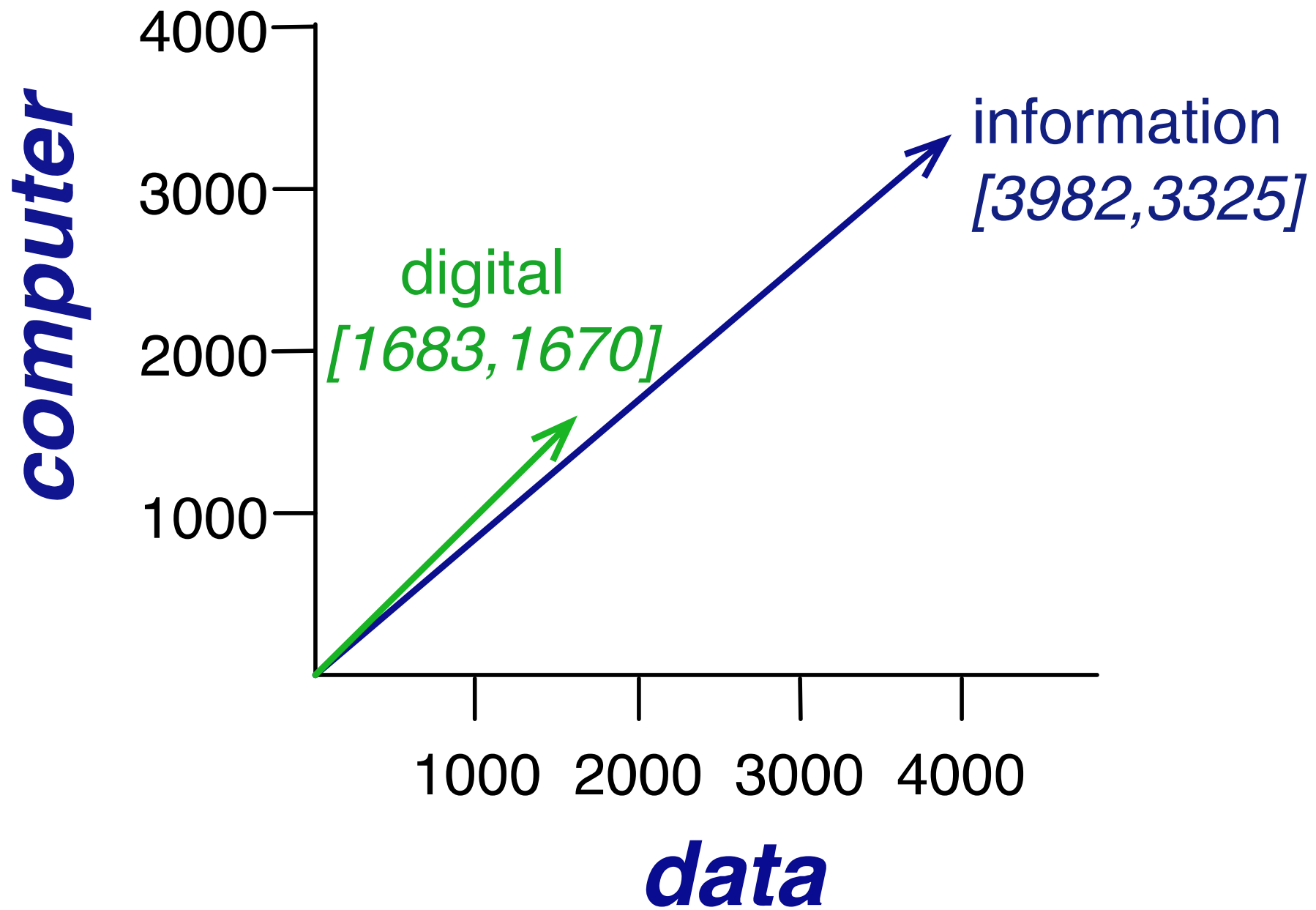
fool is "the kind of word that occurs in comedies, especially Twelfth Night"

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

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

is traditionally followed by **cherry** pie, a traditional dessert
often mixed, such as **strawberry** rhubarb pie. Apple pie
computer peripherals and personal **digital** assistants. These devices usually
a computer. This includes **information** available on the internet

| | aardvark | ... | computer | data | result | pie | sugar | ... |
|-------------|----------|-----|----------|------|--------|-----|-------|-----|
| cherry | 0 | ... | 2 | 8 | 9 | 442 | 25 | ... |
| strawberry | 0 | ... | 0 | 0 | 1 | 60 | 19 | ... |
| digital | 0 | ... | 1670 | 1683 | 85 | 5 | 4 | ... |
| information | 0 | ... | 3325 | 3982 | 378 | 5 | 13 | ... |



Vector Semantics & Embeddings

Words and Vectors

Cosine for computing word similarity

Vector
Semantics &
Embeddings

Computing word similarity: Dot product and cosine

The dot product between two vectors is a scalar:

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^N v_i w_i = v_1 w_1 + v_2 w_2 + \dots + v_N w_N$$

The dot product tends to be high when the two vectors have large values in the same dimensions

Dot product can thus be a useful similarity metric between vectors

Problem with raw dot-product

Dot product favors long vectors

Dot product is higher if a vector is longer (has higher values in many dimension)

Vector length:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2}$$

Frequent words (of, the, you) have long vectors (since they occur many times with other words).

So dot product overly favors frequent words

Alternative: cosine for computing word similarity

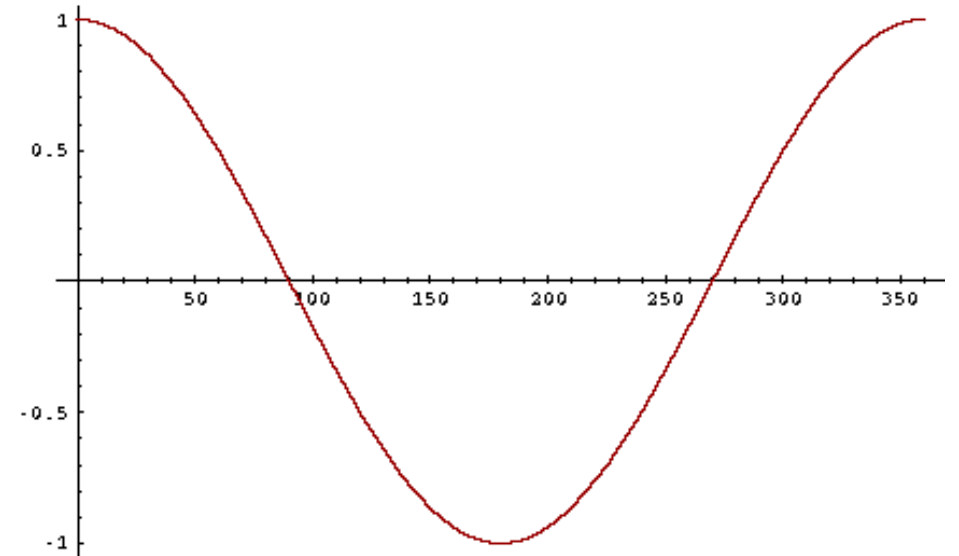
$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

Based on the definition of the dot product between two vectors **a** and **b**

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= |\mathbf{a}| |\mathbf{b}| \cos \theta \\ \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} &= \cos \theta \end{aligned}$$

Cosine as a similarity metric

- 1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal



But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1

Cosine examples

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

| | pie | data | computer |
|-------------|-----|------|----------|
| cherry | 442 | 8 | 2 |
| digital | 5 | 1683 | 1670 |
| information | 5 | 3982 | 3325 |

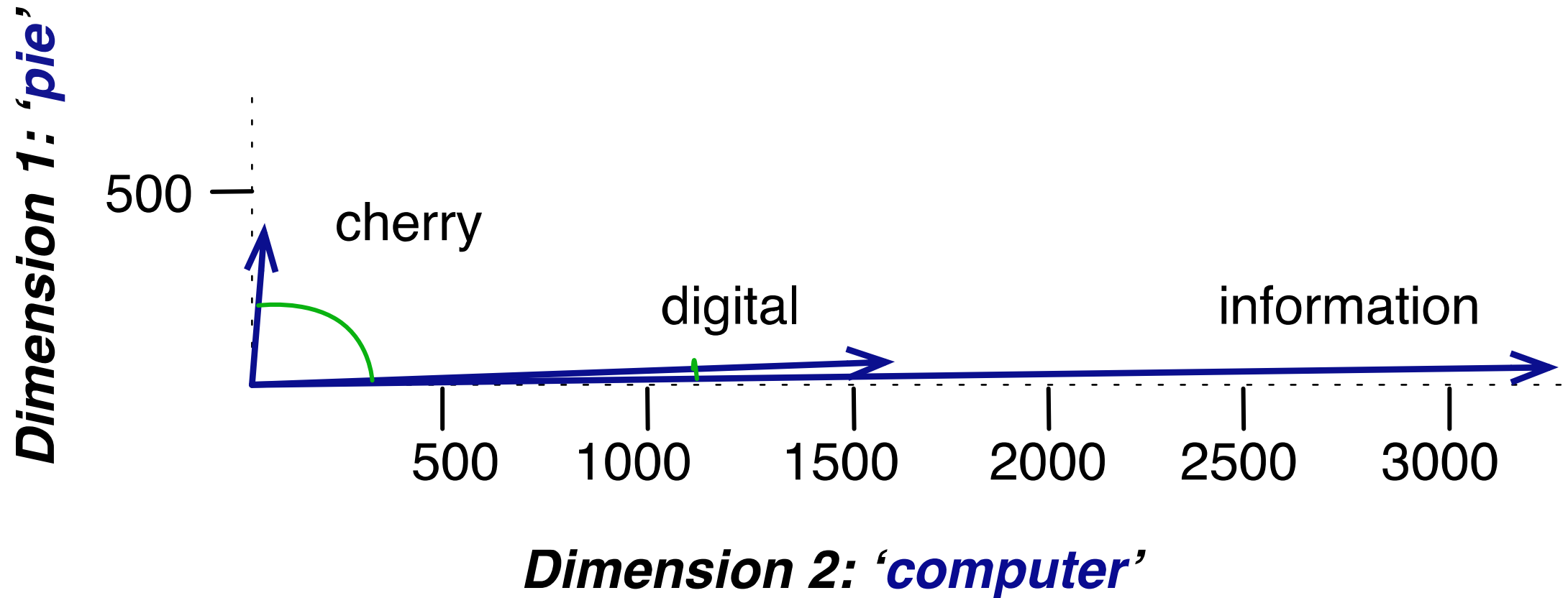
$$\cos(\text{cherry}, \text{information}) =$$

$$\frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

$$\cos(\text{digital}, \text{information}) =$$

$$\frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

Visualizing cosines (well, angles)



Vector Semantics & Embeddings

Cosine for computing word
similarity

Vector Semantics & Embeddings

TF-IDF

But raw frequency is a bad representation

- The co-occurrence matrices we have seen represent each cell by word frequencies.
- 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
- It's a paradox! How can we balance these two conflicting constraints?

Two common solutions for word weighting

tf-idf: tf-idf value for word t in document d :

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

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

PMI: (Pointwise mutual information)

- $\text{PMI}(w_1, w_2) = \log \frac{p(w_1, w_2)}{p(w_1)p(w_2)}$

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

Term frequency (tf) in the tf-idf algorithm

We could imagine using raw count:

$$\text{tf}_{t,d} = \text{count}(t,d)$$

But instead of using raw count, we usually squash a bit:

$$\text{tf}_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Document frequency (df)

df_t is the number of documents t occurs in.

(note this is not collection frequency: total count across all documents)

"*Romeo*" is very distinctive for one Shakespeare play:

| | Collection Frequency | Document Frequency |
|--------|----------------------|--------------------|
| Romeo | 113 | 1 |
| action | 113 | 31 |

Inverse document frequency (idf)

$$\text{idf}_t = \log_{10} \left(\frac{N}{\text{df}_t} \right)$$

N is the total number of documents in the collection

| Word | df | idf |
|----------|----|-------|
| Romeo | 1 | 1.57 |
| salad | 2 | 1.27 |
| Falstaff | 4 | 0.967 |
| forest | 12 | 0.489 |
| battle | 21 | 0.246 |
| wit | 34 | 0.037 |
| fool | 36 | 0.012 |
| good | 37 | 0 |
| sweet | 37 | 0 |

What is a document?

Could be a play or a Wikipedia article

But for the purposes of tf-idf, documents can be **anything**; we often call each paragraph a document!

Final tf-idf weighted value for a word

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

Raw counts:

| | 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 |

tf-idf:

| | As You Like It | Twelfth Night | Julius Caesar | Henry V |
|---------------|----------------|---------------|---------------|---------|
| battle | 0.246 | 0 | 0.454 | 0.520 |
| good | 0 | 0 | 0 | 0 |
| fool | 0.030 | 0.033 | 0.0012 | 0.0019 |
| wit | 0.085 | 0.081 | 0.048 | 0.054 |

Pointwise Mutual Information (PMI)

An alternative weighting function to tf-idf, PPMI (positive pointwise mutual information), is used for term-term-matrices, when the vector dimensions correspond to words rather than documents.

PMI

PPMI draws on the intuition that the best way to weigh the association between two words is to ask how much more the two words co-occur in our corpus than we would have a priori expected them to appear by chance.

PMI

Pointwise mutual information (Fano, 1961)⁴ is one of the most important concepts in NLP. It is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

PMI

The pointwise mutual information between a target word w and a context word c (Church and Hanks 1989, Church and Hanks 1990) is then defined as:

$$\text{PMI}(w, c) = \log_2 \frac{P(w, c)}{P(w)P(c)}$$

The numerator tells us how often we observed the two words together (assuming we compute probability by using the MLE). The denominator tells us how often we would expect the two words to co-occur assuming they each occurred independently.

PMI

PMI values range from negative to positive infinity But negative PMI values (which imply things are co-occurring less often than we would expect by chance) tend to be unreliable unless our corpora are enormous.

To distinguish whether two words whose individual probability is each 10^{-6} occur together less often than chance, we would need to be certain that the probability of the two occurring together is significantly less than 10^{-12} , and this kind of granularity would require an enormous corpus.

Positive PMI (PPMI)

For above reason it is more common to use Positive PMI (called **PPMI**) which replaces all negative PMI values with zero (Church and Hanks 1989, Dagan et al. 1993, Niwa and Nitta 1994):

$$\text{PPMI}(w, c) = \max\left(\log_2 \frac{P(w, c)}{P(w)P(c)}, 0\right)$$

Positive PMI (PPMI)

More formally, let's assume we have a co-occurrence matrix F with W rows (words) and C columns (contexts), where f_{ij} gives the number of times word w_i occurs with context c_j .

This can be turned into a PPMI matrix where PPMI_{ij} gives the PPMI value of word w_i with context c_j (which we can also express as $\text{PPMI}(w_i; c_j)$ or $\text{PPMI}(w = i; c = j)$)

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}, \quad p_{i*} = \frac{\sum_{j=1}^C f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}, \quad p_{*j} = \frac{\sum_{i=1}^W f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}$$

$$\text{PPMI}_{ij} = \max\left(\log_2 \frac{p_{ij}}{p_{i*} p_{*j}}, 0\right)$$

Positive PMI (PPMI)

So taking previous example, $F=$

| | computer | data | result | pie | sugar | count(w) |
|----------------|----------|------|--------|-----|-------|----------|
| cherry | 2 | 8 | 9 | 442 | 25 | 486 |
| strawberry | 0 | 0 | 1 | 60 | 19 | 80 |
| digital | 1670 | 1683 | 85 | 5 | 4 | 3447 |
| information | 3325 | 3982 | 378 | 5 | 13 | 7703 |
| count(context) | 4997 | 5673 | 473 | 512 | 61 | 11716 |

Figure 6.10 Co-occurrence counts for four words in 5 contexts in the Wikipedia corpus, together with the marginals, pretending for the purpose of this calculation that no other word-s/contexts matter.

Positive PMI (PPMI)

Let us compute PPMI for a single instance.

$$P(w=\text{information}, c=\text{data}) = \frac{3982}{11716} = .3399$$

$$P(w=\text{information}) = \frac{7703}{11716} = .6575$$

$$P(c=\text{data}) = \frac{5673}{11716} = .4842$$

$$\text{PPMI}(\text{information}, \text{data}) = \log_2(.3399 / (.6575 * .4842)) = .0944$$

Positive PMI (PPMI)

| | computer | data | result | pie | sugar |
|-------------|----------|------|--------|------|-------|
| cherry | 0 | 0 | 0 | 4.38 | 3.30 |
| strawberry | 0 | 0 | 0 | 4.10 | 5.51 |
| digital | 0.18 | 0.01 | 0 | 0 | 0 |
| information | 0.02 | 0.09 | 0.28 | 0 | 0 |

Figure 6.12 The PPMI matrix showing the association between words and context words, computed from the counts in Fig. 6.11. Note that most of the 0 PPMI values are ones that had a negative PMI; for example $\text{PMI}(\text{cherry}, \text{computer}) = -6.7$, meaning that *cherry* and *computer* co-occur on Wikipedia less often than we would expect by chance, and with PPMI we replace negative values by zero.

Positive PMI (PPMI)

PMI has the problem of being biased toward infrequent events; very rare words tend to have very high PMI values.

One way to address this is to slightly change the computation for $P(c)$, using a different function $P_\alpha(c)$ that discounts the probability of the context word to the power of α :

$$\text{PPMI}_\alpha(w, c) = \max\left(\log_2 \frac{P(w, c)}{P(w)P_\alpha(c)}, 0\right)$$

$$P_\alpha(c) = \frac{\text{count}(c)^\alpha}{\sum_c \text{count}(c)^\alpha}$$

Vector Semantics & Embeddings

TF-IDF

Vector Semantics & Embeddings

Word2vec

Sparse versus dense vectors

tf-idf (or PMI) 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 (fewer weights to tune)
- Dense vectors may **generalize** better than explicit counts
- Dense vectors 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**

Common methods for getting short dense vectors

“Neural Language Model”-inspired models

- Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)

- A special case of this is called LSA – Latent Semantic Analysis

Alternative to these "static embeddings":

- Contextual Embeddings (ELMo, BERT)
- Compute distinct embeddings for a word in its context
- Separate embeddings for each token of a word

Simple static 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 provides various options. We'll do:

skip-gram with negative sampling (SGNS)

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

Big idea: **self-supervision**:

- A word c that occurs near *apricot* in the corpus acts as the gold "correct answer" for supervised learning
- No need for human labels
- Bengio et al. (2003); Collobert et al. (2011)

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

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

Skip-Gram Training Data

Assume a ± 2 word window, given training sentence:

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

Skip-Gram Classifier

(assuming a +/- 2 word window)

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair

(apricot, jam)

(apricot, aardvark)

...

And assigns each pair a probability:

$$P(+ | w, c)$$

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

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:

- $\text{Similarity}(w, c) \propto w \cdot c$

We'll need to normalize to get a probability

- (cosine isn't a probability either)

Turning dot products into probabilities

$$\text{Sim}(w, c) \approx w \cdot c$$

To turn this into a probability

We'll use the sigmoid from logistic regression:

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w, c) &= 1 - P(+|w, c) \\ &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \end{aligned}$$

How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words.
We'll assume independence and just multiply them:

$$P(+|w, c_{1:L}) = \prod_{i=1}^L \sigma(c_i \cdot w)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^L \log \sigma(c_i \cdot w)$$

Skip-gram classifier: summary

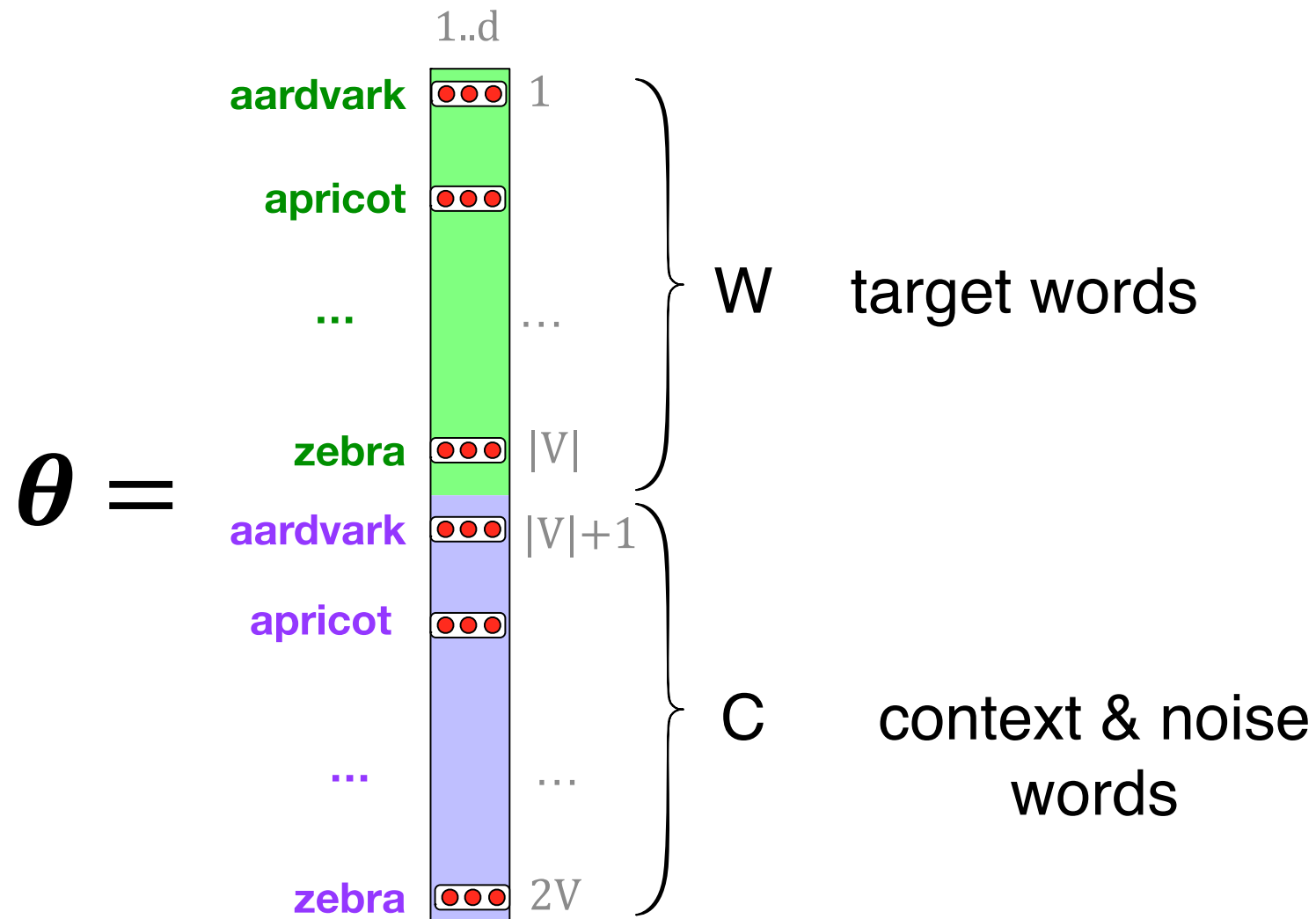
A probabilistic classifier, given

- a test target word w
- its context window of L words $c_{1:L}$

Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

To compute this, we just need embeddings for all the words.

These embeddings we'll need: a set for w , a set for c



Vector Semantics & Embeddings

Word2vec

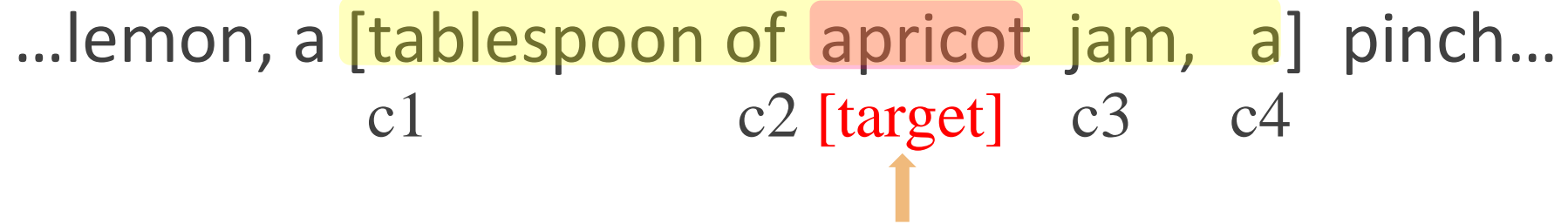
Vector Semantics & Embeddings

Word2vec: Learning the embeddings

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4



positive examples +

t

c

apricot tablespoon

apricot of

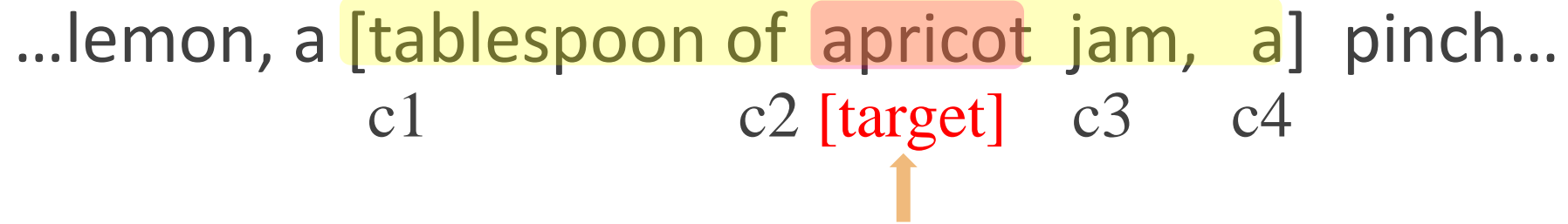
apricot jam

apricot a

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4



positive examples +

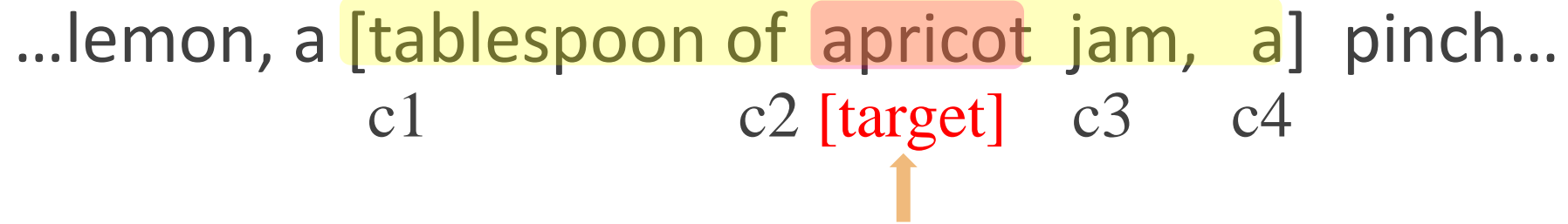
| t | c |
|---------|------------|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

For each positive example we'll grab k negative examples, sampling by frequency

Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4



positive examples +

| t | c |
|---------|------------|
| apricot | tablespoon |
| apricot | of |
| apricot | jam |
| apricot | a |

negative examples -

| t | c | t | c |
|---------|----------|---------|---------|
| apricot | aardvark | apricot | seven |
| apricot | my | apricot | forever |
| apricot | where | apricot | dear |
| apricot | coaxial | apricot | if |

Word2vec: how to learn vectors

Given the set of positive and negative training instances, and an initial set of embedding vectors

The goal of learning is to adjust those word vectors such that we:

- **Maximize** the similarity of the **target word, context word** pairs (w, c_{pos}) drawn from the positive data
- **Minimize** the similarity of the (w, c_{neg}) pairs drawn from the negative data.

Loss function for one w with c_{pos} , $c_{neg1} \dots c_{negk}$

Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

$$\begin{aligned} L_{CE} &= -\log \left[P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\ &= - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right] \end{aligned}$$

Learning the classifier

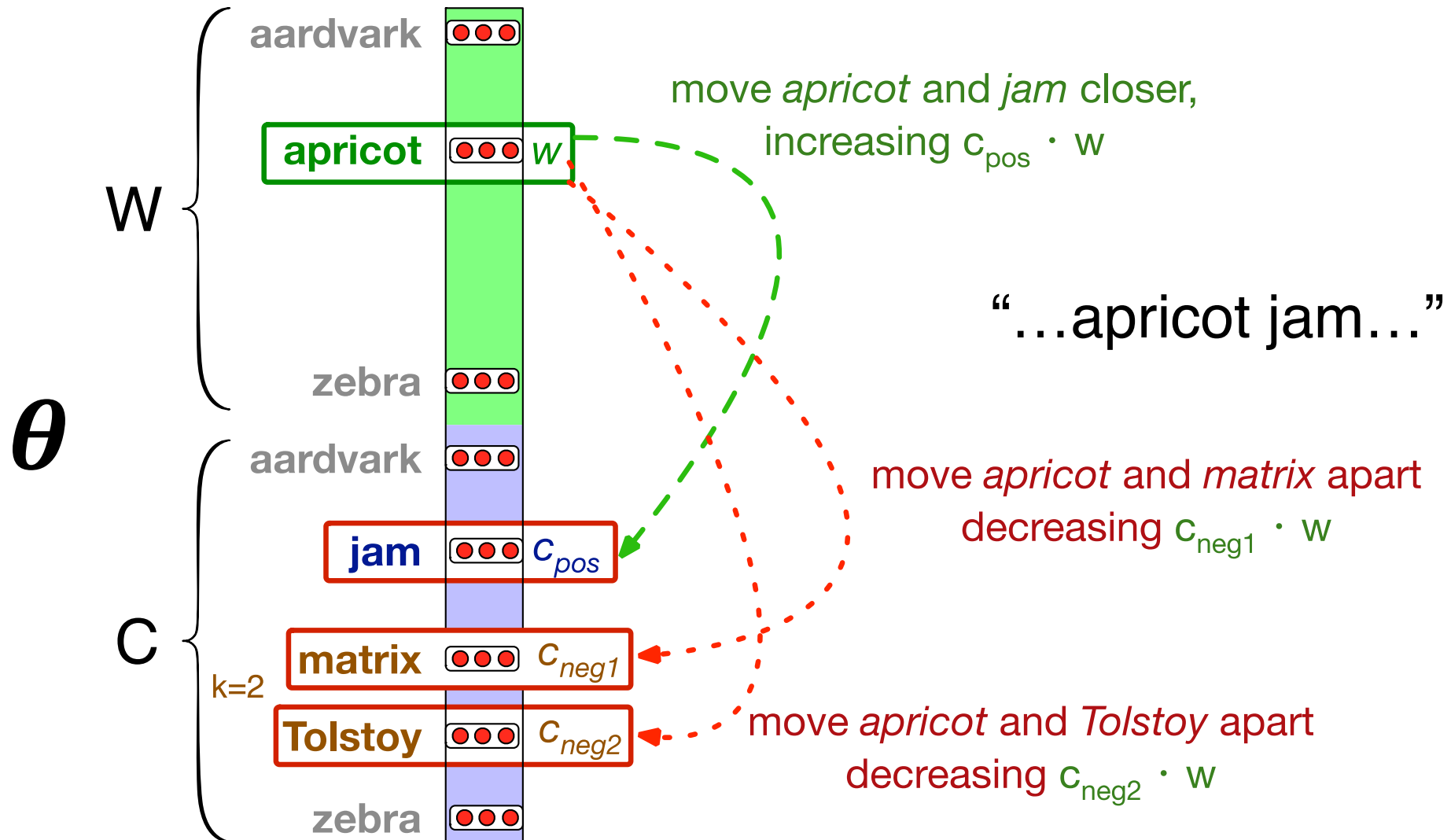
How to learn?

- 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.

Intuition of one step of gradient descent



Reminder: gradient descent

- At each step
 - Direction: We move in the reverse direction from the gradient of the loss function
 - Magnitude: we move the value of this gradient $\frac{d}{dw}L(f(x; w), y)$ weighted by a **learning rate** η
 - Higher learning rate means move w faster

$$w^{t+1} = w^t - \eta \frac{d}{dw}L(f(x; w), y)$$

The derivatives of the loss function

$$L_{CE} = - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1] w^t$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta [\sigma(c_{neg}^t \cdot w^t)] w^t$$

$$w^{t+1} = w^t - \eta \left[[\sigma(c_{pos} \cdot w^t) - 1] c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w^t)] c_{neg_i} \right]$$

Two sets of embeddings

SGNS learns two sets of embeddings

Target embeddings matrix W

Context embedding matrix C

It's common to just add them together,
representing word i as the vector $w_i + c_i$

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

Start with V random d -dimensional vectors as initial embeddings

Train a classifier based on embedding similarity

- 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.

Vector Semantics & Embeddings

Word2vec: Learning the
embeddings

Vector Semantics & Embeddings

Properties of Embeddings

The kinds of neighbors depend on window size

Small windows ($C = \pm 2$) : nearest words are syntactically similar words in same taxonomy

- *Hogwarts* nearest neighbors are other fictional schools
- *Sunnydale, Evernight, Blandings*

Large windows ($C = \pm 5$) : nearest words are related words in same semantic field

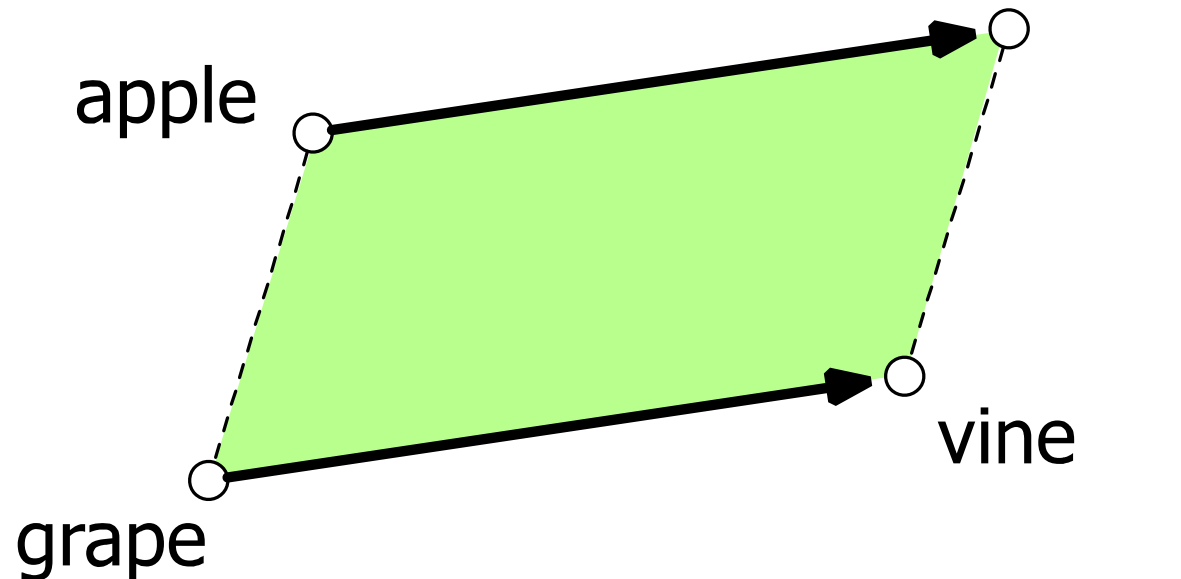
- *Hogwarts* nearest neighbors are Harry Potter world:
- *Dumbledore, half-blood, Malfoy*

Analogical relations

The classic parallelogram model of analogical reasoning
(Rumelhart and Abrahamson 1973)

To solve: "*apple is to tree as grape is to _____*"

Add $\overrightarrow{\text{tree} - \text{apple}}$ to $\overrightarrow{\text{grape}}$ to get *vine*



Analogical relations via parallelogram

The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

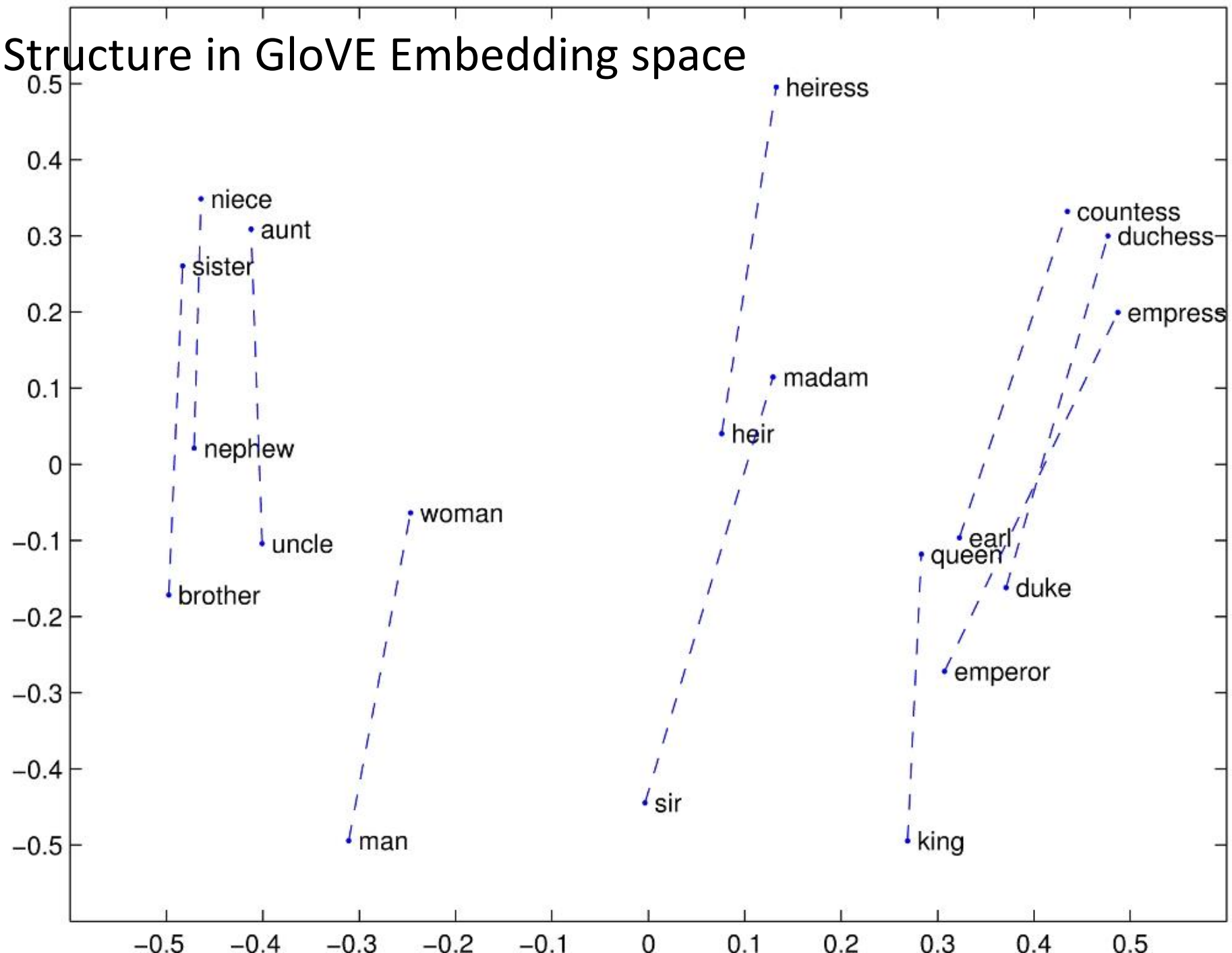
$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$

$\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ is close to $\overrightarrow{\text{Rome}}$

For a problem $a:a^*:b:b^*$, the parallelogram method is:

$$\hat{b}^* = \operatorname{argmax}_x \operatorname{distance}(x, a^* - a + b)$$

Structure in GloVe Embedding space



Caveats with the parallelogram method

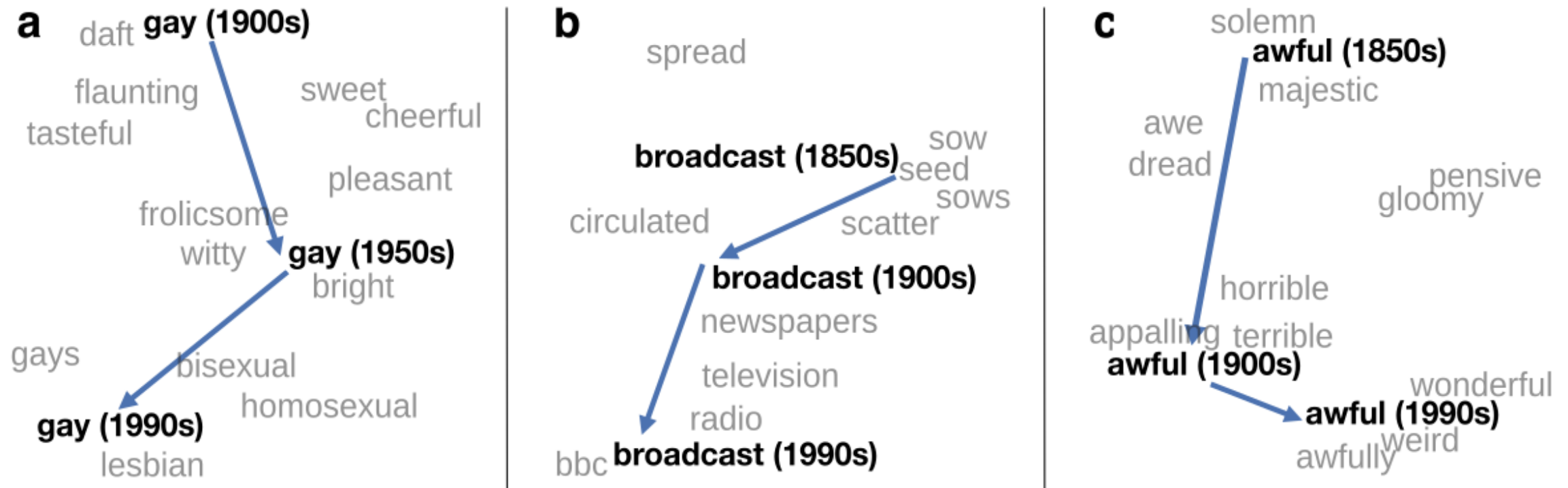
It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

Understanding analogy is an open area of research (Peterson et al. 2020)

Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

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



William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

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 *NeurIPS*, 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

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Historical embedding as a tool to study cultural biases

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences* 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
 - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical*) are biased toward men, a bias slowly decreasing 1960-1990
 - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s

Vector Semantics & Embeddings

Properties of Embeddings