

Word Vectors

Rina BUOY

[Source:](#)



AMERICAN UNIVERSITY
OF PHNOM PENH

STUDY LOCALLY. LIVE GLOBALLY.

Vector
Semantics &
Embeddings

Words and Vectors

One-hot Encoding

សាលារៀន


$$\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$
$$\in R^V$$

Term-document matrix

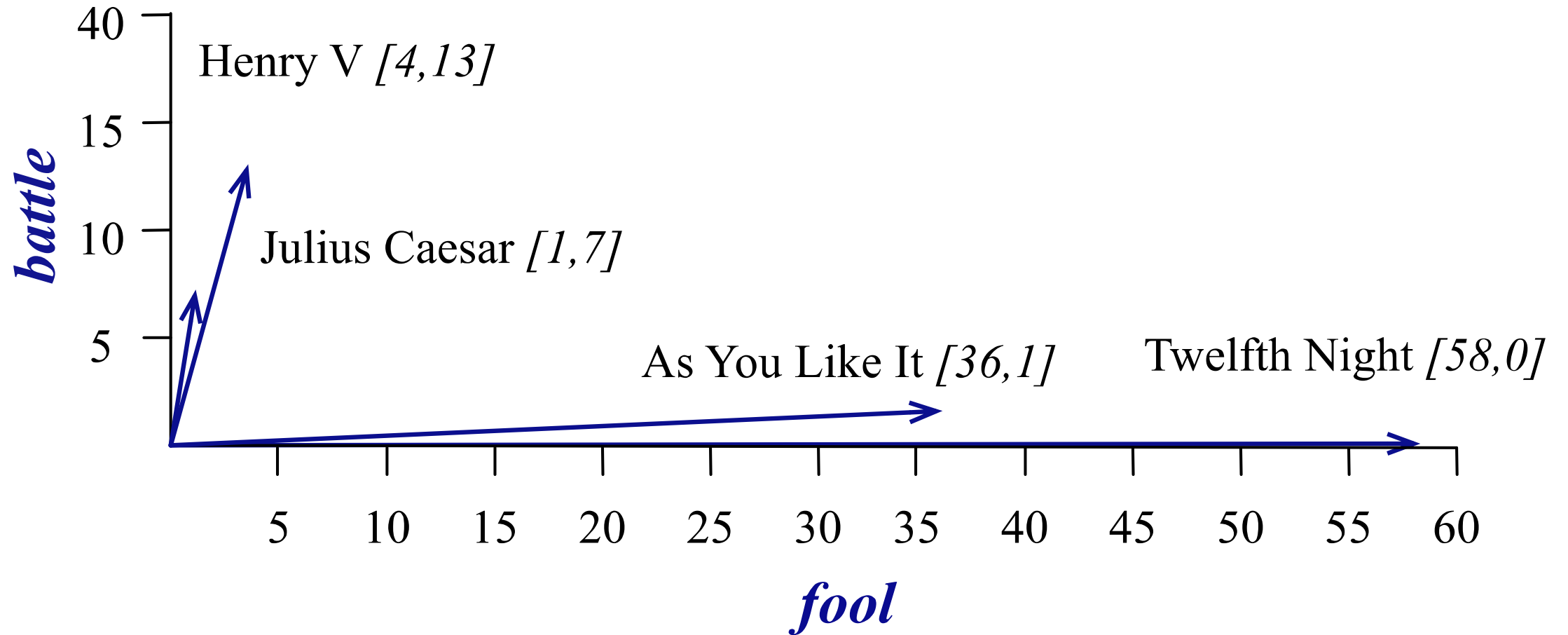
Each document is represented by a vector of words

	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

$[w_1, \dots, w_n]$


$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \dots + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

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"

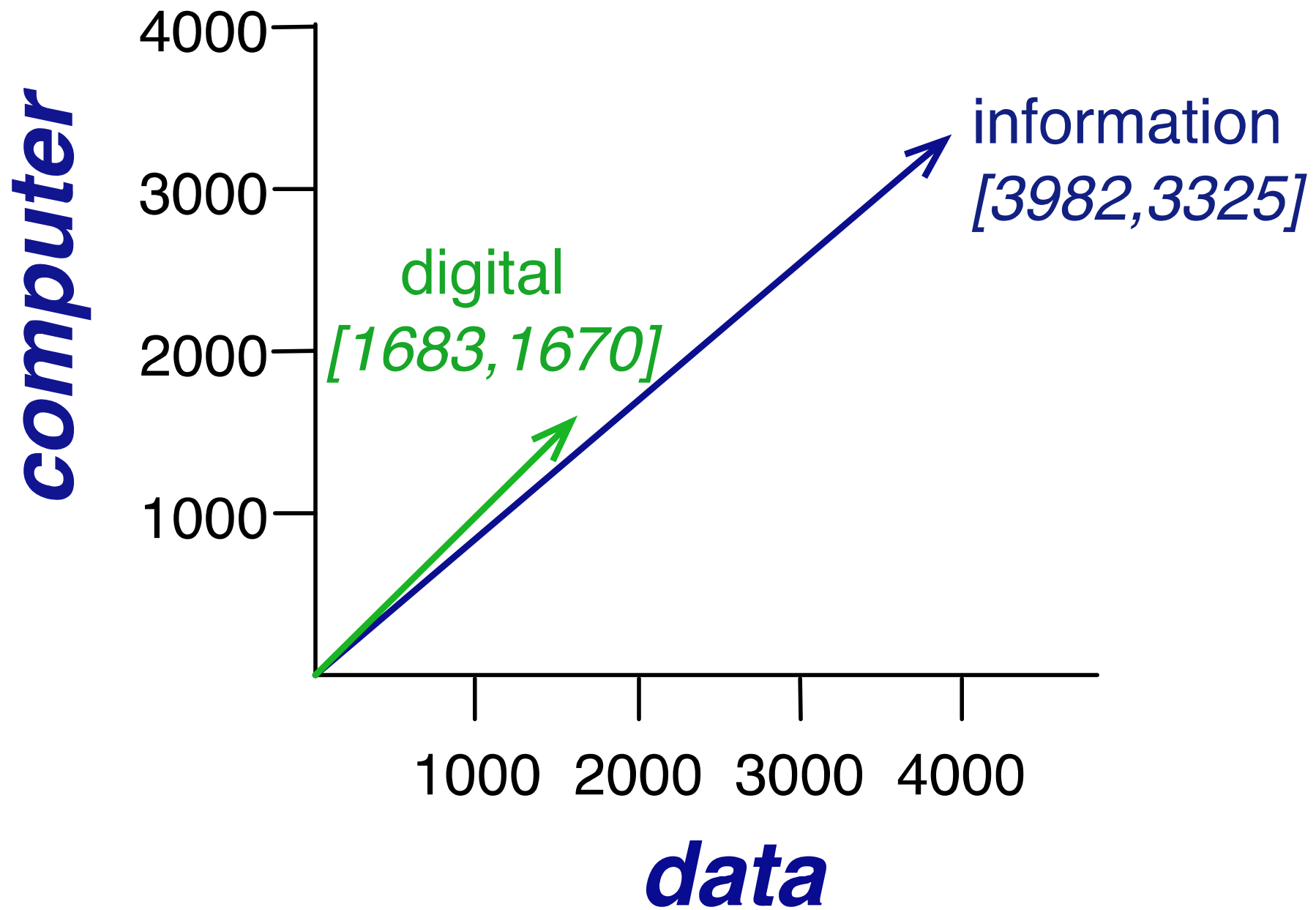
fool is "the kind of word that occurs in comedies, especially Twelfth Night"

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

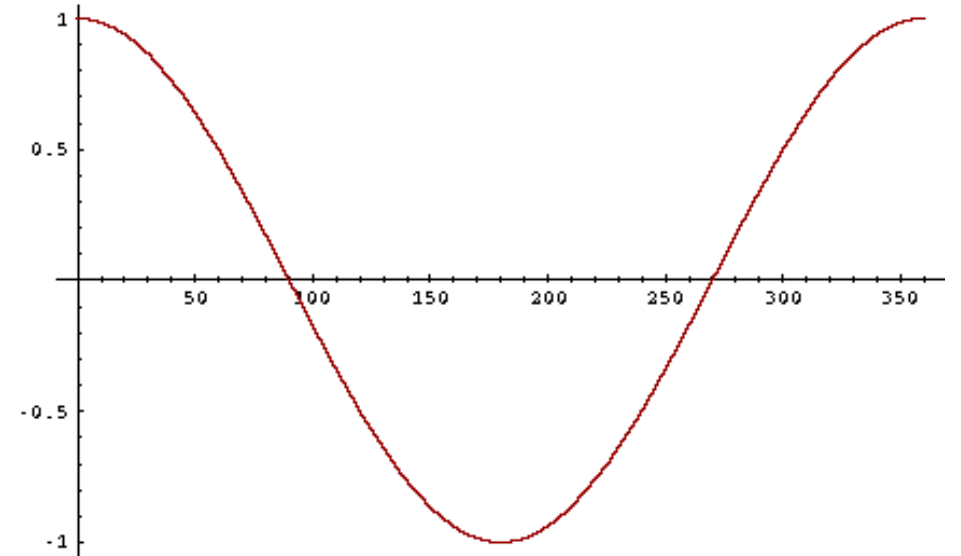
$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}}$$

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

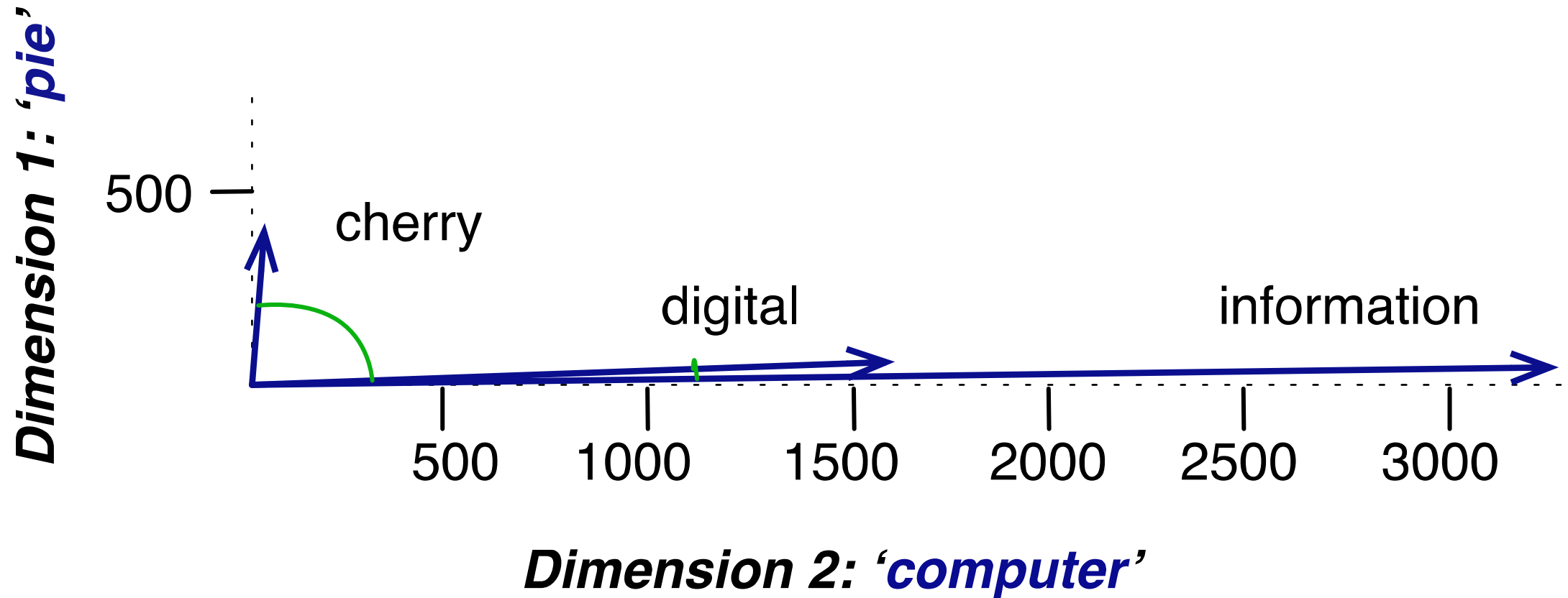
$$\cos(\text{cherry}, \text{information}) =$$

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

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

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

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

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

Vector Semantics & Embeddings

TF-IDF

Vector Semantics & Embeddings

Word2vec

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 (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)

Simple static embeddings you can download!

Word2vec (Mikolov et al)

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

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

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

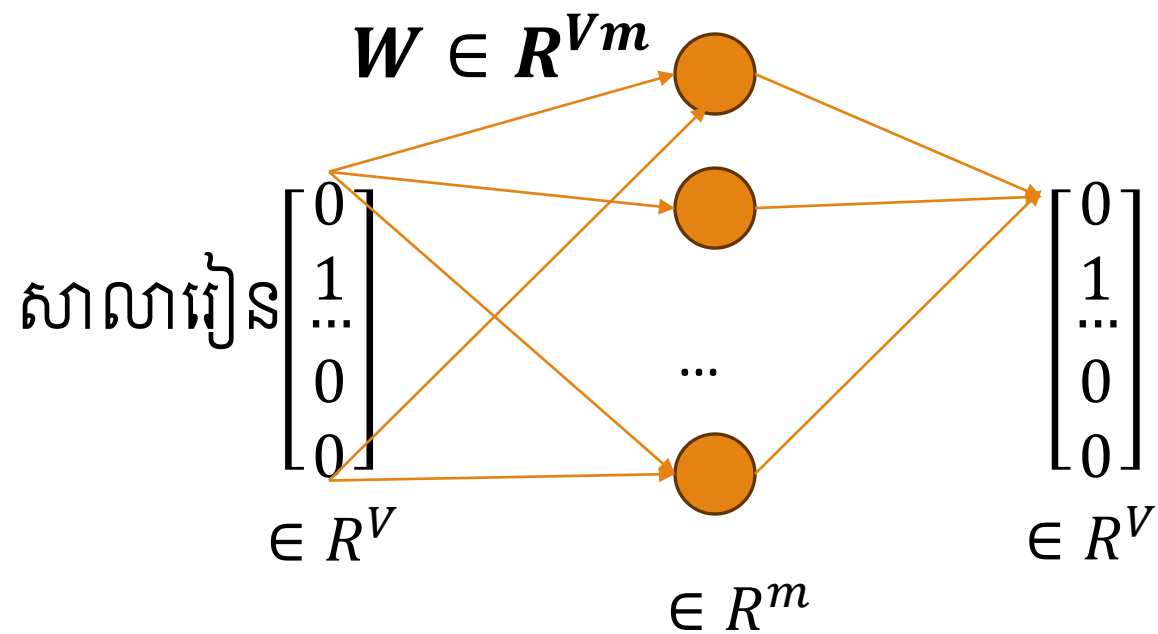
Big idea: **self-supervision**:

- 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

Model



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:

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

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

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.

Vector Semantics & Embeddings

Word2vec

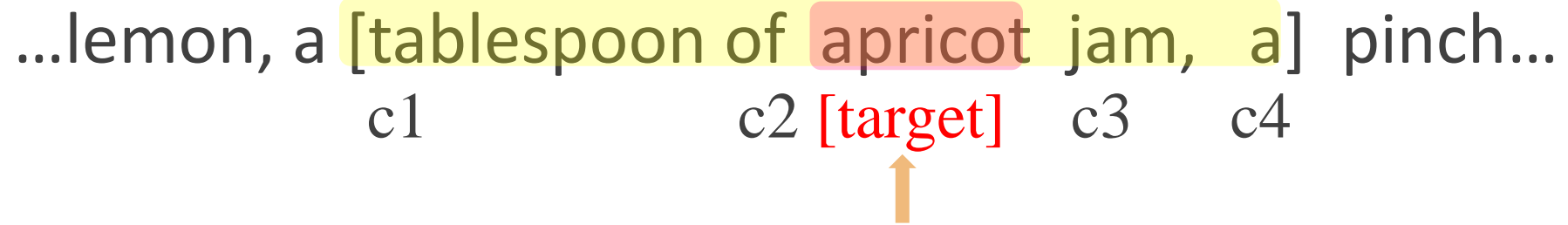
Vector Semantics & Embeddings

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
---------	----

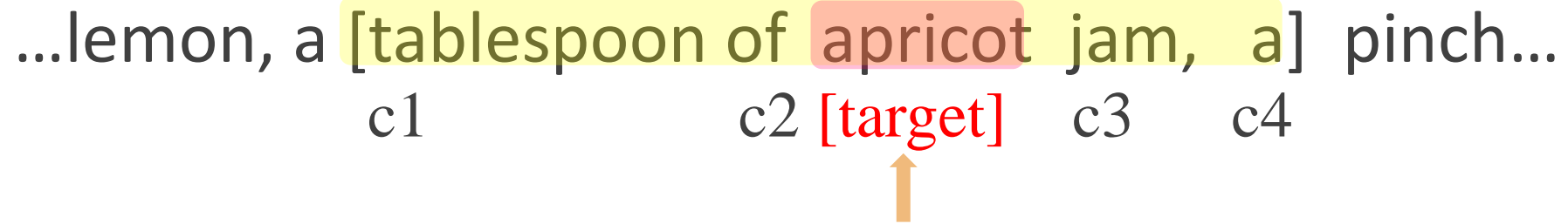
apricot	jam
---------	-----

apricot	a
---------	---

Skip-Gram Training data

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

c1 c2 [target] c3 c4



positive examples +

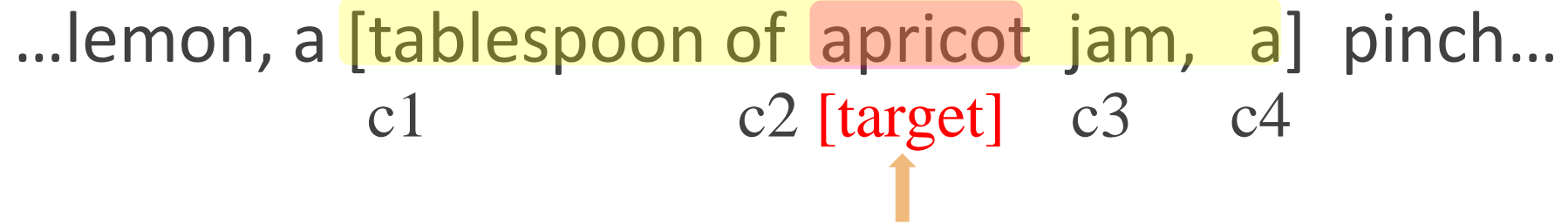
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

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.

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.

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

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

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

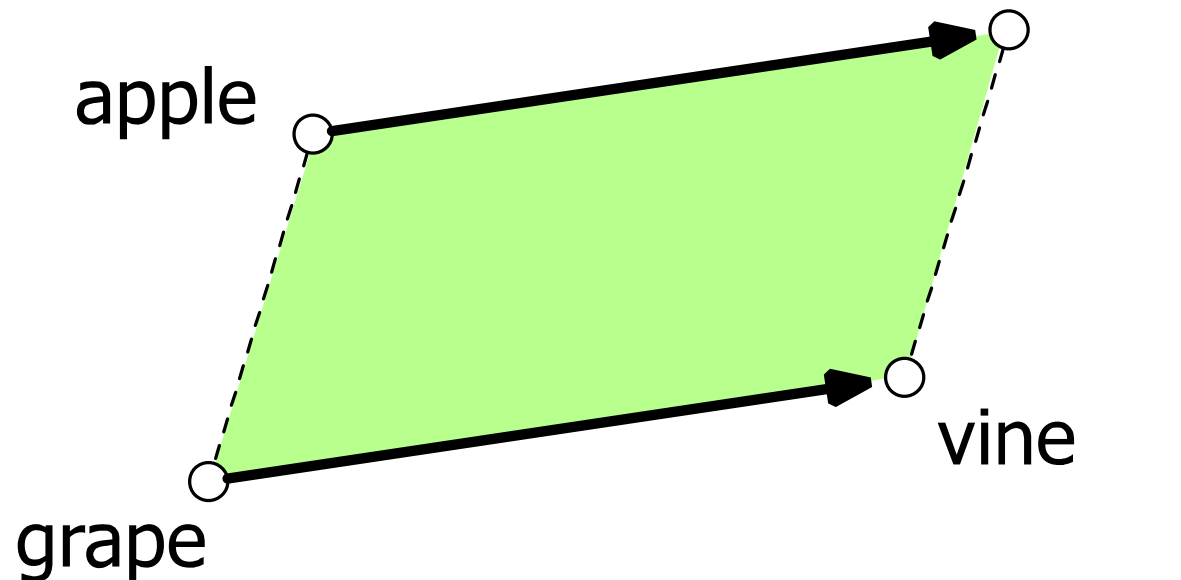
- *Hogwarts* nearest neighbors are Harry Potter world:
- *Dumbledore, half-blood, Malfoy*

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)

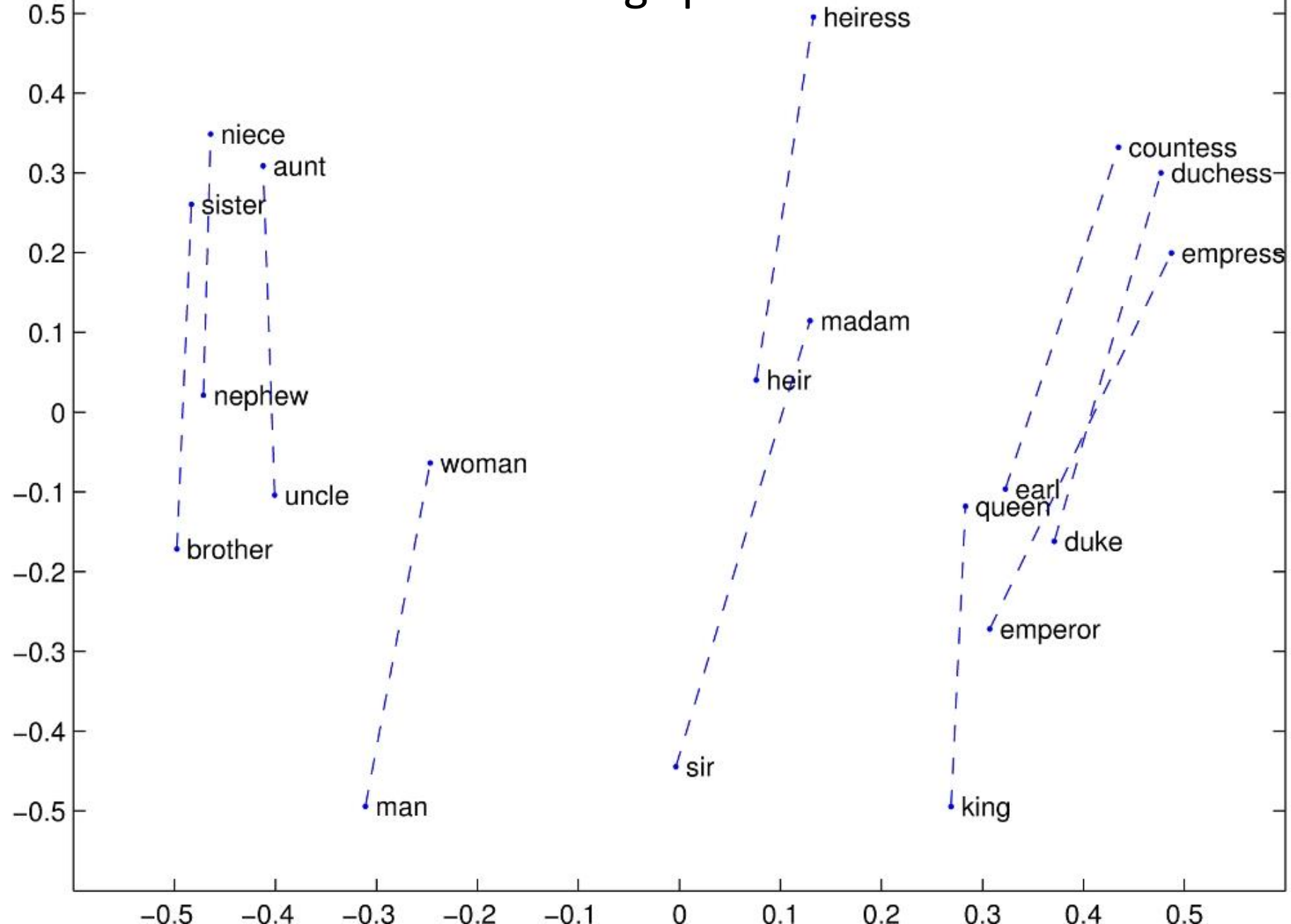
$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$

$\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ 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

Vector Semantics & Embeddings

Properties of Embeddings