Natural Language Processing

7: Lexical Semantics

en.wikipedia.org/wiki/Phaistos_Disc

Objectives

(1) Development and evaluation of language models

(2) Lexical Semantics

    (1) Word senses,

    (2) WordNet,

    (3) Word sense disambiguation

Word senses and WordNet

Word senses

(1) Words are ambiguous.

  (1) mouse1 : .... a mouse controlling a computer system in 1968.

  (2) mouse2 : .... a quiet animal like a mouse

  (3) bank1 : ...a bank can hold the investments in a custodial account ...

  (4) bank2 : ...as agriculture burgeons on the east bank, the river ...

Word senses

(1) Polysemy (from Greek ‘having many senses’, poly- ‘many’ + sema, ‘sign, mark’)

(2) A sense (or word sense) is a discrete representation of one aspect of the meaning of a word

(3) Distinction between POS ambiguity and polysemy.

How to define word senses?

(1) Two approaches:

  (1) glossaries / dictionaries / thesauri

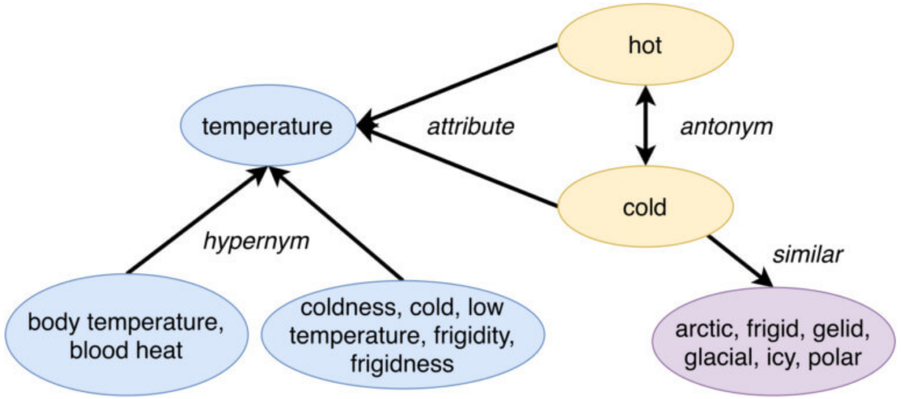
    (1) WordNets

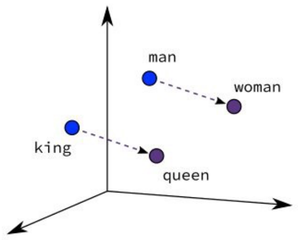
    (2) RuThes

  (2) embeddings (vectors)

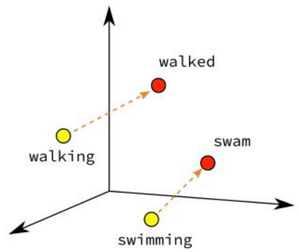
    (1) word2vec, GloVe, fasttext

    (2) ELMo, BERT

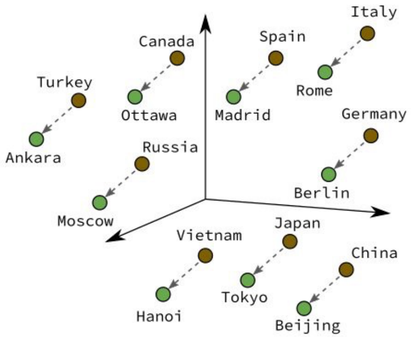




Male-Female



Verb Tense



Country-Capital

Glosses

(1) Definitions from glossary (for `bank'):

  (1) financial institution that accepts deposits and channels the money into lending activities

  (2) sloping land (especially the slope beside a body of water)

(2) They are not formal:

right	*adj.* located nearer the right hand esp. being on the right when facing the same direction as the observer.
left	*adj.* located nearer to this side of the body than the right.
red	*n.* the color of blood or a ruby.
blood	*n.* the red liquid that circulates in the heart, arteries and veins of animals.

How many senses do words have?

(1) Dictionaries tend to use many fine-grained senses

(2) Computationaly, we do not need many words senses

  (1) only those, that actually present in data...

  (2) clustering examples into senses, or senses into broader-grained categories

(18.1) They rarely *serve* red meat, preferring to prepare seafood.
(18.2) He *served* as U.S. ambassador to Norway in 1976 and 1977.
(18.3) He might have *served* his time, come out and led an upstanding life.

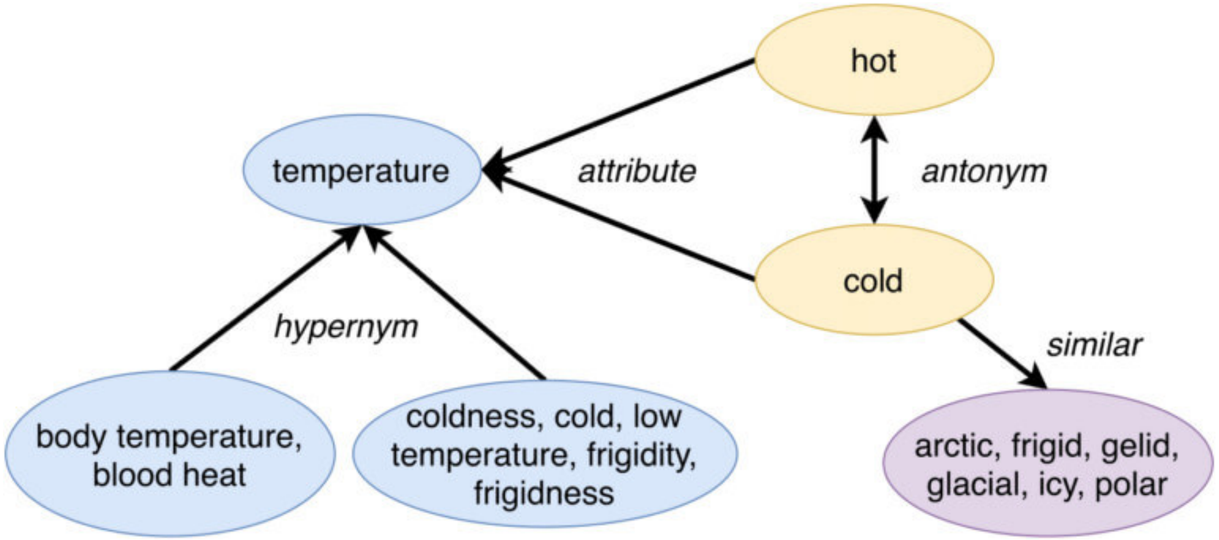
https://web.stanford.edu/~jurafsky/slp3/18.pdf

WordNet

(1) WordNet, a large online thesaurus —a database that represents word senses

(2) Synsets
set of near-synonyms

(3) Relations between synsets



WordNet

relations

(1) Synonymy

  (1) a relationship between senses rather than words (!)

  (2) forms synsets

(2) Antonymy

(3) Taxonomic relations

  (1) IS-A, part-whole

  (2) form hierarchies

Synonymy

Example: big and large

(1) How big is that plane?

(2) How large is that plane?

(5) Miss Nelson, for instance, became a kind of **big** sister to Benjamin.

(6) Miss Nelson, for instance, became a kind of **large** sister to Benjamin.

Antonymy

(1) Two senses can be antonyms if they define a binary opposition or are at opposite ends of some scale.

   (1) fast/slow cold/hot dark/light

   (2) black vs. white ?

(2) A group of antonyms, **reversives**, describe change or movement in opposite directions, such as *rise/fall or up/down*

(3) Thus, antonyms are **close** to synonyms :)

Taxonomic Relations

(1) hyponym / hypernym

  (1) mango is a *hyponym* of fruit

  (2) vehicle is a *hypernym* of car

(2) Formal structure, IS-A

  (1) $\forall x : A(x) \rightarrow B(x)$

  (2) + transitivity

Meronymy

part-whole

(1) A leg **is part of** a chair;

(2) a wheel **is part of** a car.

We say that wheel is a **meronym** of car, and car is a **holonym** of wheel.

(4) Can we form a hierarchy?

WordNet

https://wordnet.princeton.edu/download/current-version

(1) The WordNet 3.0 release has
    (1) 117,798 nouns,
    (2) 11,529 verbs,
    (3) 22,479 adjectives, and
    (4) 4,481 adverbs.
(2) The average noun has 1.23 senses, and
(3) The average verb has 2.16 senses.

Examples

(1) Synset:

{chump^{1}, fool^{2}, gull^{1}, mark^{9}, patsy^{1}, fall guy^{1}, sucker^{1}, soft touch^{1}, mug^{2}}

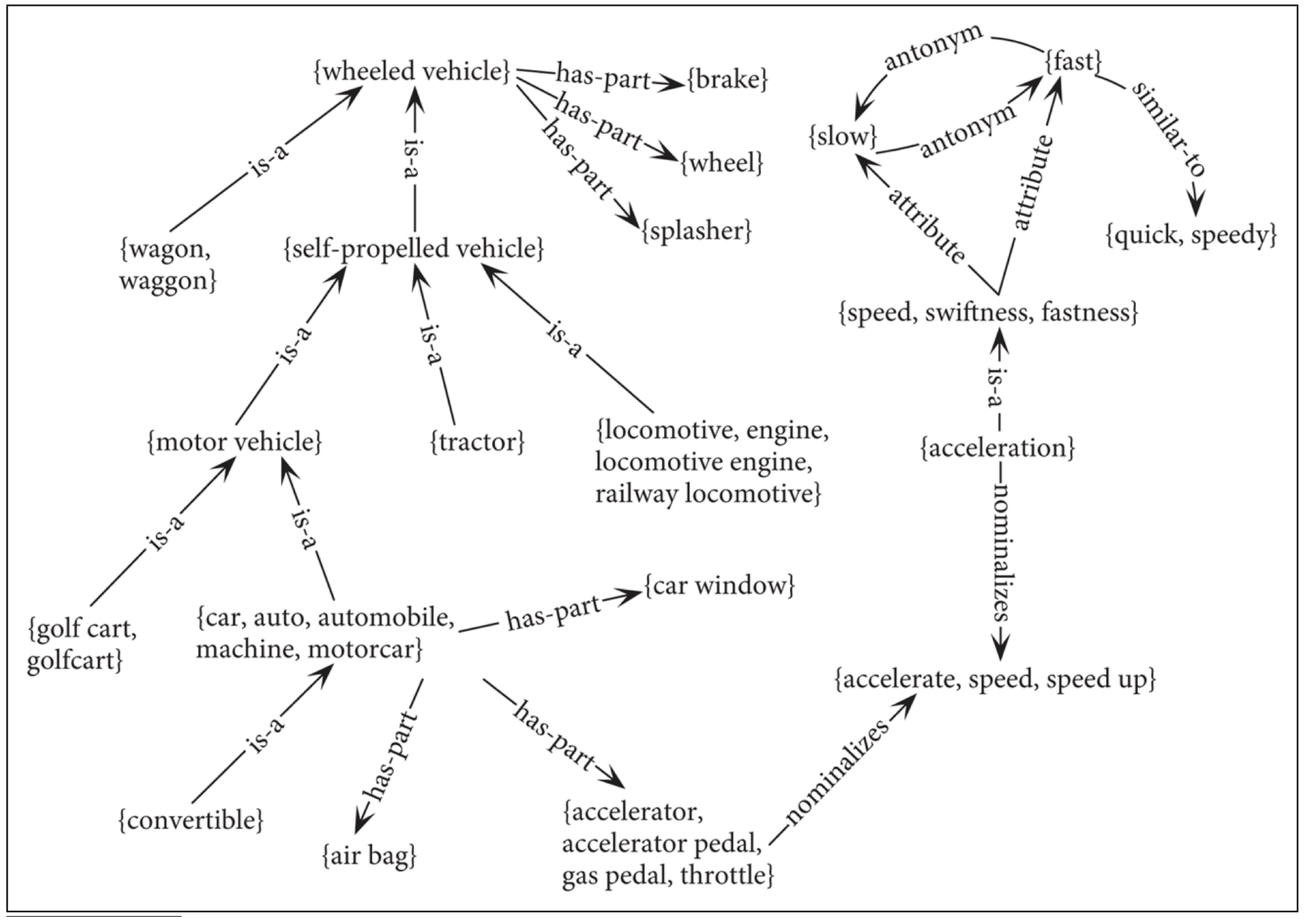
The gloss of this synset describes it as:

Gloss: a person who is gullible and easy to take advantage of.

(2) Sense relations

Relation	Also Called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	*breakfast*^{1} $\rightarrow$ *meal*^{1}
Hyponym	Subordinate	From concepts to subtypes	*meal*^{1} $\rightarrow$ *lunch*^{1}
Instance Hypernym	Instance	From instances to their concepts	*Austen*^{1} $\rightarrow$ *author*^{1}
Instance Hyponym	Has-Instance	From concepts to their instances	*composer*^{1} $\rightarrow$ *Bach*^{1}
Part Meronym	Has-Part	From wholes to parts	*table*^{2} $\rightarrow$ *leg*^{3}
Part Holonym	Part-Of	From parts to wholes	*course*^{7} $\rightarrow$ *meal*^{1}
Antonym		Semantic opposition between lemmas	*leader*^{1} $\Longleftrightarrow$ *follower*^{1}
Derivation		Lemmas w/same morphological root	*destruction*^{1} $\Longleftrightarrow$ *destroy*^{1}

WordNet as a graph



Navigli, R. 2016. Chapter 20. ontologies. In R. Mitkov, ed- itor, The Oxford handbook of computational linguistics. Oxford University Press.

Word Sense Disambiguation

WSD

(1) The **all-words** task: disambiguate all words

  (1) e.g., using the SemCor corpus with 226,036 words, manually tagged with WordNet

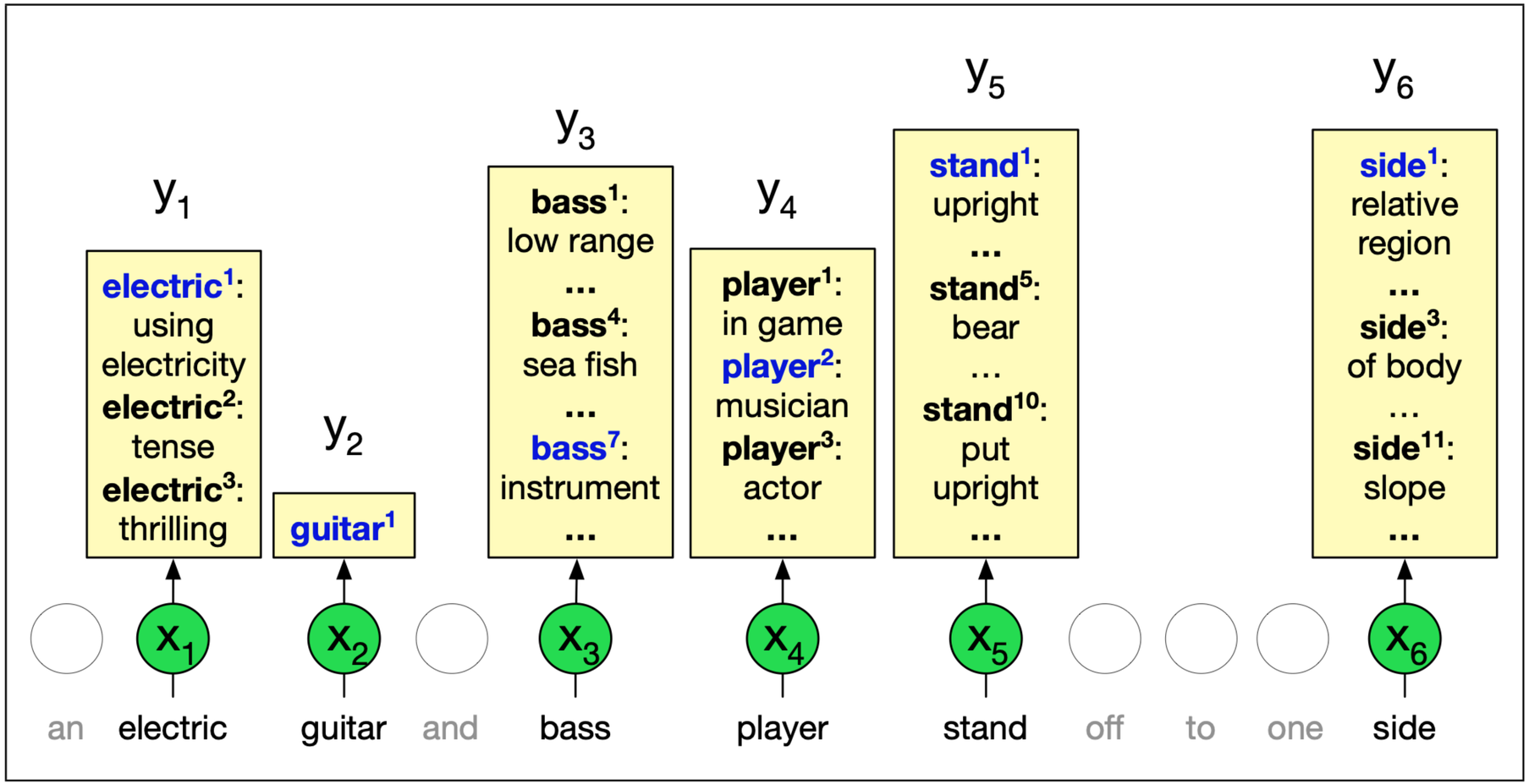
  (2) or the SemEval sense-tagged corpora

(2) Example of annotations:

$$\text{You will find}^{9}_{v} \text{ that avocado}^{1}_{n} \text{ is}^{1}_{v} \text{ unlike}^{1}_{j} \text{ other}^{1}_{j} \text{ fruit}^{1}_{n} \text{ you have ever}^{1}_{r} \text{ tasted}^{2}_{v}$$

All-words task view

senses from WordNet



WSD Baselines

(1) A surprisingly strong baseline is simply to choose **the most frequent sense** for each word from the senses in a labeled corpus

(2) Anjther heuristic is **one sense per discourse:** a word appearing multiple times in a text or discourse often appears with the same sense.

(3) The Lesk Algorithm as WSD Baseline

The Simplified Lesk algorithm

(1) The Lesk algorithm is the oldest and most powerful knowledge-based WSD method

(2) No labeled data needed, as it is a **knowledge-based** algorithm

(3) **function** SIMPLIFIED LESK(*word*, *sentence*) **returns** best sense of *word*

```
best-sense ← most frequent sense for word
max-overlap ← 0
context ← set of words in sentence
for each sense in senses of word do
  signature ← set of words in the gloss and examples of sense
  overlap ← COMPUTEOVERLAP(signature, context)
  if overlap > max-overlap then
        max-overlap ← overlap
        best-sense ← sense
end
return(best-sense)
```

https://web.stanford.edu/~jurafsky/slp3/18.pdf

Example

(1) The bank can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

(2) The second sense has less overlap with words from the context

bank[1]	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank[2]	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

Break

Word Embeddings

But what is an embedding?

(1) Let us you have an attribute of categorical type
    (1) say, it has 4 values (a,b,c,d)
    (2) how to represent this attribute in input of a ML method?
    (3) represent it as 4 binary vectors (one-hot): each vector is 3-dimensional (!)
        (1) a = (0,0,0), b = (0,0,1), c = (0,1,0), d = (1,0,0)
    (4) represent it as 1 vector (with real numbers):
        (1) a = (-1,-1); b = (1,-1); c = (-1,1); d = (1,1)
        (2) embeddings of attribute into a lower 2-dimensional space

Vectors for Words

(1) Binary 1-hot vectors that encode words have high dimensionality ((|V|))

(2) **Sparse vectors** that encode words in TF*IDF also have high dimensionality (number of documents in a collection)

(3) Dense vectors have dimensionality of 100-1000. Their advantages:

    (1) lower memory footprint

    (2) faster computation, less weights

    (3) useful distributional properties

We define meaning of a word as a vector

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

(2) The standard way to represent meaning in NLP

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

(3) Fine-grained model of meaning for similarity

Vector Semantics

Computational models of word meaning

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

(2) Vector semantics

  (1) is the standard model in language processing

  (2) handles many of our goals!

Ludwig Wittgenstein

(1) PI #43:

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

Let's define words by their usages

(1) One way to define "usage":
words are defined by their environments (the words around them)

(3) Zellig Harris (1954):
If A and B have almost identical environments we say that they are synonyms.

Idea 1: Meaning as a point in space (Osgood et al. 1957)

(1) 3 affective dimensions for a word
    (1) valence: pleasantness
    (2) arousal: intensity of emotion
    (3) dominance: the degree of control exerted

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

	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

Idea 2: Defining meaning by linguistic distribution

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

Defining meaning as a point in space based on distribution

(1) Each word = a vector (not just "good" or "w45")

(2) Similar words are "nearby in semantic space"

(3) We build this space automatically by seeing which words are nearby in text

We'll discuss 2 kinds of embeddings

(1) tf-idf

(1) ○Information Retrieval workhorse!

(2) ○A common baseline model

(3) ○Sparse vectors

(4) ○Words are represented by (a simple function of) the counts of nearby words

(2) Word2vec

(1) ○Dense vectors

(2) ○Representation is created by training a classifier to predict whether a word is likely to appear nearby

(3) ○Later we'll discuss extensions called contextual embeddings

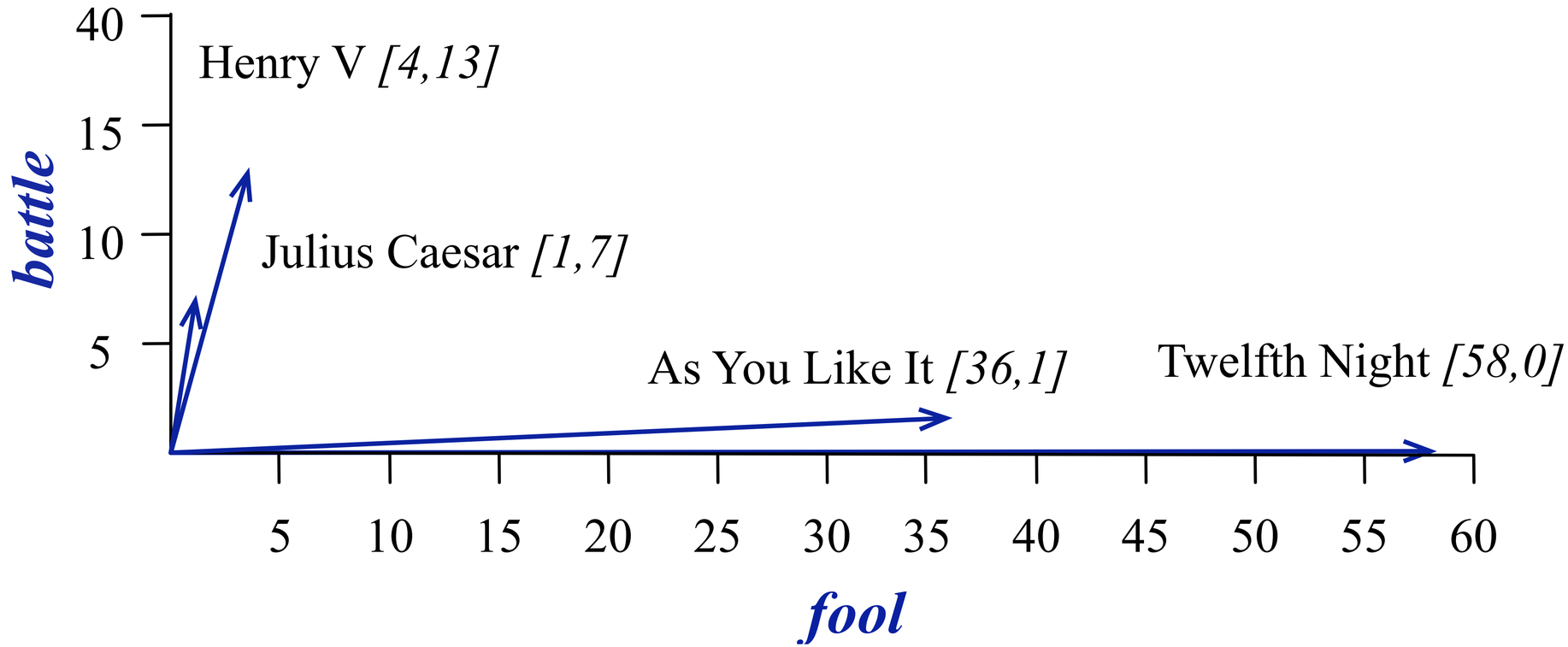
Words and Vectors

Term-document matrix

(1) 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

(1) Vectors are similar for the two comedies

(2) But comedies are different than the other two books
Comedies have more fools and wit and fewer battles.

	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

Idea for word meaning: Words can be vectors too!!!

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

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

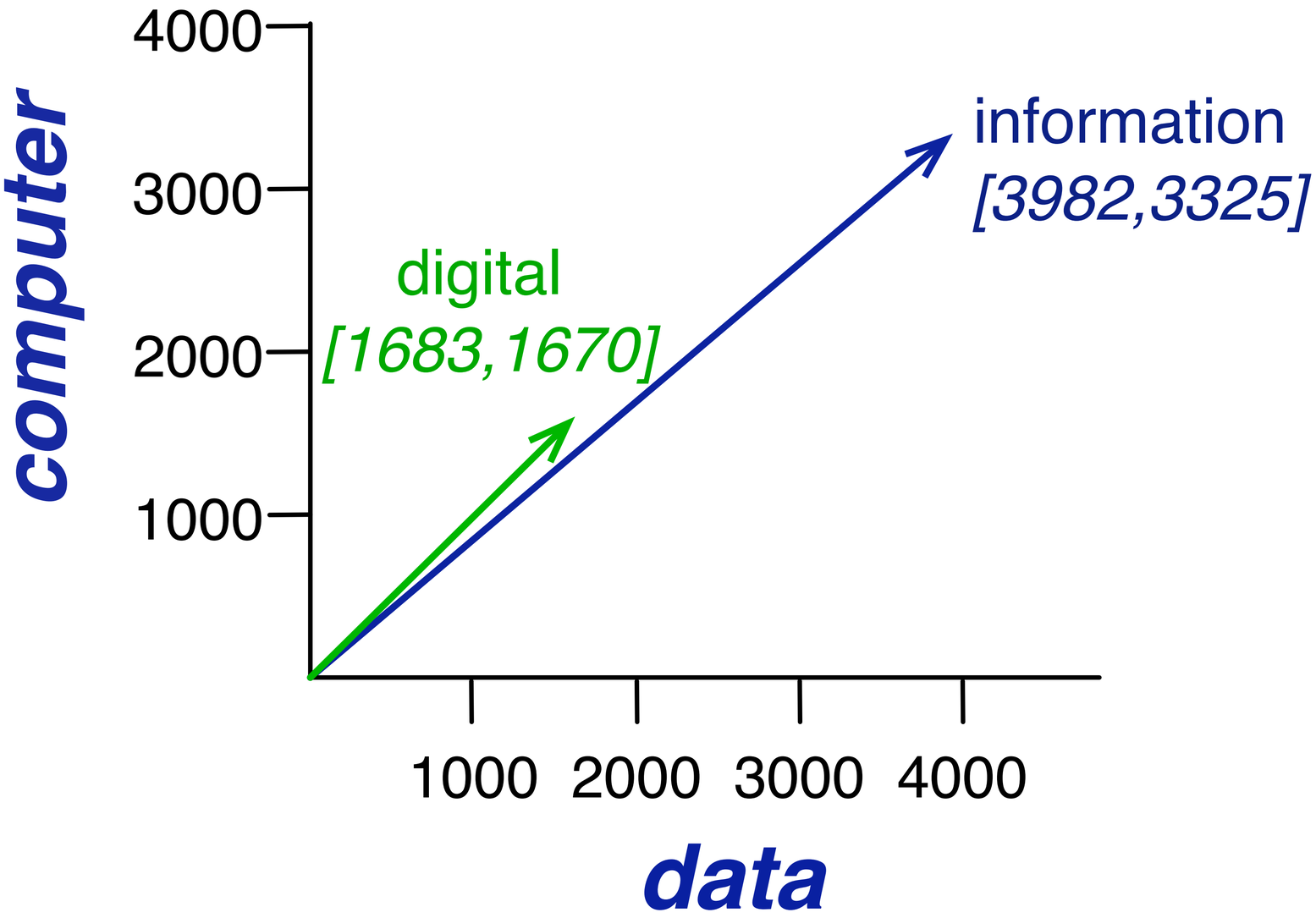
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word-word matrix (or "term-context matrix")

(1) 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	...


4000
3000
2000
1000
computer
information
[3982,3325]
digital
[1683,1670]
1000
2000
3000
4000
data

Cosine for computing word similarity

Computing word similarity: Dot product and cosine

(1) The dot product between two vectors is a scalar:

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

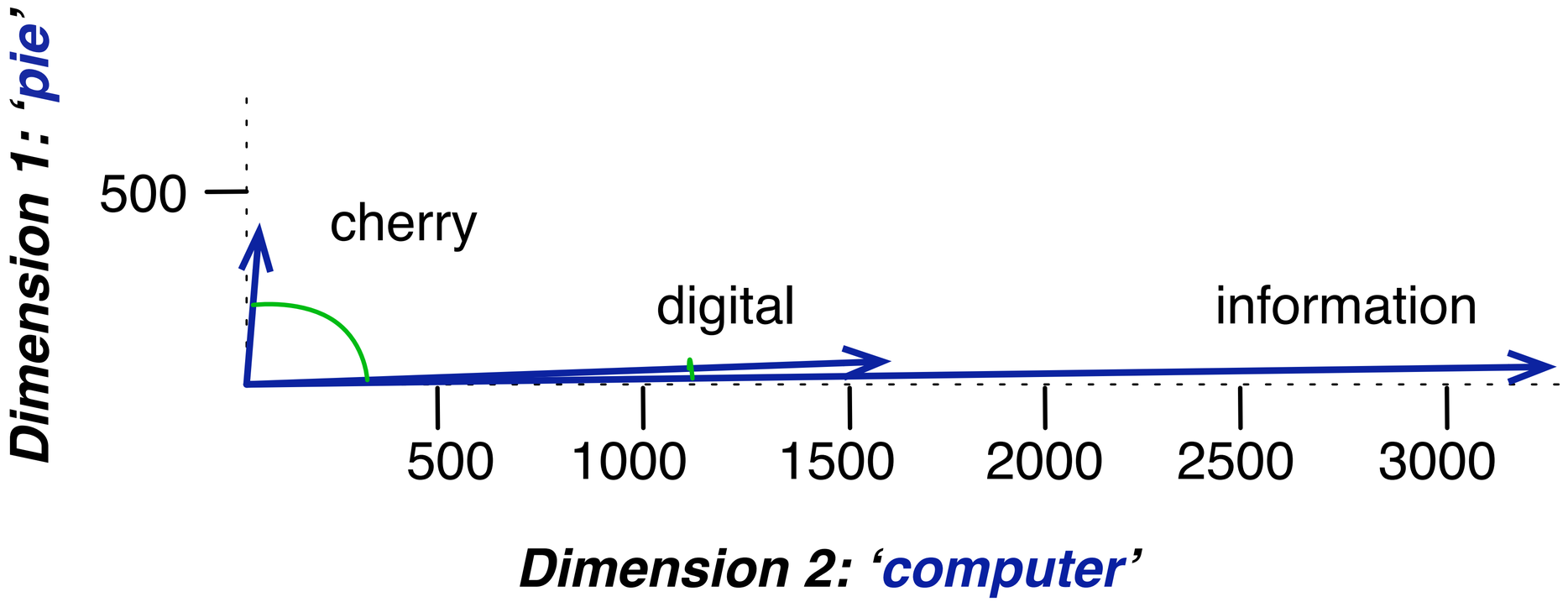
The dot product tends to be high when the two vectors have large values in the same dimensions

Dot product can thus be a useful similarity metric between vectors

Cosine for computing word similarity

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

Visualizing cosines (well, angles)



TF-IDF

Raw frequency is a bad representation

(1) The co-occurrence matrices we have seen represent each cell by word frequencies.

(2) Frequency is clearly **useful**; if *sugar* appears a lot near *apricot*, that's useful information.

(3) But overly frequent words like *the, it, or they* are not very informative about the context

(4) **It's a paradox!** How can we balance these two conflicting constraints?

Two common solutions for word weighting

(1) 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

(2) 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)

(1) $tf_{t,d} = count(t, d)$

(3) Instead of using raw count, we squash a bit:

$$tf_{t,d} = \log_{10}(count(t, d) + 1)$$

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

Final tf-idf weighted value for a word

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \quad (1)$$

	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

Final tf-idf weighted value for a word

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t \quad (1)$$

	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	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

PPMI

Pointwise Mutual Information

(1) Pointwise mutual information between **events**:

(1) Do events x and y co-occur more than if they were independent?

$$\text{PMI}(X,Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

Pointwise Mutual Information

(1) Pointwise mutual information between events:

  (1) Do events x and y co-occur more than if they were independent?

$$\text{PMI}(X,Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

(2) PMI between two **words**: (Church & Hanks 1989)

  (1) Do words x and y co-occur more than if they were independent?

$$PMI(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

Positive Pointwise Mutual Information

PPMI

(1) PMI ranges from -∞ "to" +∞

(2) But the negative values are problematic

  (1) Things are co-occurring less than we expect by chance (What?)

  (2) Unreliable without enormous corpora

(3) Replace negative PMI values by 0

(4) **Positive** PMI (PPMI) :

$$PPMI(word_1, word_2) = max(0, \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)})$$

Computing PPMI on a term-context matrix

Matrix F with W rows (words) and C columns (contexts)

f_{ij} is # of times w_i occurs in context c_j

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

	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

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}} \qquad ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

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

	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

p(w=information,c=data) = 3982/111716 = .3399

p(w=information) = 7703/11716 = .6575

p(c=data) = 5673/11716 = .4842

$$p(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{N} \qquad p(c_j) = \frac{\sum_{i=1}^{W} f_{ij}}{N}$$

	p(w,context)					p(w)
	computer	**data**	**result**	**pie**	**sugar**	**p(w)**
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

	p(w,context)					p(w)
	computer	**data**	**result**	**pie**	**sugar**	**p(w)**
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

pmi(information,data) = $\log_2$ (.3399 / (.6575*.4842)) = .0944

Resulting PPMI matrix (negatives replaced by 0)

	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

Weighting PMI

(1) PMI is biased toward infrequent events

    (1) Very rare words have very high PMI values

(2) Two solutions:

    (1) Give rare words slightly higher probabilities

    (2) Use add-one smoothing (which has a similar effect)

Weighting PMI

Giving rare context words slightly higher probability

Raise the context probabilities to $\alpha = 0.75$:

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

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

This helps because $P_\alpha(c) > P(c)$ for rare c

Consider two events, P(a) = .99 and P(b)=.01

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75}+.01^{.75}} = .97 \quad P_\alpha(b) = \frac{.01^{.75}}{.01^{.75}+.01^{.75}} = .03$$

Word2vec

Sparse versus dense vectors

(1) tf-idf (or PMI) vectors are
    (1) long (length |V|= 20,000 to 50,000)
    (2) sparse (most elements are zero)
(2) Alternative: learn vectors which are
    (1) short (length 50-1000)
    (2) dense (most elements are non-zero)

Common methods for getting short dense vectors

(1) “Neural Language Model”-inspired models
    (1) Word2vec (skipgram, CBOW), GloVe
(2) Singular Value Decomposition (SVD)
    (1) A special case of this is called LSA – Latent Semantic Analysis
(3) Alternative to these "static embeddings":
    (1) Contextual Embeddings (ELMo, BERT)
    (2) Compute distinct embeddings for a word in its context
    (3) Separate embeddings for each subtoken of a word

Simple static embeddings you can download!

... and you already applied them

(1) Word2vec (Mikolov et al)
https://code.google.com/archive/p/word2vec/

(2) GloVe (Pennington, Socher, Manning)
http://nlp.stanford.edu/projects/glove/

(3) fasttext
https://fasttext.cc

Word2vec

(1) Popular embedding method

(2) Very fast to train

(3) Code available on the web

(4) Idea: **predict** rather than **count**

(5) Word2vec provides various options.
SGNS: skip-gram with negative sampling

Word2vec

(1) Instead of counting how often each word w occurs near “apricot"

   (1) Train a classifier on a binary prediction task:

   (2) Is w likely to show up near "apricot"?

(2) We'll take the learned classifier weights as the word embeddings

(3) **Big idea: self-supervision:**
A word **c** that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning

   (1) No need for human labels

   (2) 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:

- 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

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

To turn this into a probability

We'll use the sigmoid from logistic regression:

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

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

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

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

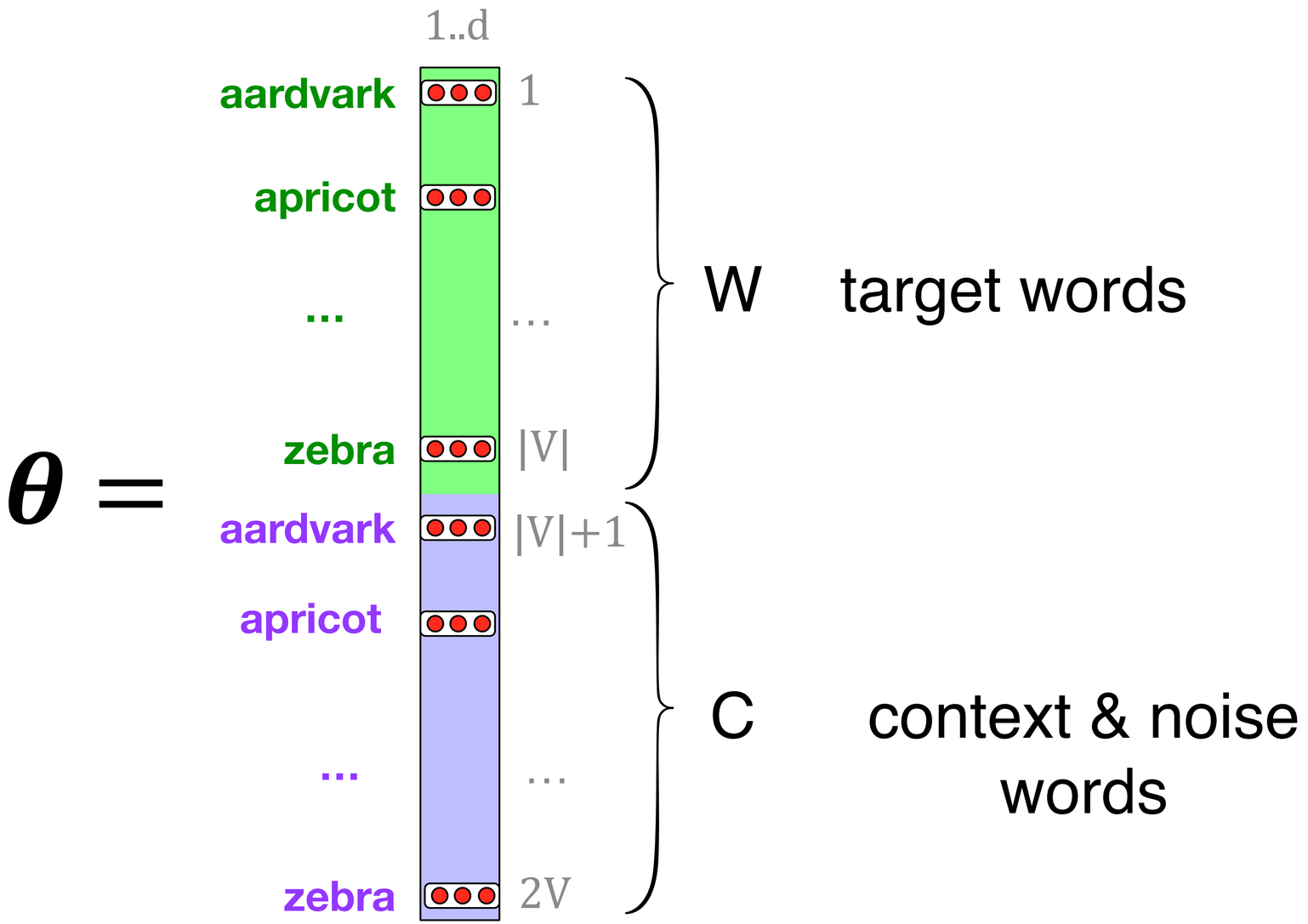
A probabilistic classifier, given

- 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



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

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

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

  (1) Maximize the similarity of the target word, context word pairs (w , C_{pos}) drawn from the positive data

  (2) Minimize the similarity of the (w , C_{neg}) pairs drawn from the negative data.

Loss function for one w with $c_{pos}, c_{neg_1}, \ldots, c_{neg_k}$

(1) 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.

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

Training the classifier

(1) How to learn?
Stochastic gradient descent!

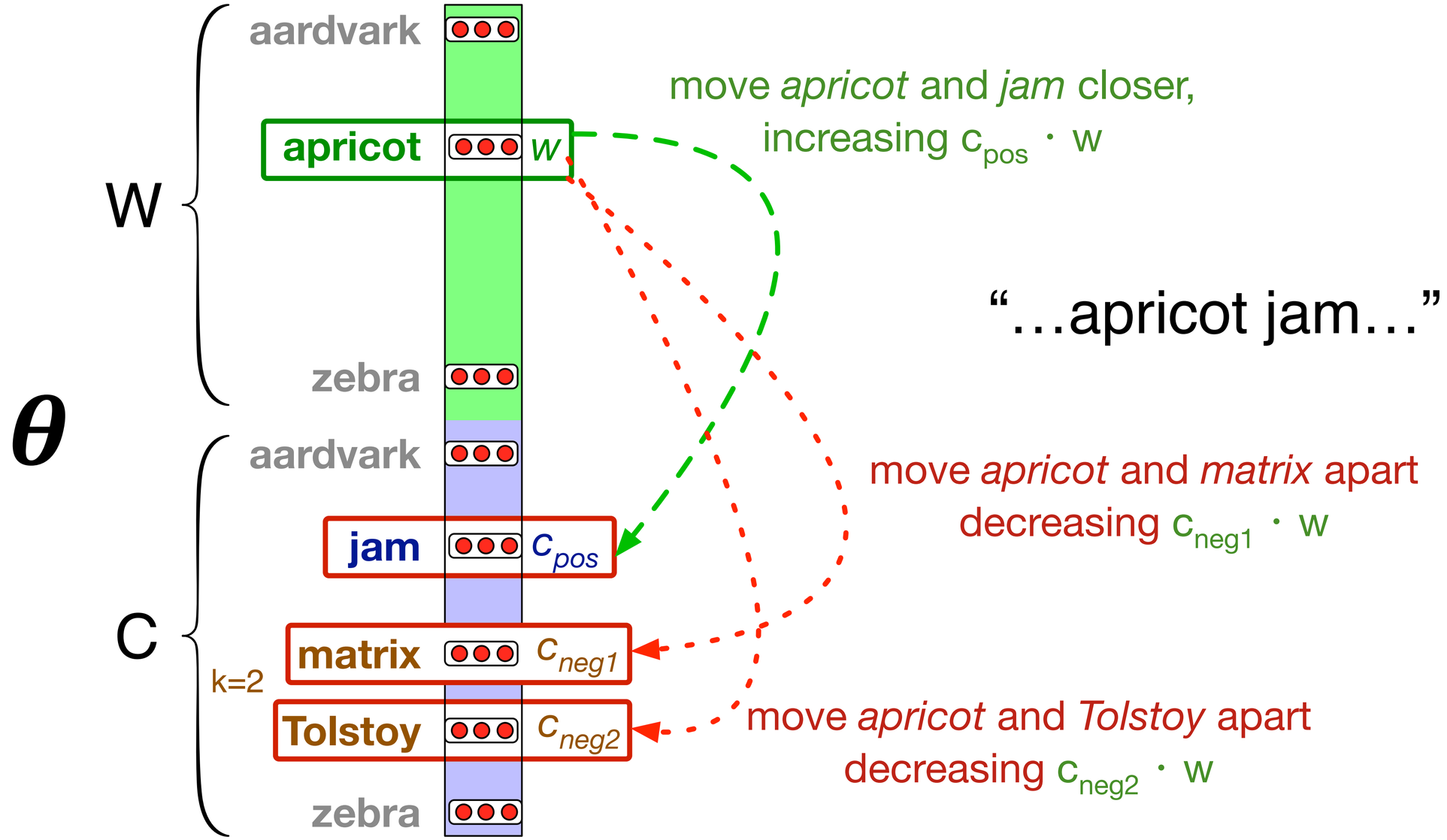
(3) We'll adjust the word weights to

(1) make the positive pairs more likely

(2) and the negative pairs less likely,

(3) 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

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

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

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

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

Update equation in SGD

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

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \eta \left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i} \right]
\end{aligned}
$$

Two sets of embeddings

(1) SGNS learns two sets of embeddings

  (1) Target embeddings matrix W

  (2) Context embedding matrix C

(2) 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

(1) Start with |V| random d-dimensional vectors as initial embeddings

(2) Train a classifier based on embedding similarity

  (1) Take a corpus and take pairs of words that co-occur as positive examples

  (2) Take pairs of words that don't co-occur as negative examples

  (3) Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance

  (4) Throw away the classifier and keep the embeddings.

Properties of Embeddings

The kinds of neighbors depend on window size

Small windows (C= +/- 2) : nearest words are syntactically similar words in same taxonomy

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

Large windows (C= +/- 5) : nearest words are related words in same semantic field

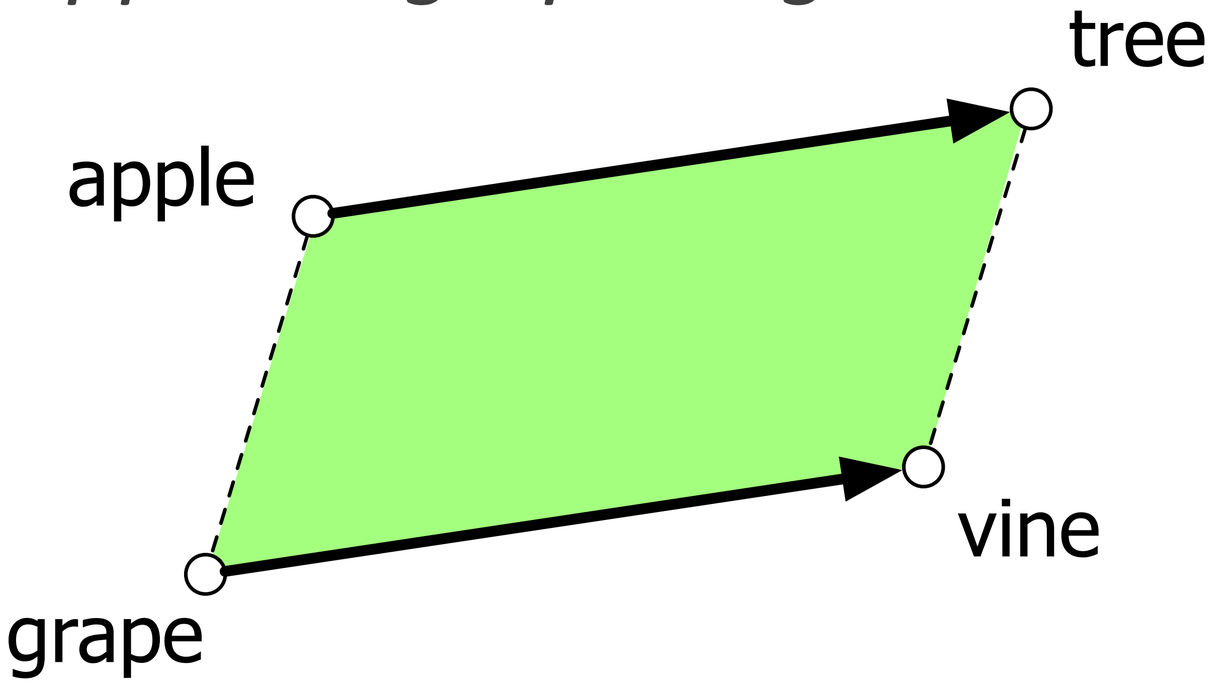
- *Hogwarts* nearest neighbors are Harry Potter world:
  - *Dumbledore, half-blood, Malfoy*

Analogical relations

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

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

Add $\overrightarrow{tree}$ – $\overrightarrow{apple}$ to $\overrightarrow{grape}$ to get $\overrightarrow{vine}$



Analogical relations via parallelogram

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

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

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

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

$$\hat{b}* = \arg\min_{x} distance(x, a* - a + b)$$

Caveats with the parallelogram method

(1) 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)

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

Demo

projector.tensorflow.org