CS 463 NATURAL LANGUAGE PROCESSING

Dr. Saleh Haridy

2022-2023

Vector Semantic and embeddings

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

- How do we represent the meaning of words in NLP systems?

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 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 / H_20

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/H_20

  "H_20" in a surfing guide?

big/large

  my big sister != my large sister

- For example, the word *H2O* is used in scientific contexts and would be inappropriate in a surfing guide—*water* would be more appropriate— and this category difference is part of the meaning of the word.
- In practice, the word *synonym* is therefore used to describe a relationship of approximate or rough synonymy.

Difference in form → difference in meaning

Relation: **Similarity**

- While words don’t have many synonyms, most words do have lots of *similar* words.
- *Cat* is not a synonym of *dog*, but *cats* and *dogs* are certainly similar words.
- The notion of word similarity is very useful in larger semantic tasks.
- Words with similar meanings. Not synonyms, but sharing some element of meaning

```
car, bicycle
cow, horse
```

- Know the similarity between two words help to understand meaning of sentence which is important for QA and text summarization.

Ask humans how similar 2 words are

- One way of getting values for word similarity is to ask humans to judge how similar one word is to another. A number of datasets have resulted from such experiments.

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

- For example the SimLex-999 dataset (Hill et al., 2015) gives values on a scale from 0 to 10, from near-synonyms (*vanish, disappear*) to pairs that scarcely seem to have anything in common (*hole, agreement*):

Relation: Word relatedness

- The meaning of two words can be related in ways other than similarity. One such class of connections is called word relatedness. Also called "word association"
- Words can be related in any way, perhaps via a semantic frame or field
    - `coffee, tea:` **similar (something people drink)**
    - `coffee, cup:` **related**, not similar

Semantic field دلالات الألفاظ

- One common kind of relatedness between words is if they belong to the same semantic field.
- A semantic field is a set of words which cover a particular semantic domain and bear structured relations with each other.

Words that

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

For example

hospitals

surgeon, scalpel, nurse, anaesthetic, hospital

restaurants

waiter, menu, plate, food, menu, chef

houses

door, roof, kitchen, family, bed

Relation: Antonymy كلمات متضادة

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 *reverses*:
  - `rise/fall, up/down`

Connotation (sentiment) دلالة أو مفهوم

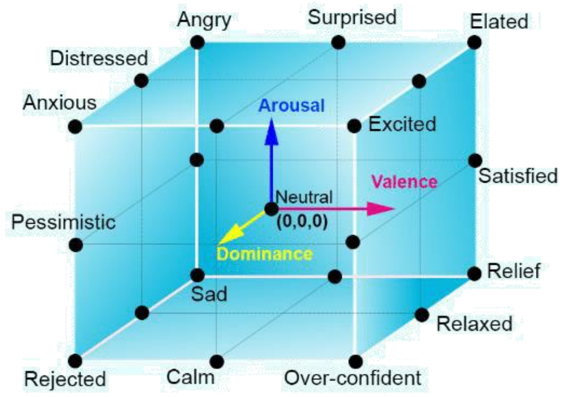
- Words have **affective** meanings معاني وجدانية أو عاطفية
  - Positive connotations (*happy*)
  - Negative connotations (*sad*)
- Connotations can be subtle:
  - Positive connotation: *copy, replica, reproduction*
  - Negative connotation: *fake, knockoff, forgery*
- Evaluation (sentiment!)
  - Positive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

Connotation

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 مستوي الرضا	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal مستوي الاثارة	delighted	0.960		mellow	0.069
	anger	0.965		napping	0.046
Dominance السيطرة	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081



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 requirement?

We'll introduce **vector semantics**

The standard model in language processing!

Handles many of our goals!

One way to find the meaning is:

"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 **cooked with garlic**.
- Ong choi is excellent **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
…

Yamaguchi, Wikimedia Commons, public domain

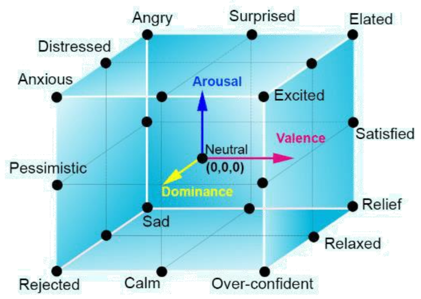
Idea 1:

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 multidimensional space



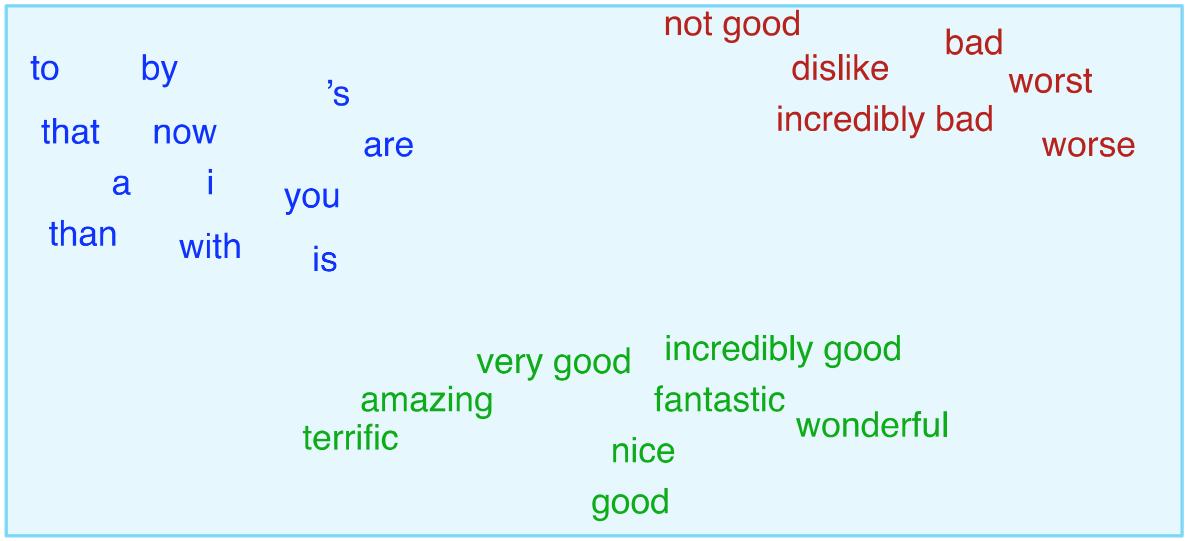
Or we can combine these two ideas:

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
- 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 pillar!
- 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

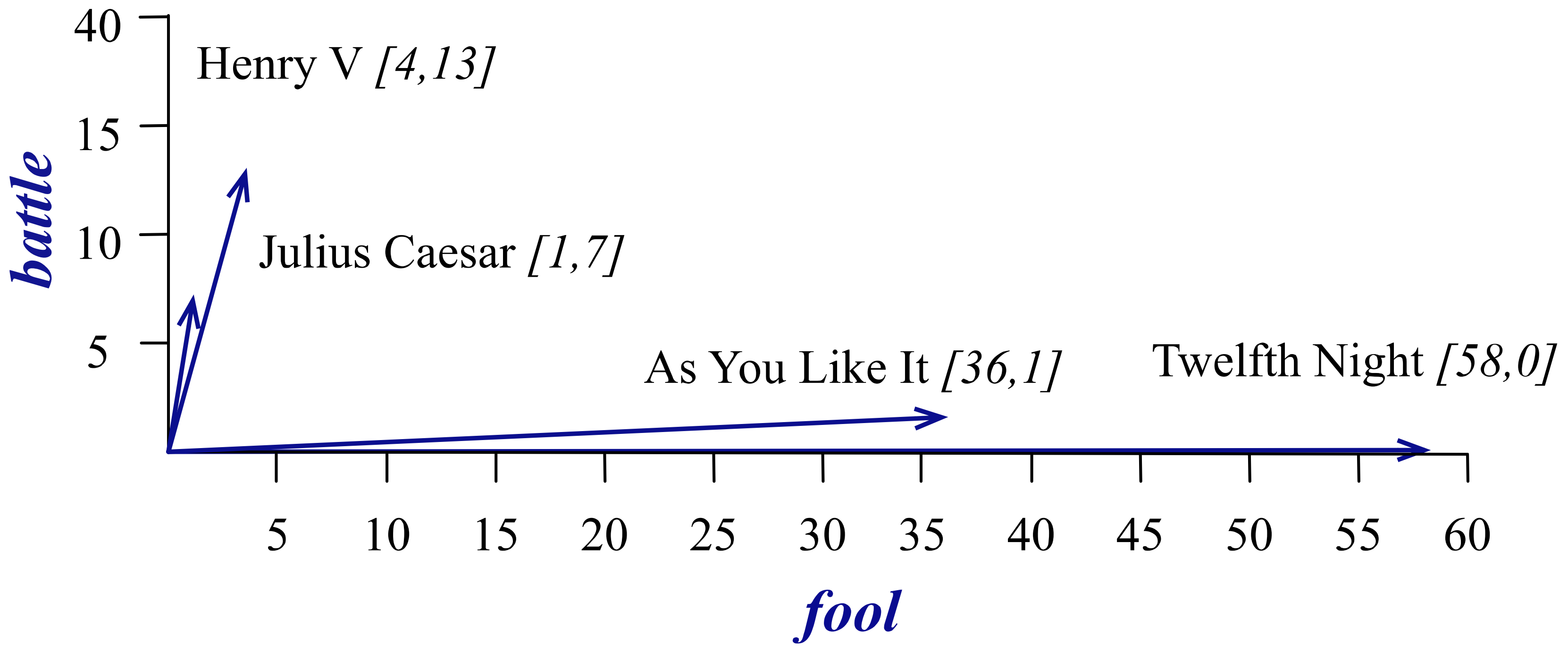
Term-document matrix

- Imagine we have a collection of documents, such as all the works of Shakespeare.
- We can represent documents in such a collection by a term-document matrix, in which each row represents a word in the vocabulary and each column represents a document from the collection.
- Here's a small selection from a term-document matrix showing the occurrence of four words in four plays by Shakespeare.
- Each cell in this matrix represents the number of times a particular word (defined by the row) occurs in a particular document (defined by the column). Thus *fool* appeared 58 times in *Twelfth Night*.
- We can think of each column as a **vector** representing for a document as a point in |*V* |-dimensional space; thus the documents in here are points in 4-dimensional space.

	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

A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.



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!!!

- Vector semantics can also be used to represent the meaning of *words*. We do this by associating each word with a word vector
- The four dimensions of the vector for *fool*, [36,58,1,4], correspond to the four Shakespeare plays.
- For documents, we saw that similar documents had similar vectors, because similar documents tend to have similar words.
- This same principle applies to words: similar words have similar vectors because they tend to occur in similar documents.

	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"

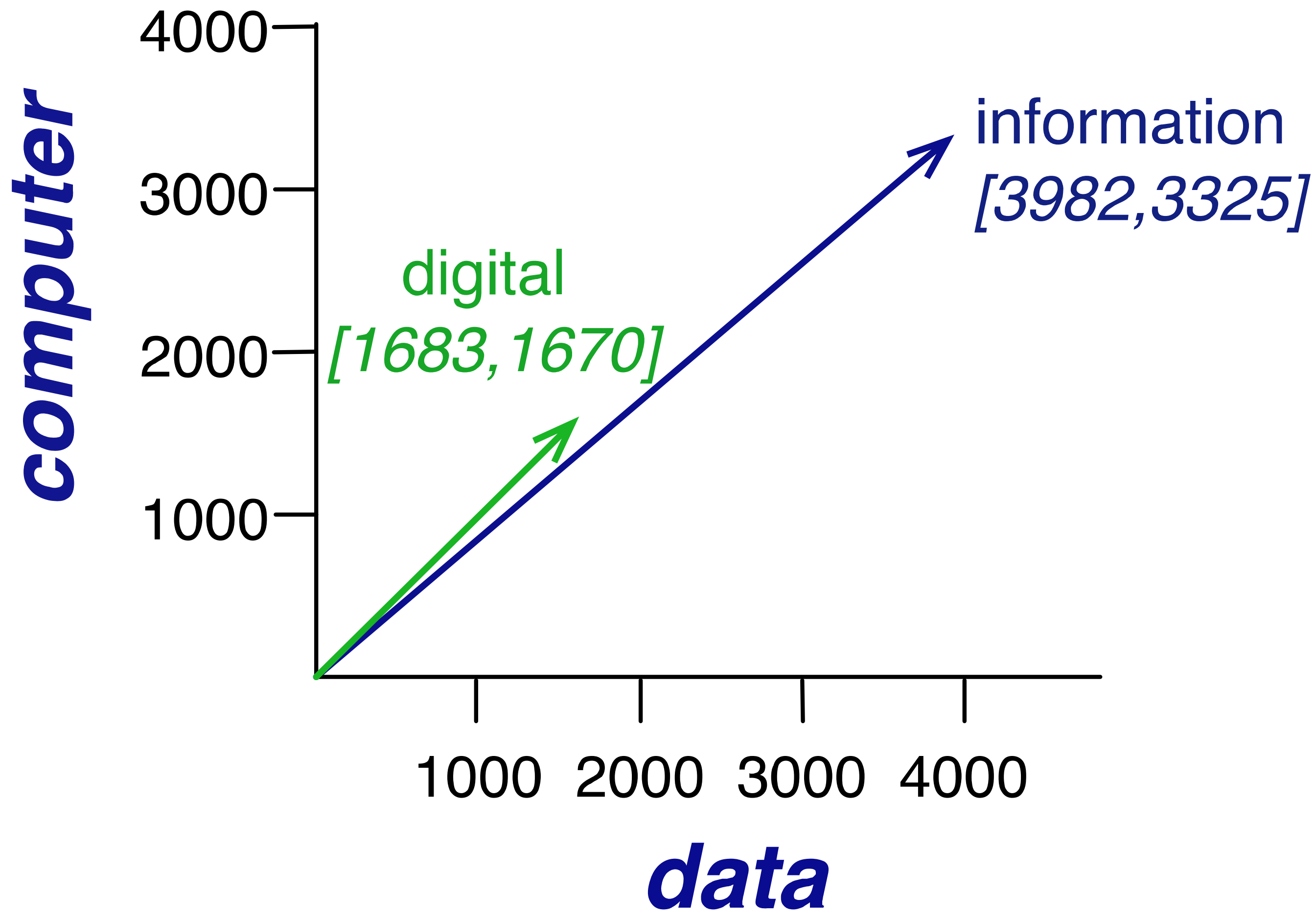
fool is "the kind of word that occurs in comedies, especially Twelfth Night"

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	...


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

Computing word similarity: Dot product and cosine

The dot product between two vectors is a scalar:

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

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

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

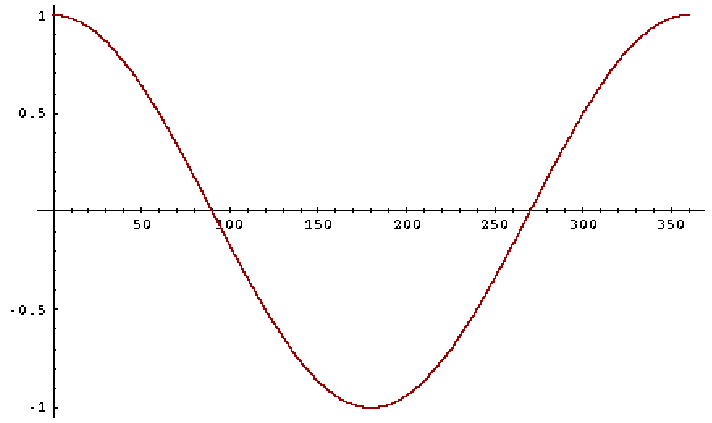
$$\begin{aligned} \mathbf{a}\cdot\mathbf{b} &= |\mathbf{a}||\mathbf{b}|\cos\theta \\ \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} &= \cos\theta \end{aligned}$$

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

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

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

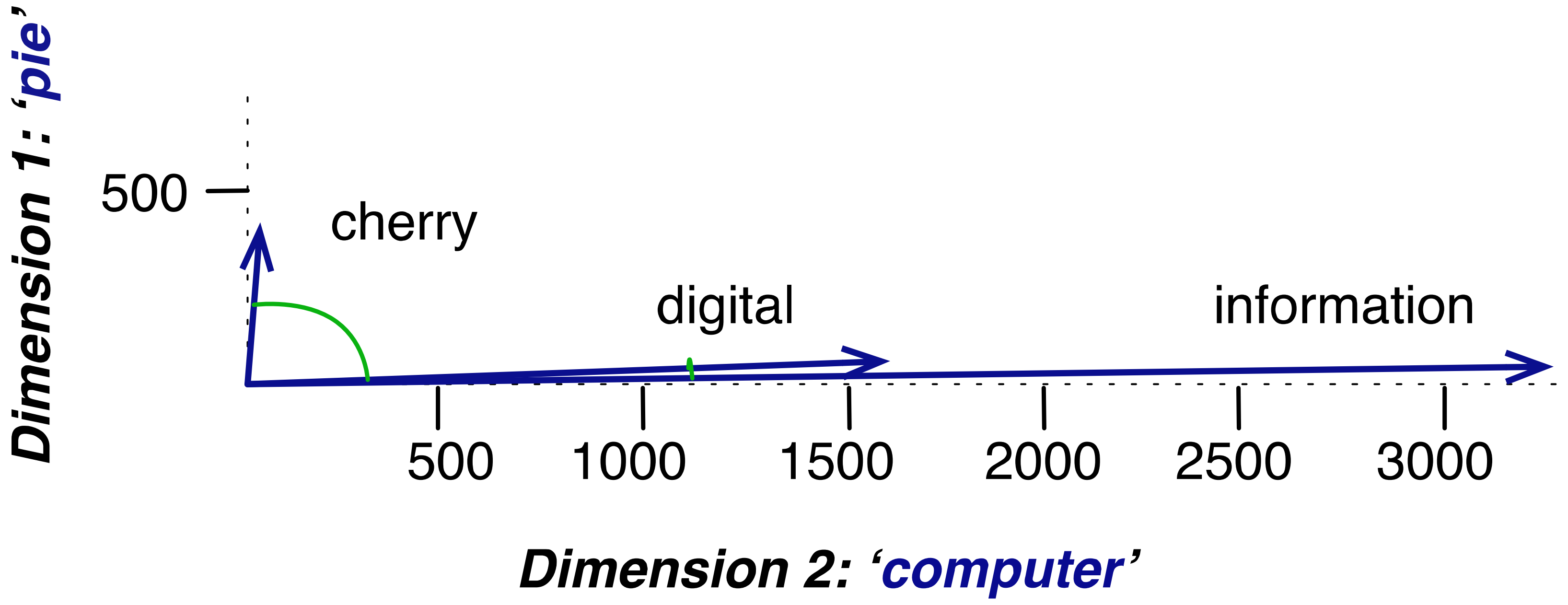
$$\cos(\text{cherry},\text{information}) =$$

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

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

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

Visualizing cosines (well, angles)



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 *apple*, 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)

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

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

Term frequency (tf)

The frequency of the word *t* in the document *d*.

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

Instead of using raw count, we squash a bit:

$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d)+1)$

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

- The second factor in tf-idf is used to give a higher weight to words that occur only in a few documents.
- Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur frequently across the entire collection aren't as helpful.
-

Inverse document frequency (idf)

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

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

Final tf-idf weighted value for a word

Raw counts:

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

	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

Pointwise Mutual Information

Pointwise mutual information:

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

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

PMI between two words: (Church & Hanks 1989)

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

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

Positive Pointwise Mutual Information

- PMI ranges from $-\infty$ to $+\infty$
- But the negative values are problematic
  - Things are co-occurring **less than** we expect by chance
  - Unreliable without enormous corpora
    - Imagine w1 and w2 whose probability is each 10^{-6}
    - Hard to be sure p(w1,w2) is significantly different than 10^{-12}
  - Plus it's not clear people are good at "unrelatedness"
- So we just replace negative PMI values by 0
- Positive PMI (**PPMI**) between word1 and word2:

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

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

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, GPT)
- Compute distinct embeddings for a word in its context
- Separate embeddings for each token of a word

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 cats 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 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

Sim(w,c) ≈ w · c

To turn this into a probability

We'll use the sigmoid from logistic regression:

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

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

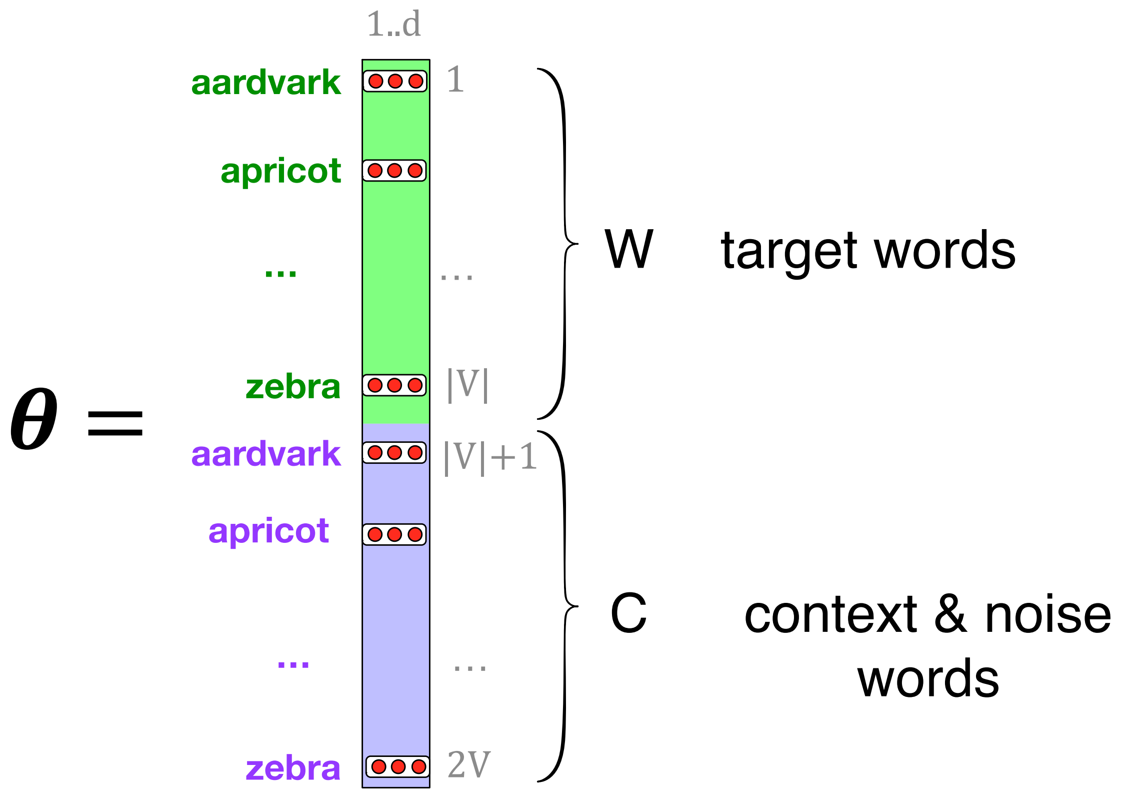
A probabilistic classifier, given

- 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



Word2vec: Learning the embeddings

Skip-Gram Training data

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

c1 c2 [target] c3 c4

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

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}, $c_{neg1} \ldots c_{negk}$

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.

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

Learning the classifier

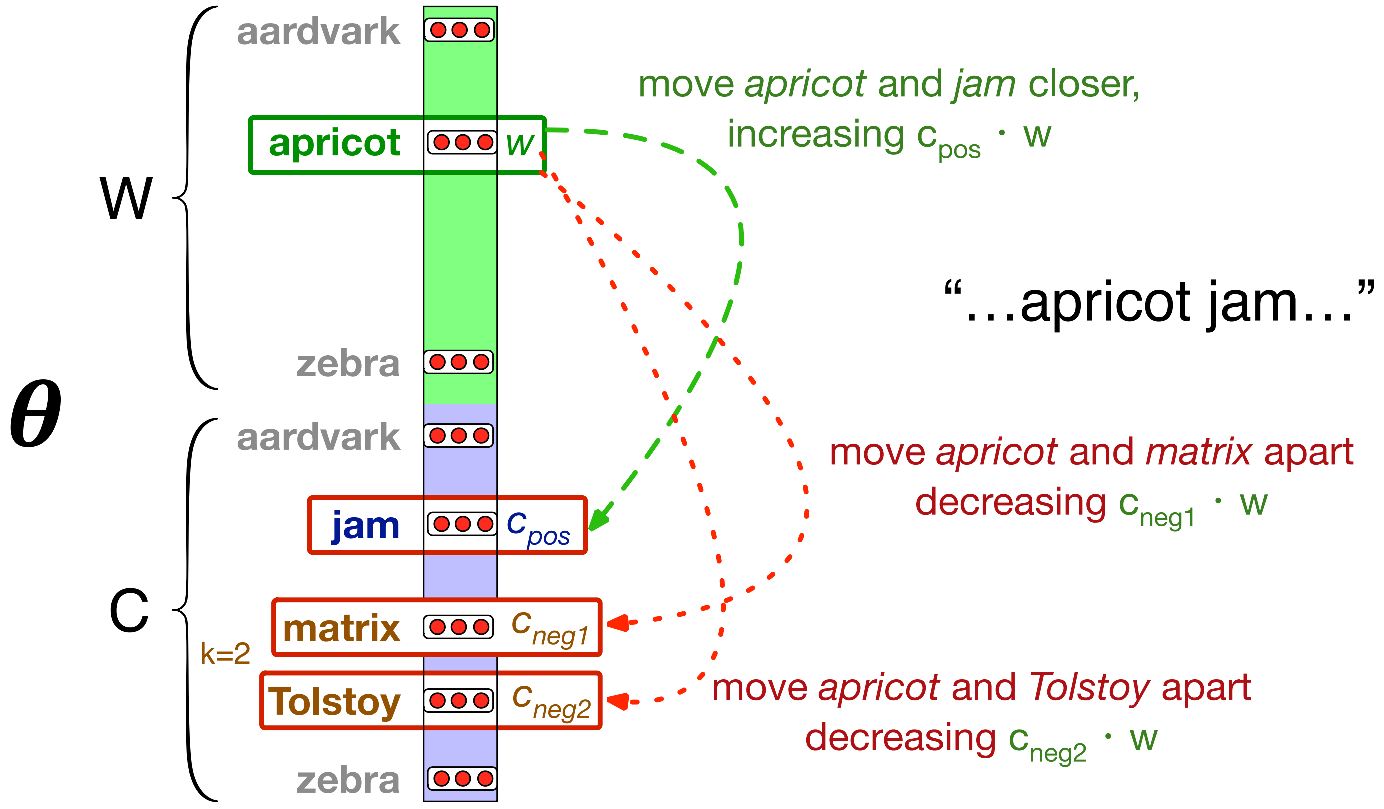
How to learn?

- 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

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

The derivatives of the loss function

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

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

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

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

Update equation in SGD

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

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.