Lecture 02:
N-gram language model (cont.)
& Word Embedding

Xiang Ren

USC CSCI 662 Advanced NLP

Spring 2026

Some slides adapted from Chris Manning and Dan Jurafsky

Announcements

Announcements + Logistics

- Project pitch on Feb 4 → find your project team ASAP! (if not individual project)
  - submit your team by Jan 28 and no more adjustment!
- Project proposal is due by Feb 11, 11:59 PM PT
  - Please use Slack to find teammates
- Classes will not be recorded, but under extreme circumstances, we might allow folks to join over zoom
- Sharing slides after class…

Project Pitch

Every project group (1-3 people) will give a **5-minute project pitch in class on x.** Feel frexxe to start forming groups using the class slack channel, or you may wait until you hear other peoples' pitches to form your final project groups.

Pitch presentation Format:

- **1 slide**
- **5-minute presentation**

(The goal is to present your idea(s) *clearly* for feedback and to find potential teammates)

- **2-minute Q&A**

(Be prepared to ask at least 1 question in class for participation)Group Formation & Proposal)

N-gram language model: Quality & Evaluation

How best to evaluate an LM?

- Extrinsic evaluation can be time-consuming; hard to design
  - Which is the best task? How many tasks to try?
- Therefore, we often use intrinsic evaluation:
  - Bad approximation
    - unless the test data looks just like the training data
  - Generally only useful in pilot experiments

Perplexity

Intuition of Perplexity

The Shannon Game: How well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

- mushrooms 0.1
- pepperoni 0.1
- anchovies 0.01
- ….
- fried rice 0.0001
- ….
- and 1e-100

- Unigrams are terrible at this game

A better model of a text

- is one which assigns a higher probability to the word that actually occurs

Perplexity

- The best language model is one that best predicts an unseen test set
  - Gives the highest $P(sentence)$

Perplexity is the inverse probability of the test set, normalized by the number of words

$$PPL(\mathbf{w}) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$PPL(\mathbf{w}) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

Minimizing perplexity is the same as maximizing probability

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

$$= \sqrt[N]{\frac{1}{\prod_i P(w_i | w_1 \ldots w_{i-1})}}$$

Applying Markov's assumption for bigrams:

$$= \sqrt[N]{\frac{1}{\prod_i P(w_i | w_{i-1})}}$$

Perplexity Example

- Let's suppose a sentence of length 10 consisting of random digits
- What is the perplexity of this sentence according to a model that assigns uniform probability to each digit?

$$P(w) = \frac{1}{10}$$

$$PPL(\mathbf{w}) = P(w_1 w_2 \ldots w_N)^{\frac{1}{N}}$$

$$= \left(\frac{1}{10}\right)^{10\,-\frac{1}{10}}$$

$$= 10$$

Lower perplexity = better model!

- Training 38 million words, test 1.5 million words, from the Wall Street Journal

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

What are the two things that might affect perplexity?

Generating from an n-gram model and Zeros

Recall: BRP

P(english|want) = .0011

P(chinese|want) = .0065

P(to|want) = .66

P(eat | to) = .28

P(food | to) = 0

P(want | spend) = 0

P (i | <s>) = .25

How can we generate sentences from this bigram model?

Generating from a bigram model

- Choose a random bigram
- (<s>, w) according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- Then string the words together

<s> I
I want
want to
to eat
eat Chinese
Chinese food
food </s>

I want to eat Chinese food

Shakespearean n-grams

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Shakespeare as a corpus

- N=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams
- So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- 4-grams (quadrigrams) are worse: What's coming out looks like Shakespeare because it is Shakespeare!

Most n-grams are never seen!

The WSJ is no Shakespeare!

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Shakespeare as a corpus

- V=29,066
- Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams
- So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- 4-grams (quadrigrams) are worse: What's coming out looks like Shakespeare because it is Shakespeare!

Most n-grams are never seen!

The WSJ is no Shakespeare!

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

So why not just sample from very high order n-gram models? Do we even need ChatGPT?

Can only produce n-grams from training data!

Shakespearean corpus cannot produce WSJ vocabulary and vice versa

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

The successes we are seeing here are due to a phenomena commonly known as overfitting

Overfitting bad!

N-grams only work well for word prediction if the test corpus looks like the training corpus

- In real life, it often doesn't
- We need to train robust models that generalize!
  - Technical terms for "doing well on the test data" or "doing well on any test data"

Machine Learning 101

- At the minimum, we would want to pick the smallest test set that gives us enough statistical power to measure a statistically significant difference between two potential models.

Machine Learning 101

- **LM Test Sets:** Given a large corpus that we want to divide into training and test, test data can either be
    - taken from some continuous sequence of text inside the corpus,
    - or we can remove smaller “stripes” of text from randomly selected parts of our corpus and combine them into a test set.

Overfitting bad!

N-grams only work well for word prediction if the test corpus looks like the training corpus

- In real life, it often doesn't
- We need to train robust models that generalize!
  - Technical terms for "doing well on the test data" or "doing well on any test data"

- One kind of generalization: **Zeros**!
  - Things that don't ever occur in the training set
    - But occur in the test set

N-gram models: Common Issues that need handling

Token Type Vocabulary

At test time, we might encounter:

- Token never seen in context (i.e. n-gram with 0 frequency)
- Token never seen (unigram with 0 frequency)
- More severe!
  - Problem: Many words like "***Petrichor***" won't appear in most training sets!
  - These are known as OOV for "out of vocabulary", or unknown tokens

Missing n-grams

Training set:

… denied the allegations

… denied the reports

… denied the claims

… denied the request

Test set

… denied the offer

… denied the loan

$$P(offer|denied\ the) = 0$$

will assign 0 probability to the test set!

And hence we cannot compute perplexity

- No one can divide by 0!

What happens to perplexity??

Missing Unigrams: the <UNK> token

- One way to handle OOV tokens is by adding a pseudo-word called <UNK>
- Closed Vocabulary: Only allow a list of predetermined tokens, everything else (in the training data) is <UNK>

Closed Vocabulary

- We can replace all words that occur fewer than n times in the training set, where n is some small number, by <UNK> and re-estimate the counts and probabilities

Open Vocabulary

- When not done carefully, may lead to artificially lower perplexity


Smoothing

Intuition for Smoothing

I like to **eat** cake but I want to **eat** pizza right now. Mary told her brother to **eat** pizza too.

→

P(next word = *pizza* | previous word = *eat*) = 2/3
P(next word = *cake* | previous word = *eat*) = 1/3
All other next words = 0 probability

- All other vocabulary tokens getting 0 probability just doesn't seem right. We want to assign some probability to other words
- We want to smooth the distribution from our counts

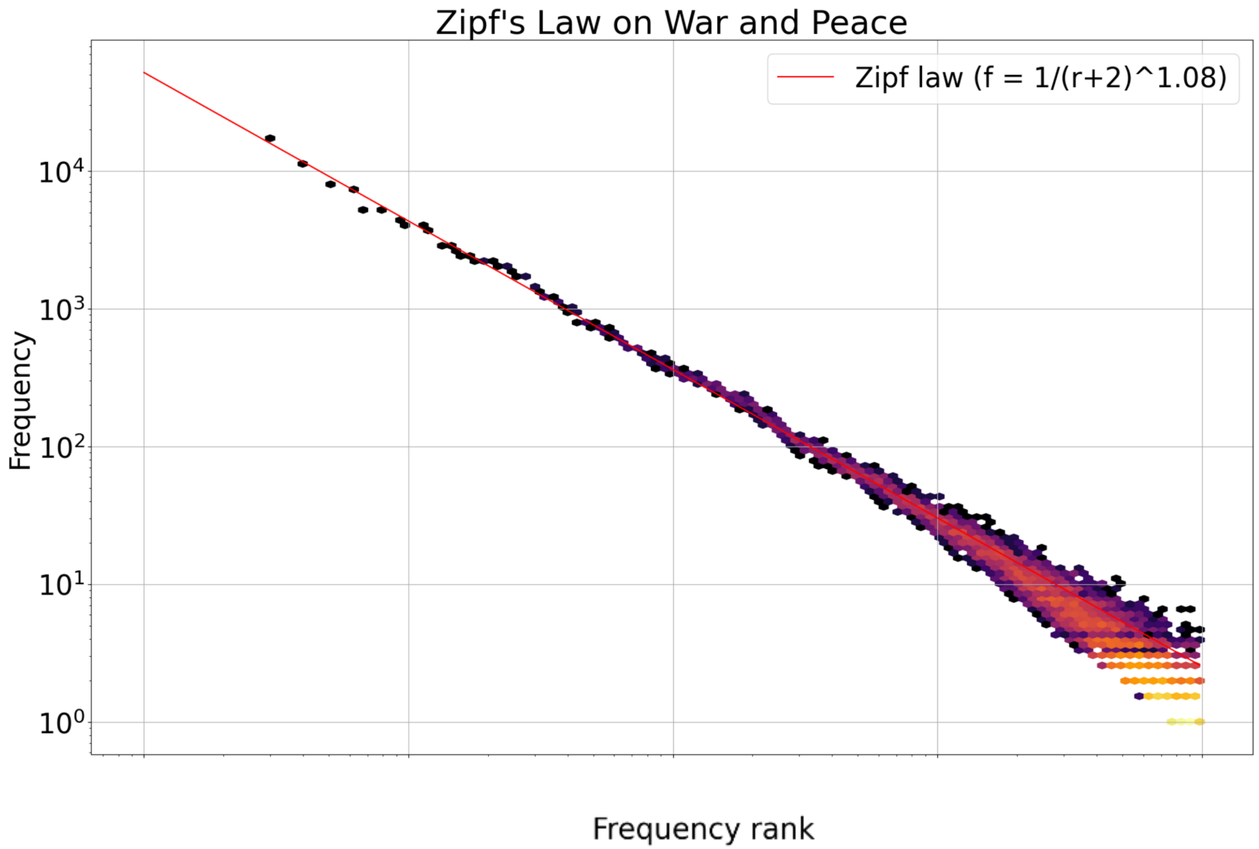
What does a count distribution look like?

Zipf's Law

The distribution over words resembles that of a power law:

- there will be a few words that are very frequent, and a long tail of words that are rare
- $freq_w(r) \approx r^{-s}$

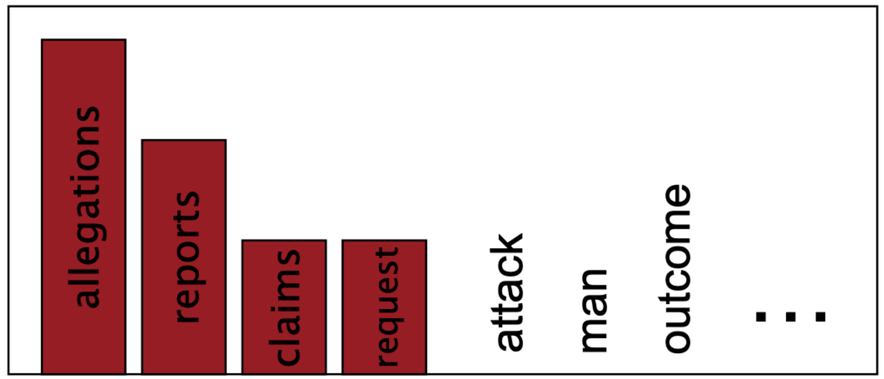
NLP algorithms must be especially robust to observations that do not occur or rarely occur in the training data



Zipf, G. K. (1949). Human behavior and the principle of least effort.

Smoothing ~ Massaging Probability Masses

- When we have sparse statistics: $Count(w|denied\ the)$
  - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request
  - 7 total



Steal probability mass to generalize better: $Count(w|denied\ the)$

2.5 allegations

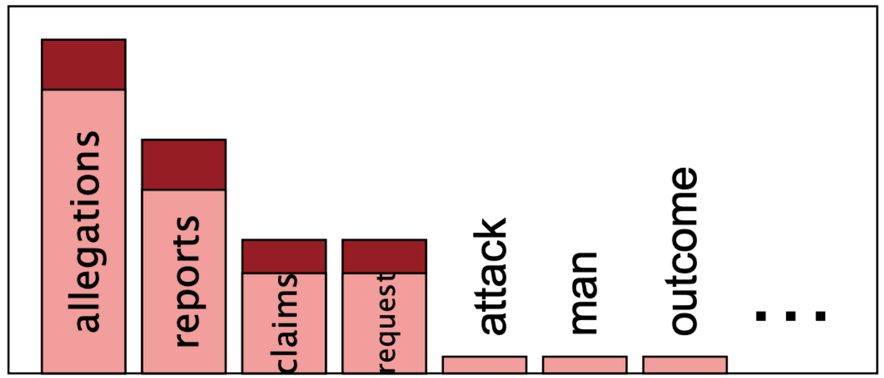
1.5 reports

0.5 claims

0.5 request

2 other

7 total



Add-One Estimation

MLE estimate

$$P_{MLE}(w_i) = \frac{c(w_i)}{\sum_w c(w)}$$

Pretend we saw each word one more time than we did

Just add one to all the counts!

Laplace smoothing

75 year old method!

All the counts that used to be zero will now have a count of 1…

Add-1 estimate

$$P_{Add-1}(w_i) = \frac{c(w_i)+1}{\sum_w (c(w)+1)} = \frac{c(w_i)+1}{V + \sum_w c(w)}$$

What happens to our P values if we don't increase the denominator?

Add-1 Estimation Bigrams

MLE estimate

$$P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)}{c(w_{i-1})}$$

Pretend we saw each bigram one more time than we did

Add-1 estimate

$$P_{Add-1}(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)+1}{c(w_{i-1})+V}$$

What does this do to the unigram counts?

Keep the same denominator as before and reconstruct bigram counts

$$= \frac{c^*(w_{i-1}w_i)}{c(w_{i-1})}$$

Recall: BRP Corpus

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

Unigrams

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Bigrams

w_i

w_{i-1}	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Laplace-smoothed bigram counts

Just add one to all the counts!

w_{i-1} \ w_i	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigram probabilities

$$P_{Add-1}(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i) + 1}{c(w_{i-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted Counts

$$c * (w_{i-1}w_i) = \frac{[c(w_{i-1}w_i) + 1]c(w_{i-1})}{c(w_{i-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Compare with raw bigram counts

Original, Raw

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Big change to the counts!

Reconstructed

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Perhaps 1 is too much, add a fraction?

Add-k smoothing

Add-1 Estimation: Last thoughts

So add-1 isn't used for n-grams, being something of a blunt instrument

- One-size-fits-all

Add-1 is used to smooth other NLP models though…

- For text classification
- In domains where the number of zeros isn't so huge

Interpolation

Perhaps use some pre-existing evidence

- Condition on less context for contexts you haven't learned much about

Interpolation

- mix unigram, bigram, trigram probabilities for a trigram LM
- mix n-gram, (n-1)gram, … unigram probabilities for an n-gram LM

Interpolation works better than Add-1 / Laplace

Linear Interpolation

Simple Interpolation

$$\hat{P}(w_i|w_{i-2}w_{i-1}) = \lambda_1 P(w_i) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i|w_{i-2}w_{i-1})$$

$$\sum_k \lambda_k = 1$$

Hyperparameters!

Context-Conditional Interpolation

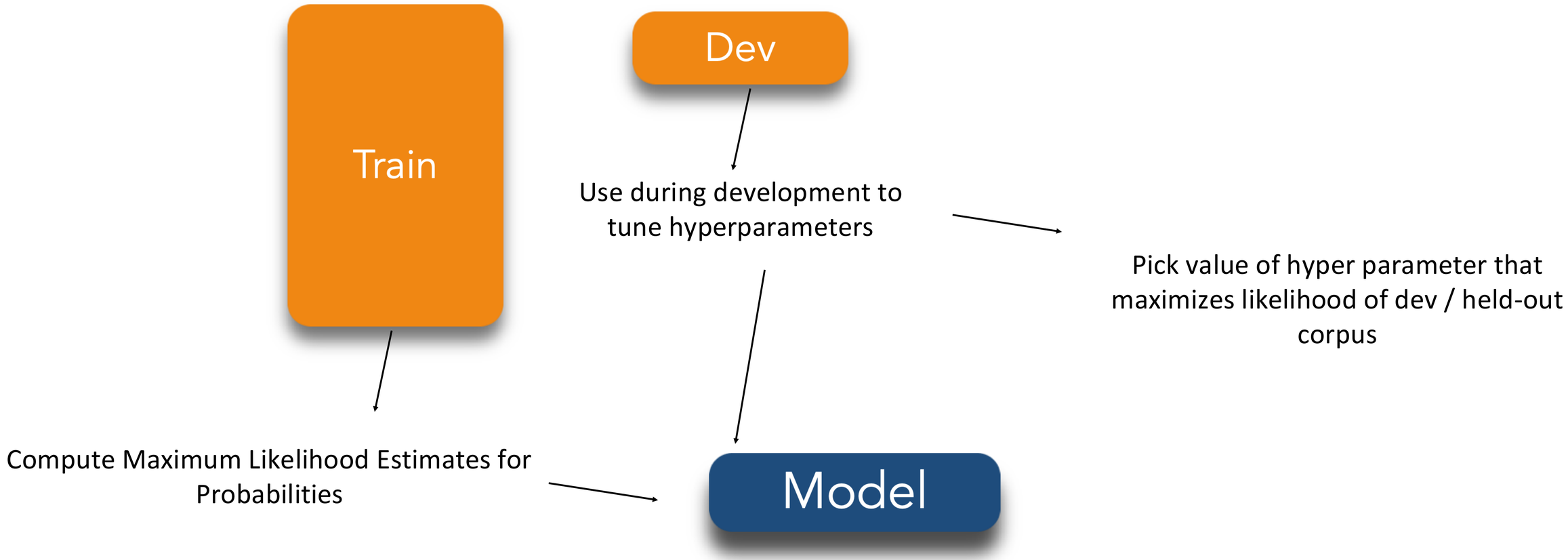
$$\hat{P}(w_i|w_{i-2}w_{i-1}) = \lambda_3(w_{i-2}^{i-1}) P(w_i|w_{i-2}w_{i-1}) + \lambda_2(w_{i-2}^{i-1}) P(w_i|w_{i-1}) + \lambda_1(w_{i-2}^{i-1}) P(w_i)$$

Reconstituted Counts

Different for every unique context

How to set the λs?

Language Model Development



How to set the λs?

- Choose λs to maximize the probability of held-out data:
  - Fix the n-gram probabilities (on the training data)
  - Then search for λs that give largest probability to held-out set:

$$logP(w_1 \ldots w_n | M(\lambda_1 \ldots \lambda_k)) = \sum_i logP_{M(\lambda_1 \ldots \lambda_k)}(w_i | w_{i-1})$$

Backoff and Discounting

Backoff

- use trigram if you have good evidence,
- otherwise bigram, otherwise unigram

- Still need a correct probability distribution!
  - Discount higher order n-grams by d to save some probability mass for the lower order n-grams
  - need a function α to distribute this probability mass to the lower order n-grams

Stupid Backoff

- No discounting, just use relative frequencies
- Don't care about a valid language model

Not a probability distribution (usually denoted as P)

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4\, S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

Hyperparameter!

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

Brants *et al.* 2007

Absolute discounting: just subtract a little from each count

- Consider an n-gram with count of 4. Suppose we wanted to subtract a little from this to save probability mass for the zeros
  - How much to subtract ?
- Church and Gale (1991)'s clever idea
- Divide up 22 million words of AP Newswire
- Training and held-out set
  - for each bigram in the training set
    - see the actual (averaged) count in the held-out set!
- It sure looks like $c^* \approx (c - .75)$

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Absolute Discounting Interpolation

- Save ourselves some time and just subtract 0.75 (or some d, such that $0 \leq d < 1$)!
- Maybe keeping a couple extra values of d (for counts 1 and 2 and 3)

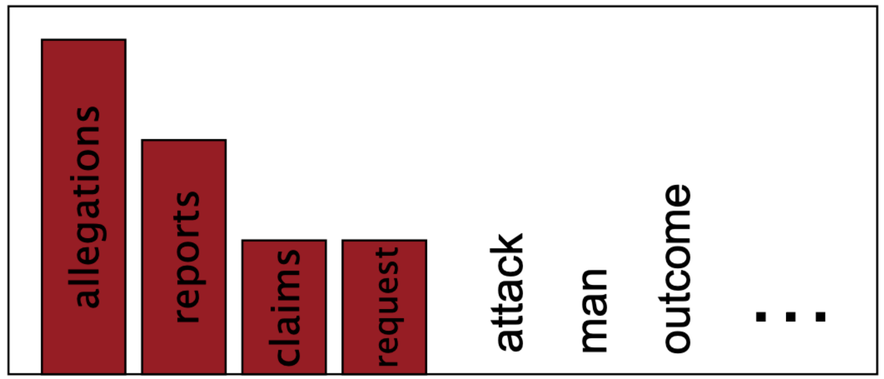
$$P_{AbsoluteDiscounting}(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i) - d}{\sum_w c(w_{i-1}w)} + \lambda(w_{i-1})P(w_i)$$

Discounted Bigram

But should we really just use the regular unigram $P(w_i)$?

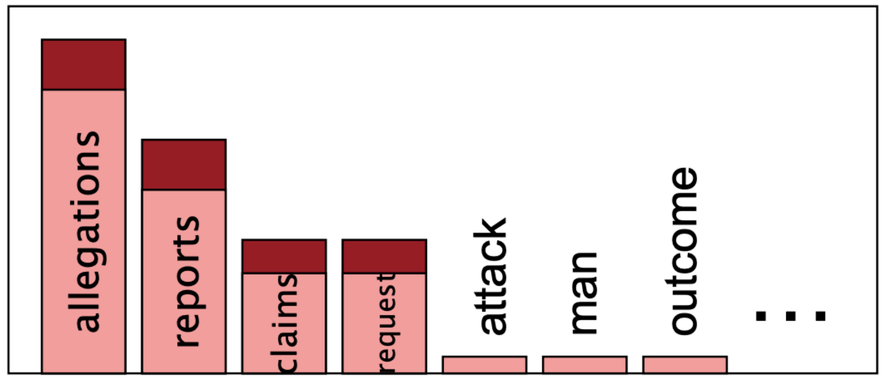
Smoothing ~ Massaging Probability Masses

- When we have sparse statistics: $Count(w|denied\ the)$
  - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request
  - 7 total



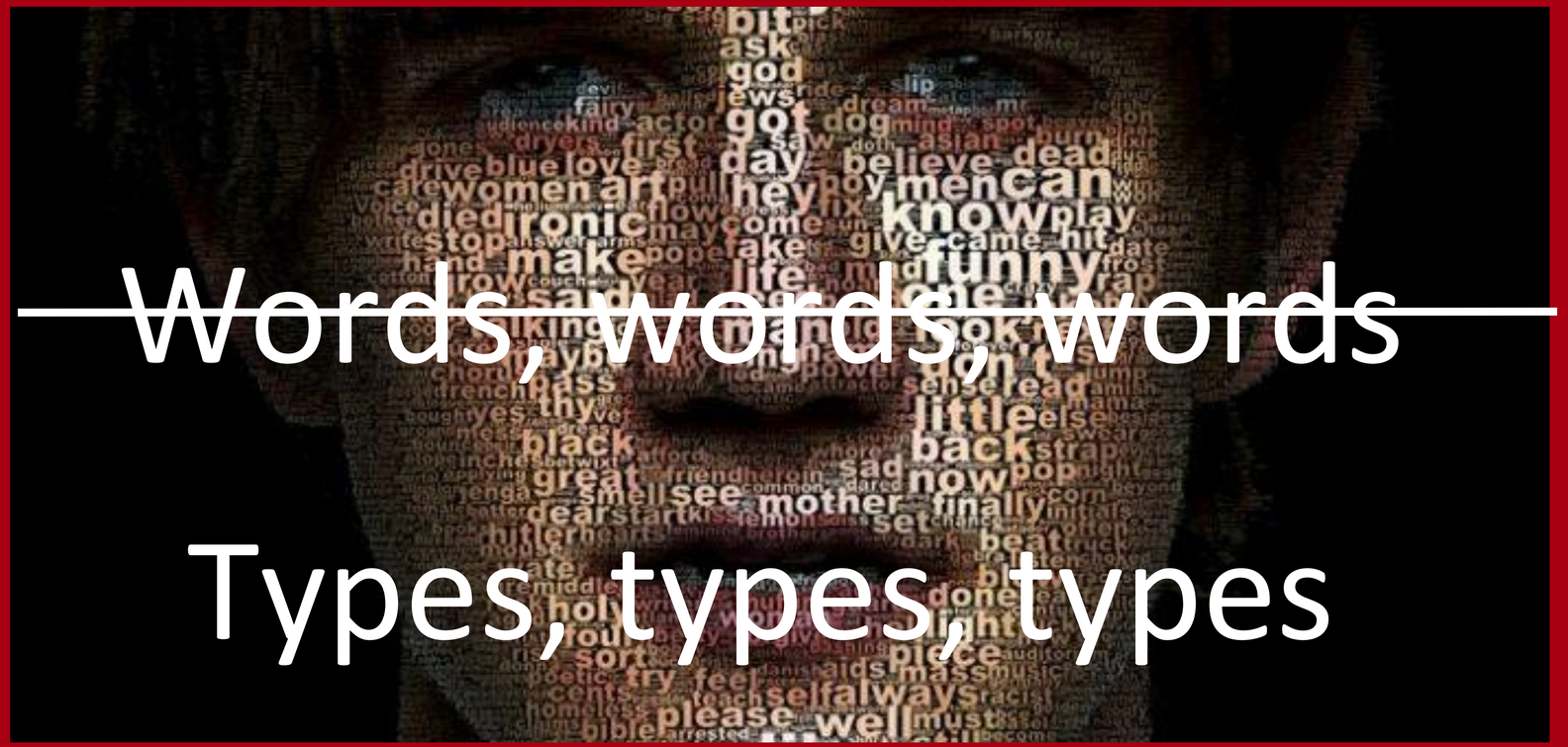
Steal probability mass to generalize better: $Count(w|denied\ the)$

- 2.5 allegations
- 1.5 reports
- 0.5 claims
- 0.5 request
- 2 other (or, **<UNK>**)
- 7 total



Early Neural Language Models

This class: Word Embeddings — the most important component of a neural LM


~~Words, words, words~~
Types, types, types

Word: Types & Tokens

"The cat chased the cat."

- Word types: *the*, *cat*, *chased* → **3 types**
- Word tokens: *the*, *cat*, *chased*, *the*, *cat* → **5 tokens**

So:

- **Word** = the linguistic item
- **Type** = the category representing identical items

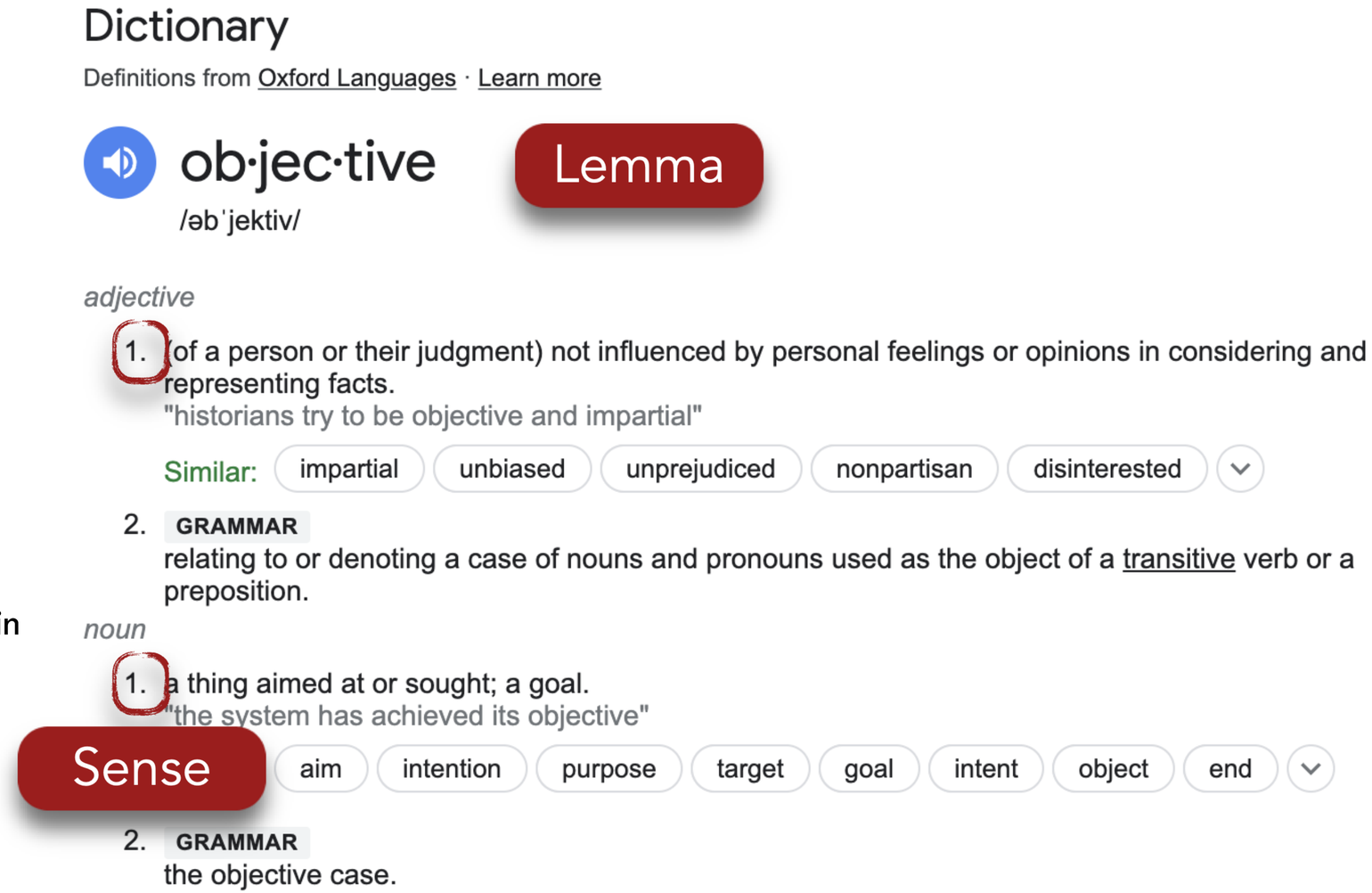
What do words mean?

A sense or "concept" is the ***meaning*** component of a word

Lemmas

- Canonical form
- For example,
  break, breaks, broke, broken and breaking all share the lemma "break"

Can be polysemous (have multiple senses)



Synonymy

Synonyms: words that have the same ***meaning*** in some or all contexts

- couch / sofa
- automobile / car
- water / H_20

Is perfect synonymy possible?

- Even if many aspects of meaning are identical
- Still may differ based on politeness, slang, register, genre, etc.
  - e.g. cannot use H_20 in a surfing guide!

Similarity

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

Human assessment of word similarity

Simlex-999 dataset (Hill et al., 2015)

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

Not to be confused with word association / relatedness:

- couch / sofa vs. couch / pillow

Antonymy

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

Otherwise, they are very similar!

- e.g. 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
    - e.g. long/short, fast/slow
- be reversives: denote opposing processes
    - rise/fall, up/down

WordNet

- WordNet® is a large lexical database of English
- Nouns, verbs, adjectives and adverbs are grouped into sets of synonyms (synsets), each expressing a distinct concept
- Relations between synsets:
  - Super-subordinate relations (hyperonymy, hyponymy or ISA relation)
    - an armchair is a kind of chair, chair is a kind of furniture
  - Meronymy (part-of)
    - chair has legs
  - Antonymy

n-grams and Semantics

- n-grams do not represent meaning well
  - Do not tell us that the word “rancor” is close in meaning to the word “hatred”
  - Or that “Rise” and “Fall” have opposite meanings
  - Let alone more complex is-a or part-of relations

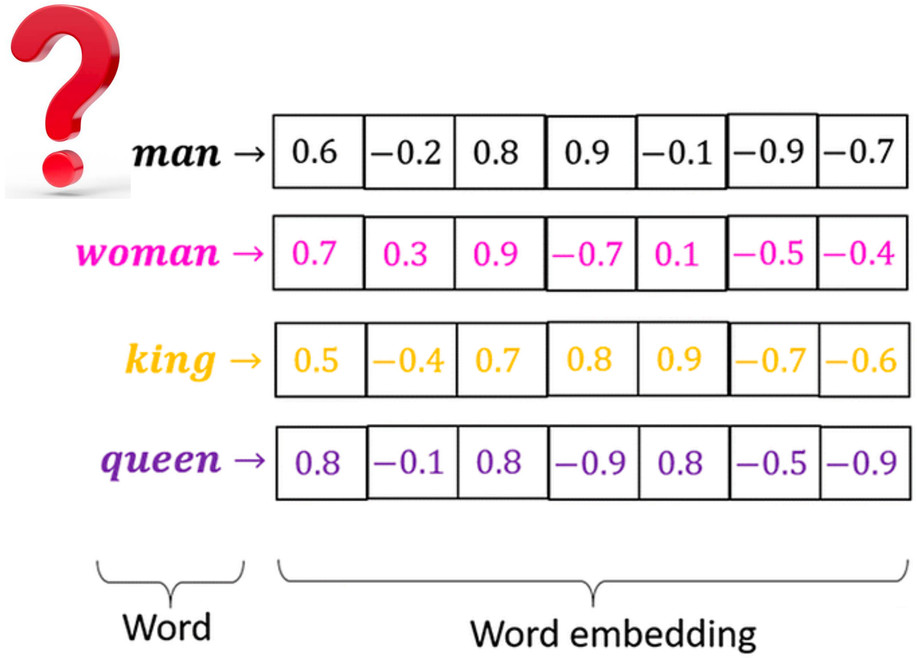
n-grams and Semantics

- n-grams do not represent meaning well
  - Do not tell us that the word “rancor” is close in meaning to the word “hatred”
  - Or that “Rise” and “Fall” have opposite meanings
  - Let alone more complex is-a or part-of relations
- Discrete representations of meaning!
- Next: feature representations which are continuous

Words as Vectors

In NLP, we commonly represent word types with vectors!

- But why?
- Very useful in capturing similarity between words, and other forms of lexical semantics (e.g. synonymy, hypernyms, antonymy)
- Computing the similarity between two words (or phrases, or documents) is extremely useful for many NLP tasks
  - Q: How tall is Mount Everest?
  - A: The official height of Mount Everest is 29029 ft



- Similarity for plagiarism detection
- Word similarity can lead to sentence and document similarity

enough scale for companies to make profit from it. In order to be competitive with new technologies, the challenge of today's large companies is to create new business within their business (Garvin & Levesque, 2006). Furthermore, the two researchers emphasize a switch from downsizing and cost cutting to the creation, development and assistance of innovative new businesses. For existing companies the implementation of corporate entrepreneurship, in order to develop innovative businesses, is risky. Are the three types of entrepreneurship linked together over time? How long does it take to change behavior of the firm as a whole? If the five attributes are created, do all grow together equally, or do some grow faster and earlier than others? How do the importance and intensities of the attributes differ both absolutely and relatively in each type? These are the questions that a longitudinal study such as this can attempt to answer to shed light on the nature of organizations' adjustments to hostile environments. According to Garvin and Levesque (2006) implementing new ventures face several barriers, and can only be successful if a blend of old and new organizational traits is done. To achieve a blend of old and new, an organization needs to rely on employee innovative behavior in order to succeed in dynamic business environments (Yuan & Woodman, 2010).

The downside is potentially very damaging to a startup's lifespan: if a startup lands a pilot or POC with the corporation running the accelerator, they have very little bartering power or time to find other partners to test their solution with. The transition from manufacturing economy to service economy has led to a shift in business agenda from

1 Original source
onlinelibrary.wiley.com/stor...

...all grow together equally, or do some grow faster and earlier than others? These are the questions that a longitudinal study such as this can attempt to answer to shed light on the nature of organiza-tions' adjustments to hostile environments. Of the many ways to adjust, two stand out at

- Visualizing semantic change over time

- New words: `dank, cheugy, rizz, shook, situationship`

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

Discrete representations

- So far, words are regarded as atomic symbols: hotel, conference, walk
- Missed nuances: synonyms, antonyms
- WordNet taxonomy

Words as Vectors

"You shall know a word by the company it keeps."

- Firth (1957)

Word Meaning via Language Use

- The meaning of a word can be given by its distribution in language usage:
  - One way to define "usage": words are defined by their environments
    - Neighboring words or grammatical environments
- Intuitions: Zellig Harris (1954):
  - "oculist and eye-doctor … occur in almost the same environments"
  - "If A and B have almost identical environments we say that they are synonyms."

A bottle of tesgüino is on the table
Everybody likes tesgüino
Tesgüino makes you drunk
We make tesgüino out of corn.

Two words are similar if they have similar word contexts

Defining meaning as a point in space based on distribution

- Each word = a vector
  - not just "good" or "word#45"
- Similar words are "nearby in semantic space"
- We build this space automatically by seeing which words are nearby in text
- 2-D representation

Word Meanings via Language Properties

- Meaning of a word can be determined by some properties of the word
- Point in space (Osgood et al., 1957)
- Example Properties: Affective Dimensions

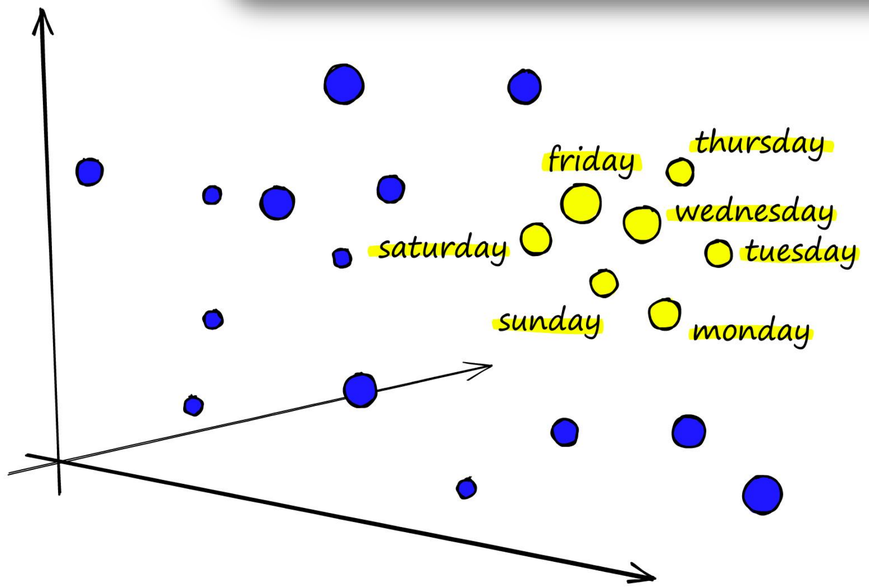
	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

NRC VAD Lexicon (Mohammad 2018)

Word Embeddings

Vector Semantics

- Represent a word as a point in a multidimensional semantic space
  - Space itself constructed from distribution of word neighbors
- Called an "embedding" because it's embedded into a space
- Fine-grained model of meaning for similarity



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

Image Credit: Pinecone

Intuition: Why Vectors?

- Consider generation:
  - With word strings, a feature is a word identity
    - Feature 5: 'The previous word was "horror"'
    - 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] … perhaps corresponding to "horrific"
    - We can generalize to similar but unseen words!!!

Cosine Similarity for Word Similarity

Cosine similarity of two vectors

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v}.\vec{w}}{|\vec{v}||\vec{w}|}$$

$$= \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Based on the definition of the dot product between two vectors $\vec{a}$ and $\vec{b}$

$$\vec{v}.\vec{w} = |\vec{v}||\vec{w}|\cos\theta$$

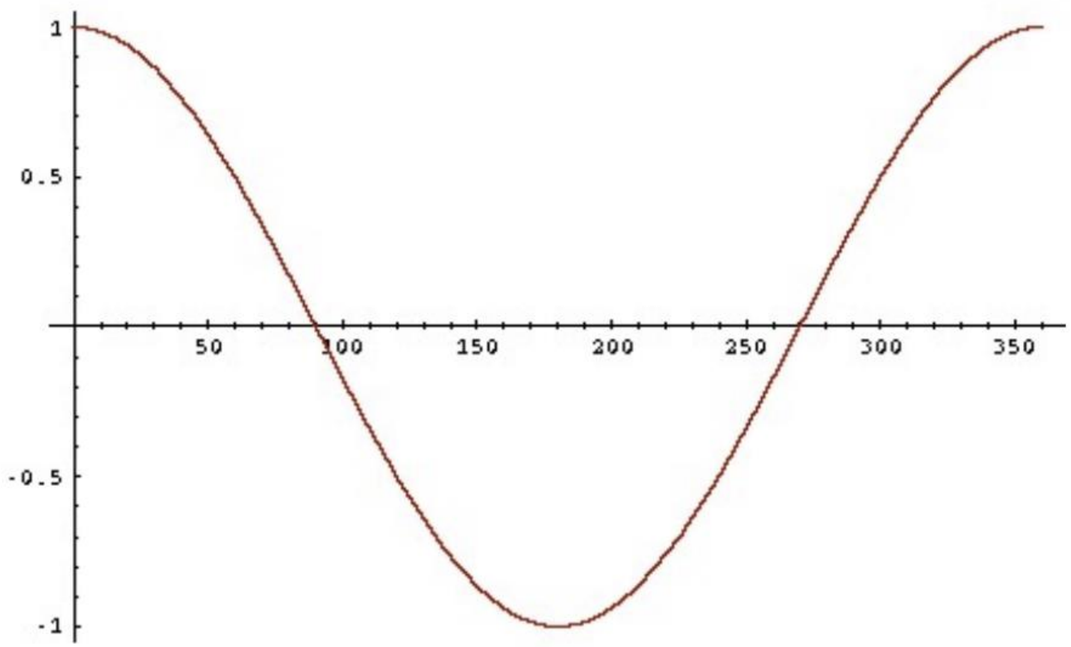
$$\cos\theta = \frac{\vec{v}.\vec{w}}{|\vec{v}||\vec{w}|}$$

Cosine as a similarity metric

-1: vectors point in opposite directions

+1: vectors point in same directions

0: vectors are orthogonal



Greater the cosine, more similar the words

n-grams as One-hot Vectors

Unigram Vectors: Represent each word as a vector of zeros with a single 1 identifying the index of the word

vocabulary
i
hate
love
the
movie
film

movie = <0, 0, 0, 0, 1, 0>
film = <0, 0, 0, 0, 0, 1>

One hot vector

How can we compute a vector representation such that the dot product correlates with word similarity?

Dot product is zero! These vectors are orthogonal

Term-document matrix

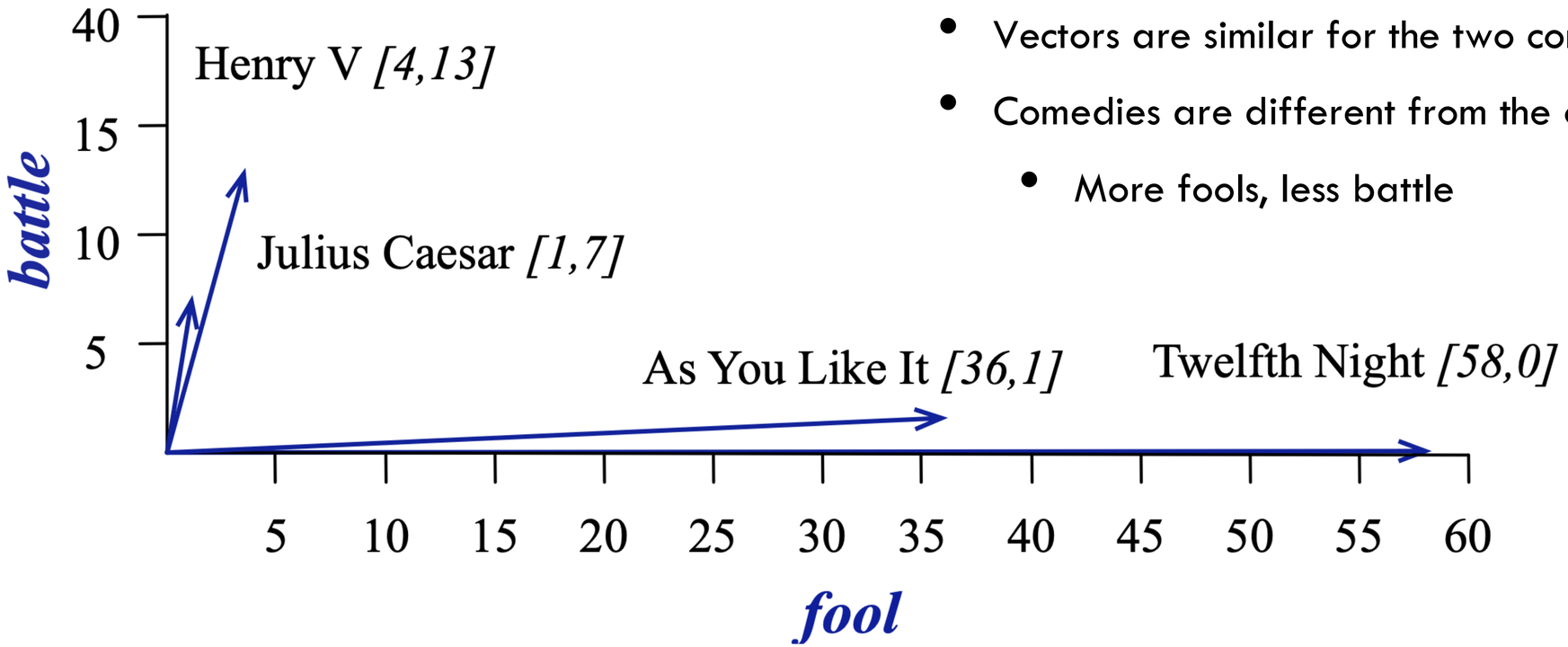
Let us consider a collection of documents and count how frequently a word (term) appears in each. A document could be a play or a Wikipedia article. In general, documents can be anything; we often call each paragraph a document!

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

	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
- Comedies are different from the other two (tragedies)
  - More fools, less battle

Words as vectors in a co-occurrence matrix

	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 appears in Julius Caesar and Henry V

"Fool" is the kind of word that appears in As You Like It and Twelfth Night

Word-word co-occurrence matrix

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

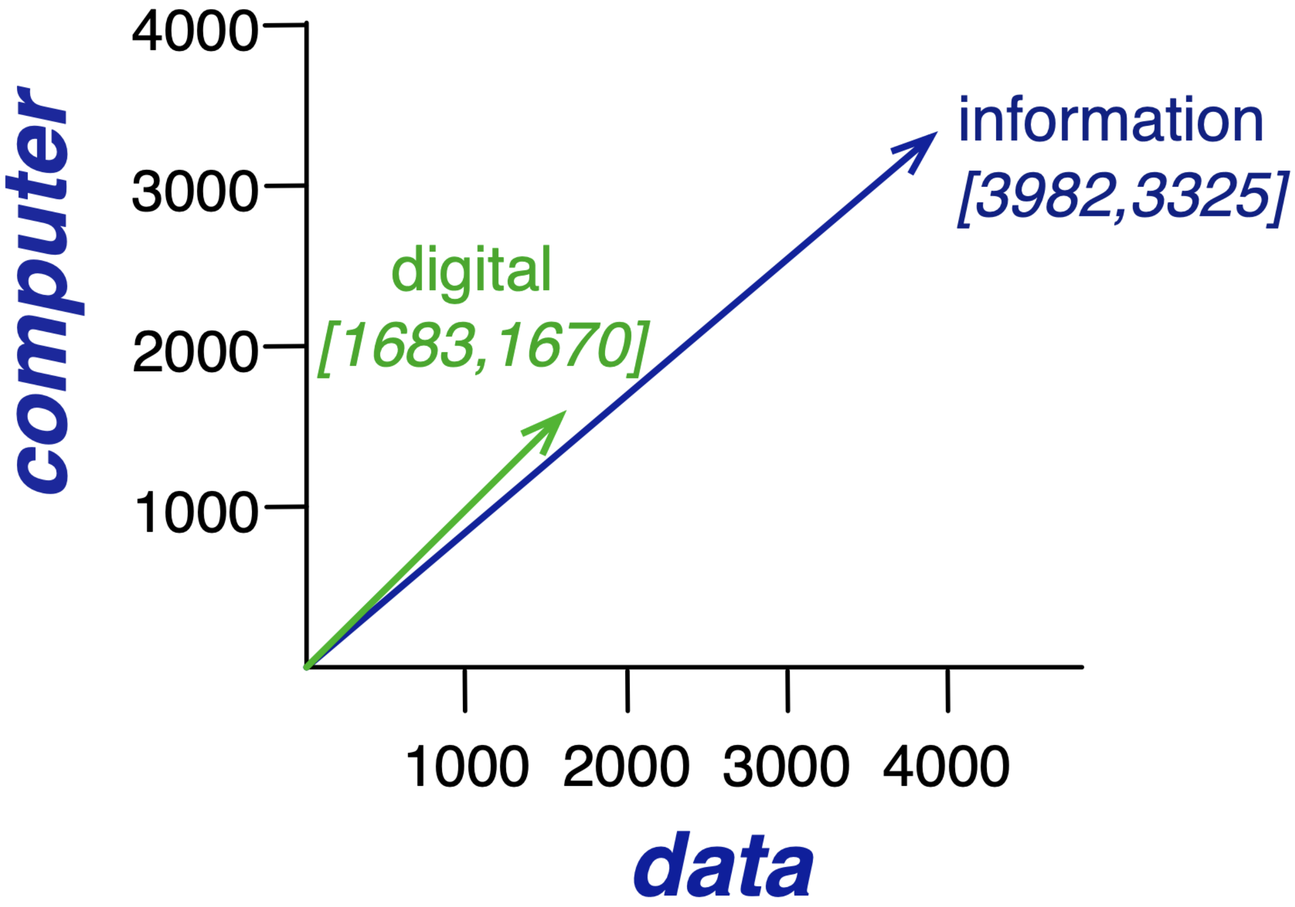
Context Window

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

Words, not documents

	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	...

	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	...



Cosine Similarity

$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|}$$

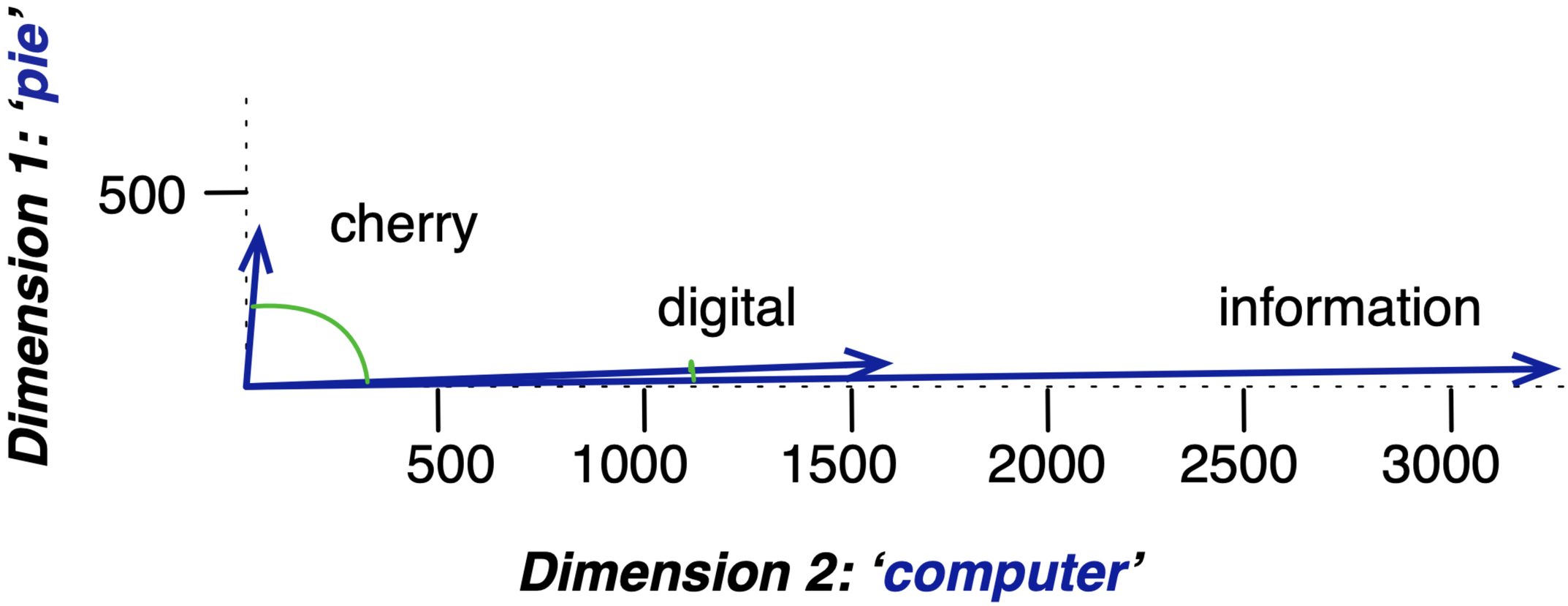
$$= \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}$$

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

$$\cos(\text{cherry, information}) = \frac{442*5 + 8*3982 + 2*3325}{\sqrt{442^2 + 8^2 + 2^2}\sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

$$\cos(\text{digital, information}) = \frac{5*5 + 1683*3982 + 1670*3325}{\sqrt{5^2 + 1683^2 + 1670^2}\sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

Visualizing cosines



Raw frequencies though…

- …are 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?

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

Need some form of weighting!

Two different kinds of weighting

tf-idf: Term Frequency - Inverse Document Frequency

- Downweighting words like “the” or “if”
- Term-document matrices

PMI: Pointwise Mutual Information

- Considers the probability of words like “good” and “great” co-occurring
- Word co-occurrence matrices

Term Frequency

Term Frequency: frequency counting (usually log transformed)

$$tf_{t,d} = \begin{cases} 1 + \log(count(t,d)), & if\ count(t,d) > 0 \\ 0, & otherwise \end{cases}$$

$count(t,d)$ = # occurrences of word t in document d

Inverse Document Frequency

- Document Frequency: df_t is the number of documents t occurs in.
  - NOT collection frequency: total count across all documents
  - "Romeo" is very distinctive for one Shakespeare play
- Inverse Document Frequency: idf_t

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

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

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

N = total number of documents in the collection

tf-idf

Final tf-idf weighted value for a word

$$tf_{t,d} \times idf_{t,d}$$

	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

Raw Counts

tf-idf Weighted Counts

	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 (PMI)

- PMI between two words:
  - Do words x and y co-occur more than if they were independent?

$$PMI(w_1, w_2) = \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

- PMI ranges from $-\infty$ to $+\infty$
  - Negative values are problematic: words are co-occurring less than we expect by chance
  - Only reliable under an enormous corpora
    - Imagine w_1 and w_2 whose probability is each 10^{-6}
    - Hard to be sure $P(w_1, w_2)$ is significantly different than 10^{-12}
  - So we just replace negative PMI values by 0

- Positive PMI

$$PPMI(w_1, w_2) = max\left(0, \log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}\right)$$

Church & Hanks 1989

Computing PPMI on a term-context matrix

Context C

Term W

	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

Frequency $f(w_i, c_j)$ or f_{ij} is the # times w_i occurs in context c_j

$$P_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{i,j}} \qquad P_i = \sum_{j=1}^{C} P_{ij} \qquad P_j = \sum_{i=1}^{W} P_{ij}$$

$$PPMI(w_i, c_j) = PPMI_{i,j} = max\left(0, \log \frac{P_{ij}}{P_i P_j}\right)$$

	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

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

$$P_i = \sum_{j=1}^{C} P_{ij}$$

$$P_j = \sum_{i=1}^{W} P_{ij}$$

	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	

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

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

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

	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	

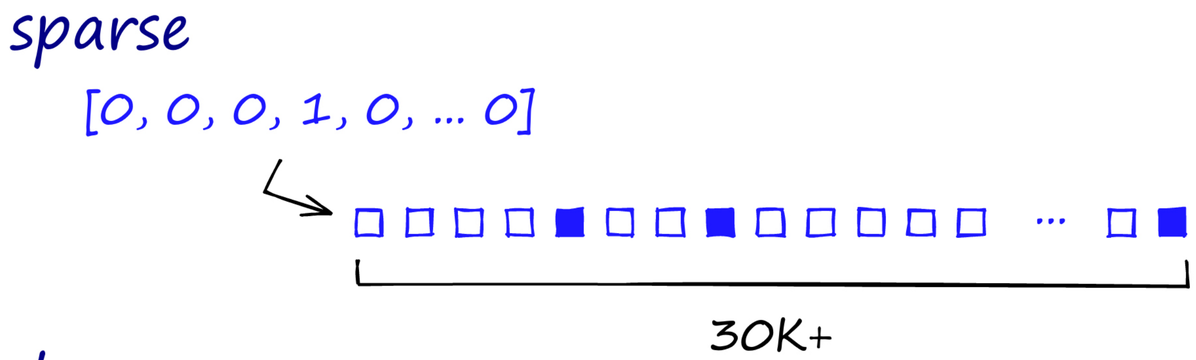
	computer	**data**	**result**	**pie**	**sugar**
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

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

$$PPMI_{i,j} = max\left(0, \log \frac{P_{ij}}{P_i P_j}\right)$$

The problem…

- 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)

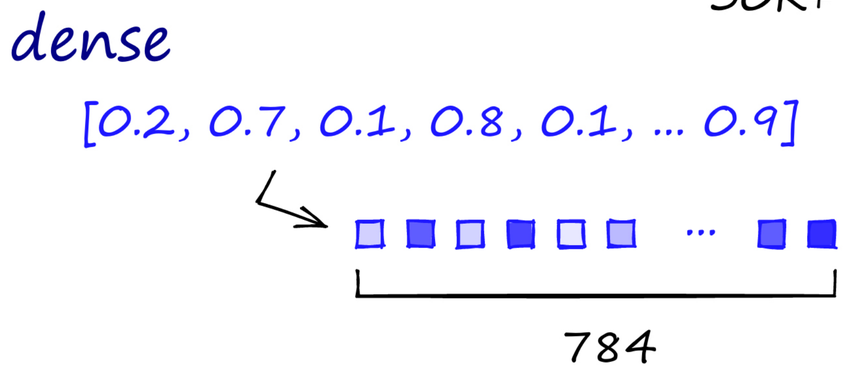


Image Credit: Pinecone

Sparse vs. Dense Vectors

- Why dense vectors?
  - Memory efficiency is not a problem for sparse 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

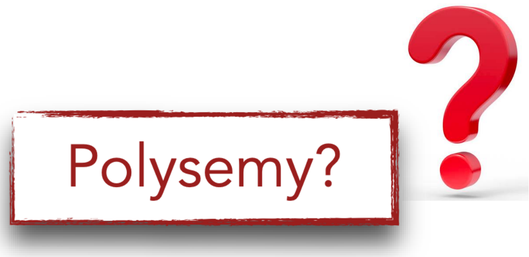
word2vec

word2vec

Mikolov et al., ICLR 2013. Efficient estimation of word representations in vector space.

Mikolov et al., NeurIPS 2013. Distributed representations of words and phrases and their compositionality.

- Short, dense vector or embedding
- Static embeddings
  - One embedding per word type
  - Does not change with context change
- Two algorithms for computing:
  - Skip-Gram with Negative Sampling or SGNS
  - CBOW or continuous bag of words
  - But we will study a slightly different version…
- Efficient training
- Easily available to download and plug in



word2vec : Intuition

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

Instead of counting how often each word W occurs near another, e.g. “cherry”

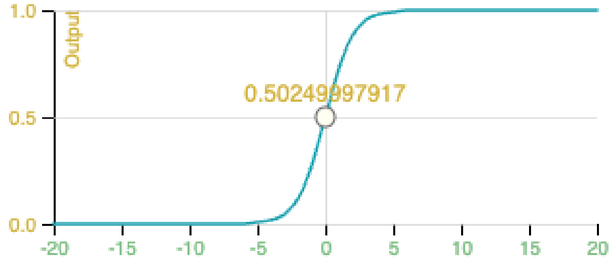
- Train a classifier on a binary prediction task:
  - Is W likely to show up near “cherry”?
- We don’t actually care about this task!!!
  - But we'll take the learned **classifier weights** as the word embeddings

x? y?

Word embedding itself is the learned parameter!

Binary Text Classification

- Goal: Given an input, predict label or class from a discrete set
  - e.g. Predict the sentiment (positive or negative) for a sentence
- Input: x represented by feature vector of size d, given by $\mathbf{x} \in \mathbb{R}^d$
- Output: $y \in \{0,1\}$ for binary classification
- Suffices to learn conditional probabilities
  - Parameterized by $\theta \in \mathbb{R}^d$
- Could estimate by cooccurrence counts, but a single feature
  - Better option: dot product (assigning a weight to every feature)
  - Returns a real value: $z \in \mathbb{R}$
- How to get a probability?
  - Consider the Sigmoid function:
- Argmax for prediction:



Logistic Regression

$$P(y|\mathbf{x};\theta)$$

$$z = \theta \cdot \mathbf{x}$$

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

$$P(y=1|\mathbf{x};\theta) = \sigma(\theta \cdot \mathbf{x})$$

$$P(y=0|\mathbf{x};\theta) = 1 - \sigma(\theta \cdot \mathbf{x}) = \sigma(-\theta \cdot \mathbf{x})$$

$$\hat{y} = \arg\max_{y' \in \{0,1\}} P(y'|\mathbf{x};\theta)$$

word2vec: Self-supervision

One missing piece: where to get the (x, y) pairs from?

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

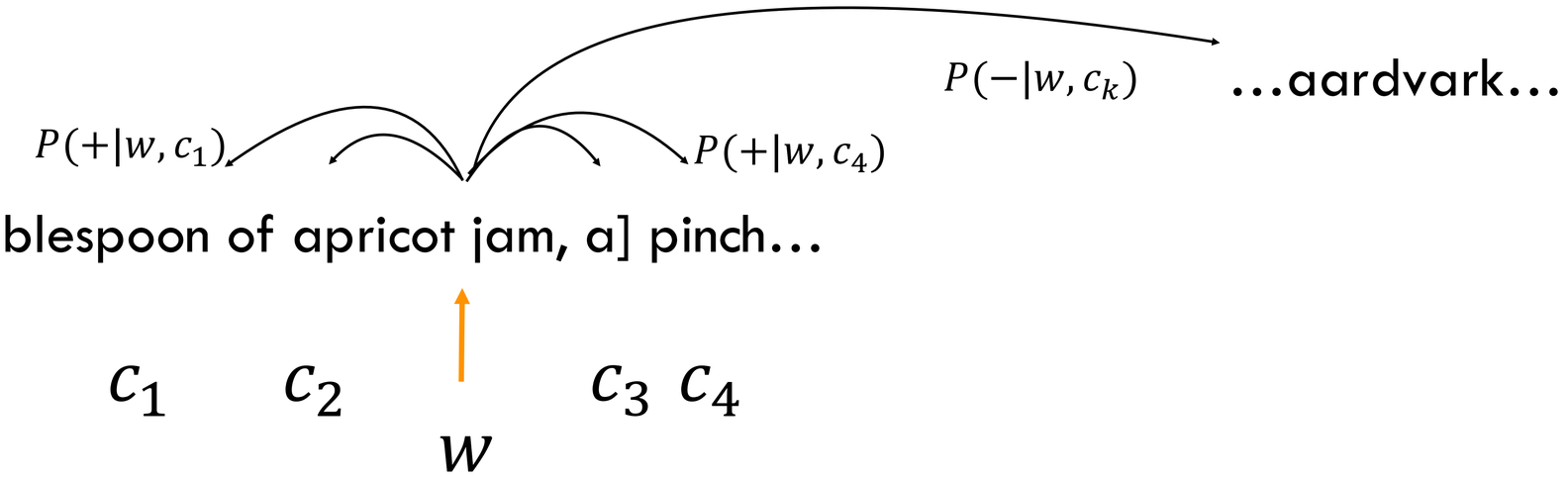
- A word C that occurs near "cherry" in the corpus acts as the gold "correct answer" for supervised learning
- No need for human labels!

What about incorrect labels?

Bengio et al. (2003); Collobert et al. (2011)

word2vec: Goal

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



Goal: train a classifier that is given a candidate (word, context) pair:

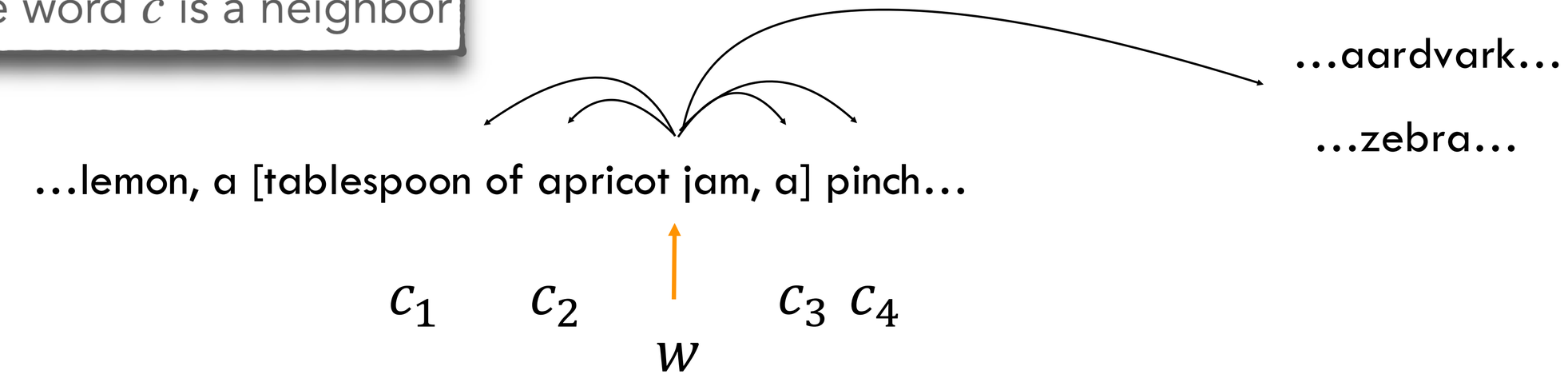
- (apricot, jam)
- (apricot, aardvark)
- …

And assigns each pair a probability:

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

word2vec: Pseudocode

Predict if candidate word c is a neighbor



1. Treat the target word w 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

word2vec: Probability Estimates

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

- Central intuition: Base this probability on embedding similarity!
- Remember: two vectors are similar if they have a high dot product
  - Cosine similarity is just a normalized dot product
- So:

$$sim(w,c) \propto \mathbf{w} \cdot \mathbf{c}$$

Vectors, not scalars!

- Still not a probability!
  - We'll need to normalize to get a probability

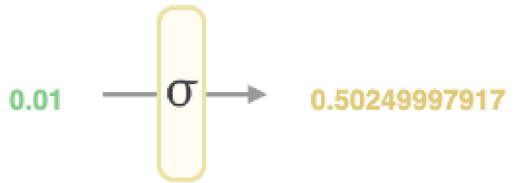
Can we just use cosine?

Turning dot products into probabilities

Sigmoid

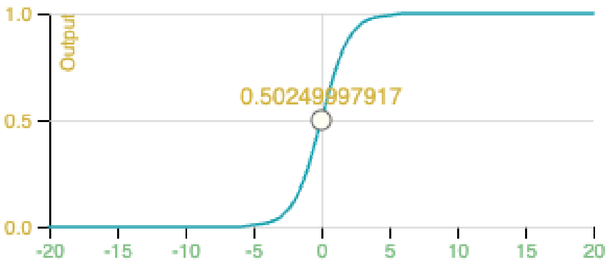
Similarity:

$$sim(w, c) \approx \mathbf{w} \cdot \mathbf{c}$$



Turn into a probability using the sigmoid function:

$$f(0.01) = \frac{1}{1 + e^{-(0.01)}} = 0.50249997917$$



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

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

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

Logistic Regression!

Accounting for a context window

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

Single Context Word

But we have lots of context words

- Depends on window size
- We'll assume independence and just multiply them

Same with negative context words!

c_{neg} } ...aardvark...
...zebra...

$$\log P(-|w, c_{neg}) = \sum_{c' \in c_{neg}} \log \sigma(-\mathbf{c}' \cdot \mathbf{w})$$

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

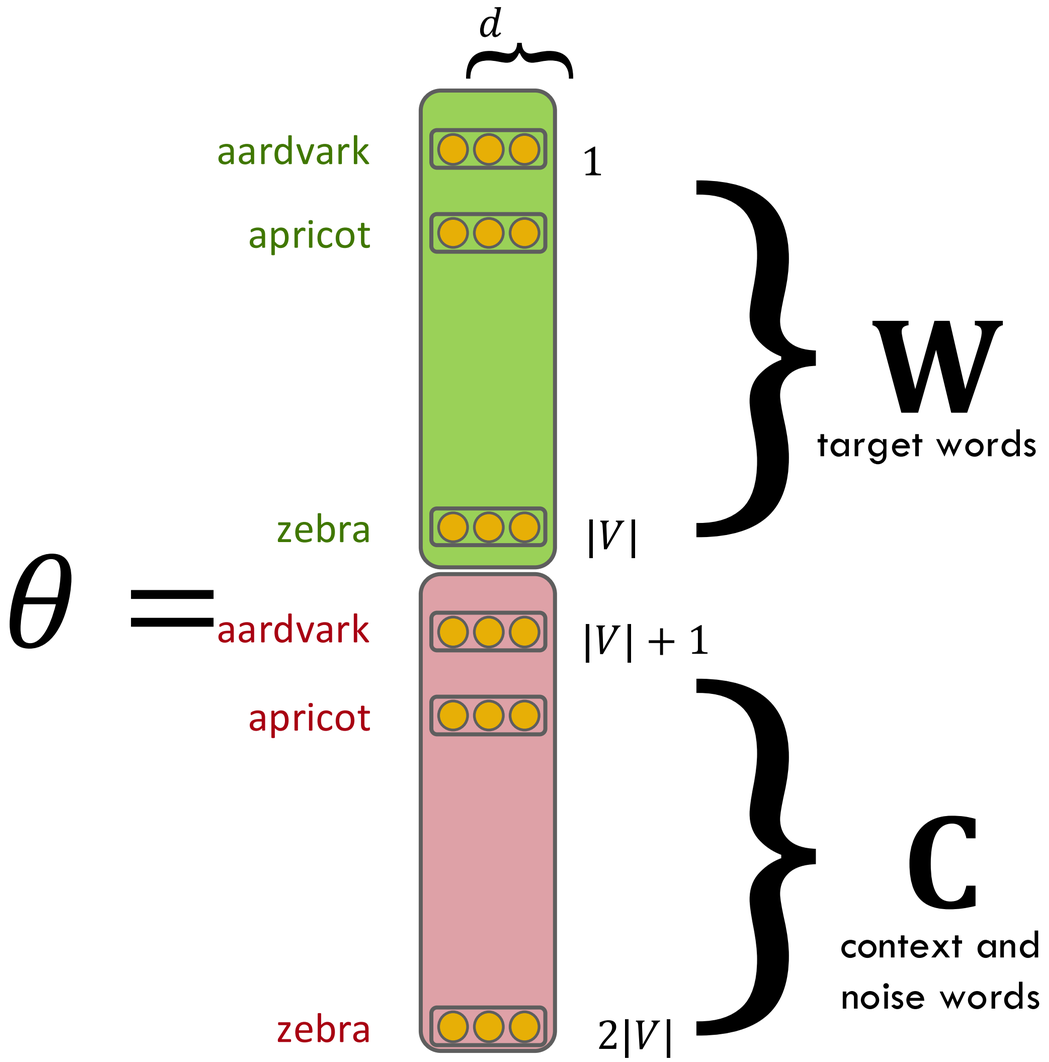
c_1 c_2 w c_3 c_4

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

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

word2vec 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
  - Separate representations for targets and contexts
  - Same as the parameters we need to estimate!



Learning word2vec embeddings

Word2vec: Training Data

For each positive example we'll grab a set of negative examples, sampling by weighted unigram frequency

c_{neg} } ...aardvark...
...zebra...

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

c_1 c_2 w c_3 c_4

Negative examples

w	c_{neg}
apricot	aardvark
apricot	zebra
apricot	where
apricot	adversarial

Positive examples

w	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Word2vec: Learning Problem

c_{neg} } …aardvark…
…zebra…

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

c_1 c_2 w c_3 c_4

Given

- the set of positive and negative training instances, and
- a set of randomly initialized embedding vectors of size $2|V|$,

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

- Maximize the similarity of the target word, context word pairs $(w, c_{1:L})$ drawn from the positive data
- Minimize the similarity of the (w, c_{neg}) pairs drawn from the negative data

Loss function

Maximize the similarity of the target with the actual context words in a window of size L, and minimize the similarity of the target with the $K > L$ negative sampled non-neighbor words

For every word, context pair…

Cross Entropy

$$L_{CE} = -\log[P(+|\mathbf{w}, \mathbf{c}_{pos})P(-|\mathbf{w}, \mathbf{c}_{neg})]$$

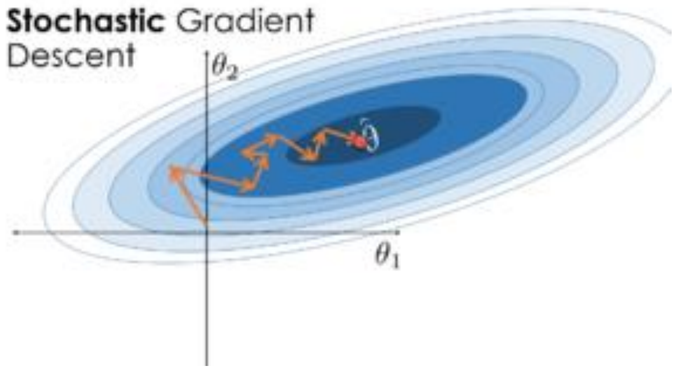
$$= -[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log P(-|\mathbf{w}, \mathbf{c}_{neg_j})]$$

$$= -[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log(1 - P(+|\mathbf{w}, \mathbf{c}_{neg_j})]$$

$$= -[\log \sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log \sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})]$$

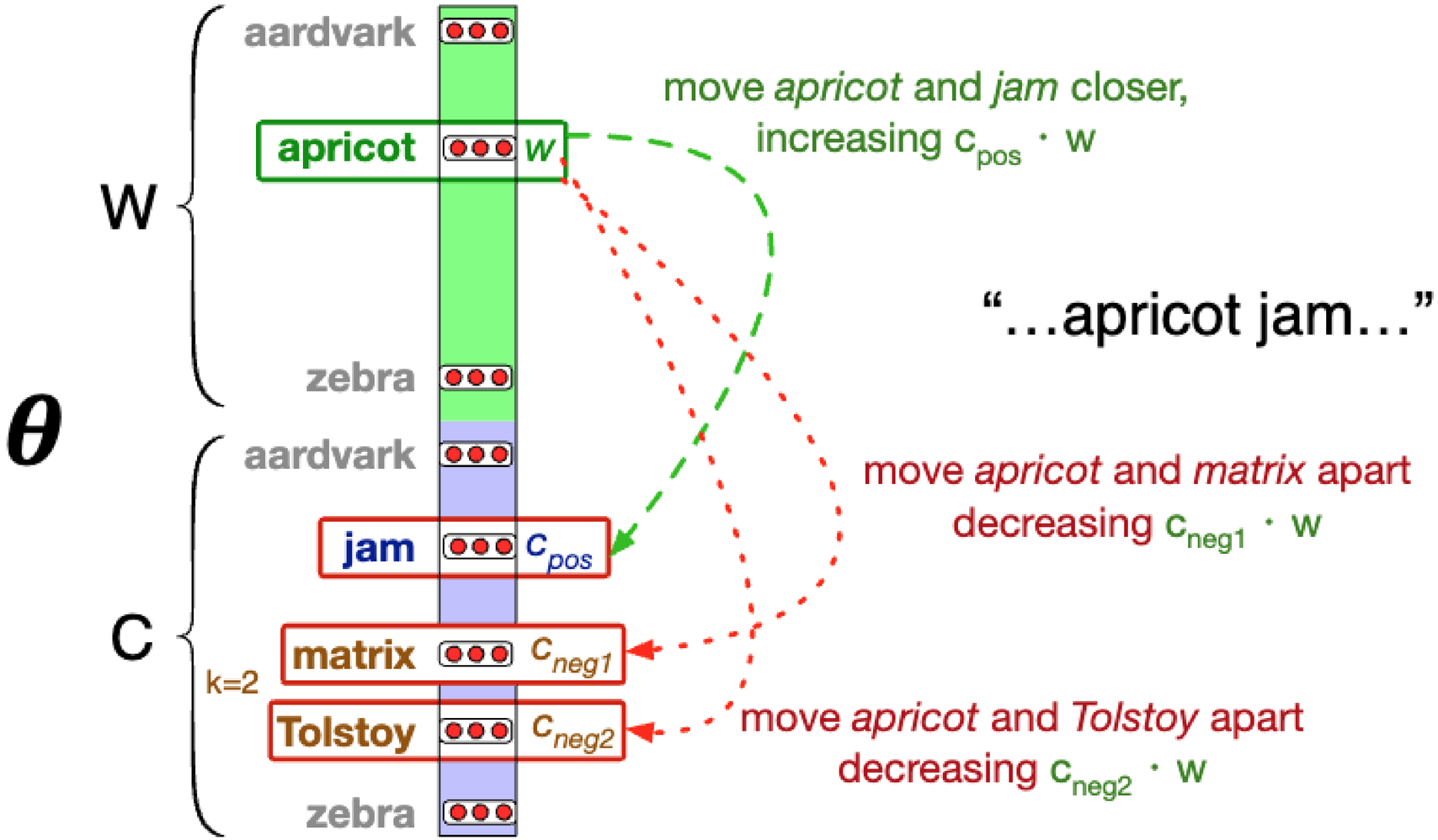
Learning the classifier

- How to learn?
  - Stochastic gradient descent!
  - Iterative process
  - Start with randomly initialized weights
    - Update the parameters (coming up)
  - Stop when the parameters do not change much…



- 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

$$w_{t+1} = w_t - \eta \frac{\partial}{\partial w} L(f(x; w), y^*)$$

At each step of gradient descent, we update the parameter w

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

SGD: Derivates

$$L_{CE} = -[\log\sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log\sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})]$$

3 different parameters

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

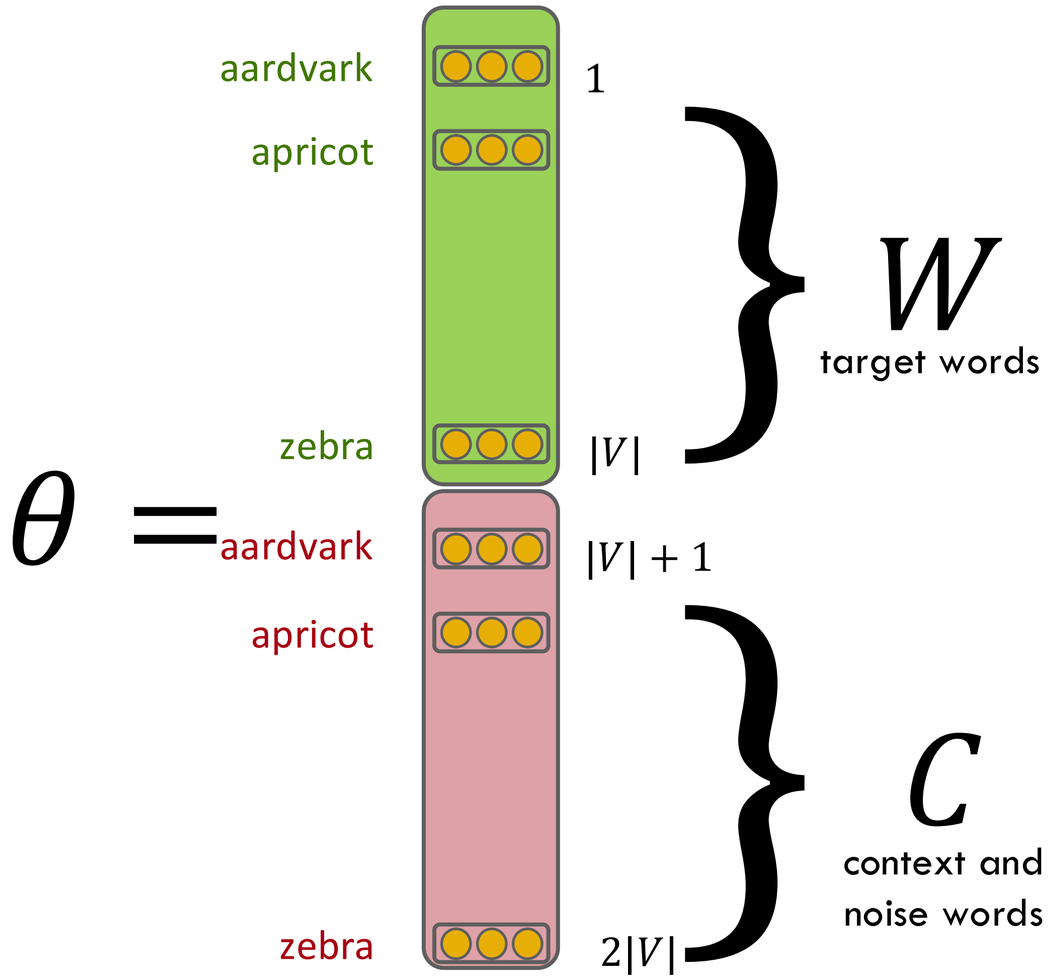
$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{neg_j}} = [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{w}$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{j=1}^{K} [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{c}_{neg_j}$$

Update the parameters by subtracting respective η-weighted gradients

word2vec: Learned Embeddings

- SGNS learns two sets of embeddings:
  - Target embeddings matrix $\mathbf{W}$
  - Context embedding matrix $\mathbf{C}$
- It's common to just add them together, representing word i as the vector $\mathbf{w}_i + \mathbf{c}_i$



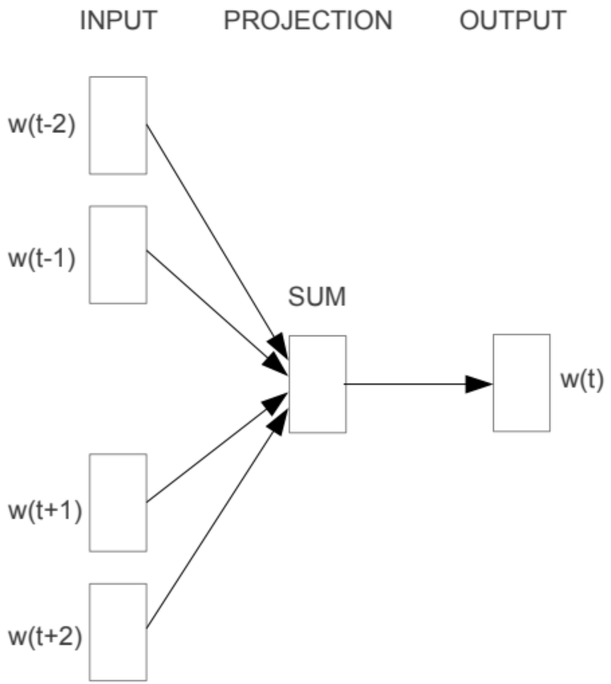
CBOW and Skipgram

- CBOW: continuous bag of words - given context, predict which word might be in the target position

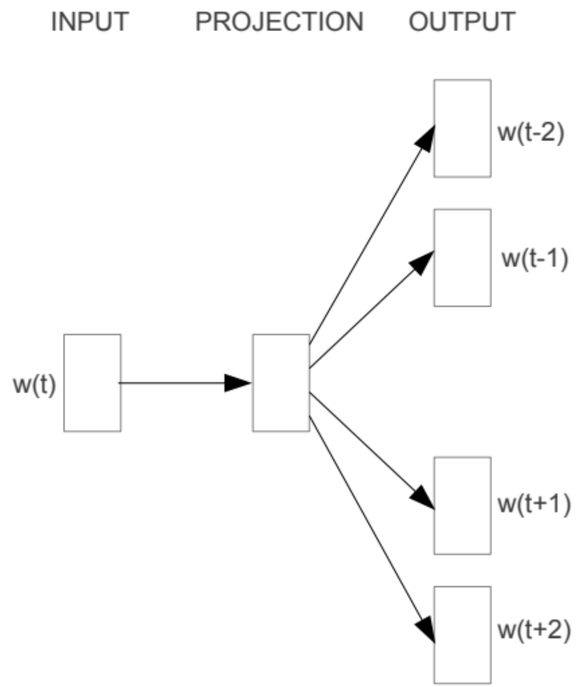
- Skip-gram: given word, predict which words make the best context

- CBOW is faster than Skip-gram. Why?

- Skip-gram generally works better



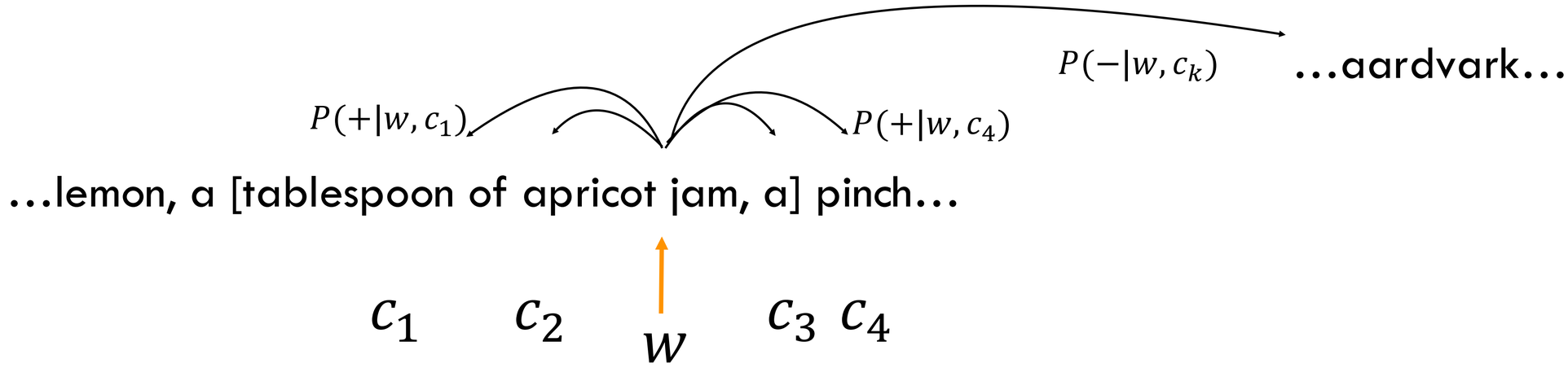
CBOW



Skip-gram

Mikolov et al., 2013. Exploiting Similarities among Languages for Machine Translation.

word2vec: Summary



- Start with $2|V|$ 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.

GloVe: Global Vectors

- Another very widely used static embedding model
  - model is based on capturing global corpus statistics
    - based on ratios of probabilities from the word-word co-occurrence matrix,
    - intuitions of count-based models like PPMI
- Builds on matrix factorization
  - Idea: store most of the important information in a fixed, small number of dimensions: a dense vector
  - Goal: Create a low-dimensional matrix for the embedding while minimizing reconstruction loss
- Fast training, scalable to huge corpora

Pennington et al., 2014

Welcome

CSCI 662 Spring 2026: NLP

🐝 Spring 2026 ⏰ Wed 2:00 - 5:20pm 📍 WPH 102

- TODOs for you
  - Finalize project teams (due on Jan 28)
  - Brainstorm project ideas
    - Project pitch on Feb 4

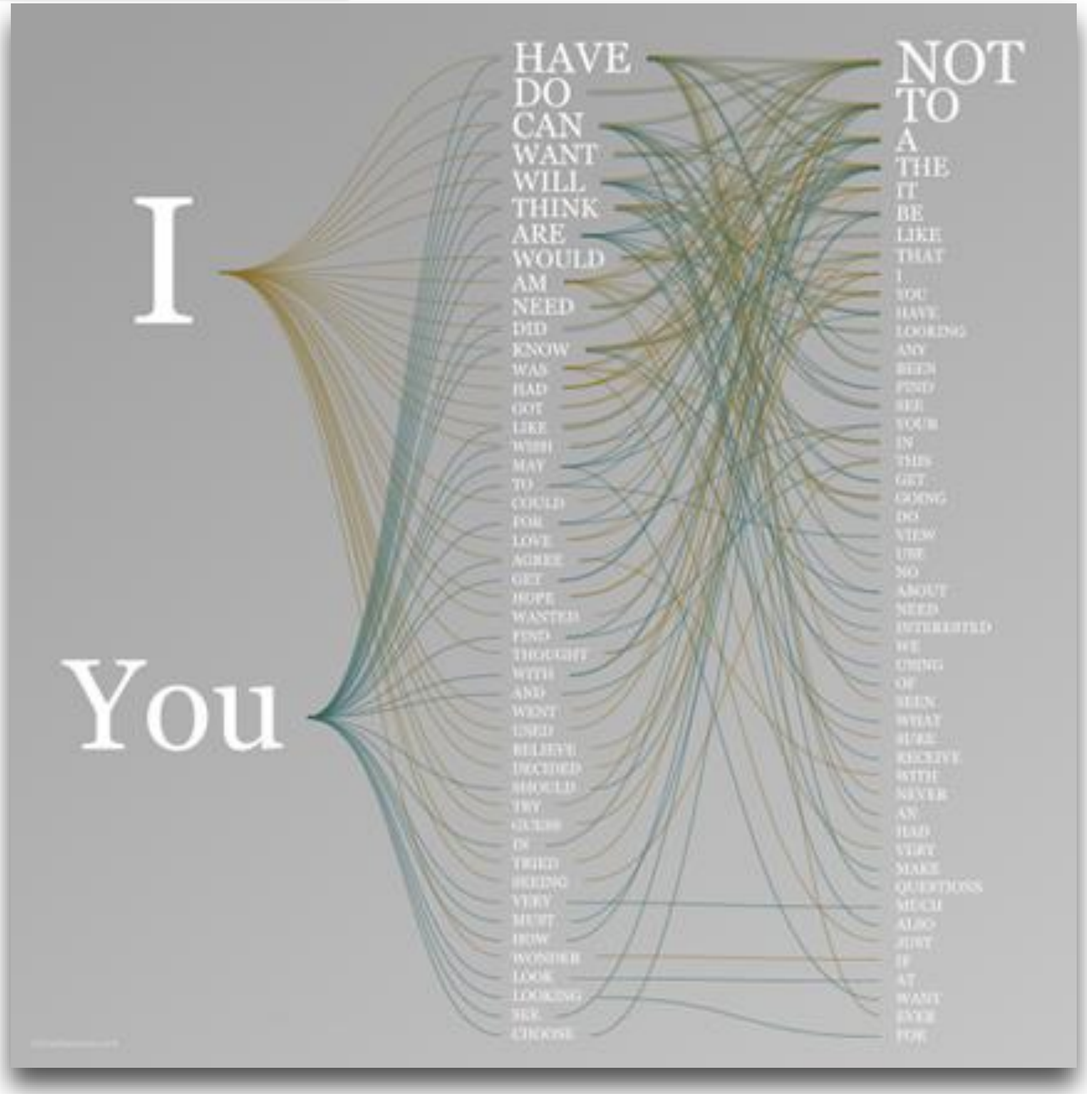


Image Courtesy: Chris Harrison