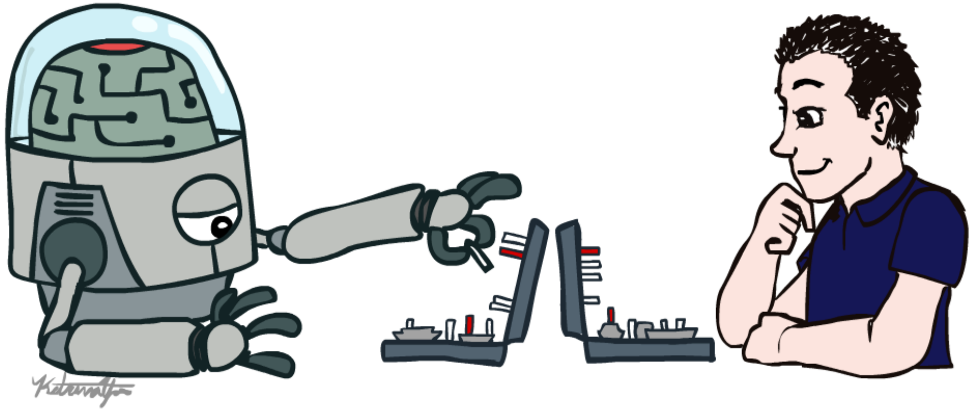
Lecture 13



Ashis Kumar Chanda
chanda@rowan.edu

Slides are adapted from Stanford NLP course

Background

- **Natural languages cannot be neatly characterized**
  - Ex: Not to be invited is sad. (grammatically correct)
  - Ex: To be not invited is sad. (grammatically incorrect)
- **One word could have multiple meanings.**
  - Ex: kitty. (it can be a pet name or a child name)
- **Natural language is ambiguous and vague.**
  - Ex: It’s a great day. (not clear how great it is)

Challenges

- **How to present a word so that a machine can understand?**
  - Using a number to a specific word
- **How to present a text/sentence/paragraph so that a machine can understand?**
  - Using a sequence of word numbers

Bag of word representations

- Suppose we have V words in a vocabulary.
- Every word has an index in vocabulary.
- Now, a text can be represented by an array of size V where an index of the array presents a word in vocabulary.
- If a word is present in the text, then the word index value would be 1 in the array.
- Otherwise, it would be 0.

Bag of word representations

Sentence: My school bag is red

Array:	0	1	2	3	4	5	6	.. .. ..	V
	1	1	0	1	1	1	0	.. .. ..	0

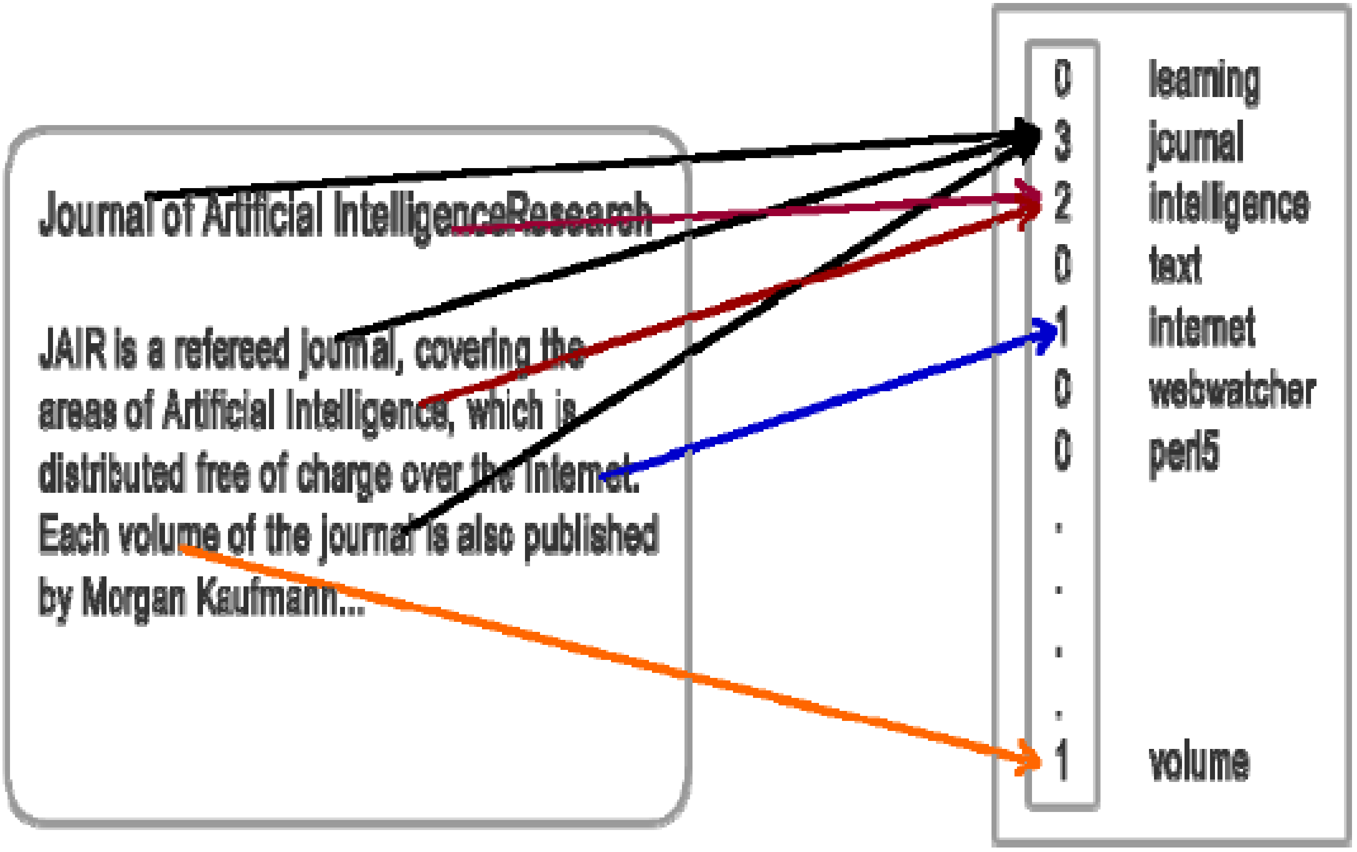
We can say the array as a word vector

0 my
1 is
2 the
3 school
4 bag
5 red
…
…
…

V zoo

English vocabulary with V words

Bag of word representations



Bag of word representations

- Discards word position information
  - Is it a problem?
  - Depends on the type of tasks
    - Topic categorization (Maybe not)
    - Sentiment Recognition (Yes)

"This movie is not good. It was not exciting at all. The best part was the popcorn. Good luck finding someone who enjoyed any minute of it."

Problem: Understanding the sentiment of a movie review

- Suppose we have 5000 movie review dataset.
- 2500 reviews are positive, and 2500 reviews are negative.

Review	Sentiment
A technical marvel, it also manages to be emotionally charged and involving	positive
3 hours long movie! Again, I don't like to see that Jack died at the end!	negative

Problem: Understanding the sentiment of a movie review

- Suppose we have 5000 movie review dataset.
- 2500 reviews are positive, and 2500 reviews are negative.
- If we have a new movie review, can we identify it as "positive" or "negative" using an intelligent system?
  - Binary classification problem
  - How to represent a movie review?
    - BOW? BOW can't understand the meaning of "A technical marvel".

Relation: **Similarity**

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

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

Ask humans how similar 2 words are

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

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

Computational models of word meaning

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

- We'll introduce **vector semantics**
- The standard model in language processing!
- Handles many of our goals!

Let's define words by their usages

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

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

What does recent English borrowing *ongchoi* mean?

- Suppose you see these sentences:
  - Ong choi is delicious **sautéed with garlic**.
  - Ong choi is superb **over rice**
  - Ong choi **leaves** with salty sauces
- And you've also seen these:
  - …spinach **sautéed with garlic over rice**
  - Chard stems and **leaves** are **delicious**
  - Collard greens and other **salty** leafy greens
- Conclusion:
  - Ongchoi is a leafy green like spinach, chard, or collard greens

Ongchoi: *"Water Spinach"*

Vietnamese: Ongchoi
Bangla: কলমি শাক

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 space (Osgood et al. 1957)

- 3 affective dimensions for a word
  - **valence**: pleasantness
  - **arousal**: intensity of emotion
  - **dominance**: the degree of control exerted

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

NRC VAD Lexicon
(Mohammad 2018)

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

Idea 1: Defining meaning by linguistic distribution

Idea 2: Meaning as a point in multidimensional space

Defining meaning as a point in space based on distribution

- Each word = a vector   (not just "good" or "w_{45}")
- Similar words are "**nearby in semantic space**"
- We build this space automatically by seeing which words are **nearby in text**

We define meaning of a word as a vector

- Called an "embedding" because it's embedded into a space
- The standard way to represent meaning in NLP
- **Every modern NLP algorithm uses embeddings as the representation of word meaning**
- Fine-grained model of meaning for similarity

Intuition: why vectors?

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

We'll discuss 2 kinds of embeddings

- tf-idf
  - Information Retrieval workhorse!
  - A common baseline model
  - **Sparse** vectors
  - Words are represented by (a simple function of) the **counts** of nearby words
- Word2vec
  - **Dense** vectors
  - Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
  - Later we'll discuss extensions called **contextual embeddings**

From now on:

Computing with meaning representations instead of string representations

荃者所以在鱼，得鱼而忘荃	Nets are for fish; Once you get the fish, you can forget the net.
言者所以在意，得意而忘言	Words are for meaning; Once you get the meaning, you can forget the words

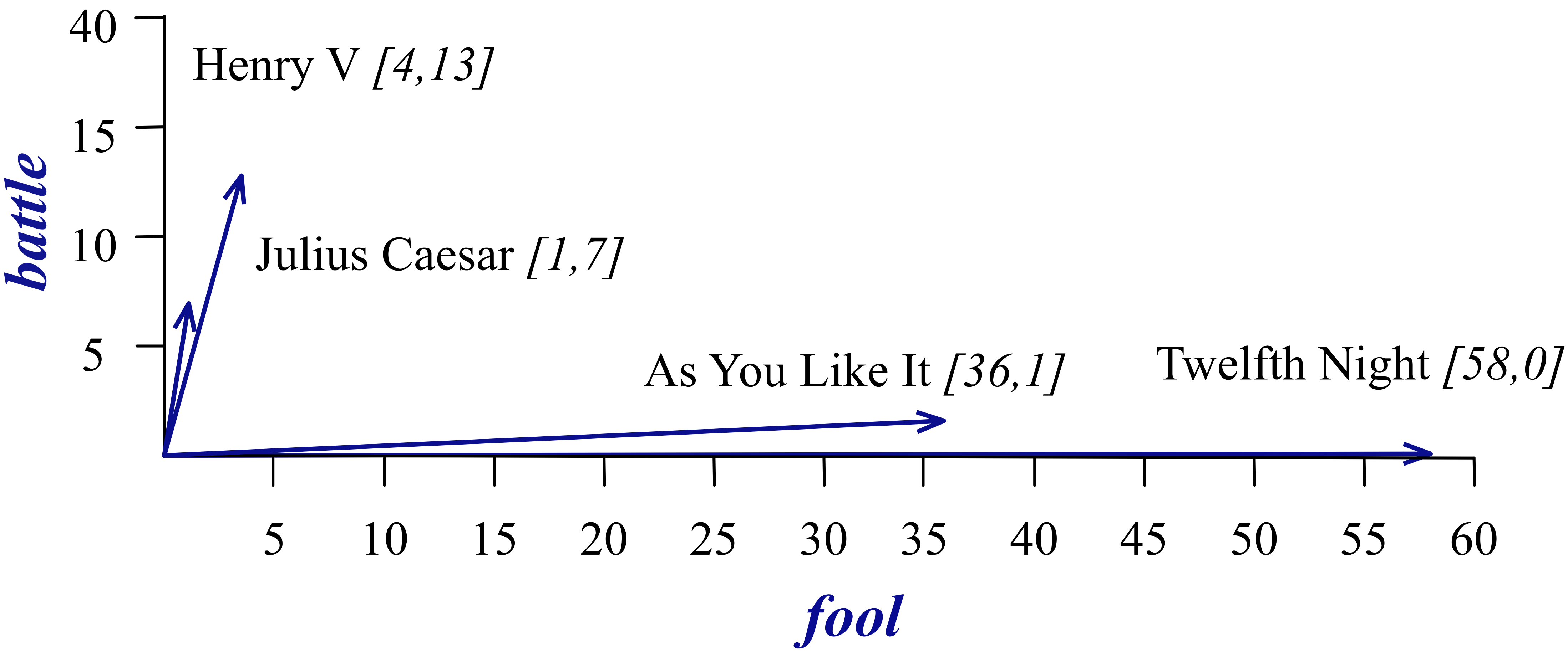
庄子(Zhuangzi), Chapter 26

Term-document matrix

Each document is represented by a vector of words

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

Visualizing document vectors



Vectors are the basis of information retrieval

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

- Vectors are similar for the two comedies
- But comedies are different than the other two
- Comedies have more *fools* and *wit* and fewer *battles*.

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

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

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

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

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

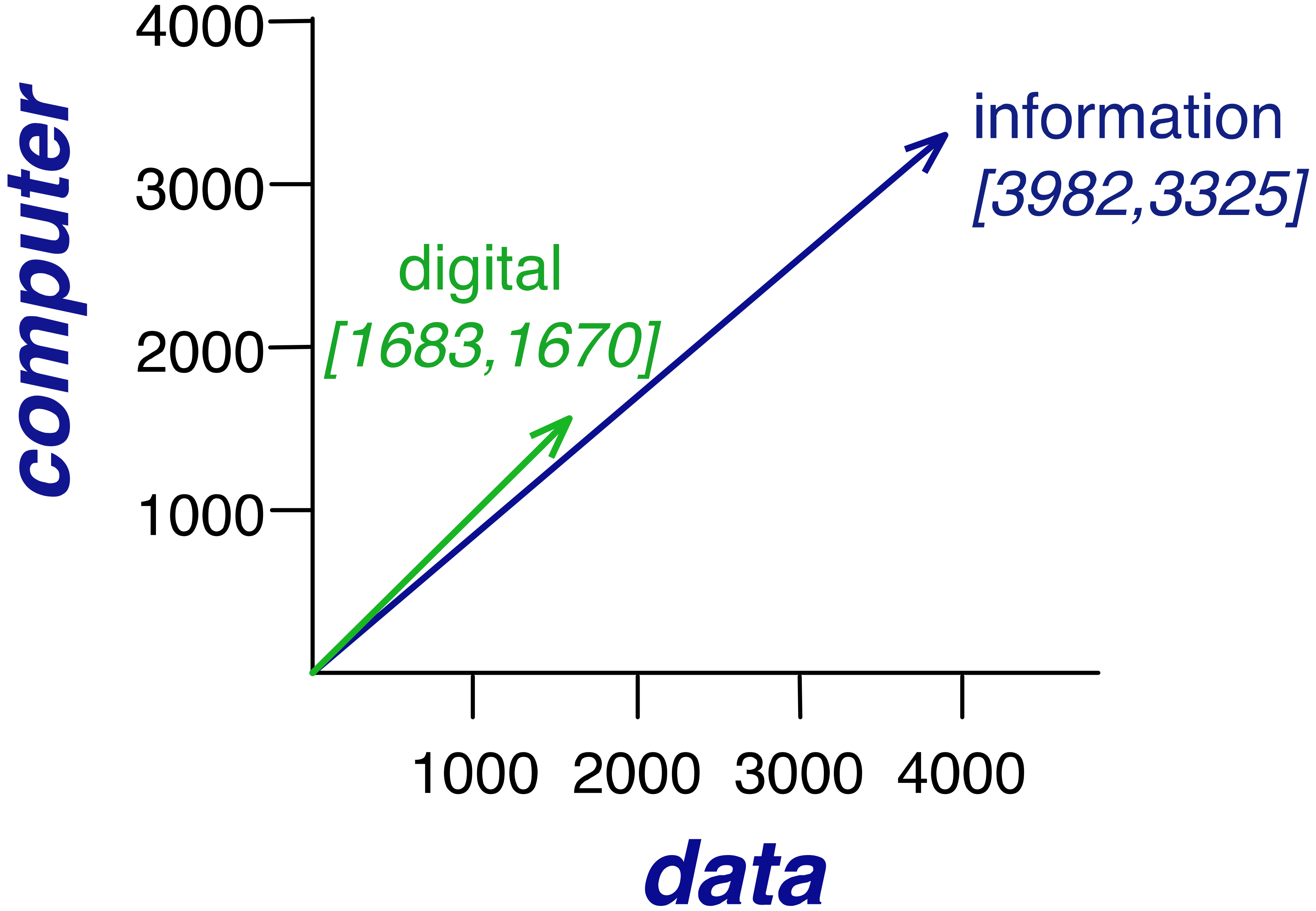
- Two **words** are similar in meaning if their context vectors are similar

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

Word →

Word ↓

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

Words and Vectors

- We've seen that a word can be simply represented as a vector of counts, a fundamental idea that underlies all embedding representations.

Words and Vectors

- To measure similarity between two words, we need a metric that compares two vectors.
- By far the most common similarity metric is the **cosine** of the angle between the vectors.

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 (right angles)

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

Cosine examples

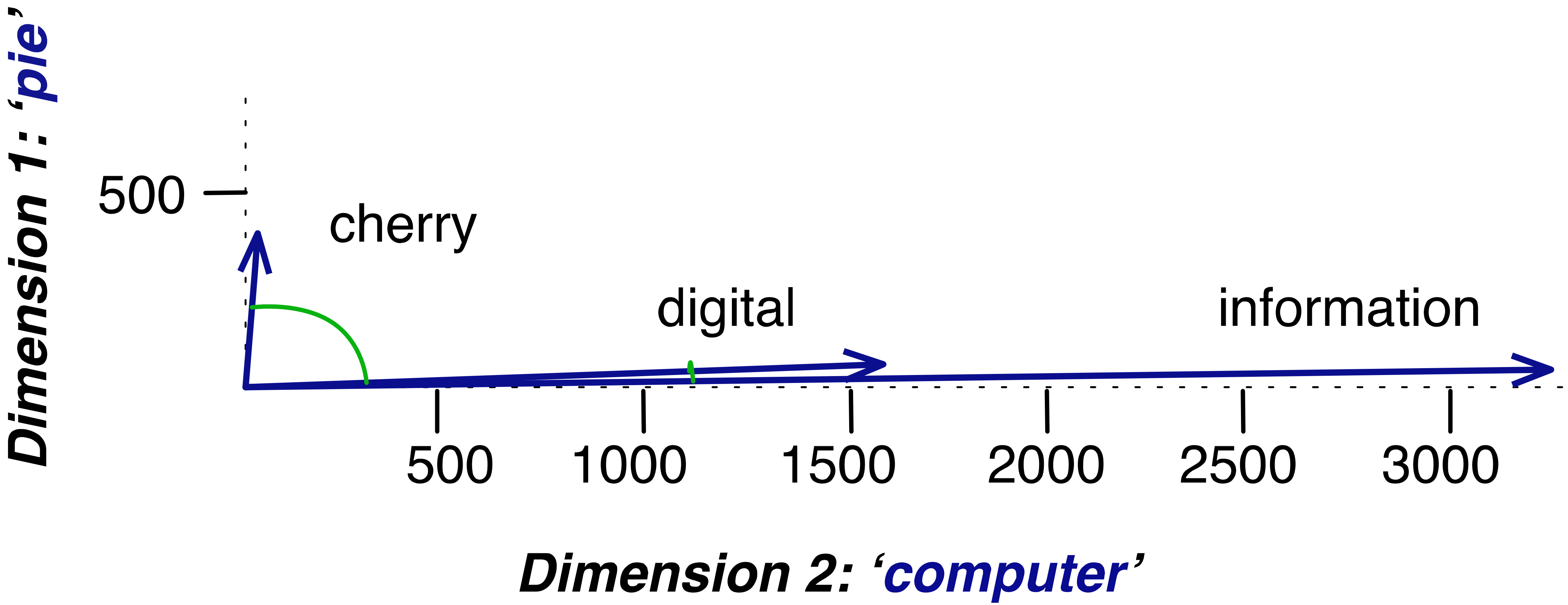
$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\cdot\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

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

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

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

Visualizing cosines (well, angles)



TF-IDF

a common way to reweight counts in term-document matrices

But raw frequency is a bad representation

- The co-occurrence matrices we have seen represent each cell by word frequencies.
- Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.
- But overly frequent words like *the*, *it*, or *they* are not very informative about the context
- It's a paradox! How can we balance these two conflicting constraints?

Two common solutions for word weighting

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

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

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

- **PMI:** (Pointwise mutual information)
  - 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)

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

- Instead of using raw count, we squash a bit:

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

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

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

Log(1) = 0

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

What is a document?

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

Final tf-idf weighted value for a word

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

- Raw counts:

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

- tf-idf:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	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

- **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 large corpora
    - Imagine w1 and w2 whose probability is each 10^{-6}
- 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

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

	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

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

- 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

- PMI is biased toward infrequent events
  - Very rare words have very high PMI values
- Two solutions:
  - Give rare words slightly higher probabilities
  - 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

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

Sparse versus dense vectors

- Why dense vectors?
  - Short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
  - Dense vectors may **generalize** better than explicit counts
  - Dense vectors may do better at capturing synonymy:
    - *car* and *automobile* are synonyms; but are distinct dimensions
      - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
  - **In practice, they work better**

Common methods for getting short dense vectors

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

Simple static embeddings you can download!

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

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

Word2vec

- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: **predict** rather than **count**
- Word2vec provides various options. We'll do:
- **skip-gram with negative sampling (SGNS)**

Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don't actually care about this task
    - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
    - A word c that occurs near apricot in the corpus 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 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 Training Data

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

…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:

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

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

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

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

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

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

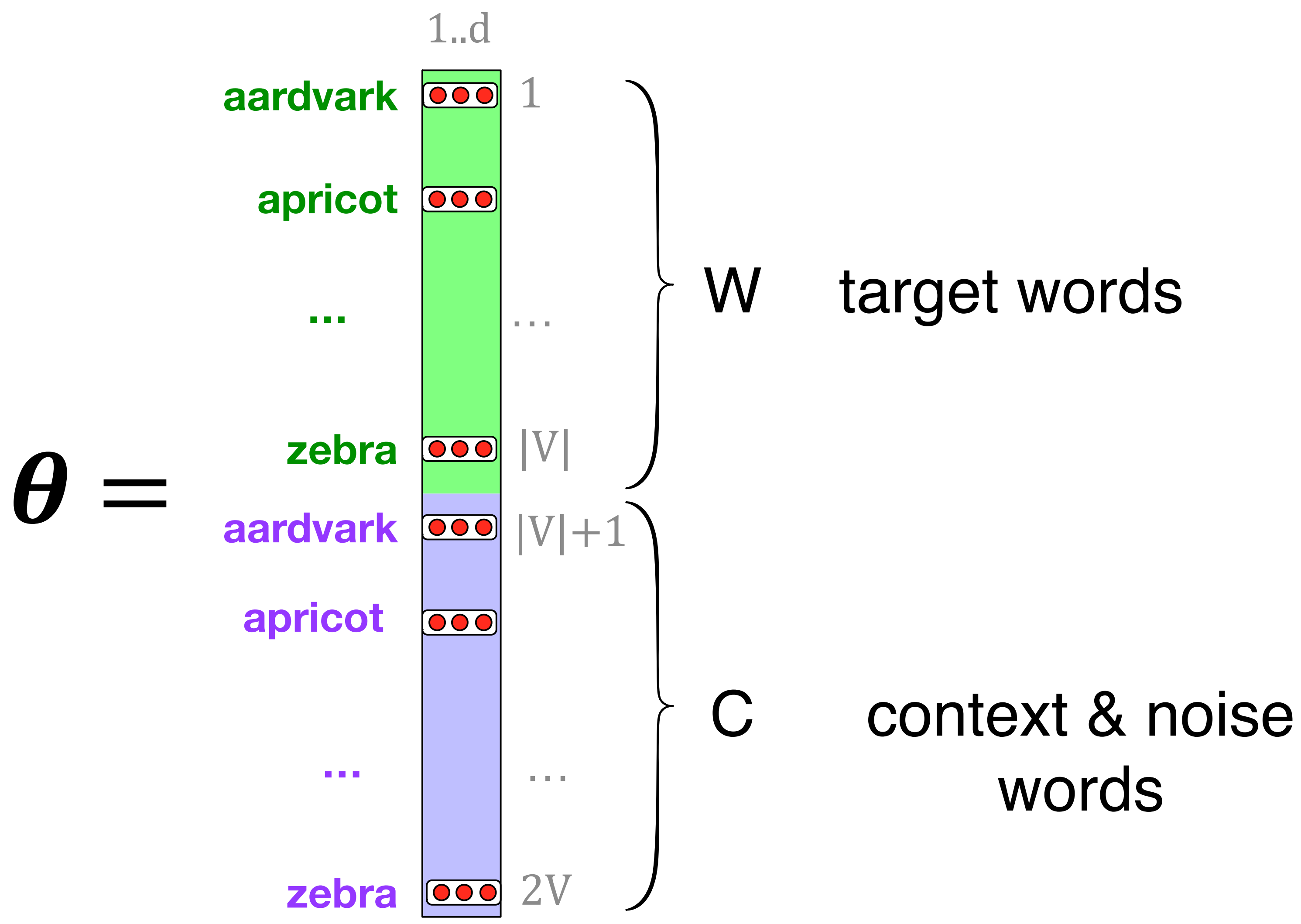
$$\log P(+|w,c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

Skip-gram classifier: summary

- A probabilistic classifier, given
  - a test target word w
  - its context window of L words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

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

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



Word2vec

Let's talk about various properties and parameters of embeddings

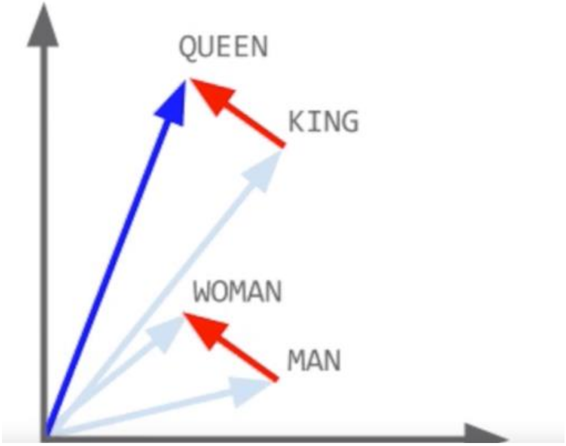
The kinds of neighbors depend on window size

- **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*

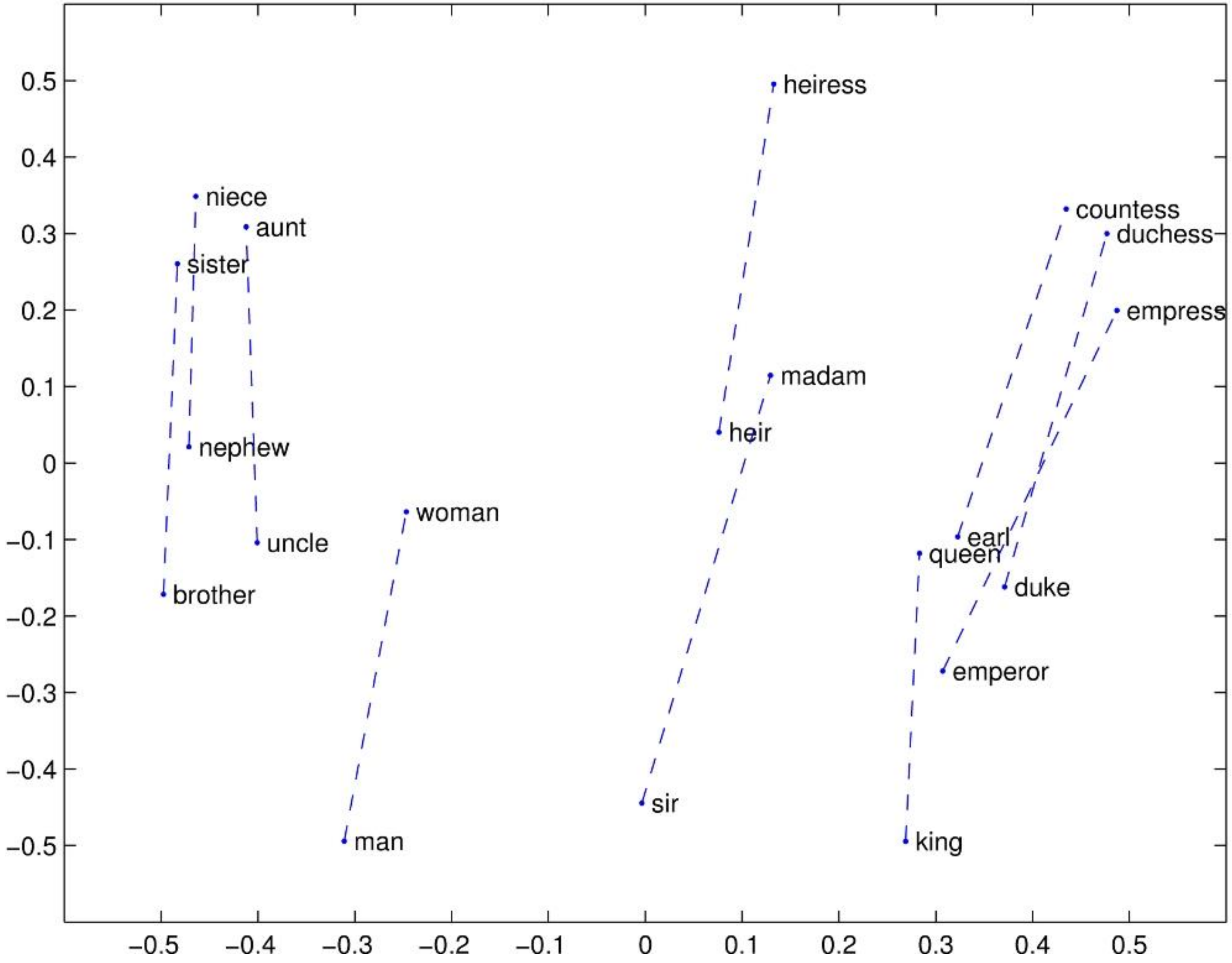
Analogical relations via parallelogram

- The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)
- king – man + woman is close to queen
- Paris – France + Italy is close to Rome
- For a problem a:a*::b:b*, the parallelogram method is:

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



Structure in GloVE Embedding space



Caveats with the parallelogram method

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

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

Embeddings as a window onto historical semantics

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

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

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

Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

- Ask "Paris : France :: Tokyo : x"
  - x = Japan
- Ask "father : doctor :: mother : x"
  - x = nurse
- Ask "man : computer programmer :: woman : x"
  - x = homemaker

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

Let’s go back to our old problem

Movie review classification

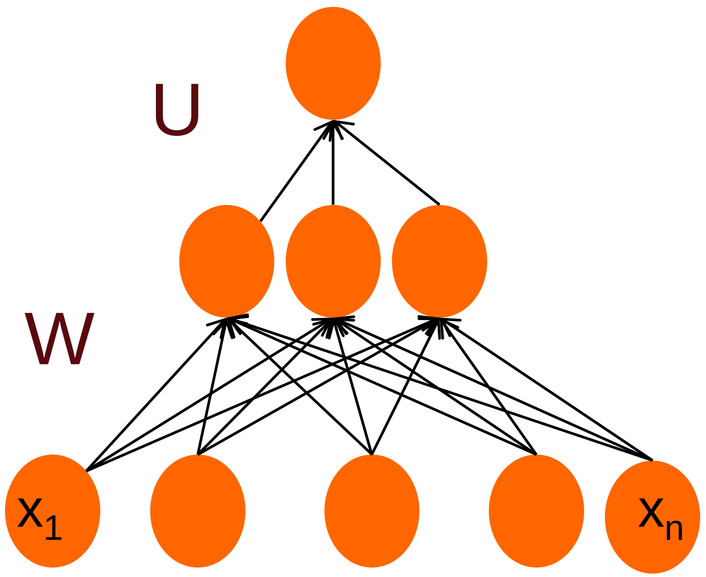
Problem: Understanding the sentiment of a movie review

- Suppose we have 5000 movie review dataset.
- 2500 reviews are positive, and 2500 reviews are negative.

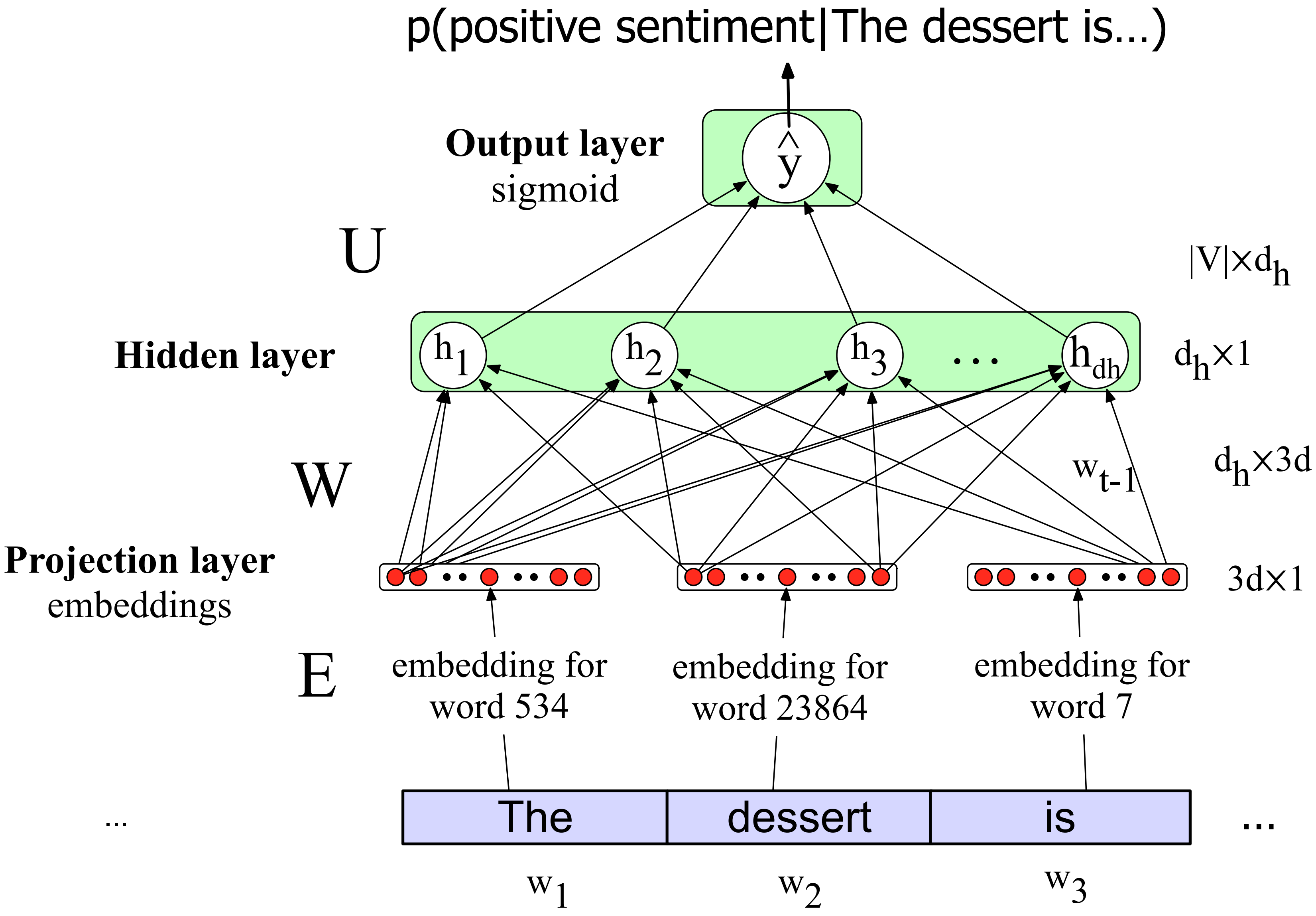
Review	Sentiment
A technical marvel, it also manages to be emotionally charged and involving	positive
3 hours long movie! Again, I don't like to see that Jack died at the end!	negative

Problem: Understanding the sentiment of a movie review

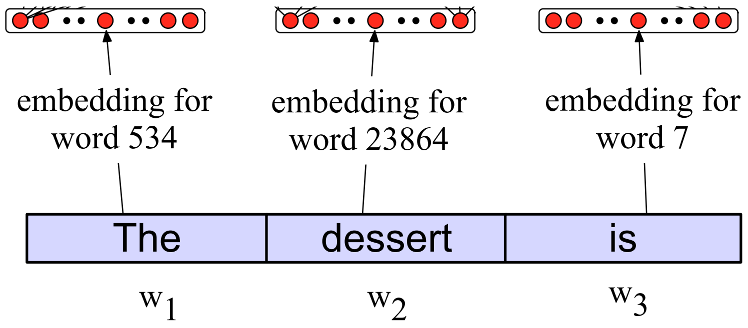
- Plan to use feedforward networks
- We could do exactly what we did with XOR example
- Output layer is 0 or 1 as before
- Input layer are word vectors



Neural Net Classification with embeddings as input features!



Issue: texts come in different sizes



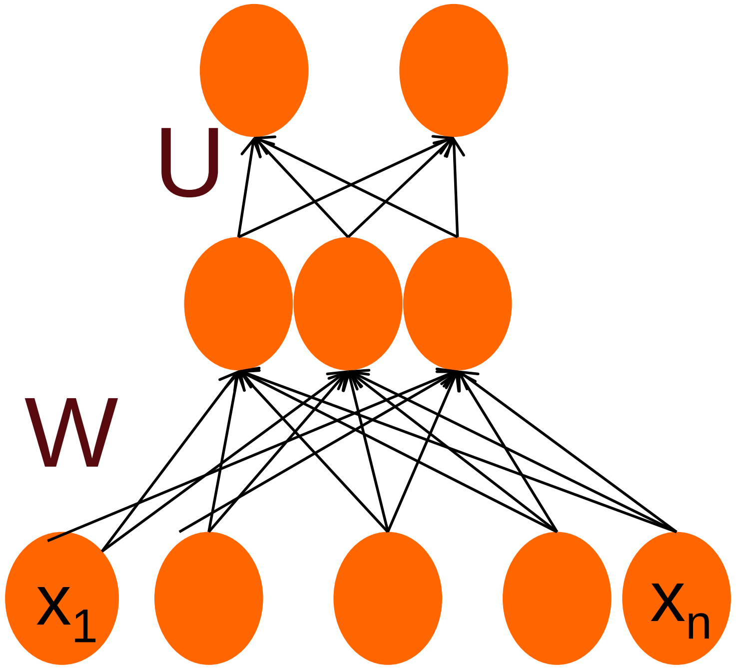
- This assumes a fixed size length (3)!
- Kind of unrealistic.
- Some simple solutions

1. Make the input the length of the longest review
   - If shorter then pad with zero embeddings
   - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings
   - Take the element-wise max of all the word embeddings
     - For each dimension, pick the max value from all words

Reminder: Multiclass Outputs

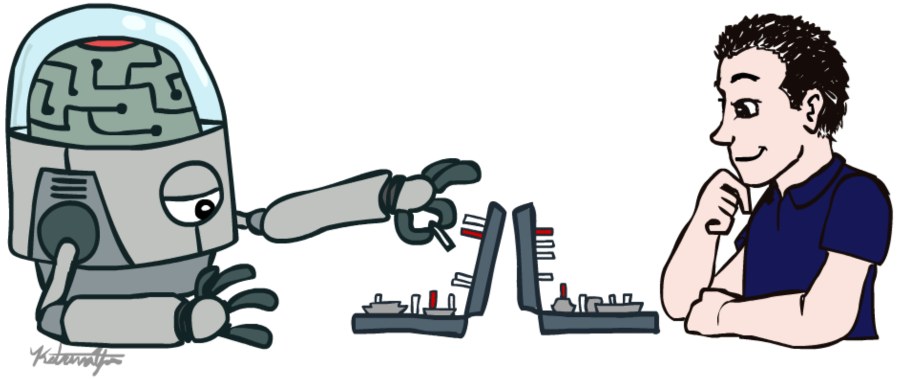
- What if you have more than two output classes?
- Ex: Positive, Negative, Neutral movie review
  - Add more output units (one for each class)
  - And use a "softmax layer"

$$\mathrm{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq D$$



Reference

- https://web.stanford.edu/~jurafsky/slp3/6.pdf



Thanks!