CIS 421/521:
ARTIFICIAL INTELLIGENCE

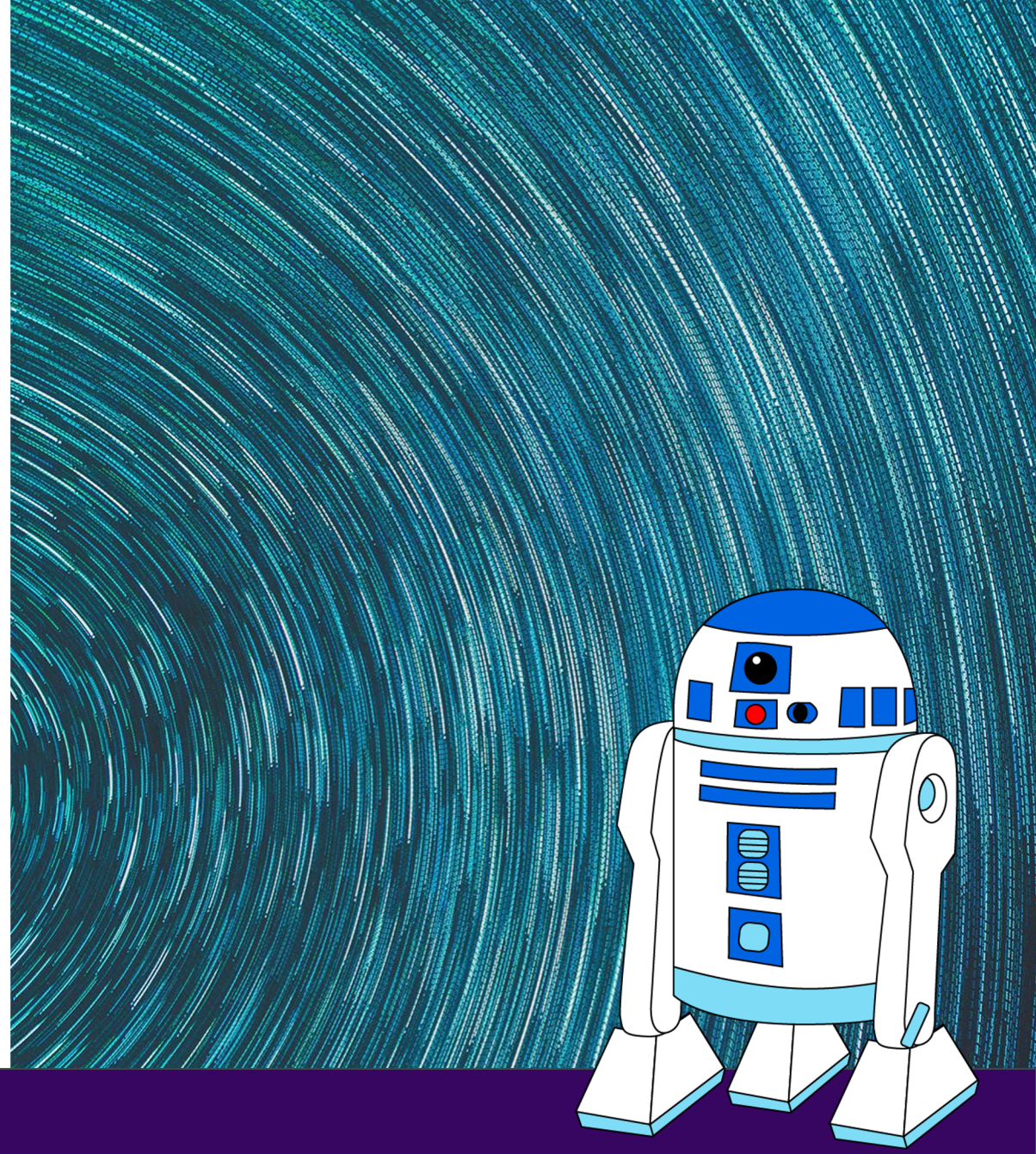
Module 14: Natural Language Processing

Penn Engineering
University of Pennsylvania

CIS 421/521:
ARTIFICIAL INTELLIGENCE

Neural Networks and Neural LMs

Jurafsky and Martin Chapters 7 and 9



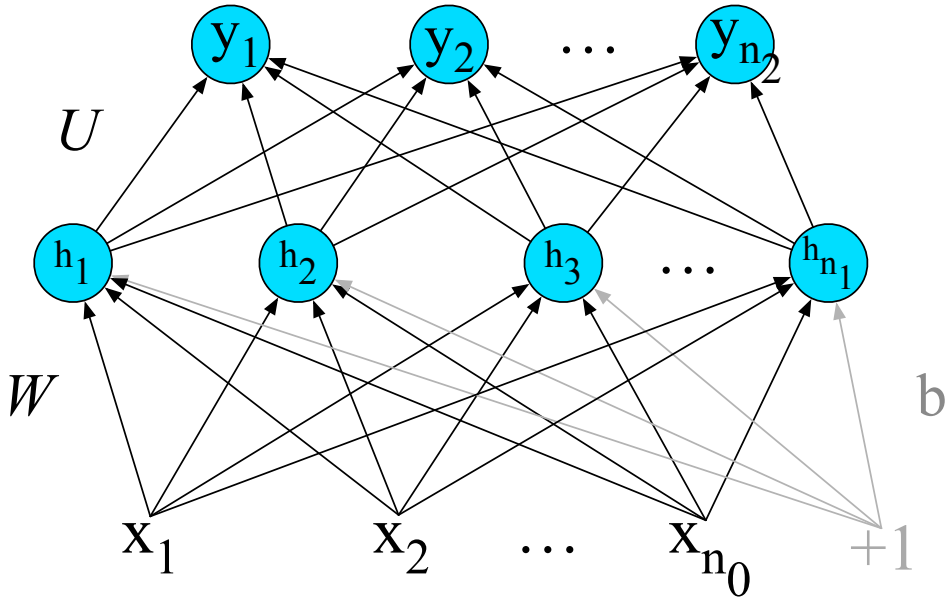
Penn Engineering
University of Pennsylvania

Review: Feed-Forward Neural Network

The simplest kind of is the **Feed-Forward Neural Network**

Multilayer network, all units are usually **fully-connected**, and **no cycles**.

The outputs from each layer are passed to units in the next higher layer, and no outputs are passed back to lower layers.



Layer 2 (output layer)

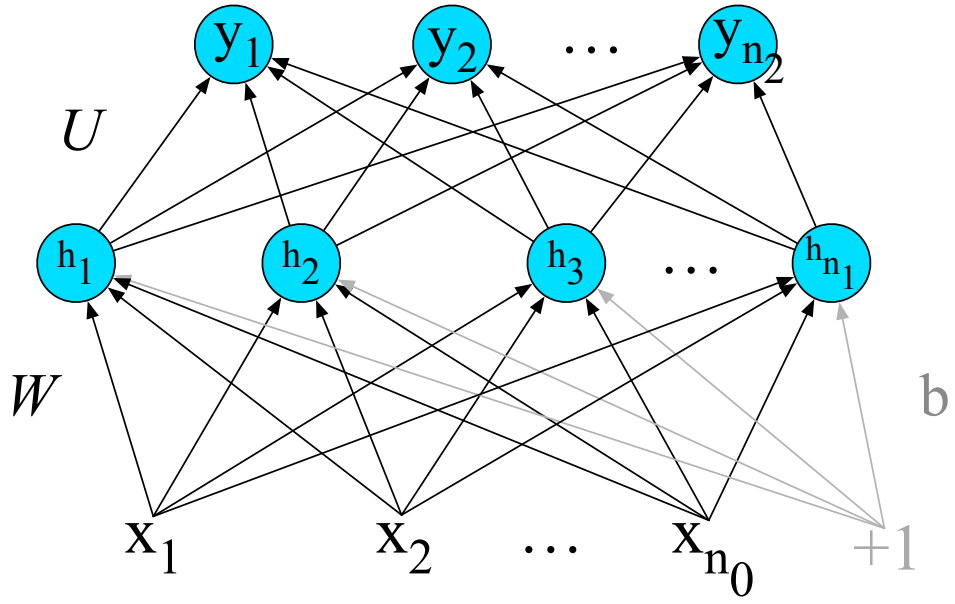
Layer 1 (hidden layer)

Layer 0 (input layer)

Review: Feed-Forward Neural Network

A single hidden unit has parameters $\boldsymbol{w}$ (the weight vector) and $\boldsymbol{b}$ (the bias scalar).

We represent the parameters for the **entire hidden layer** by combining the weight vector $\mathbf{w}_i$ and bias $\boldsymbol{b}_i$ for each unit $\boldsymbol{i}$ into a single weight matrix $\boldsymbol{W}$ and a single bias vector $\mathbf{b}$ for the whole layer.



Layer 2 (output layer)

Layer 1 (hidden layer)

Layer 0 (input layer)

Review: Feed-Forward Neural Network

The advantage of using a single matrix ***W*** for the weights of the entire layer is the hidden layer computation can be done efficiently with simple matrix operations.

The computation has three steps:

1. multiplying the weight matrix by the input vector *x*,
2. adding the bias vector *b*, and
3. applying the activation function *g* (such as Sigmoid)

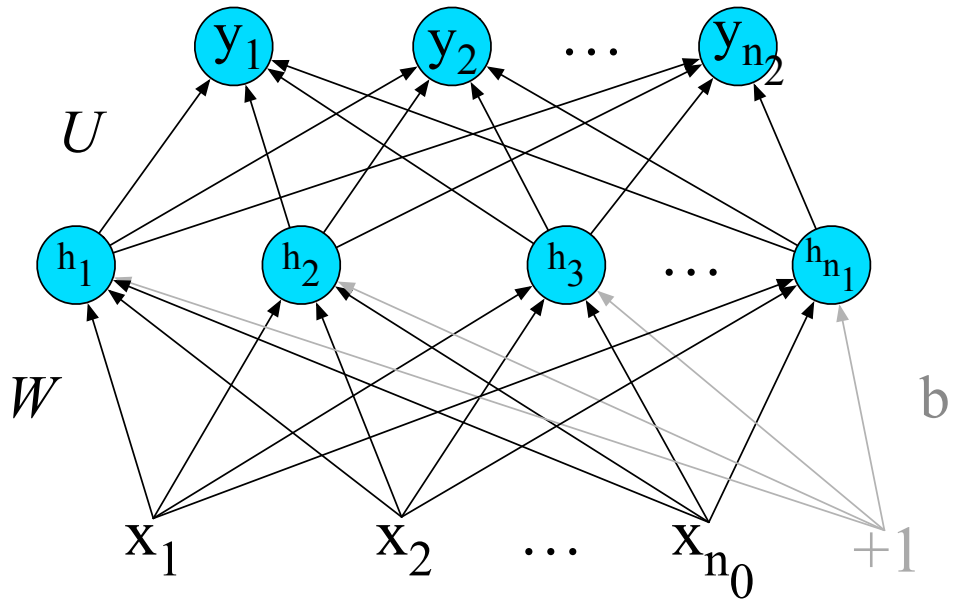
The output of the hidden layer, the vector *h*, is thus the following, using the sigmoid function σ:

$$h = \sigma(Wx+b)$$

Review: Feed-Forward Neural Network

Like the hidden layer, the output layer has a weight matrix ***U***.
Its weight matrix is multiplied by its input vector (***h***) to produce the intermediate output ***z***.

$$z=Uh$$



Layer 2 (output layer)

Layer 1 (hidden layer)

Layer 0 (input layer)

Review: Feed-Forward Neural Network

Here are the final equations for a feedforward network with a single hidden layer, which takes an input vector x, outputs a probability distribution y, and is parameterized by weight matrices W and U and a bias vector b:

$h = \sigma(Wx+b)$
$z = Uh$
$y = \text{softmax}(z)$

Like with logistic regression, softmax normalizes the output and turns it into a probability distribution.

Training Neural Nets

Like logistic regression, we want to learn the best parameters for the neural net to make its predictions $\hat{y}$ as close to possible as the gold standard labels in our training data *y*.

What do we need?

A loss function – cross-entropy loss

An optimization algorithm – gradient descent

A way of computing the gradient of the loss function – error propagation

Cross-Entropy Loss

If the neural network is a binary classifier with a sigmoid at the final layer, the loss function is exactly the same as we saw in logistic regression:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

Cross-Entropy Loss

If the neural network is a binary classifier with a sigmoid at the final layer, the loss function is exactly the same as we saw in logistic regression:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

For multinomial classification

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log \hat{y}_i$$

If there is only one correct answer, where the truth is $y_i=1$, then this simplifies to be

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i$$

Plugging into softmax:

$$L_{CE}(\hat{y}, y) = -\log \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Computing the gradient

Logistic regression can be thought of as a network with just one weight layer and a sigmoid output. In that case the gradient is:

$$\frac{\partial LCE(w,b)}{\partial wj} = (\hat{y} - y)\, xj$$

$$= (\sigma(w \cdot x + b) - y)x_j$$

But these derivatives only give correct updates for the last weight layer!

For deeper networks, computing the gradients requires looking back through all the earlier layers in the network, even though the loss is only computed with respect to the output of the network.

Solution: **error backpropagation algorithm**

Computation Graphs

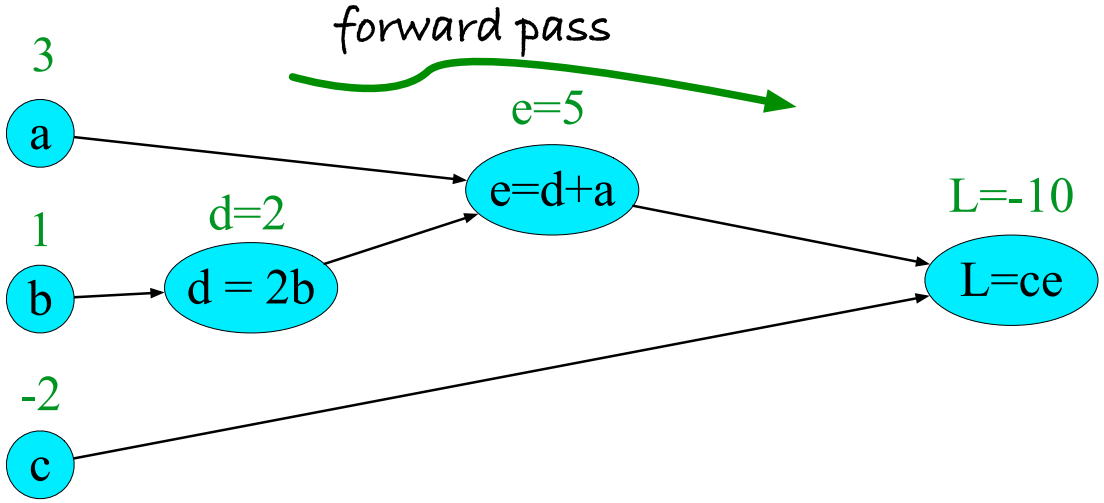
Although backpropagation was invented for neural nets, it is related to general procedure called **backward differentiation**, which depends on the notion of **computation graphs**.

A computation graph represents the process of computing a mathematical expression. The computation is broken down into separate operations. Each operation is a node in a graph.

$$L(a, b, c) = c(a + 2b)$$

$$d = 2 * b$$
$$e = a + d$$
$$L = c * e$$

Forward pass



$L(a, b, c) = c(a + 2b)$

$d = 2*b$
$e = a+d$
$L = c*e$

inputs $a = 3$, $b = 1$, $c = -2$,

Backward differentiation

The importance of the computation graph comes from the backward pass, which is used to compute the derivatives that we'll need for the weight update.

How do we compute the derivative of our output function L with respect to the input variables a, b, and c?

Backwards differentiation uses the **chain rule** from calculus.

$$\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b}, \text{ and } \frac{\partial L}{\partial c}$$

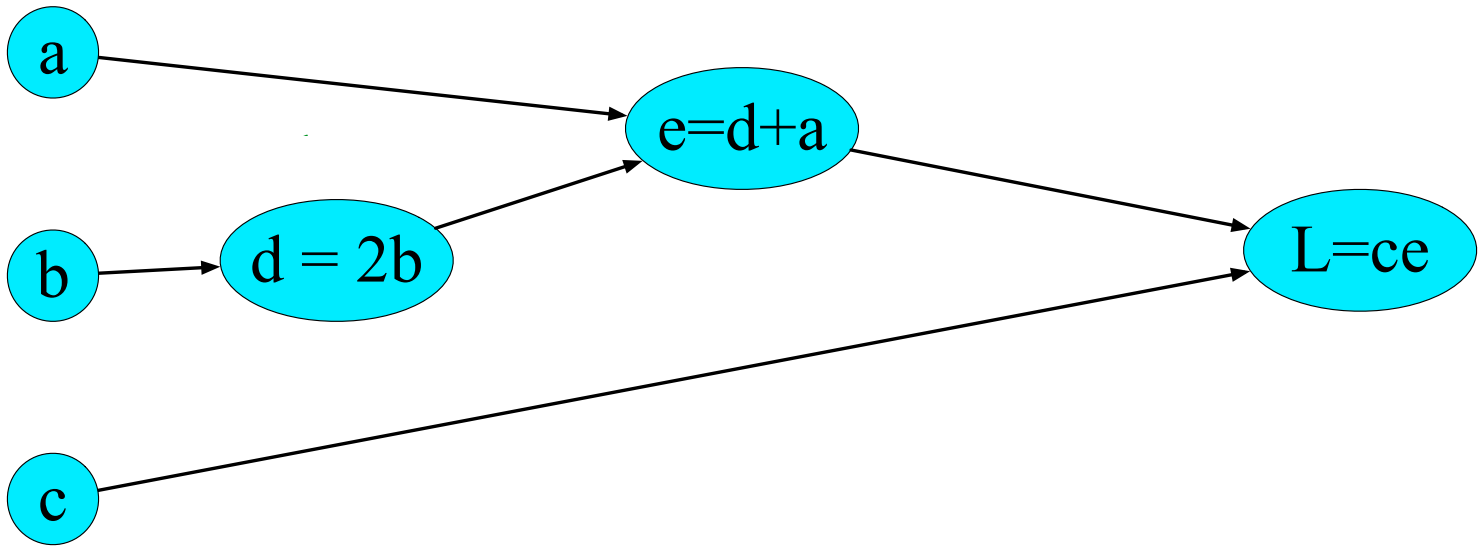
Chain rule

For a composite function $f(x) = u(v(x))$, the derivative of $f(x)$ is:

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

Similarly for, $f(x) = u(v(w(x)))$, the derivative of $f(x)$ is:

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx}$$

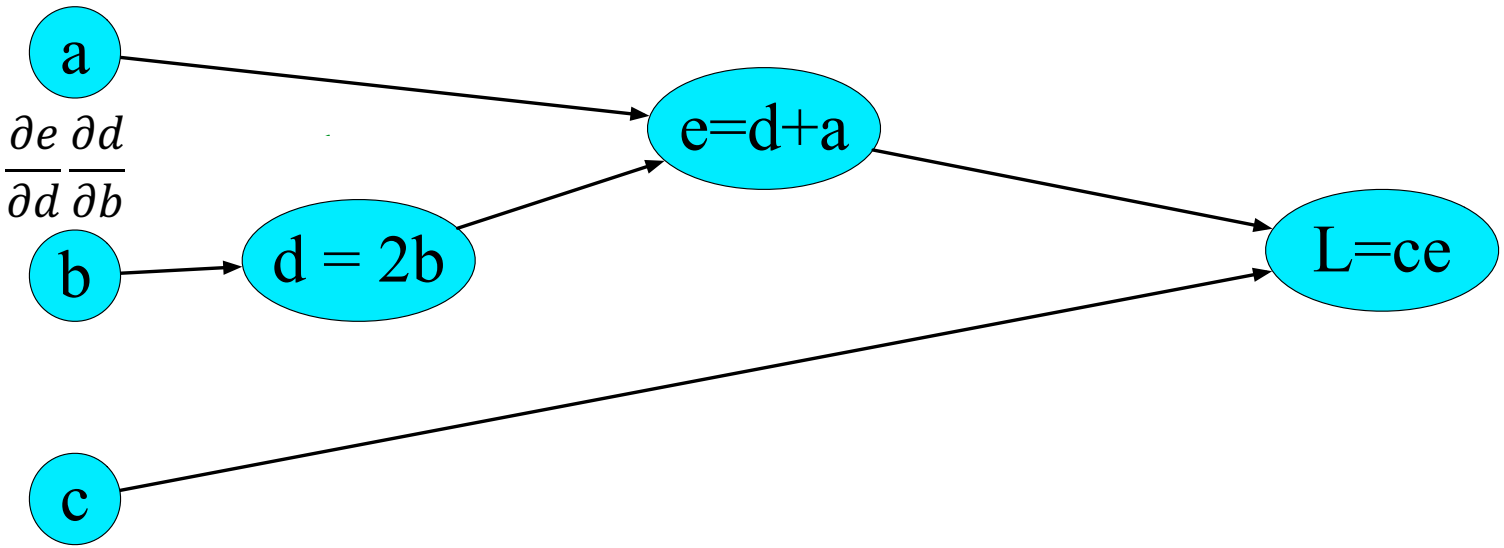


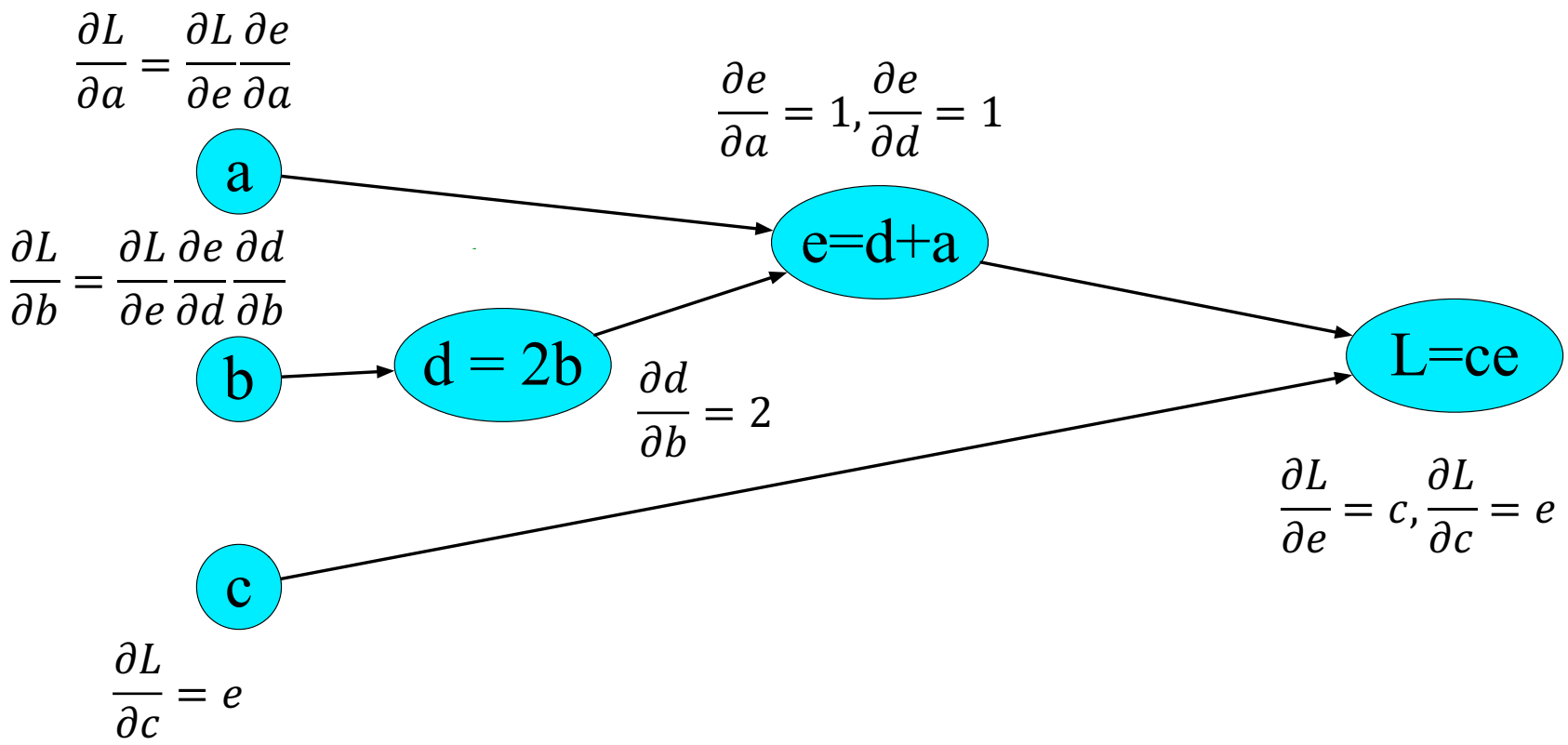
$$\frac{\partial L}{\partial c} = e$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

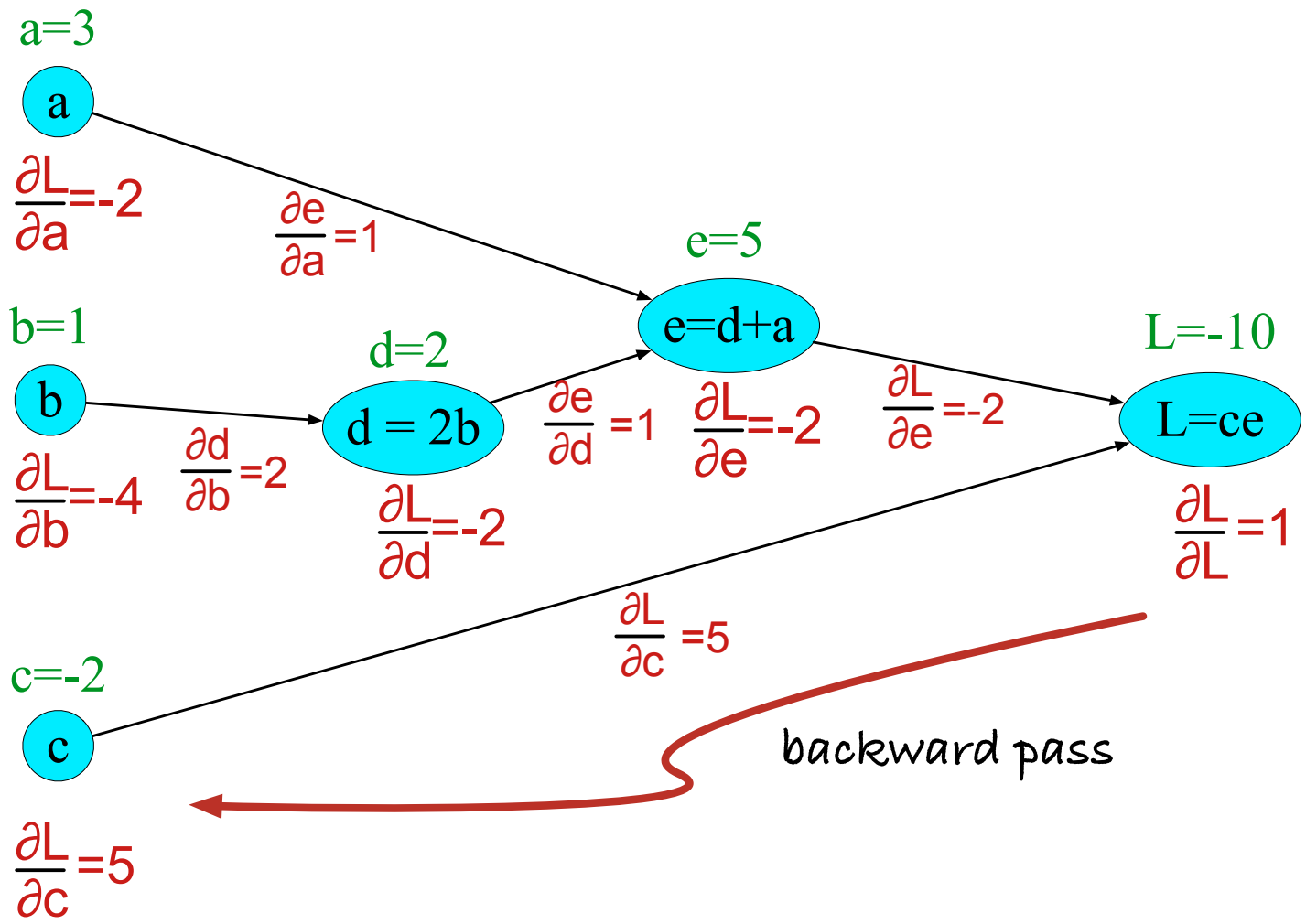
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$\frac{\partial L}{\partial c} = e$$

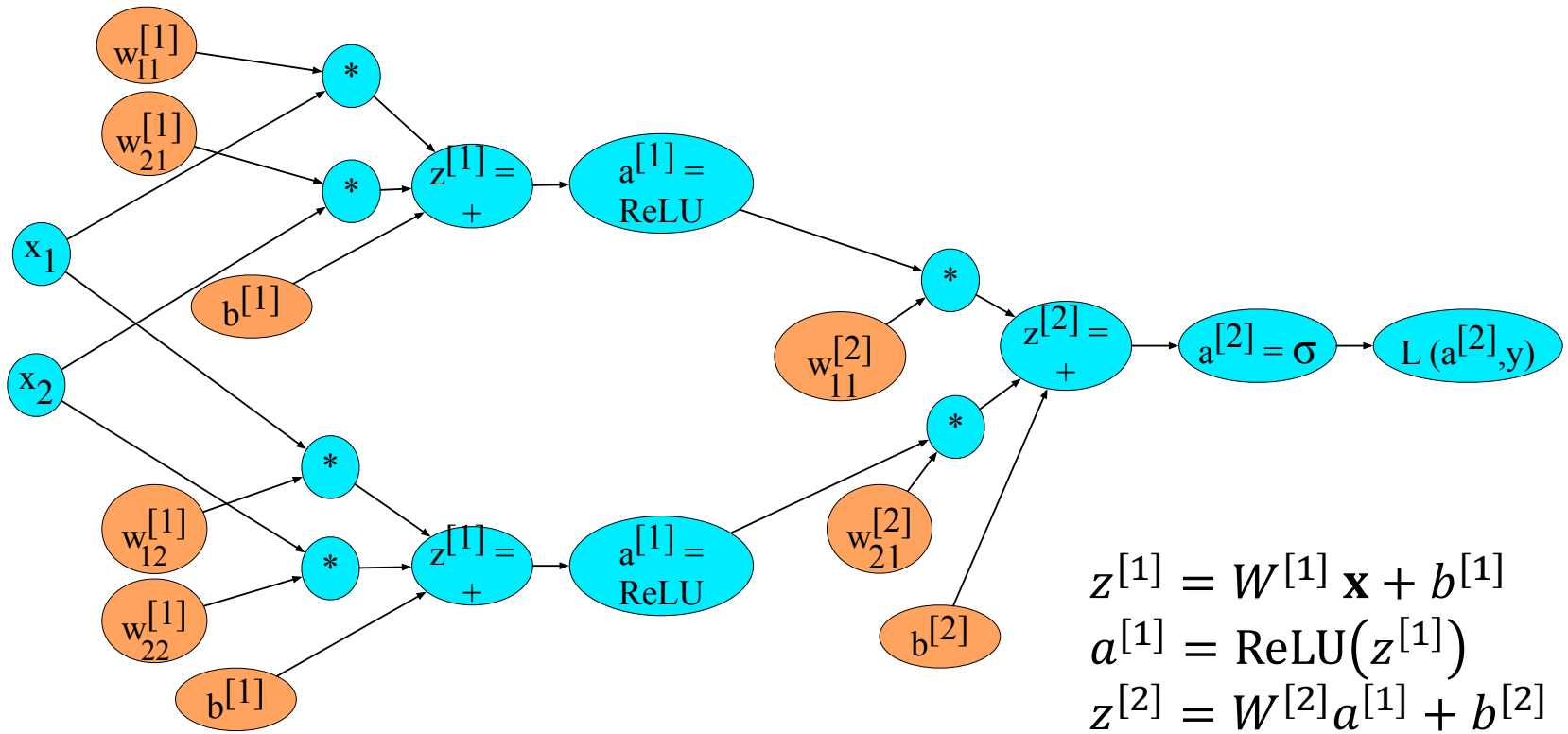



$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$
$\frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$
a
e=d+a
$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$
b
d = 2b
$\frac{\partial d}{\partial b} = 2$
L=ce
$\frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$
c
$\frac{\partial L}{\partial c} = e$

Backward pass



Computation Graph for a NN



$$z^{[1]} = W^{[1]}\,\mathbf{x} + b^{[1]}$$
$$a^{[1]} = \text{ReLU}\left(z^{[1]}\right)$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma\left(z^{[2]}\right)$$
$$\hat{y} = a^{[2]}$$

Neural Language Models

Language Models

Estimate the probability of a sentence consisting of word sequence $w_{1:n}$

$$P(w_{1:n}) \approx \prod_{i=1}^{n} P(w_i \mid w_{i-k:i-1})$$

We need to estimate the probability of $P(w_{i+1} \mid w_{k-i:i})$ from a large corpus.

$$\hat{p}_{MLE}(w_{i+1} = m \mid w_{i-k:i}) = \frac{\#(w_{i-k:i+1})}{\#(w_{i-k:i})}$$

$$\hat{p}_{add-\alpha}(w_{i+1} = m \mid w_{i-k:i}) = \frac{\#(w_{i-k:i+1}) + \alpha}{\#(w_{i-k:i}) + \alpha|V|}$$

Neural Language Models

Neural Language models have several advantages over n-gram LMs:

1. They don’t need smoothing
2. They can handle much longer histories.
3. They can generalize over contexts of similar words.
4. Neural LMs tend to have much higher predictive accuracy than n-gram LMs.

Disadvantage: slower to train than traditional n-gram LMs

Neural LMs (Bengio et al 2003)

1. Associate each word in the vocabulary with a vector-representation, thereby creating a notion of similarity between words.
2. Express the joint probability *function* of a word sequence in terms of the word vectors for the words in that sequence.
3. Simultaneously learn the word vectors and the parameters of the *function.*

The word vectors are low-dimensional (d=30 to d=100) dense vectors, like we've seen before.

The probability function is expressed the product of conditional probabilities of the next word given the previous word, *using a multi-layer, feed forward neural network.*

Neural LMs

The input to the neural network is a k-gram of words $w_{1:k}$.

The output is a probability distribution over the next word.

The *k* context words are treated as a word window. Each word is associated with an **embedding** vector:

The input vector **x** just concatenates v(w) for each of the k words:

$$v(w) \in \mathbb{R}^{d_w}$$

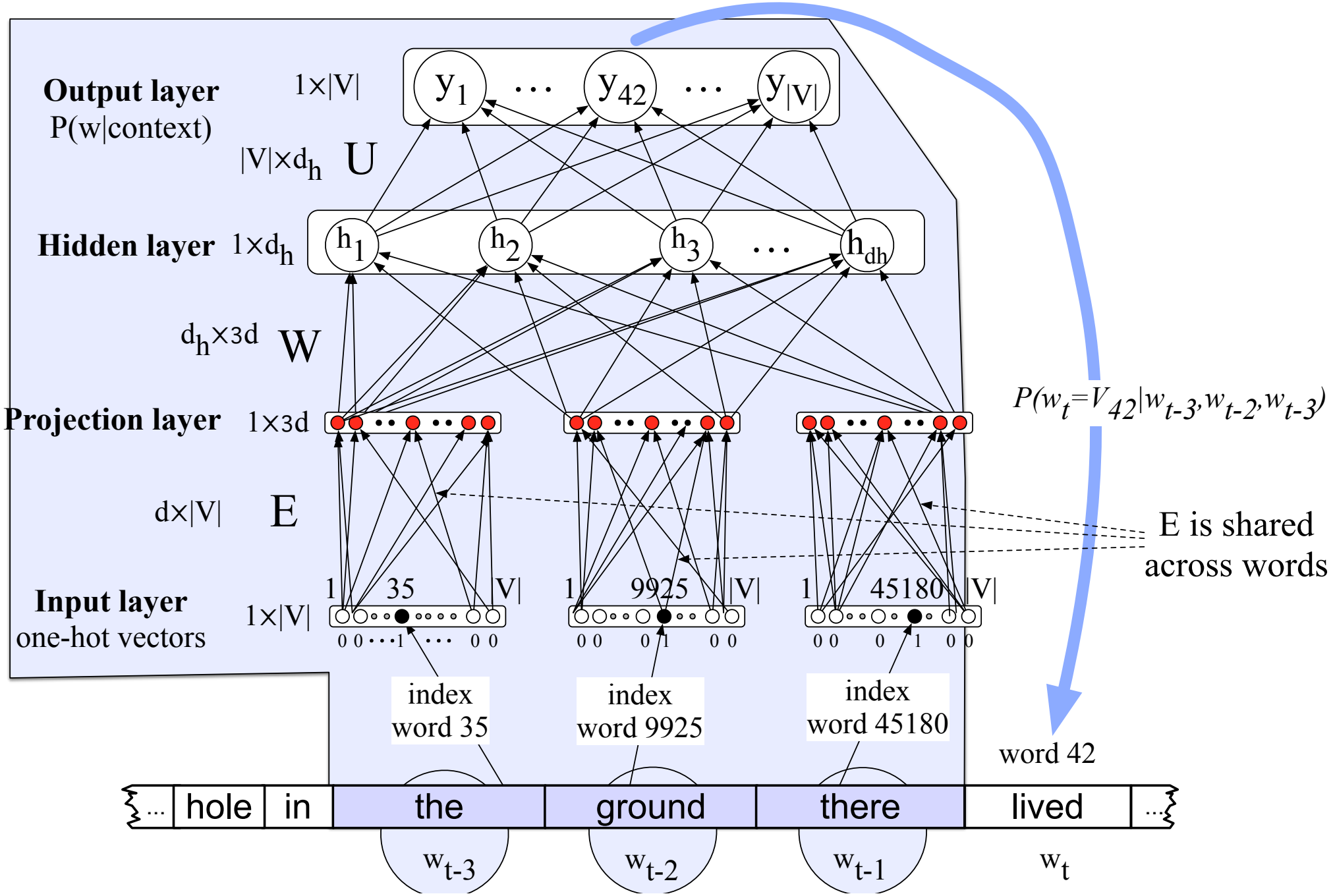
$$\boldsymbol{x} = [v(w_1); v(w_2); \ldots; v(w_k)]$$

Neural LMs

The input **x** is fed into a neural network with 1 or more hidden layers:

$$\hat{\boldsymbol{y}} = P(w_i|w_{1:k}) = LM(w_{1:k}) = \text{softmax}(\boldsymbol{h}\boldsymbol{W}^{\boldsymbol{2}} + \boldsymbol{b}^{\boldsymbol{2}})$$
$$\boldsymbol{h} = g(\boldsymbol{x}\boldsymbol{W}^{\boldsymbol{1}} + \boldsymbol{b}^{\boldsymbol{1}})$$
$$\boldsymbol{x} = [v(w_1); v(w_2); \ldots; v(w_k)]$$
$$v(w) = \boldsymbol{E}_{[w]}$$

$$w_i \in V \quad \boldsymbol{E} \in \mathbb{R}^{|V| \times d_w} \quad \boldsymbol{W}^{\boldsymbol{1}} \in \mathbb{R}^{k \cdot d_w \times d_{\text{hid}}} \quad \boldsymbol{b}^{\boldsymbol{1}} \in \mathbb{R}^{d_{\text{hid}}} \quad \boldsymbol{W}^{\boldsymbol{2}} \in \mathbb{R}^{d_{\text{hid}} \times |V|} \quad \boldsymbol{b}^{\boldsymbol{2}} \in \mathbb{R}^{|V|}$$



Training

The training examples are simply word k-grams from the corpus

The identities of the first k-1 words are used as features, and the last word is used as the target label for the classification.

Conceptually, the model is trained using cross-entropy loss.

Advantages of NN LMs

Better results. They achieve better preplexity scores than SOTA n-gram LMs.

Larger N. NN LMs can scale to much larger orders of n. This is achievable because parameters are associated only with individual words, and not with n-grams.

They generalize across contexts. For example, by observing that the words *blue, green, red, black*, etc. appear in similar contexts, the model will be able to assign a reasonable score to the *green car* even though it never observed in training, because it did observe *blue car* and *red car*.

A by-product of training are **word embeddings**

Language Modeling

Goal: Learn a **function** that returns the joint probability

Primary difficulty:

1. There are too many parameters to accurately estimate. This is sometimes called the “curse of dimensionality”
2. In n-gram-based models we fail to generalize to related words / word sequences that we have observed.

Curse of dimensionality / sparse statistics

Suppose we want a joint distribution over 10 words.
Suppose we have a vocabulary of size 100,000.

$$100{,}000^{10} = 10^{50} \text{ parameters}$$

This is too high to estimate from data.

Chain rule

In LMs we user chain rule to get the conditional probability of the next word in the sequence given all of the previous words:

$$P(w_1 w_2 w_3 \ldots w_t) = \prod_{t=1}^{T} P(w_t | w_1 \ldots w_{t-1})$$

What assumption do we make in n-gram LMs to simplify this?

The probability of the next word only depends on the previous *n*-1 words.

A small *n* makes it easier for us to get an estimate of the probability from data.

Probability tables

We construct tables to look up the probability of seeing a word given a history.

curse of	$P(w_t \mid w_{t-n} \ldots w_{t-1})$
dimensionality	
azure	
knowledge	
oak	

The tables only store observed sequences.

What happens when we have a new (unseen) combination of n words?

Unseen sequences

What happens when we have a new (unseen) combination of n words?

1. Back-off
2. Smoothing / interpolation

We are basically just stitching together short sequences of observed words.

Alternate idea

Let's try **generalizing**.

Intuition: Take a sentence like

The cat is walking in the bedroom

And use it when we assign probabilities to similar sentences like

The dog is running around the room

A Neural Probabilistic LM

Bengio et al NIPS 2003

1. Use a vector space model where the words are vectors with real values $\mathbb{R}^m$. m=30, 60, 100. This gives a way to compute word similarity.
2. Define a function that returns a joint probability of words in a sequence based on a sequence of these vectors.
3. Simultaneously learn the word representations **and** the probability function from data.

Seeing one of the cat/dog sentences allows them to increase the probability for that sentence **and** its combinatorial # of **"neighbor" sentences** in vector space.

A Neural Probabilistic LM

Given:

A training set $w_1 \ldots w_t$ where $w_t \in V$

Learn:

$f(w_1 \ldots w_t) = P(w_t | w_1 \ldots w_{t-1})$

Subject to giving a high probability to an unseen text/dev set (e.g. minimizing the perplexity)

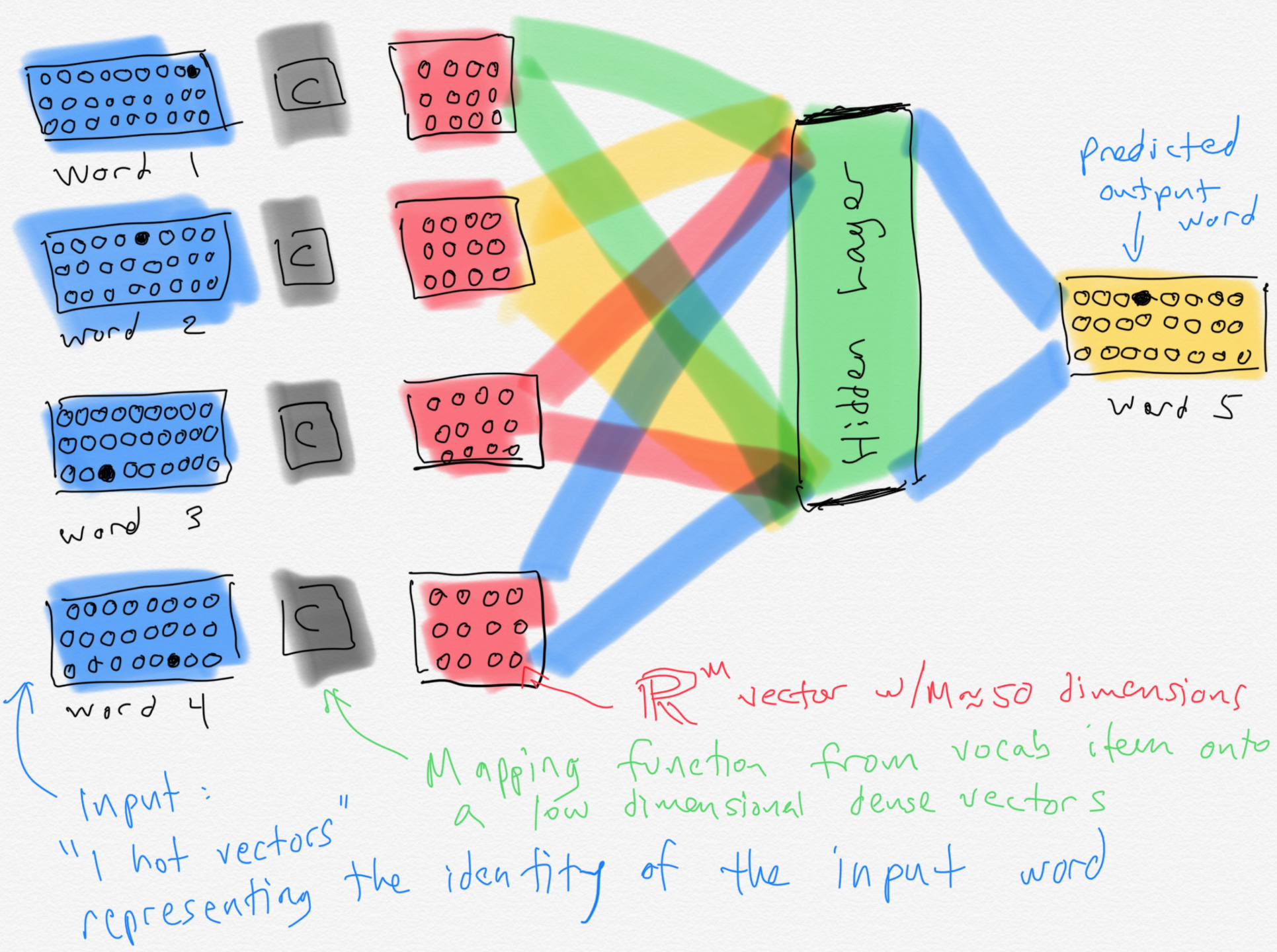
Constraint:

Create a proper probability distribution (e.g. sums to 1) so that we can take the product of conditional probabilities to get the joint probability of a sentence

A Neural Probabilistic LM

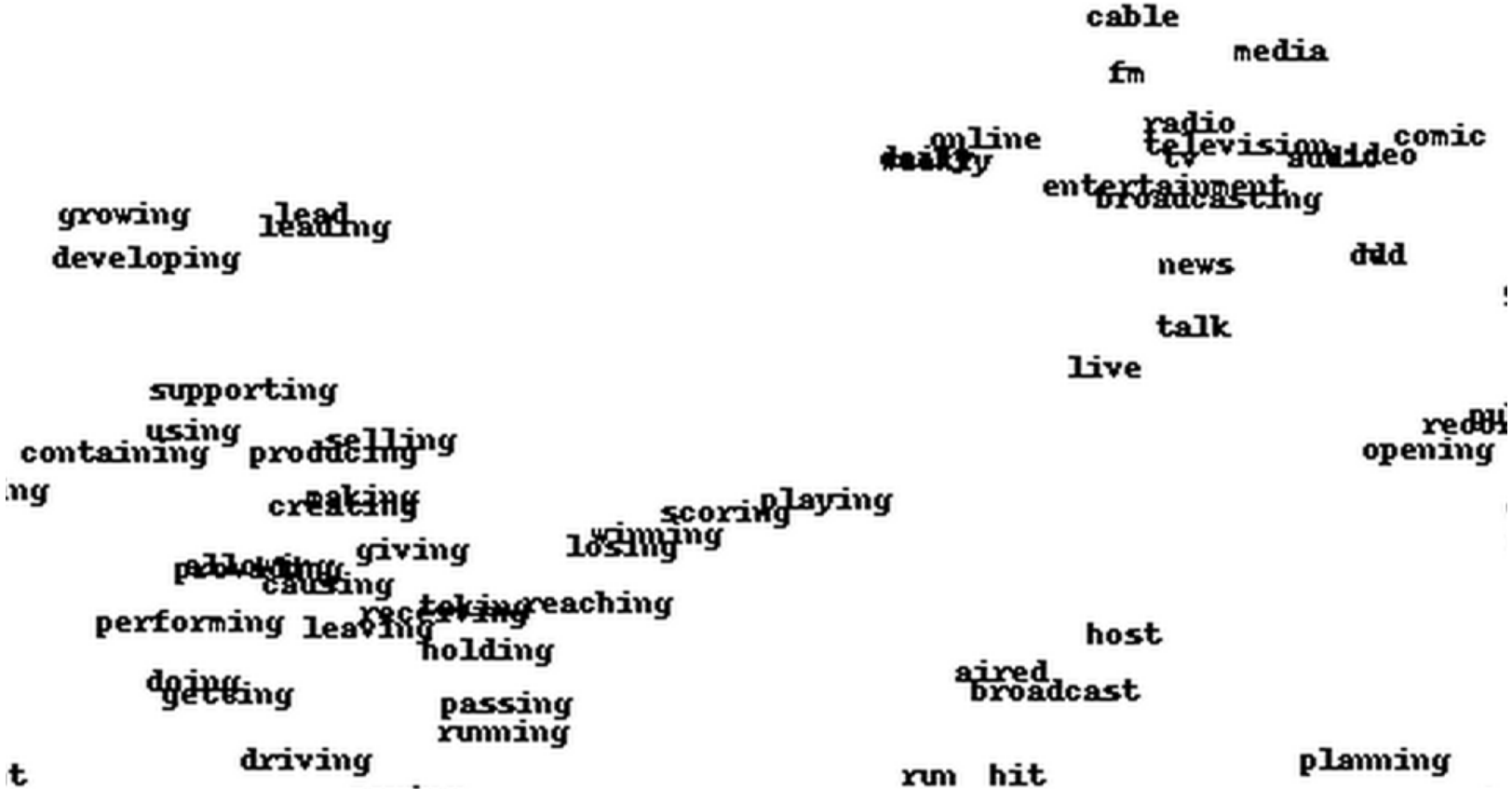
1. Create a mapping function C from any word in V onto $\mathbb{R}^M$. Store this in a V-by-M matrix. Initialize it with singular value decomposition (SVD).
2. The neural architecture: a function *g* maps sequence of word vectors onto a probability distribution over the vocabulary V

$$g(C(w_{t-n}) \ldots C(w_{t-1})) = P(w_t | w_{t-n} \ldots w_{t-1})$$



Word embeddings

When the ~50 dimensional vectors that result from training a neural LM are projected down to 2-dimensions, we see a lot of words that are intuitively similar to each other are close together.



Current state of the art neural LMs

ELMo

GPT

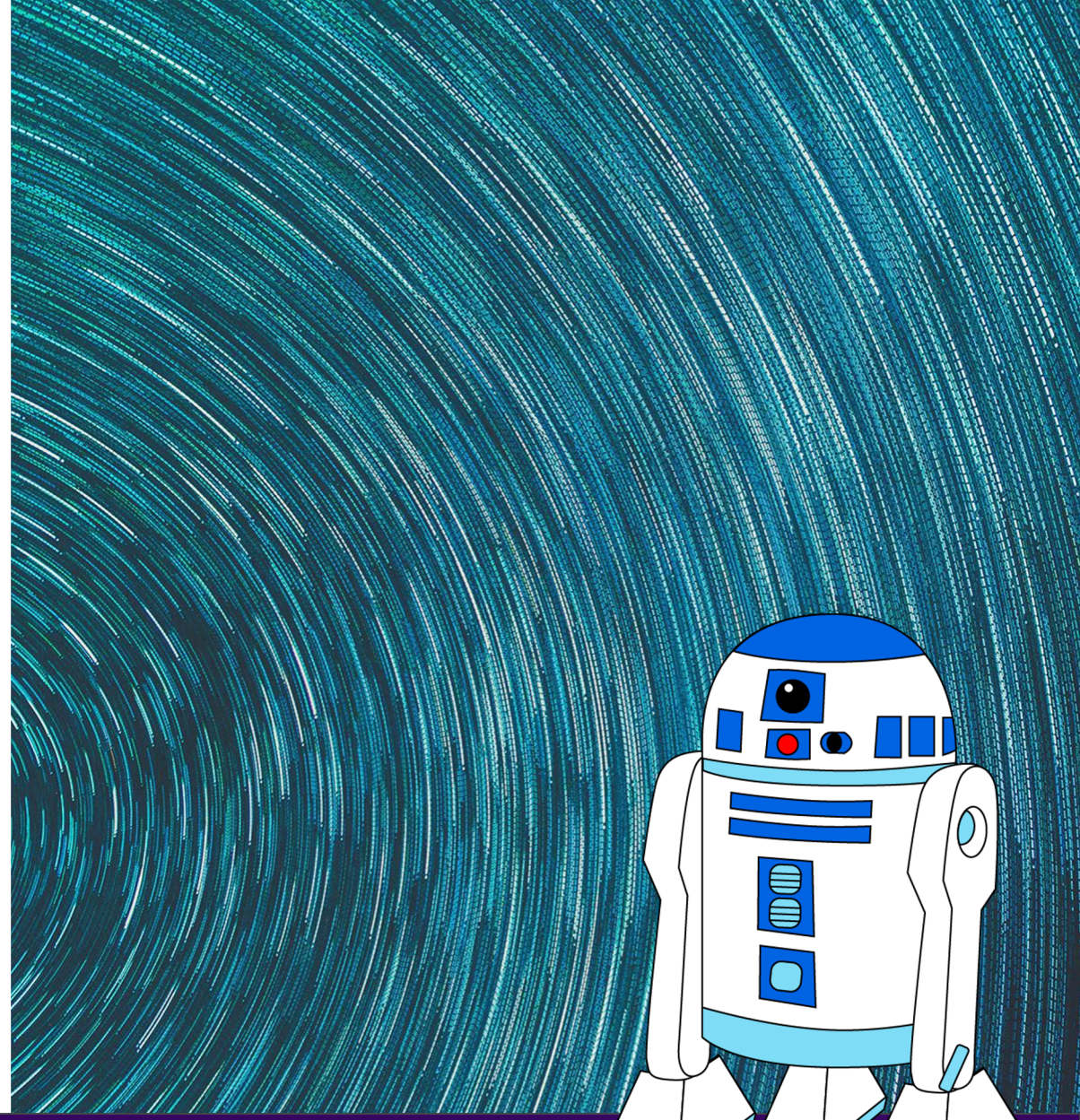
BERT

GPT-2

CIS 421/521:
ARTIFICIAL INTELLIGENCE

NLP: Vector Semantics

Jurafsky and Martin Chapter 6



Penn Engineering
University of Pennsylvania

Word Meaning

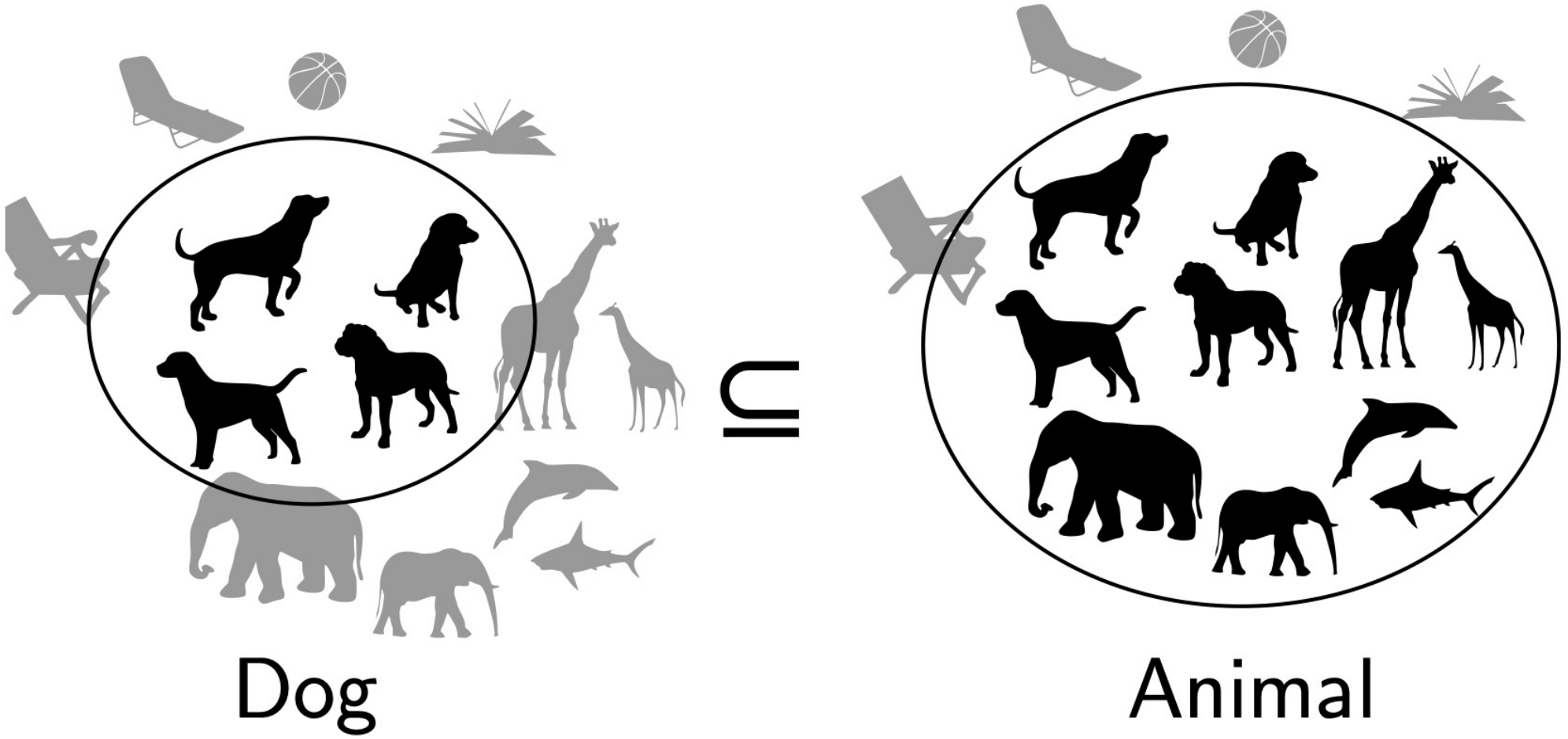
How should we **represent** the **meaning** of a word?

In N-gram LMs we represented words as a string of letters or as an index in a vocabulary list.

Ideally, we want a meaning representation to encode:

1. **Synonyms** – words that have similar meanings
2. **Antonyms** – words that have opposite meanings
3. **Connotations** – words that are positive or negative
4. **Semantic Roles** – *buy, sell*, and *pay* are different parts of the same underlying *purchasing* event
5. Support for **entailment**

Entailment in formal semantics



Entailment in formal semantics

All animals have an ulnar artery

⇒

All dogs have an ulnar artery

+ Mathematically well-understood
+ Powerful machinery for handling logical operations
- Knowledge must come from somewhere else

Noun

- S: (n) **dog**, domestic dog, Canis familiaris (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds) *"the dog barked all night"*
- S: (n) frump, **dog** (a dull unattractive unpleasant girl or woman) *"she got a reputation as a frump"; "she's a real dog"*
- S: (n) **dog** (informal term for a man) *"you lucky dog"*
- S: (n) cad, bounder, blackguard, **dog**, hound, heel (someone who is morally reprehensible) *"you dirty dog"*
- S: (n) frank, frankfurter, hotdog, hot dog, **dog**, wiener, wienerwurst, weenie (a smooth-textured sausage of minced beef or pork usually smoked; often served on a bread roll)
- S: (n) pawl, detent, click, **dog** (a hinged catch that fits into a notch of a ratchet to move a wheel forward or prevent it from moving backward)
- S: (n) andiron, firedog, **dog**, dog-iron (metal supports for logs in a fireplace) *"the andirons were too hot to touch"*

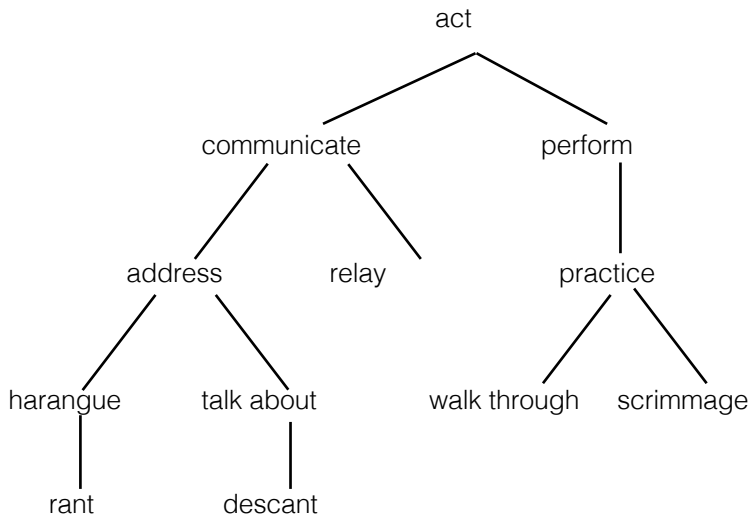
Verb

Noun

- S: (n) **dog**, domestic dog, Canis familiaris (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds) *"the dog barked all night"*
  - *direct hyponym* / *full hyponym*
  - *part meronym*
  - *member holonym*
  - ***direct hypernym*** / *inherited hypernym* / *sister term*
    - S: (n) canine, canid (any of various fissiped mammals with nonretractile claws and typically long muzzles)
    - S: (n) domestic animal, domesticated animal (any of various animals that have been tamed and made fit for a human environment)
- S: (n) frump, **dog** (a dull unattractive unpleasant girl or woman) *"she got a reputation as a frump"; "she's a real dog"*
- S: (n) **dog** (informal term for a man) *"you lucky dog"*
- S: (n) cad, bounder, blackguard, **dog**, hound, heel (someone who is morally reprehensible) *"you dirty dog"*
- S: (n) frank, frankfurter, hotdog, hot dog, **dog**, wiener, wienerwurst, weenie

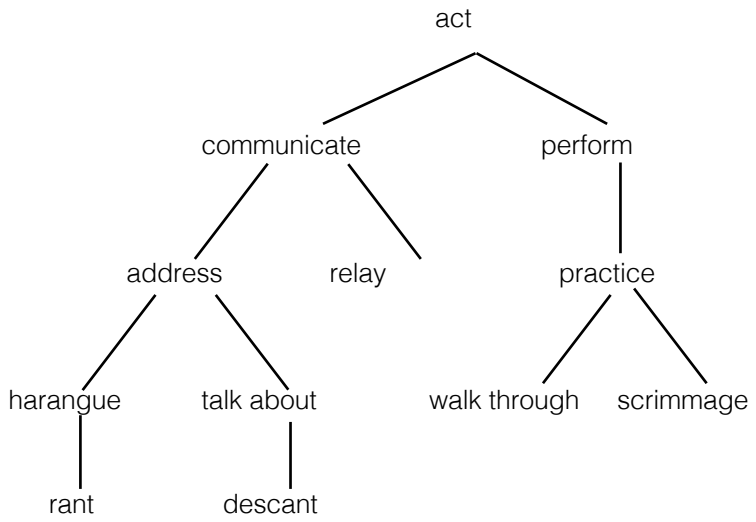
- S: (n) canine, canid (any of various fissiped mammals with nonretractile claws and typically long muzzles)
  - S: (n) carnivore (a terrestrial or aquatic flesh-eating mammal) *"terrestrial carnivores have four or five clawed digits on each limb"*
    - S: (n) placental, placental mammal, eutherian, eutherian mammal (mammals having a placenta; all mammals except monotremes and marsupials)
      - S: (n) mammal, mammalian (any warm-blooded vertebrate having the skin more or less covered with hair; young are born alive except for the small subclass of monotremes and nourished with milk)
        - S: (n) vertebrate, craniate (animals having a bony or cartilaginous skeleton with a segmented spinal column and a large brain enclosed in a skull or cranium)
          - S: (n) chordate (any animal of the phylum Chordata having a notochord or spinal column)
            - S: (n) animal, animate being, beast, brute, creature, fauna (a living

Lexical Semantics

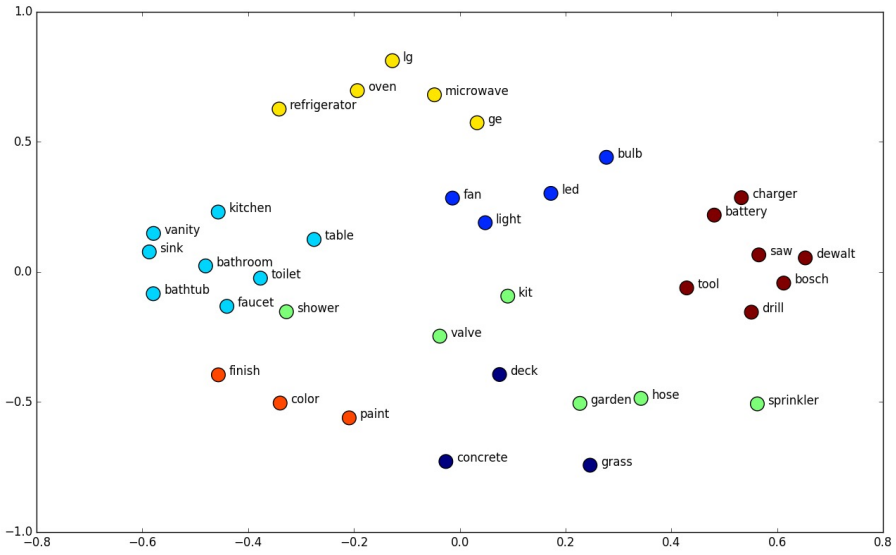


WordNet

Lexical Semantics

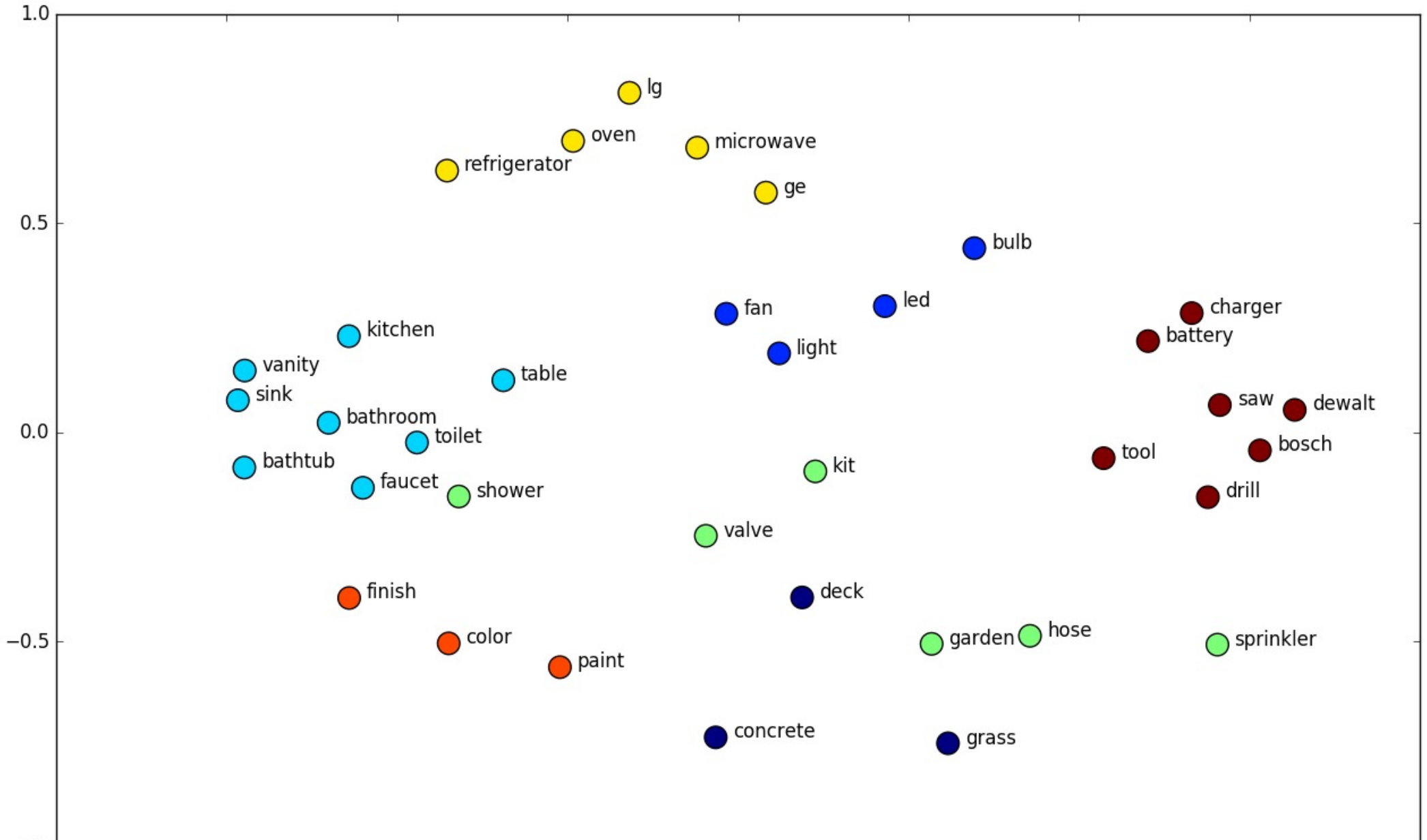


WordNet



Vector Space Models

Vector Space Models



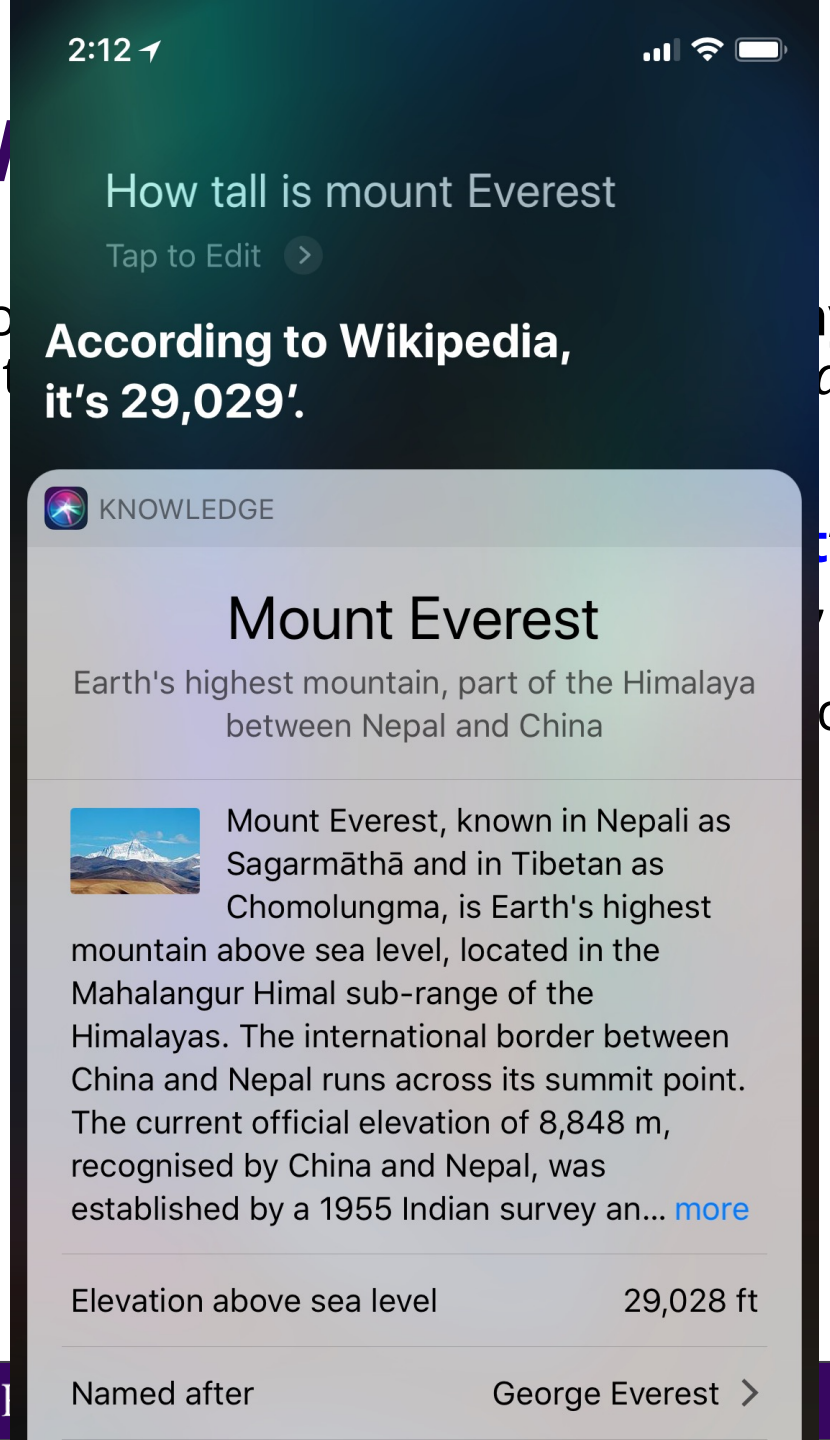
Word similarity

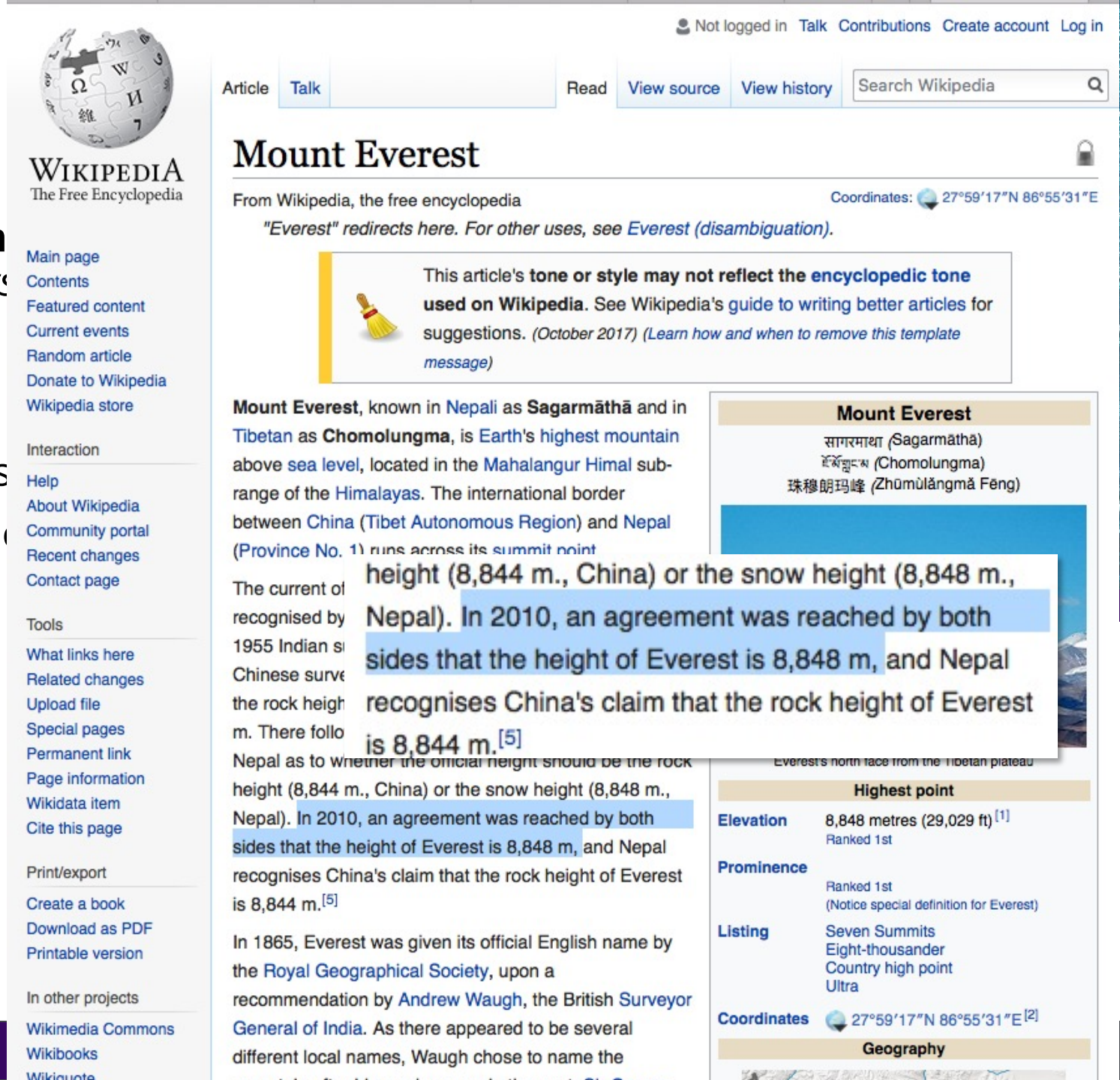
Most words don't have many **synonyms**, but they do have a lot of **similar** words. *Cat* is not a synonym of *dog*, but *cats* and *dogs* are certainly similar words.

"**fast**" is similar to "**rapid**"

"**tall**" is similar to "**height**"

Useful for applications like question answering





height (8,844 m., China) or the snow height (8,848 m., Nepal). In 2010, an agreement was reached by both sides that the height of Everest is 8,848 m, and Nepal recognises China's claim that the rock height of Everest is 8,844 m.[5]

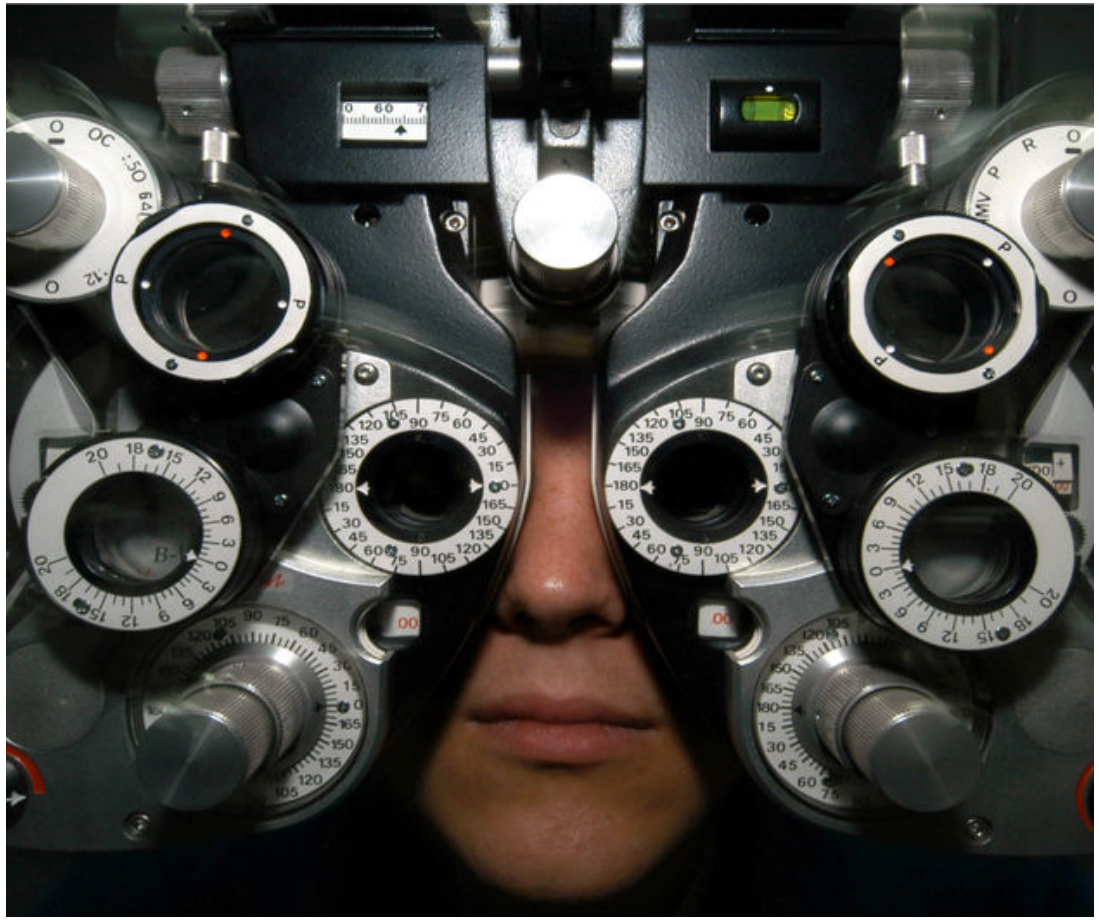
Distributional Hypothesis

If we consider *optometrist* and *eye-doctor* we find that, as our corpus of utterances grows, these two occur in almost the same environments. In contrast, there are many sentence environments in which *optometrist* occurs but *lawyer* does not…

It is a question of the relative frequency of such environments, and of what we will obtain if we ask an informant to substitute any word he wishes for optometrist (not asking what words have the same meaning).

These and similar tests all measure the probability of particular environments occurring with particular elements… If A and B have almost identical environments we say that they are synonyms.

–Zellig Harris (1954)



Intuition of distributional word similarity

Nida (1975) example:

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

From context words humans can guess **tesgüino** means
an alcoholic beverage like beer

Intuition for algorithm:
Two words are similar if they have similar word contexts.

History of Vector Space Models

Vector Space Models were initially developed in the SMART information retrieval system (Salton, 1971)

Each document in a collection is represented as point in a space (a vector in a vector space)

A user's query is a pseudo-document and is represented as a point in the same space as the documents

Perform IR by retrieving documents whose vectors are close together in this space to the query vector

Term-Document Matrix

	D1	D2	D3	D4	D5
abandon					
abdicate					
abhor					
academic					
...					
zygodactyl					
zymurgy					

Term-Document Matrix

	D1	D2	D3	D4	D5
abandon					
abdicate					
abhor					
academic					
...					
zygodactyl					
zymurgy					

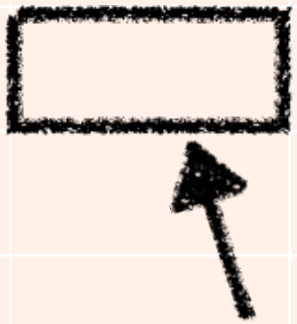
Each column vector represents a Document

Term-Document Matrix

	D1	D2	D3	D4	D5
abandon					
abdicate					
abhor					
academic					
...					
zygodactyl					
zymurgy					

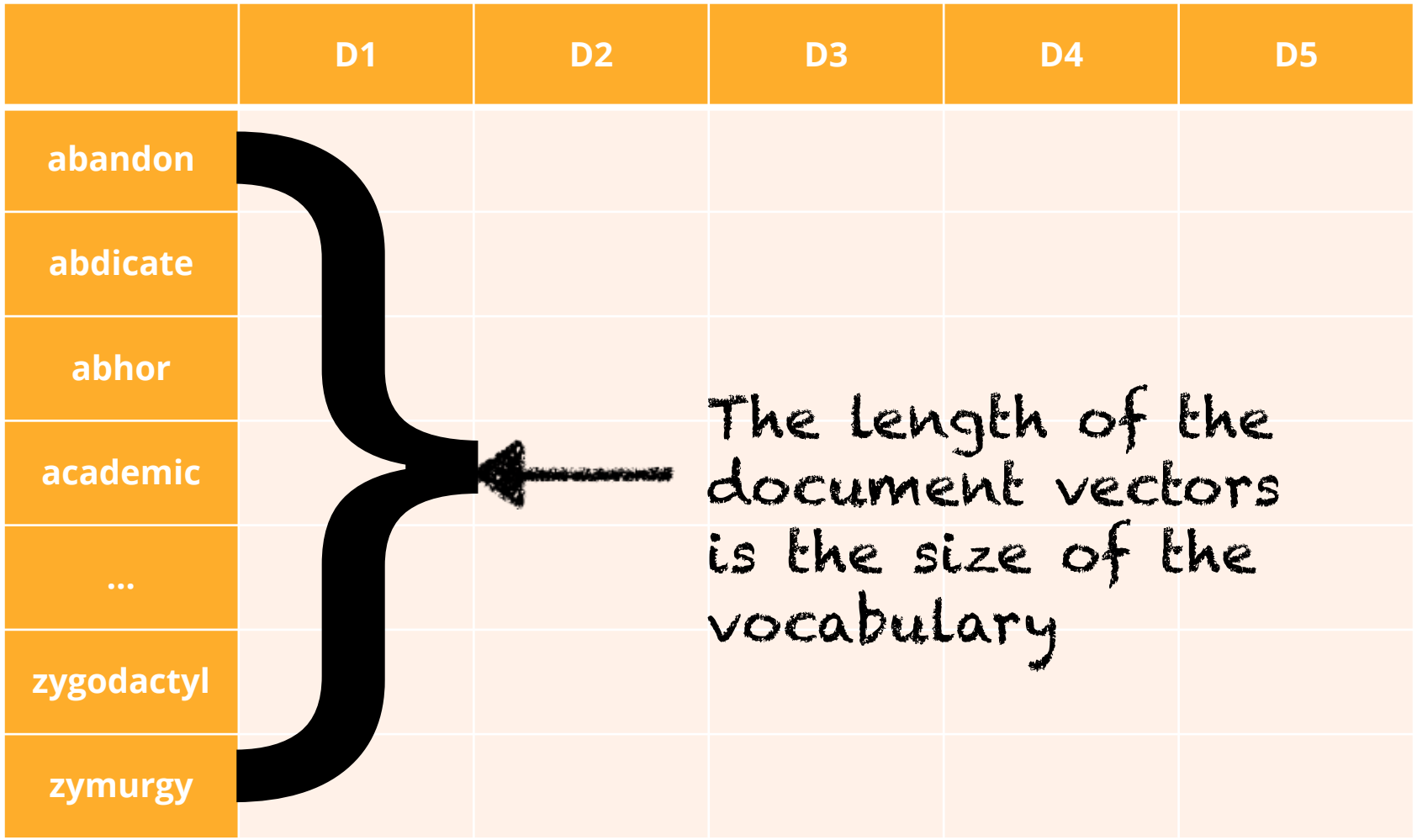
Term-Document Matrix

	D1	D2	D3	D4	D5
abandon					
abdicate					
abhor					
academic					
...					
zygodactyl					
zymurgy					



The value in a cell is based on how often that term occurred in that document

Term-Document Matrix

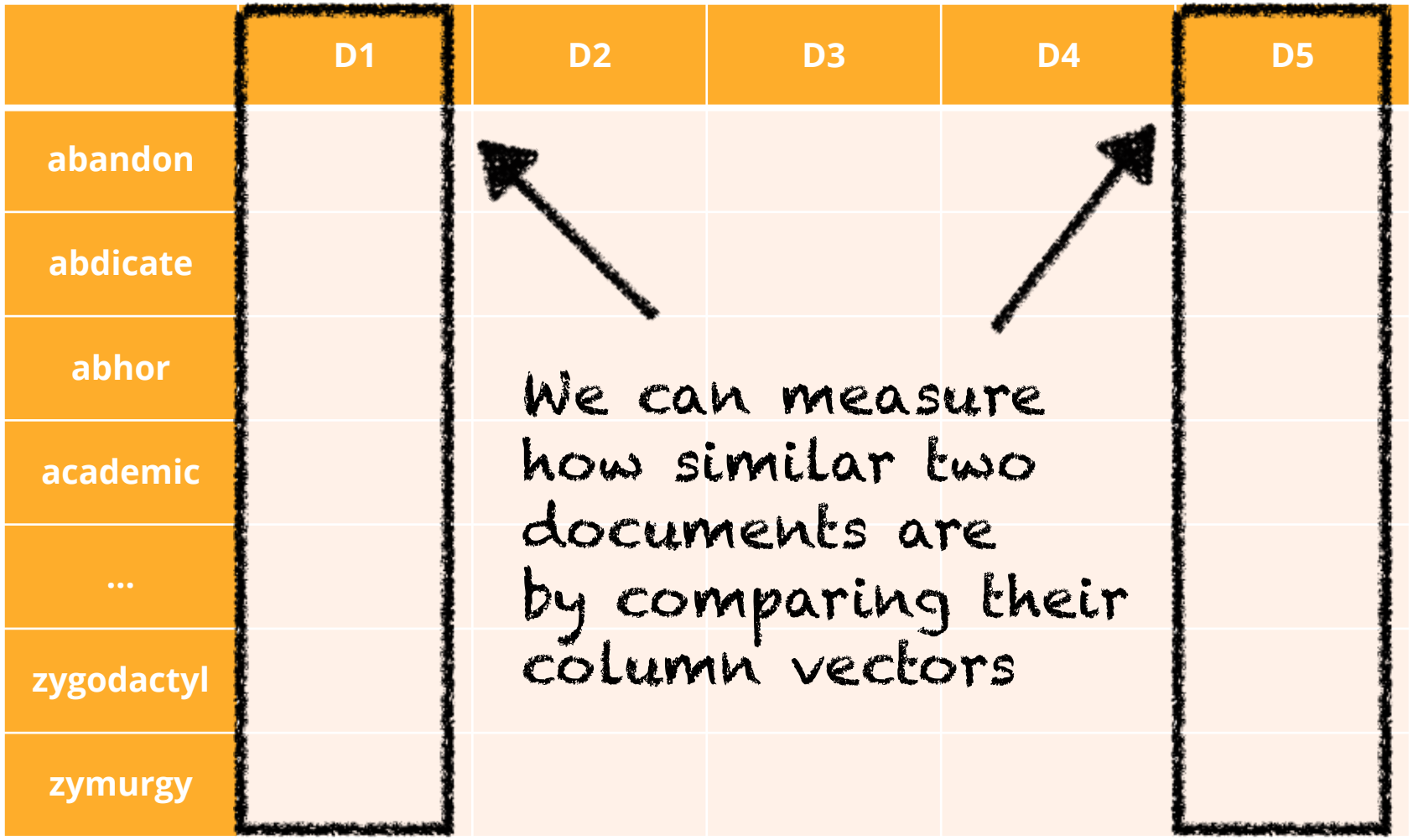


Term-Document Matrix

	D1	D2	D3	D4	D5
abandon					
abdicate					
abhor					
academic					
...					
zygodactyl					
zymurgy					

Document vectors can be sparse (most values are 0)

Term-Document Matrix



What can document similarity let you do?

Word similarity for plagiarism detection

MAINFRAMES

Mainframes are primarily referred to large computers with rapid, advanced processing capabilities that can execute and perform tasks equivalent to many Personal Computers (PCs) machines networked together. It is characterized with high quantity Random Access Memory (RAM), very large secondary storage devices, and high-speed processors to cater for the needs of the computers under its service.

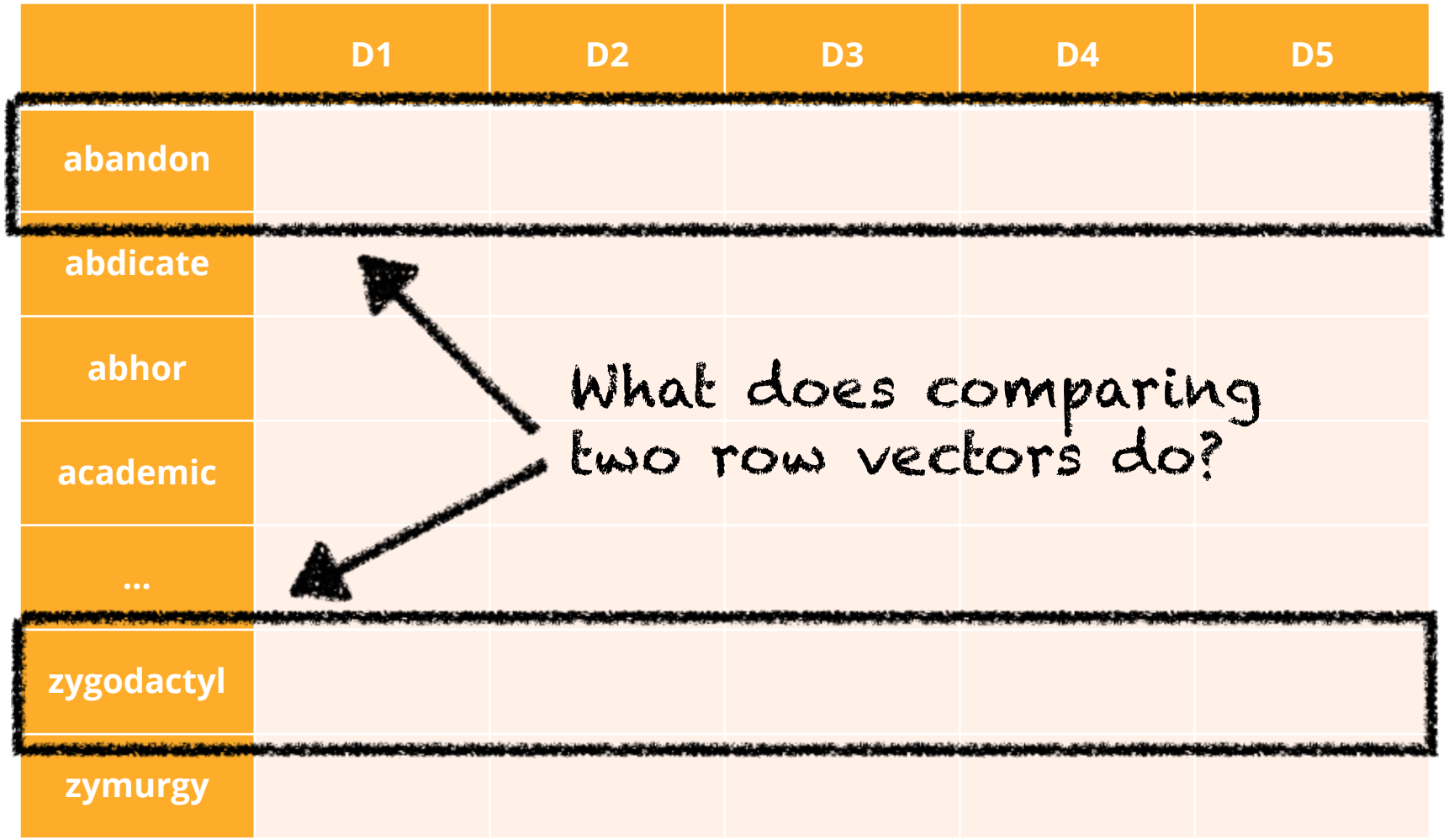
Consisting of advanced components, mainframes have the capability of running multiple large applications required by many and most enterprises and organizations. This is one of its advantages. Mainframes are also suitable to cater for those applications (programs) or files that are of very high demand by its users (clients). Examples of such organizations and enterprises using mainframes are online shopping websites such as Ebay, Amazon, and computing-giant

MAINFRAMES

Mainframes usually are referred those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater for the needs of those computers under its service.

Due to the advanced components mainframes have, these computers have the capability of running multiple large applications required by most enterprises, which is one of its advantage. Mainframes are also suitable to cater for those applications or files that are of very large demand by its users (clients). Examples of these include the large online shopping websites -i.e. : Ebay, Amazon, Microsoft, etc.

Term-Document Matrix



Vector comparisons

	doc_X	doc_Y
A	2	4
B	10	15
C	14	10

Vector comparisons

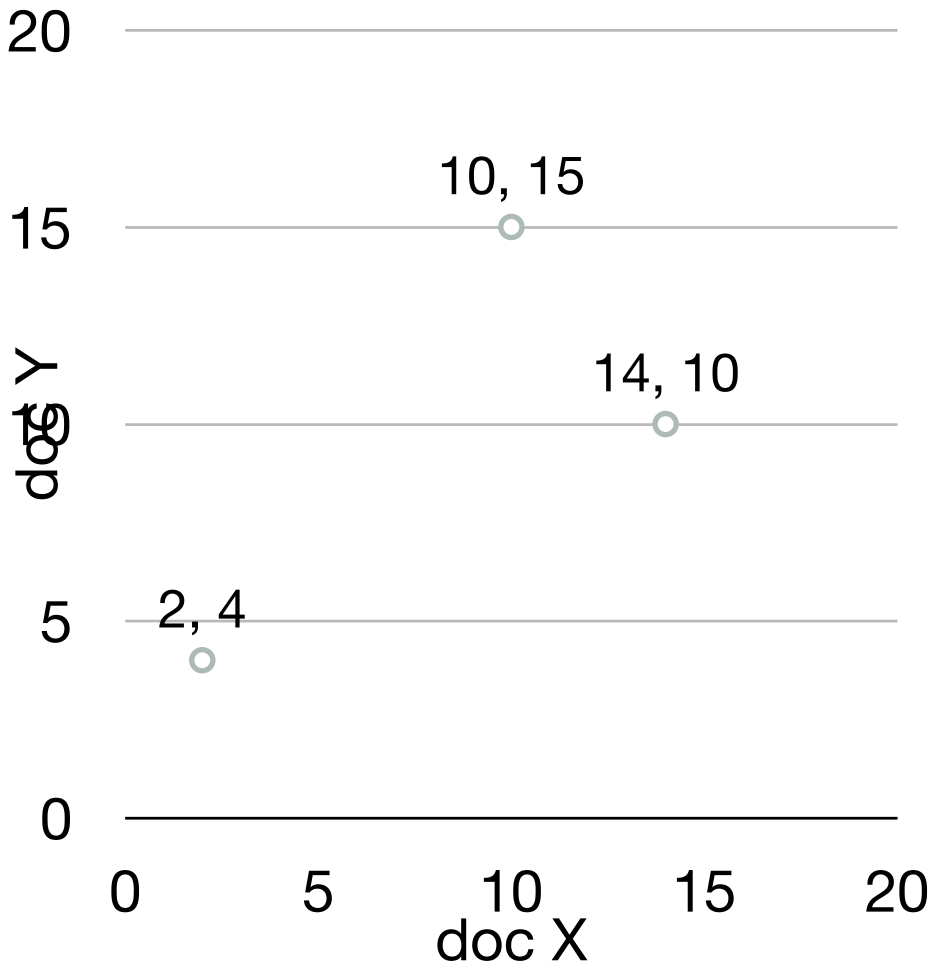
	doc_x	doc_y
A	2	4
B	10	15
C	14	10

doc_y is a positive movie review
doc_x is a less positive movie review

A = "superb" positive / low frequency
B = "good" positive / high frequency
C = "disappointing" negative / high frequency

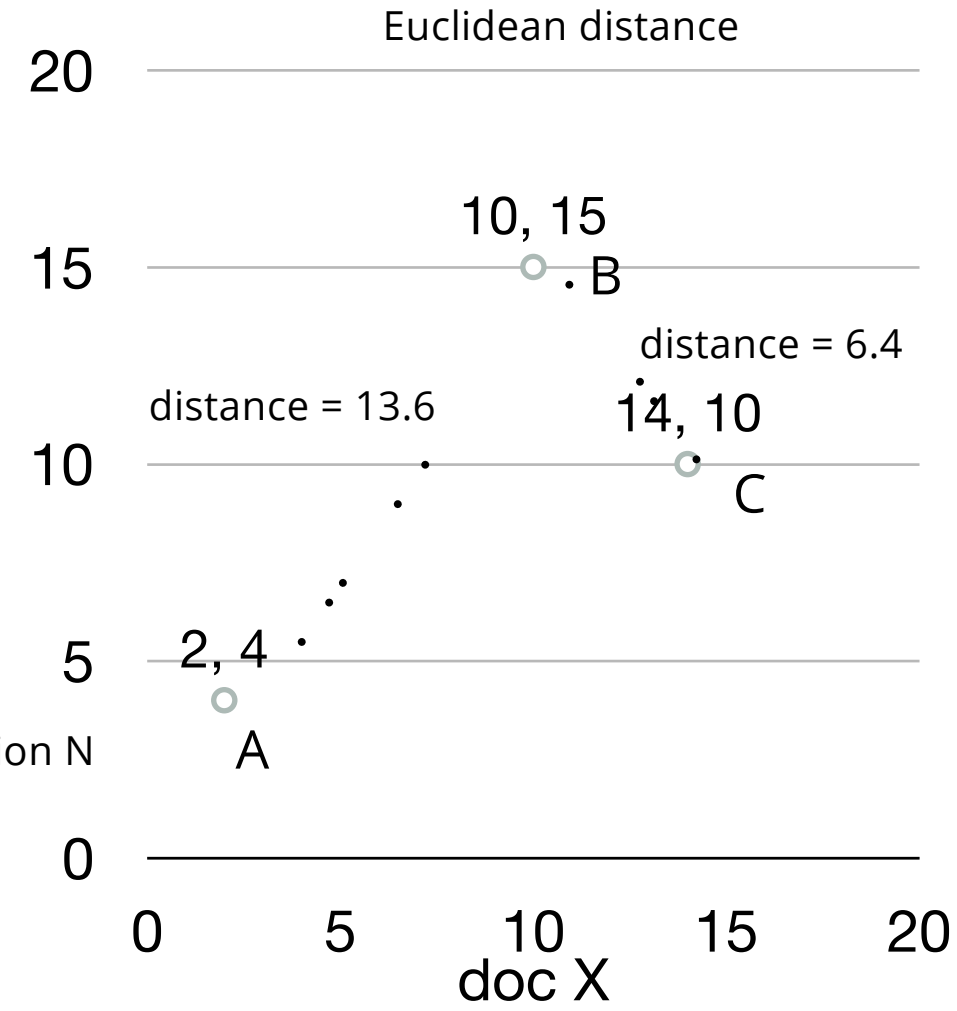
Vector comparisons

	doc_X	doc_Y
A	2	4
B	10	15
C	14	10



Vector comparisons

	doc_X	doc_Y
A	2	4
B	10	15
C	14	10



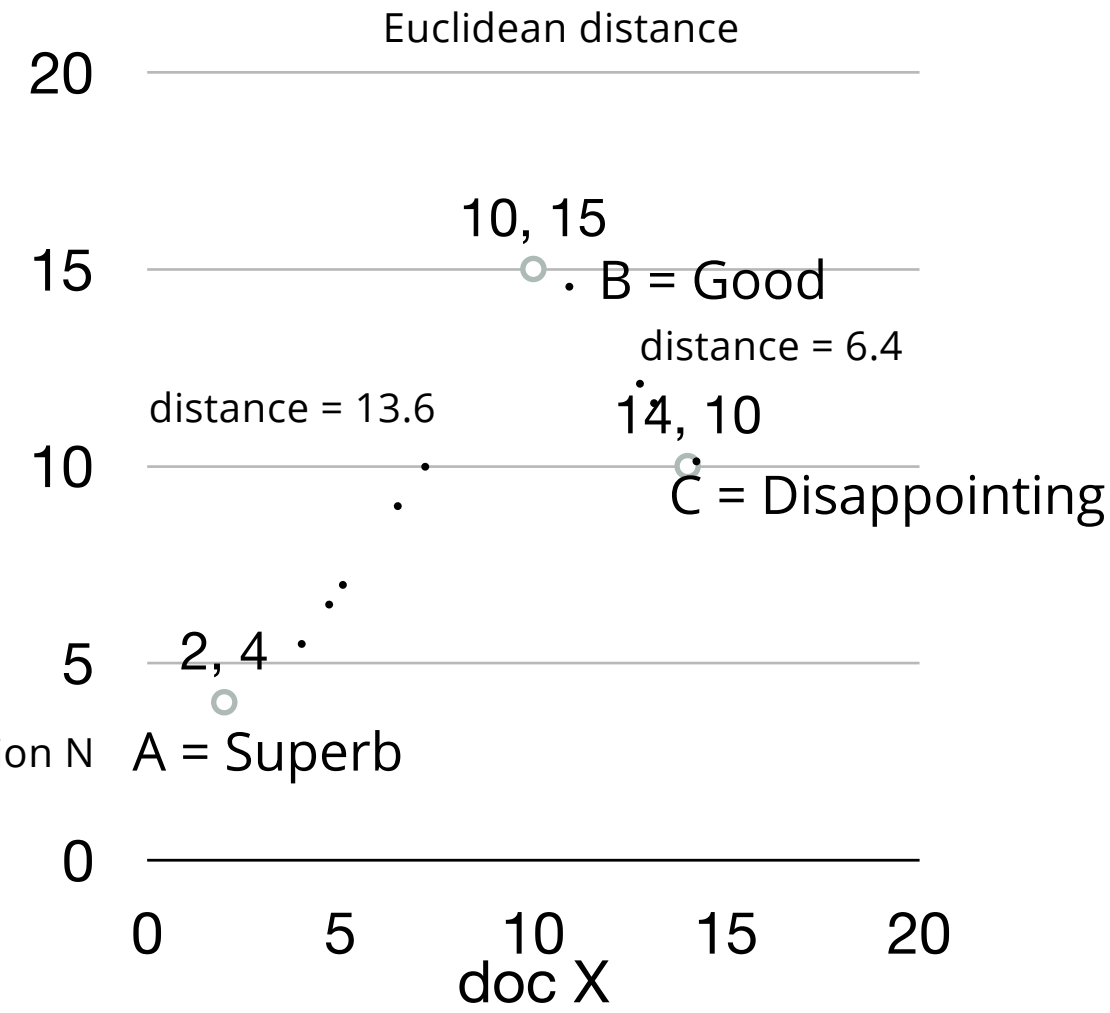
Euclidean distance : vectors u, v of dimension N

$$\sqrt{\sum_{i=1}^{N} |u_i - v_i|^2}$$

Vector comparisons

Oh no! Good is closer to Disappointing than to Superb.

	doc_X	doc_Y
A	2	4
B	10	15
C	14	10



Euclidean distance : vectors u, v of dimension N

$$\sqrt{\sum_{i=1}^{N} |u_i - v_i|^2}$$

Vector L2 (length) Normalization

	doc_X	doc_Y	$\|\|u\|\|$
A	2	4	4.47
B	10	15	18.02
C	14	10	17.20

$$\|u\| = \sqrt{\sum_{i=1}^{n} u_i^2}$$

Vector L2 (length) Normalization

	doc_X	doc_Y	$\|\|u\|\|$
A	2/4.47	4/4.47	4.47
B	10/18.02	15/18.02	18.02
C	14/17.2	10/17.2	17.20

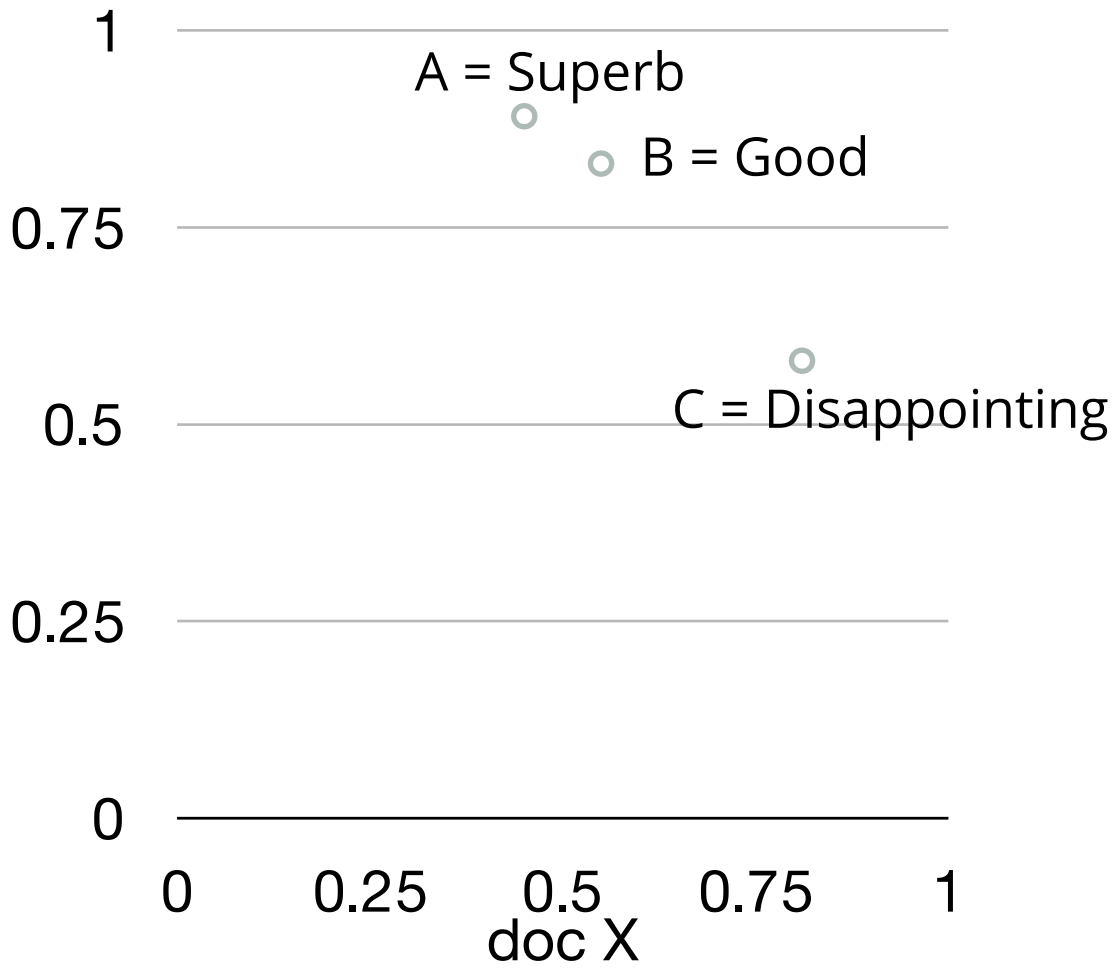
$$\|u\| = \sqrt{\sum_{i=1}^{n} u_i^2}$$

Divide each vector by its L2 length

Vector L2 (length) Normalization

	doc_X	doc_Y
Ȧ	0.45	0.89
Ḃ	0.55	0.83
Ċ	0.81	0.58

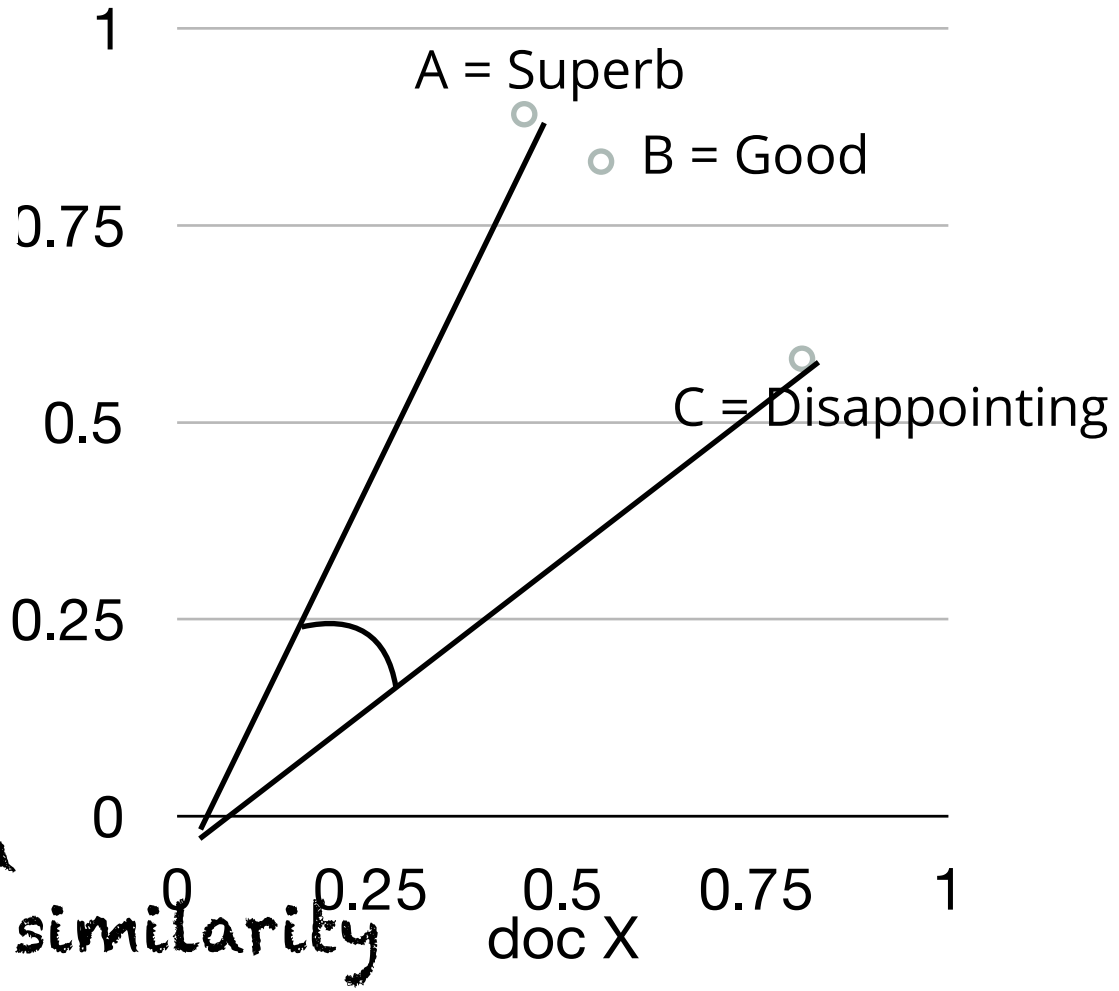
Now Good is closer to Superb than to Disappointing



Cosine Distance

$$1 - \frac{\sum_{i=1}^{n} u_i \times v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \times \sqrt{\sum_{i=1}^{n} v_i^2}}$$

Cosine does the L2 normalization too



Cosine angle between vectors tells us their similarity

Term-Term Matrix

	abandon	abdicate	abhor	...	zymurgy
abandon					
abdicate					
abhor					
academic					
...					
zygodactyl					
zymurgy					

Term-Term Matrix

AKA
Term-Context
Matrix

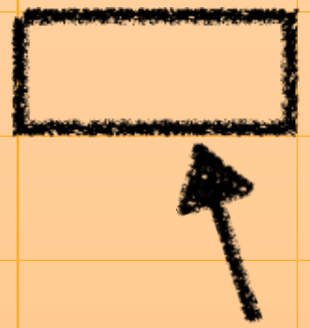
	abandon	abdicate	abhor	…	zymurgy
abandon					
abdicate					
abhor					
academic					
…					
zygodactyl					
zymurgy					

Length of the vector is now |V|
instead of number of documents

Term-Term Matrix

AKA
Term-Context
Matrix

	abandon	abdicate	abhor	...	zymurgy
abandon					
abdicate					
abhor					
academic					
...					
zygodactyl					
zymurgy					



The value in a cell indicates how often abandon appears in a context window surrounding abdicate

Context windows

w-2, w-1 **target_word** w+1 w+2

The government most not **abdicate** responsibility to non-elected
it has led men to **abdicate** their family responsibilities
other demands, but declining to **abdicate** his responsibility
leaders **abdicate** their role and present people with no plans

	his	leaders	not	responsibility	to
abdicate	1	1	1	2	3

Context windows

Occur in a window of +/- 2 words, in the same sentence, in the same document

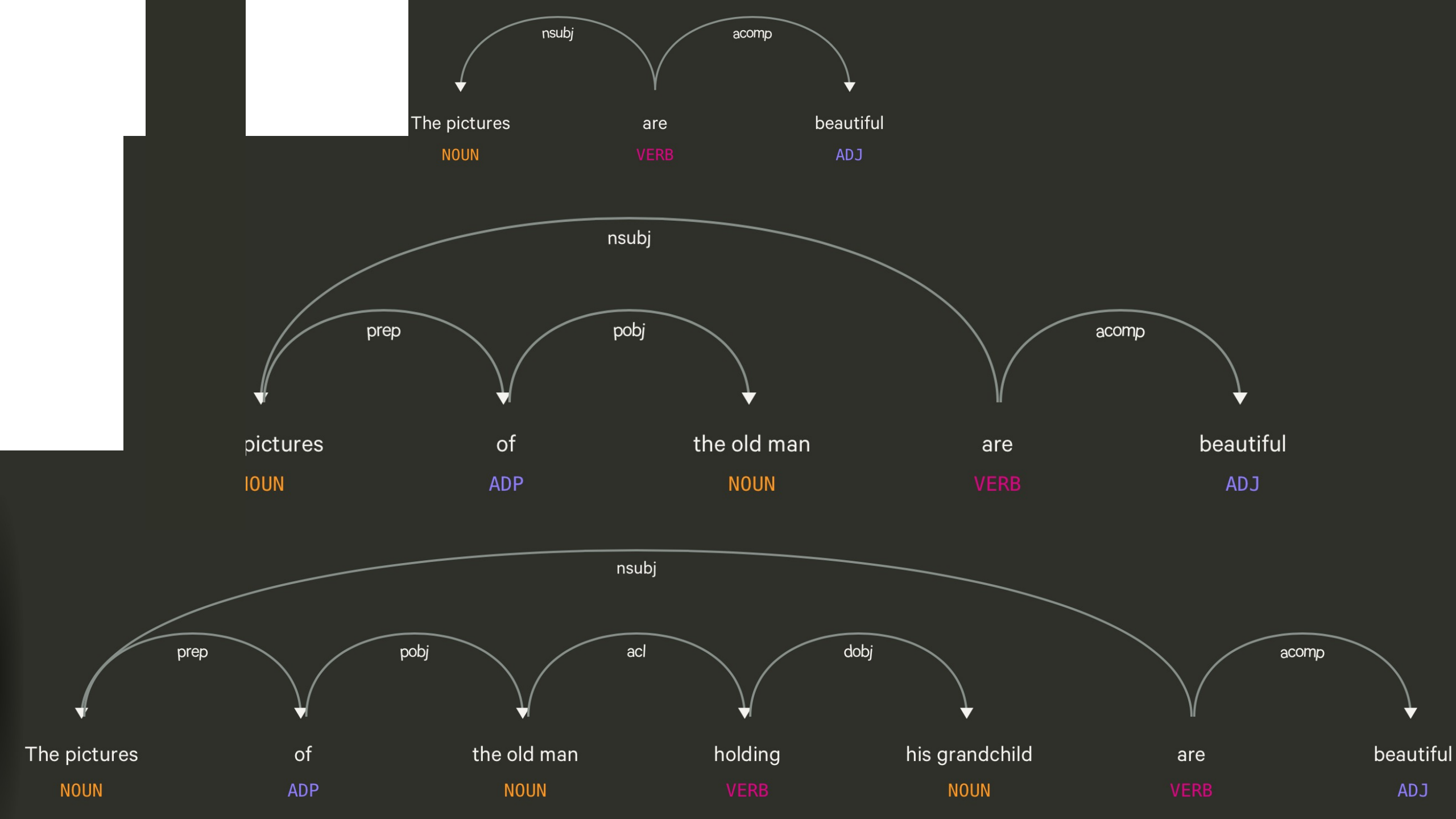
Instead of window of words use more complex contexts: dependency patters. Subj-of-verb, adj-mod, obj-of-verb

Languages have long distance dependencies

*The **pictures are** beautiful.*

*The **pictures** of the old man **are** beautiful.*

*The **pictures** of the old man holding his grandchild **are** beautiful.*


nsubj
acomp
The pictures
NOUN
are
VERB
beautiful
ADJ
nsubj
prep
pobj
acomp
pictures
IOUN
of
ADP
the old man
NOUN
are
VERB
beautiful
ADJ
nsubj
prep
pobj
acl
dobj
acomp
The pictures
NOUN
of
ADP
the old man
NOUN
holding
VERB
his grandchild
NOUN
are
VERB
beautiful
ADJ

Using syntax to define a word's context

Zellig Harris (1968) "The meaning of entities, and the meaning of grammatical relations among them, is related to the restriction of combinations of these entities relative to other entities"

Duty and Responsibility have similar syntactic distributions

Modified by adjectives	additional, administrative, assumed, collective, congressional, constitutional …
Object of verbs	assert, assign, assume, attend to, avoid, become, breach..

Alternates to counts

Raw word frequency is not a great measure of association between words. It's very skewed "the" and "of" are very frequent, but maybe not the most discriminative

We'd rather have a measure that asks whether a context word is particularly informative about the target word.

Instead of raw counts, it's common to transform vectors using TF-IDF or PPMI

TF-IDF

*Term frequency * inverse document frequency*

How often a word occurred in a document

1 over the number of documents that it occurred in

Sparse v. Dense Vectors

Co-occurrence matrix (weighted by TF-IDF or mutual information)

- **Long** (length |V| = 50,000+)
- **Sparse** (most elements are zeros)

Alternative: learn vectors that are

- **Short** (length 200-1000)
- **Dense** (most elements are non-zero)

How do we get dense vectors?

One recipe: train a classifier!

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples.
3. Use logistic regression to train a classifier to distinguish those two cases.
4. Use the weights as the embeddings.

Word2Vec

Mikolov et al. 2013

Learn embeddings as part of the process of word prediction.

Train a classifier to predict neighboring words

Inspired by neural net language models.

In so doing, learn dense embeddings for the words in the training corpus.

Advantages:

Fast, easy to train (much faster than SVD)

Available online in the word2vec package
Including sets of pretrained embeddings!

Word2Vec

Predict each neighboring word in a context window of 2C of surrounding words

So for C=2, we are given a word w_t and we try to predict its 4 surrounding words

$$[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$$

Uses "negative sampling" for training

Negative sampling

lemon, a [tablespoon of apricot preserves or] jam

c1 c2 w c3 c4

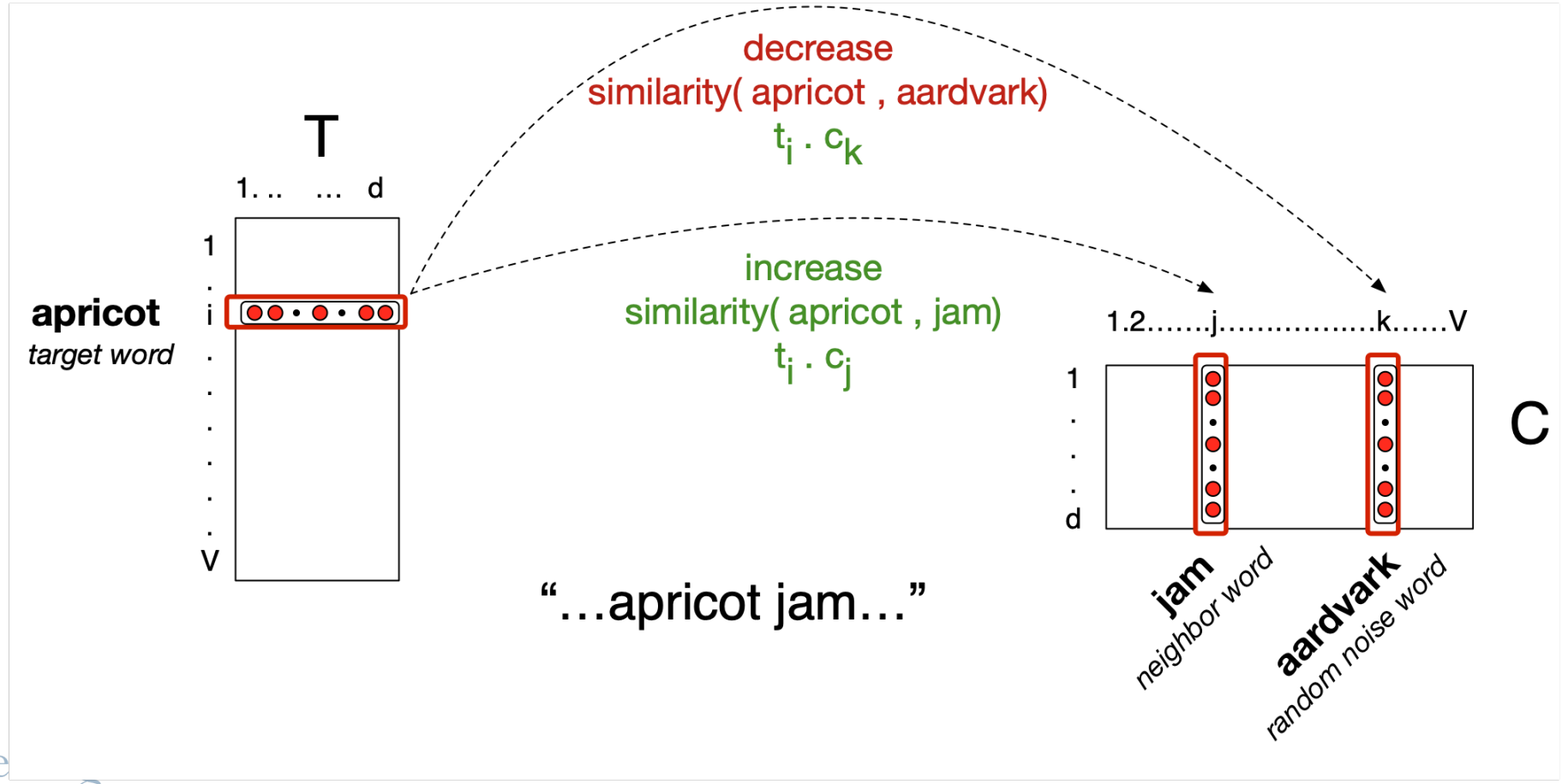
We want predictions
of these words to be high

And these words to be low

[cement metaphysical dear coaxial apricot attendant whence forever puddle]

n1 n2 n3 n4 n5 n6 n7 n8

Neural Network



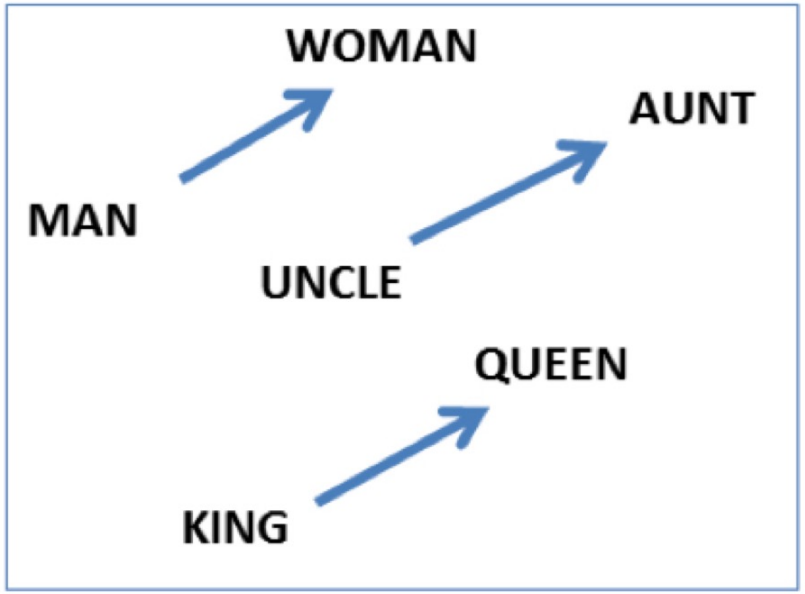
Properties of Embeddings

Nearest Neighbors are surprisingly good

target:	Redmond	Havel	ninjutsu	graffiti	capitulate
	Redmond Wash.	Vaclav Havel	ninja	spray paint	capitulation
	Redmond Washington	president Vaclav Havel	martial arts	grafitti	capitulated
	Microsoft	Velvet Revolution	swordsmanship	taggers	capitulating

Embeddings capture relational meanings

.vector('king') - vector('man') + vector('queen') ≅ vector('woman')



Magnitude: A Fast, Efficient Universal Vector Embedding Utility Package

Ajay Patel
Plasticity Inc.
San Francisco, CA
ajay@plasticity.ai

Alexander Sands
Plasticity Inc.
San Francisco, CA
alex@plasticity.ai

Chris Callison-Burch
Computer and Information
Science Department
University of Pennsylvania
ccb@upenn.edu

Marianna Apidianaki
LIMSI, CNRS
Université Paris-Saclay
91403 Orsay, France
marapi@seas.upenn.edu

Abstract

Vector space embedding models like word2vec, GloVe, and fastText are extremely popular representations in natural language processing (NLP) applications. We present Magnitude, a fast, lightweight tool for utilizing and processing embeddings. Magnitude is an open source Python package with a compact vector storage file format that allows for efficient manipulation of huge numbers of embeddings. Magnitude performs common operations up to 60 to 6,000 times faster than Gensim. Magnitude introduces several

Metric	Cold	Warm
Initial load time	97x	–
Single key query	1x	110x
Multiple key query (n=25)	68x	3x
k-NN search query (k=10)	1x	5,935x

Table 1: Speed comparison of Magnitude versus Gensim for common operations. The 'cold' column represents the first time the operation is called. The 'warm' column indicates a subsequent call with the same keys.

file, a 97x speed-up. Gensim uses 5GB of RAM versus 18KB for Magnitude.

Demo of word vectors

```
# Install Magnitude
pip3 install pymagnitude

# Download Google's word2vec vectors
wget
http://magnitude.plasticity.ai/word2vec+approx/GoogleNews-vectors-negative300.magnitude
# Warning it's 11GB large

# Start Python, and try the commands
# on the next slide
python3
```

Demo of word vectors

```
from pymagnitude import *
vectors = Magnitude("GoogleNews-vectors-
negative300.magnitude")

queen = vectors.query('queen')
king = vectors.query("king")
vectors.similarity(king, queen)
# 0.6510958

vectors.most_similar_approx(king, topn=5)
#[('king', 1.0), ('kings', 0.72), ('prince', 0.62),
('sultan', 0.59), ('ruler', 0.58)]
```

Many possible models

Matrix type
Term-document
Term-context
Pattern-pair

Dim. Reduction
word2vec
GloVe
PCA
LDA
LSA

Reweighting
length norm.
TF-IDF
PPMI
probabilities

Comparisons
cosine
Manhattan
Jaccard
KL divergence
JS distance
DICE

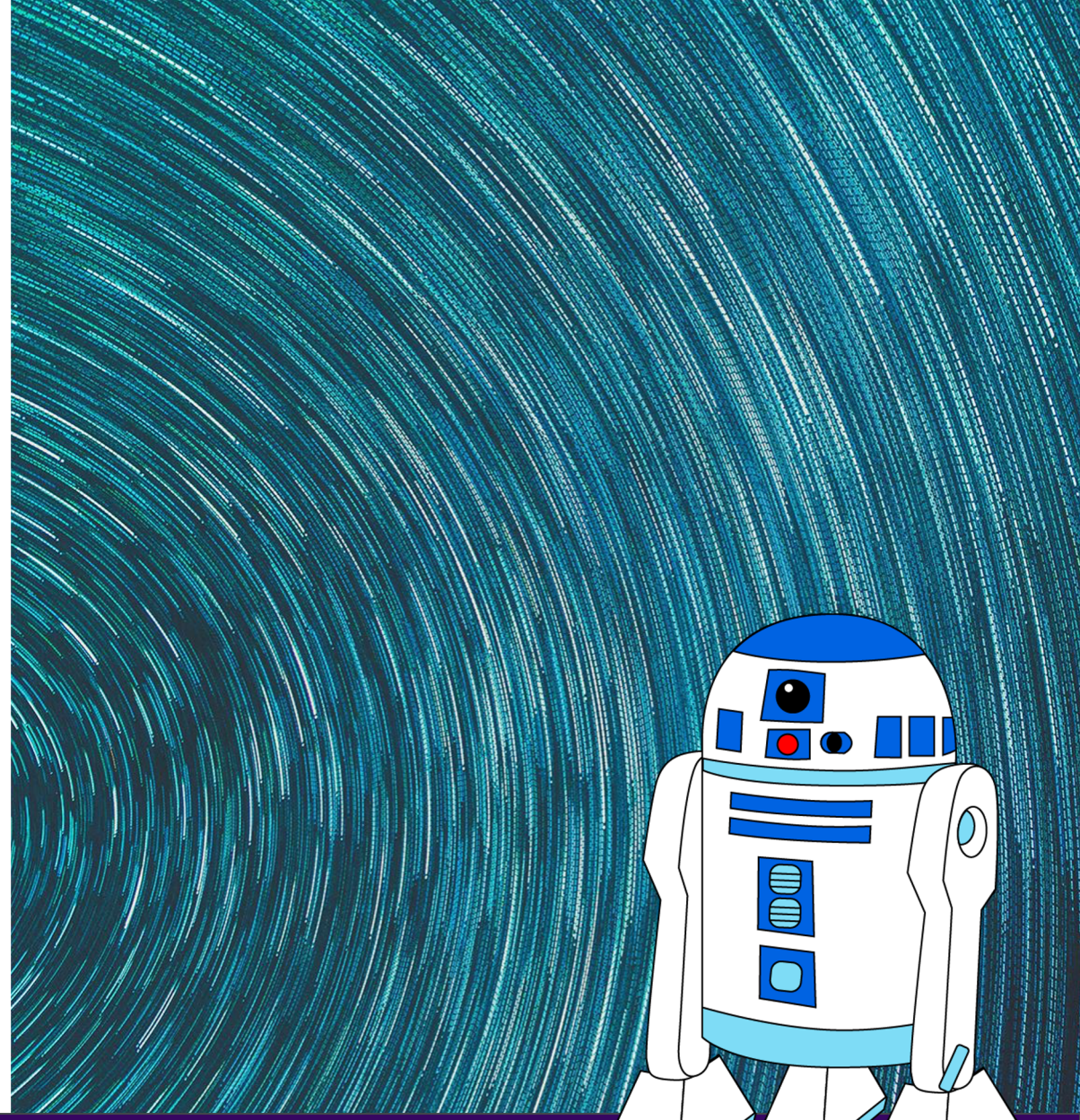
How many dimensions?

What modifications should we make to the input?

CIS 421/521:
ARTIFICIAL INTELLIGENCE

Vector Semantics part 2

Jurafsky and Martin Chapter 6



Penn Engineering
University of Pennsylvania

Sparse vectors

Documents created by term-by-document or term-context matrices are

- **long** (length $|V| = 20{,}000$ to $50{,}000$)
- **sparse** (most elements are zero)

Alternative: dense vectors

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 storing explicit counts
- They may do better at capturing synonymy:
  - *car* and *automobile* are synonyms; but are distinct dimensions in sparse vectors
    - 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**

Dense embeddings you can download!

Word2vec (Mikolov et al.)

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

Fasttext http://www.fasttext.cc/

Glove (Pennington, Socher, Manning)

http://nlp.stanford.edu/projects/glove/

Magnitude (Patel and Sands)

https://github.com/plasticityai/magnitude

Word2vec

Popular embedding method

Very fast to train

Code available on the web

Idea: **predict** rather than **count**

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

Brilliant insight

- Use running text as implicitly supervised training data!
- A word *s* near *apricot*
  - Acts as gold ‘correct answer’ to the question
  - “Is word *w* likely to show up near *apricot*?”
- No need for hand-labeled supervision
- The idea comes from **neural language modeling** (Bengio et al. 2003))

Word2Vec: Skip-Gram Task

Word2vec provides a variety of options. Let's do

- "skip-gram with negative sampling" (SGNS)

Skip-gram algorithm

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

Skip-Gram Training Data

Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 target c3 c4

Assume context words are those in
+/- 2 word window

Skip-Gram Goal

Given a tuple (t,c) = target, context

- (*apricot*, *jam*)
- (*apricot*, *aardvark*)

Return probability that c is a real context word:

$P(+|t,c)$

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

How to compute p(+|t,c)?

Intuition:

- Words are likely to appear near similar words
- Model similarity with dot-product!
- Similarity(t,c) $\approx$ t $\cdot$ c

Problem:

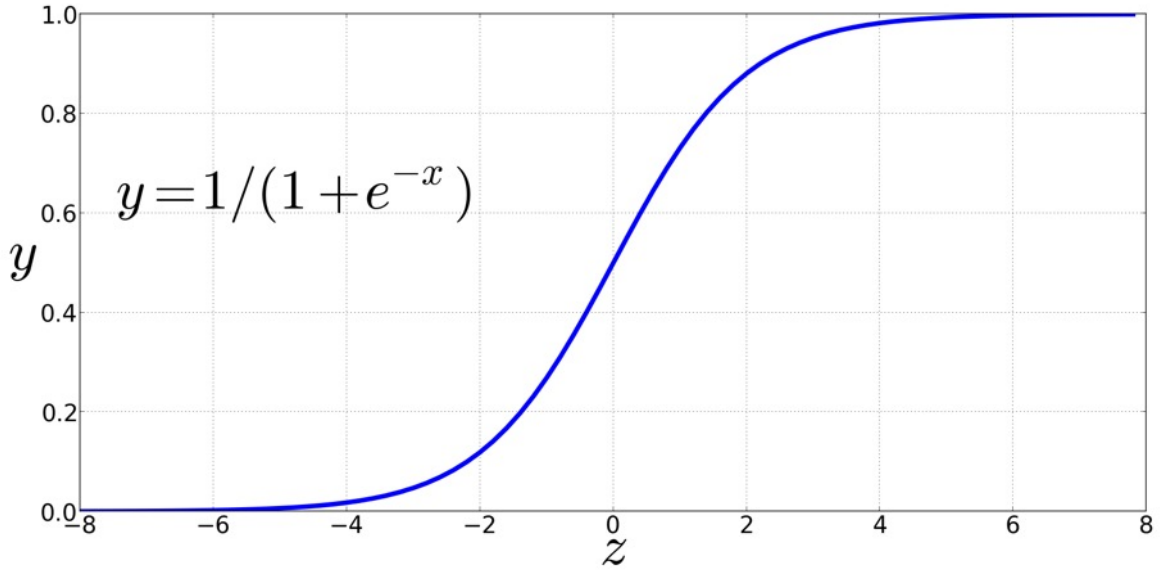
- *Dot product is not a probability!*
  - *(Neither is cosine)*

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

Turning dot product into a probability

The sigmoid lies between 0 and 1:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Turning dot product into a probability

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

Turning dot product into a probability

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

$$\begin{aligned} P(-|t,c) &= 1 - P(+|t,c) \\ &= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}} \end{aligned}$$

For all the context words:

Assume all context words are independent

$$P(+|t, c_{1:k}) = \prod_{i=1}^{\kappa} \frac{1}{1 + e^{-t \cdot c_i}}$$

For all the context words:

Assume all context words are independent

$$P(+|t, c_{1:k}) = \prod_{i=1}^{\kappa} \frac{1}{1 + e^{-t \cdot c_i}}$$

$$\log P(+|t, c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1 + e^{-t \cdot c_i}}$$

Popping back up

Now we have a way of computing the probability of p(+|t,c), which is the probability that c is a real context word for t.

But, we need embeddings for **t** and **c** to do it.

Where do we get those embeddings?

Word2vec learns them automatically!

It starts with an initial set of embedding vectors and then iteratively shifts the embedding of each word *w* to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don't occur nearby.

Skip-Gram Training Data

Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 t c3 c4

Training data: input/output pairs centering on *apricot*

Assume a +/- 2 word window

Skip-Gram Training

Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 t c3 c4

positive examples + t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

For each positive example, we'll create *k* negative examples.
Using *noise* words
Any random word that isn't *t*

How many noise words?

Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 t c3 c4

k=2

positive examples +		**negative examples -**	
t	c	t	c
apricot	tablespoon	apricot	aardvark
apricot	of	apricot	puddle
apricot	preserves	apricot	where
apricot	or	apricot	coaxial
		apricot	twelve
		apricot	hello
		apricot	dear
		apricot	forever

Choosing noise words

Could pick w according to their unigram frequency P(w)

More common to chosen then according to $p_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

α= ¾ works well because it gives rare noise words slightly higher probability

To show this, imagine two events p(a)=.99 and p(b) = .01:

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

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

Learning the classifier

Iterative process.

We'll start with 0 or random weights

Then adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely

over the entire training set:

Setup

Let's represent words as vectors of some length (say 300), randomly initialized.

So we start with 300 * V random parameters

Over the entire training set, we'd like to adjust those word vectors such that we

- Maximize the similarity of the target word, context word pairs (t,c) drawn from the positive data
- Minimize the similarity of the (t,c) pairs drawn from the negative data.

Objective Criteria

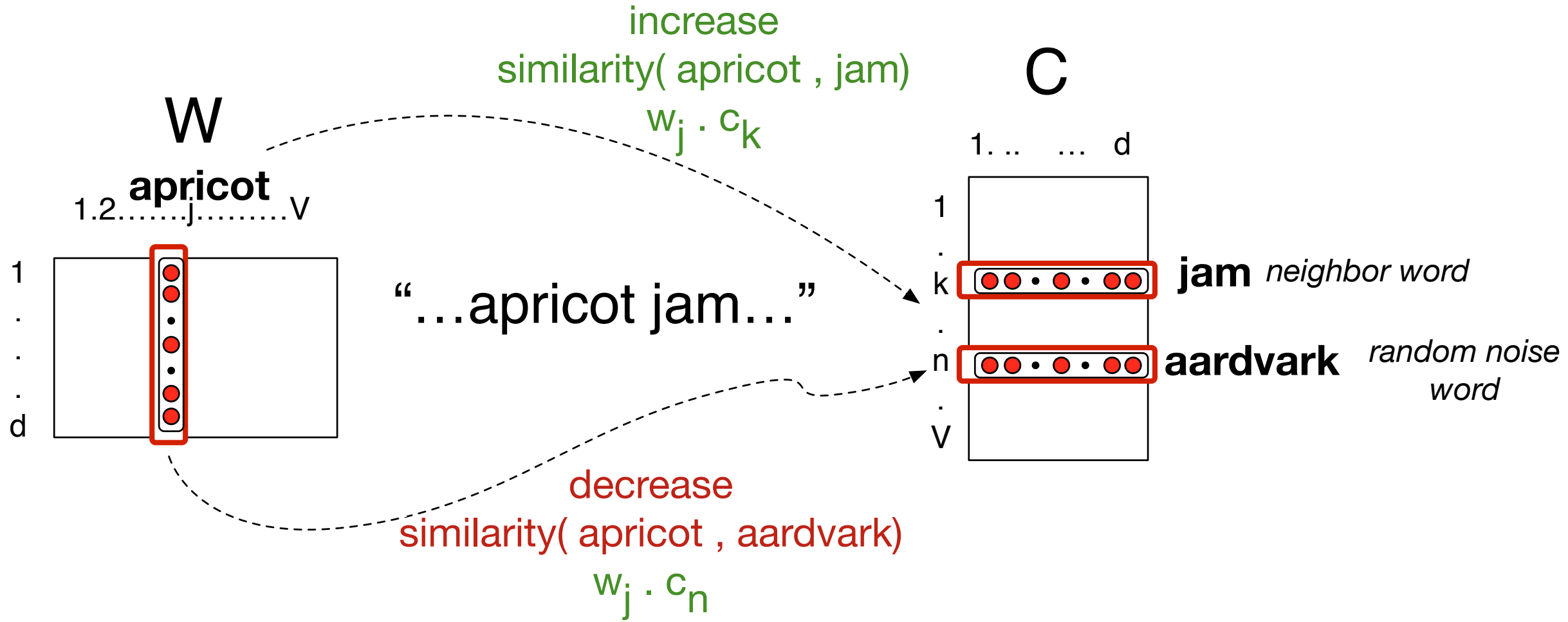
We want to maximize…

$$\sum_{(t,c)\in+} logP(+|t,c) + \sum_{(t,c)\in-} logP(-|t,c)$$

Maximize the + label for the pairs from the positive training data, and the – label for the pairs sample from the negative data.

Focusing on one target word t:

$$
\begin{aligned}
L(\theta) &= \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i) \\
&= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t) \\
&= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}
\end{aligned}
$$


increase
similarity(apricot , jam)
$w_j \cdot c_k$
W
apricot
1.2……j……V
1
.
.
.
d
"…apricot jam…"
C
1. ..
…
d
1
.
k
.
n
.
V
jam
neighbor word
aardvark
random noise word
decrease
similarity(apricot , aardvark)
$w_j \cdot c_n$

Train using gradient descent

Actually learns two separate embedding matrices W and C

Can use W and throw away C, or merge them somehow

Summary: How to learn word2vec (skip-gram) embeddings

Start with V random 300-dimensional vectors as initial embeddings

Use logistic regression

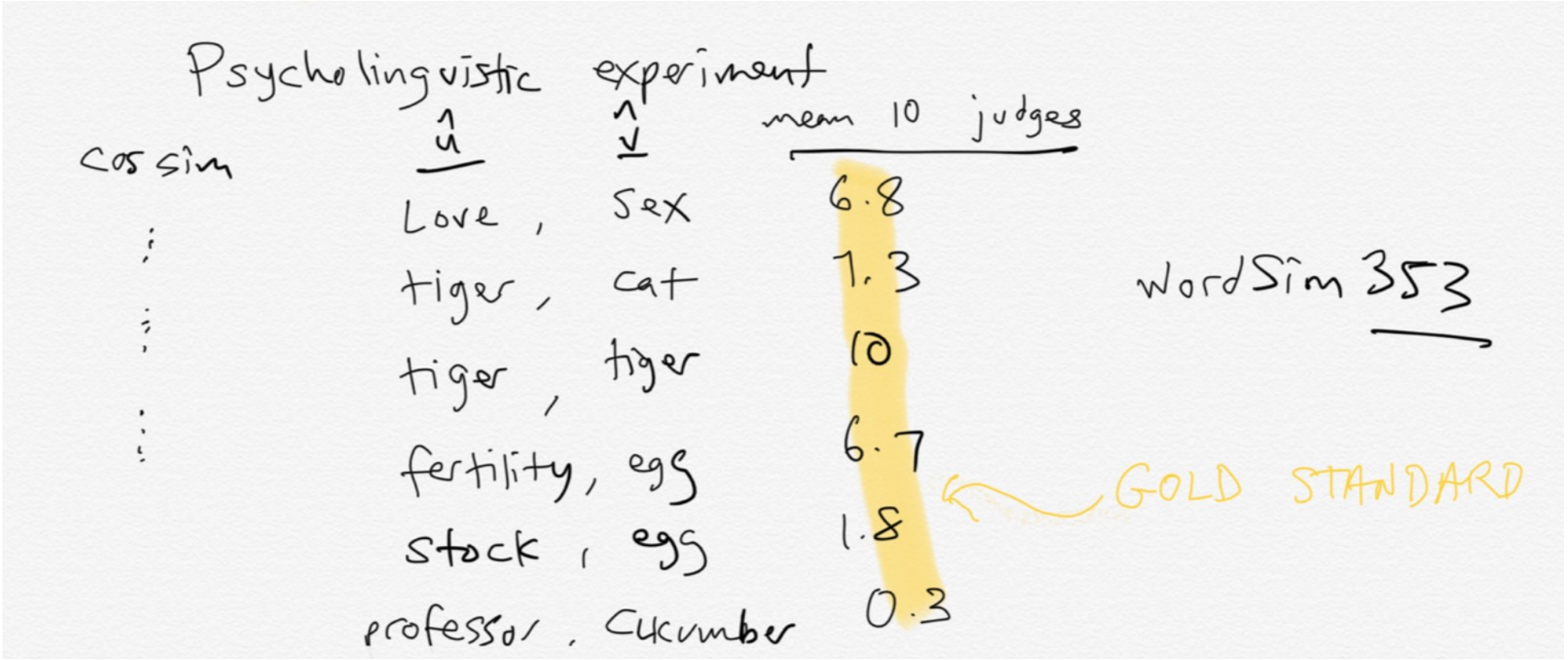
- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

Evaluating embeddings

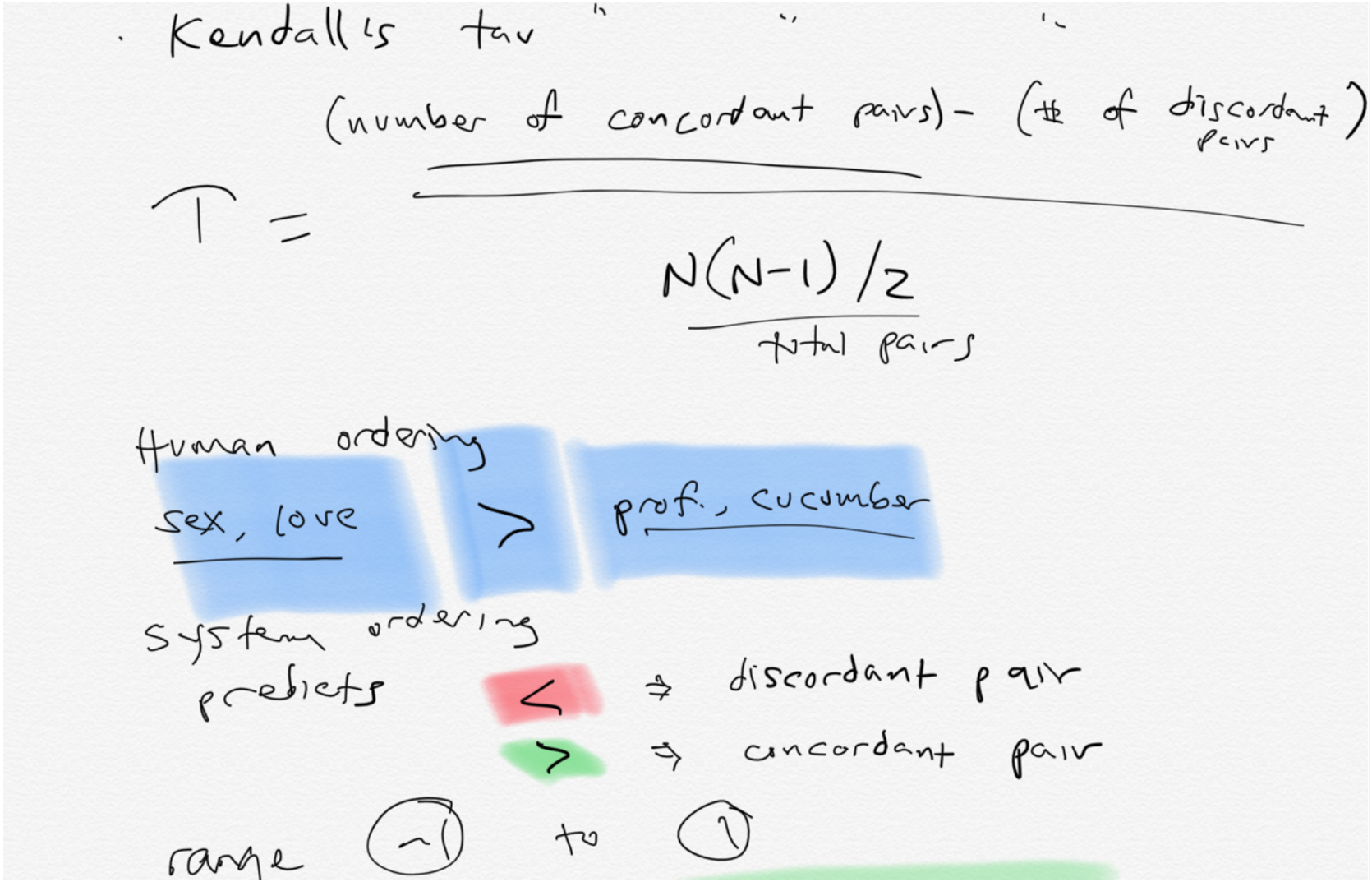
Compare to human scores on word similarity-type tasks:

- WordSim-353 (Finkelstein et al., 2002)
- SimLex-999 (Hill et al., 2015)
- Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
- TOEFL dataset: *"levied" is closest in meaning to:*
  (a) imposed, (b) believed, (c) requested, (d) correlated

Intrinsic evaluation



Compute correlation



Properties of embeddings

Similarity depends on window size C

C = ±2 The nearest words to *Hogwarts:*

- *Sunnydale*
- *Evernight*

C = ±5 The nearest words to *Hogwarts:*

- *Dumbledore*
- *Malfoy*
- *halfblood*

How does context window change word emeddings?

Target Word	BoW5	BoW2	Deps
batman	nightwing	superman	superman
	aquaman	superboy	superboy
	catwoman	aquaman	supergirl
	superman	catwoman	catwoman
	manhunter	batgirl	aquaman
hogwarts	dumbledore	evernight	sunnydale
	hallows	sunnydale	collinwood
	half-blood	garderobe	calarts
	malfoy	blandings	greendale
	snape	collinwood	millfield
florida	gainesville	fla	texas
	fla	alabama	louisiana
	jacksonville	gainesville	georgia
	tampa	tallahassee	california
	lauderdale	texas	carolina
	aspect-oriented	aspect-oriented	event-driven

Solving analogies with embeddings

In a word-analogy task we are given two pairs of words that share a relation (e.g. “man:woman”, “king:queen”).

The identity of the fourth word (“queen”) is hidden, and we need to infer it based on the other three by answering

“*man* is to *woman* as *king* is to — ?”

More generally, we will say **a:a∗** as **b:b∗.**

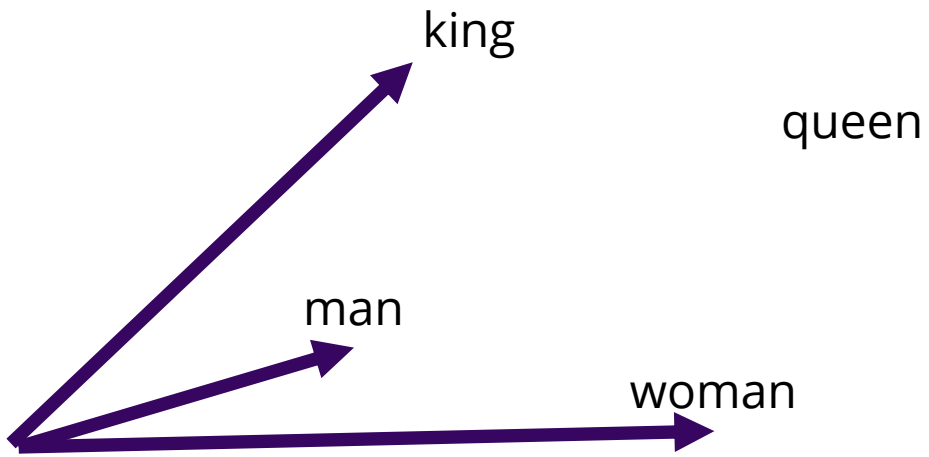
Can we solve these with word vectors?

Vector Arithmetic

a:a∗ as **b:b∗.** **b∗** is a hidden vector.

b∗ should be similar to the vector b − a + a∗

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

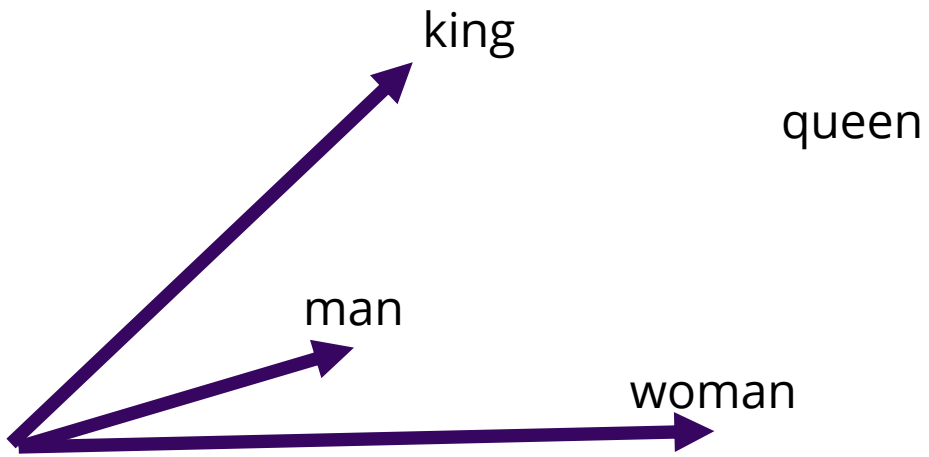


Vector Arithmetic

a:a∗ as **b:b∗.** **b∗** is a hidden vector.

b∗ should be similar to the vector b − a + a∗

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

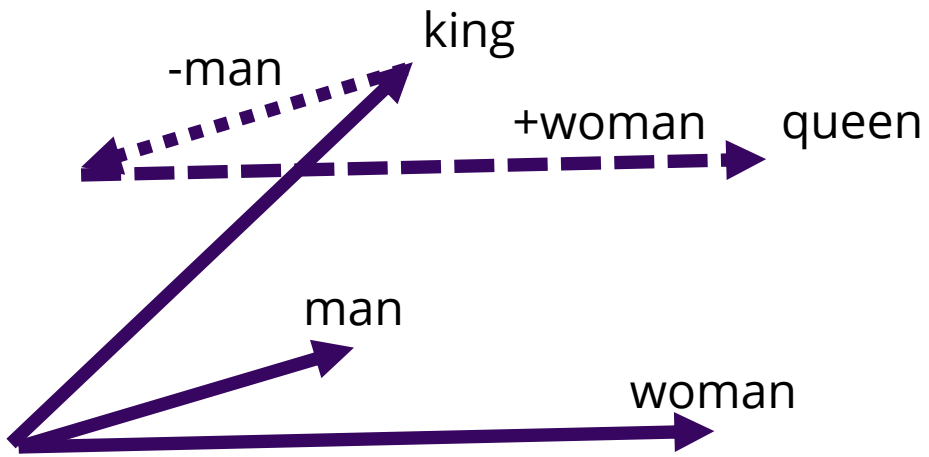


Analogy: Embeddings capture relational meaning!

a:a∗ as **b:b∗.** **b∗** is a hidden vector.

b∗ should be similar to the vector b − a + a∗

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

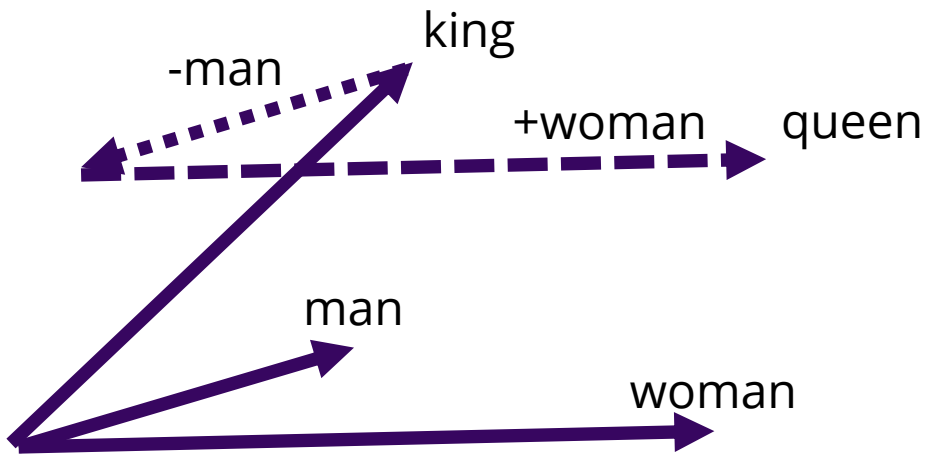


Vector Arithmetic

a:a∗ as **b:b∗.** **b∗** is a hidden vector.

b∗ should be similar to the vector b − a + a∗

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

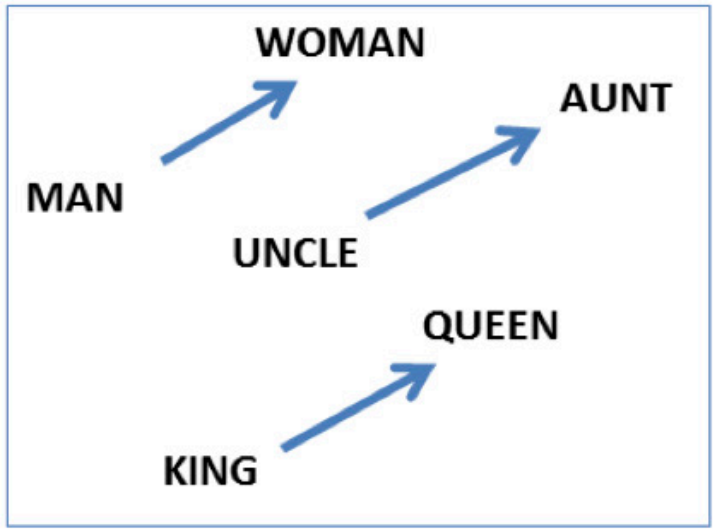


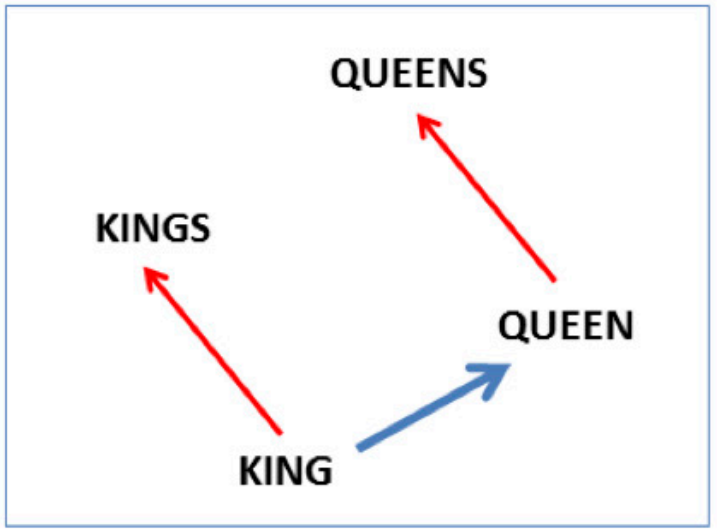
The analogy question can be solved by optimizing: $\arg\max_{b^* \in V} (\cos(b^*, b - a + a^*))$

Analogy: Embeddings capture relational meaning!

vector(‘*king*’) - vector(‘*man*’) + vector(‘*woman*’) ≈ vector(‘queen’)

vector(‘*Paris*’) - vector(‘*France*’) + vector(‘*Italy*’) ≈ vector(‘Rome’)





Vector Arithmetic

If all word-vectors are normalized to unit length then

$$\arg\max_{b^* \in V} \left(\cos\left(b^*, b - a + a^*\right)\right)$$

is equivalent to

$$\arg\max_{b^* \in V} \left(\cos\left(b^*, b\right) - \cos\left(b^*, a\right) + \cos\left(b^*, a^*\right)\right)$$

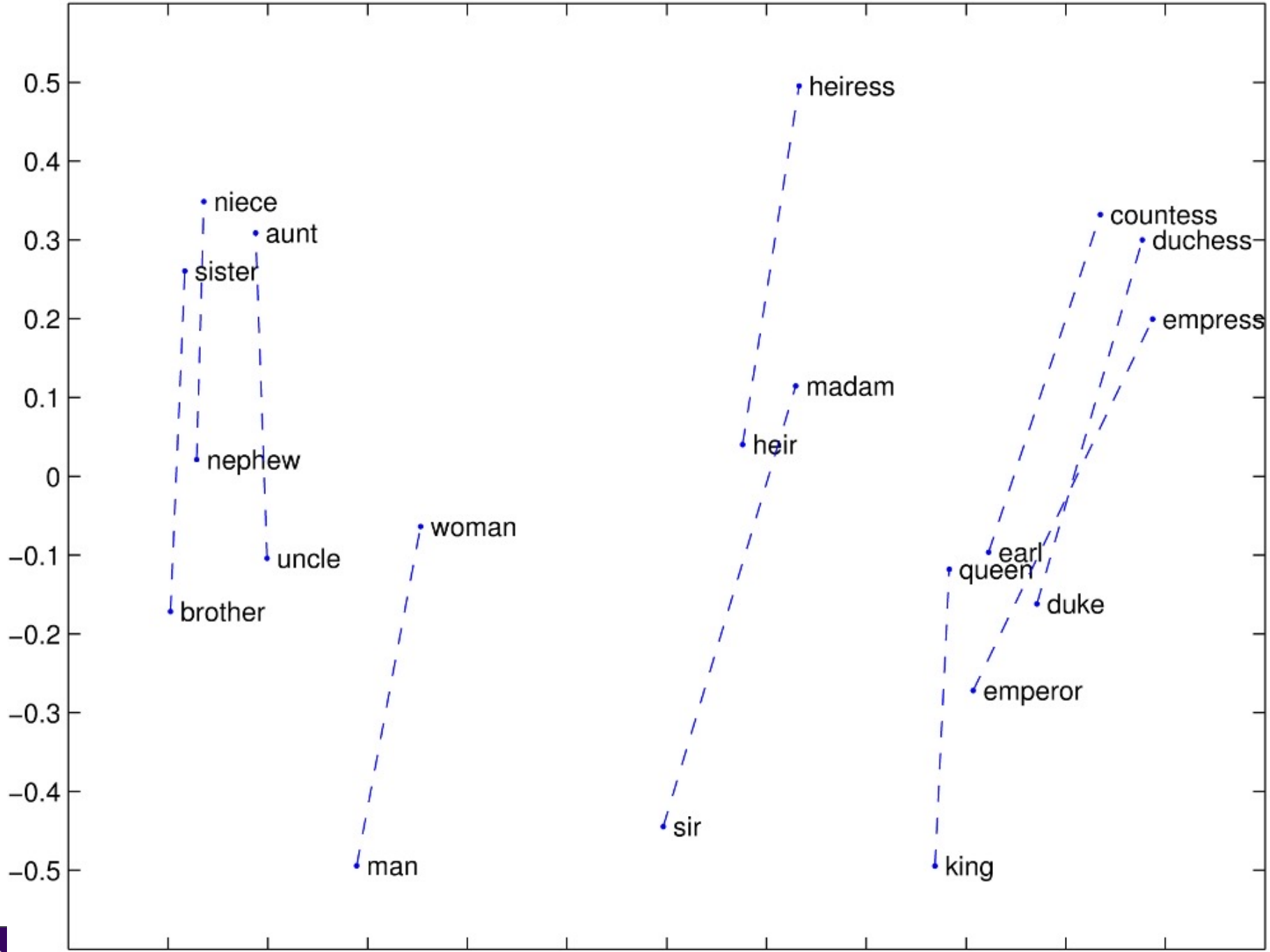
Vector Arithmetic

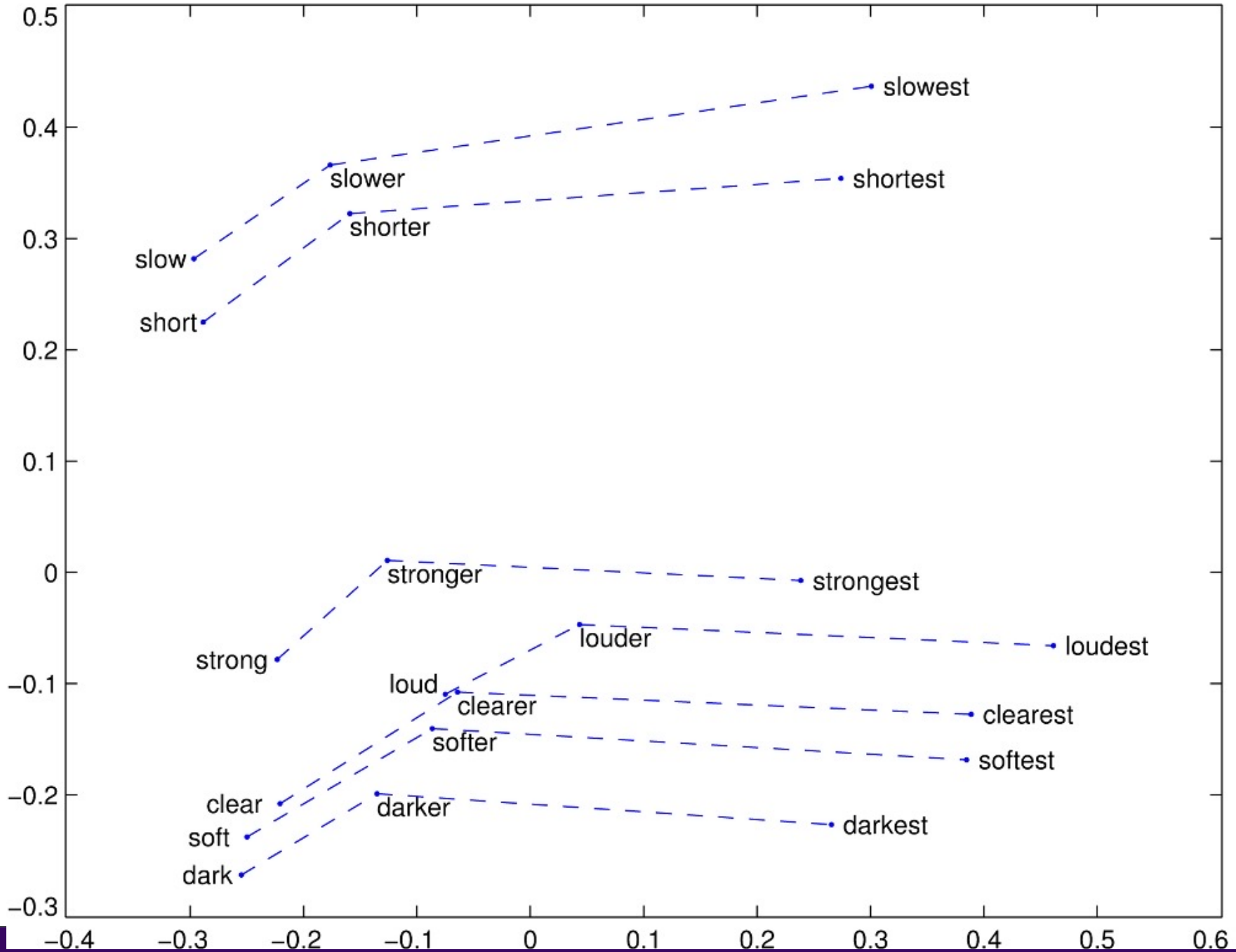
Alternatively, we can require that the direction of the transformation be maintained.

$$\arg\max_{b^* \in V} (\cos(b^*, b - a + a^*))$$

$$\arg\max_{b^* \in V} (\cos(b^* - b, a^* - a))$$

This basically means that b_* − b **shares the same direction with** a_* − a, ignoring the distances


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king


slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6

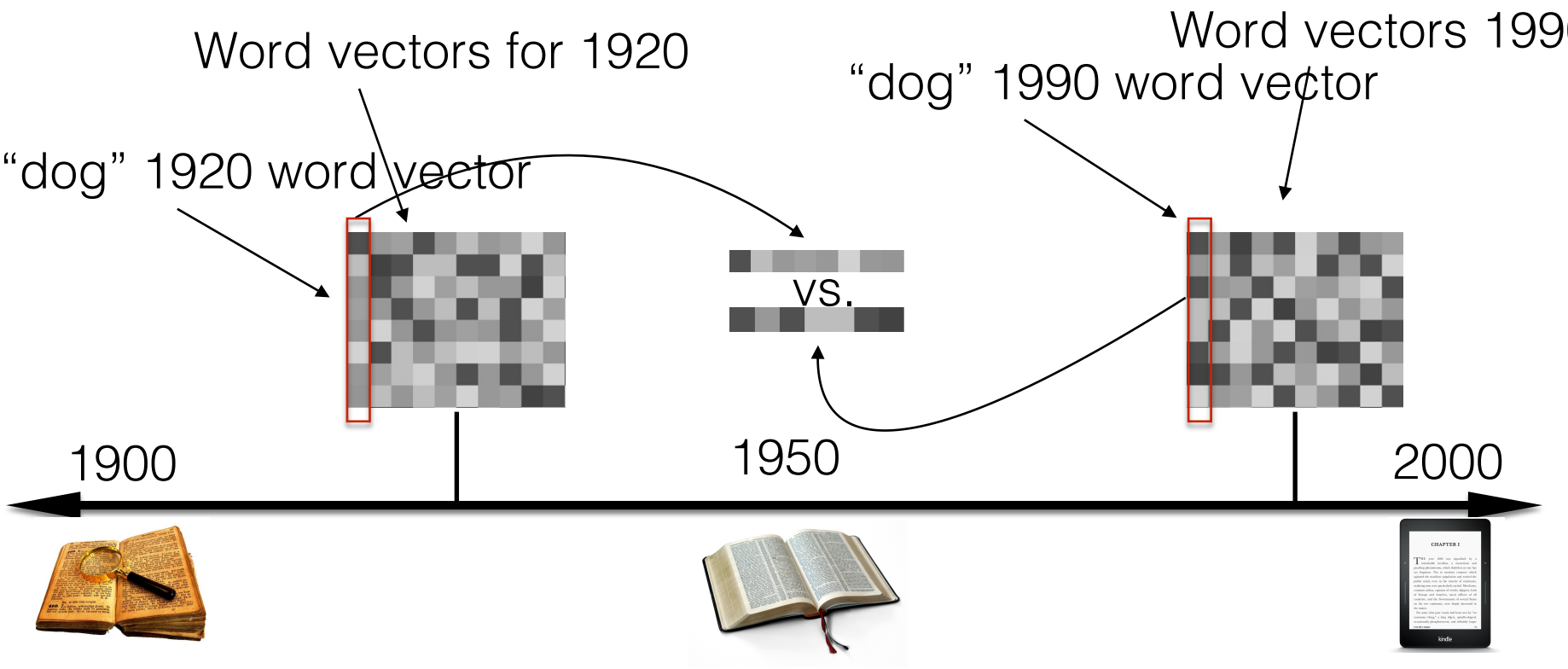
Embeddings can help study word history!

Train embeddings on old books to study changes in word meaning!!

Dan Jurafsky

Will Hamilton

Diachronic word embeddings for studying language change!



Visualizing changes

Project 300 dimensions down into 2

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

gay | gā |

adjective (**gayer, gayest**)

1 (of a person) homosexual (used especially of a man): *that friend of yours, is he gay?*
- relating to or used by homosexuals: *a gay bar* | *the gay vote can decide an election.*

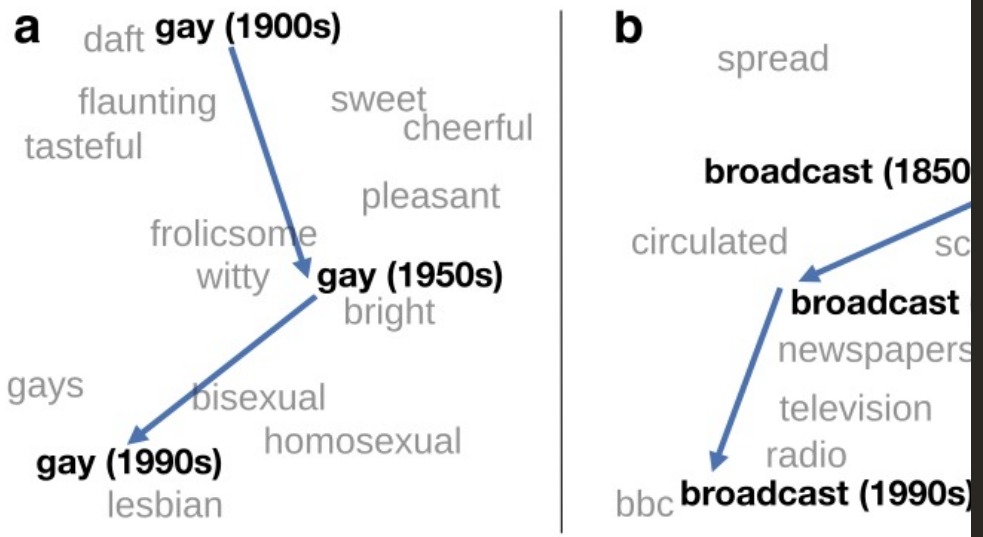
2 *dated* lighthearted and carefree: *Nan had a gay disposition and a very pretty face.*
- brightly colored; showy; brilliant: *a gay profusion of purple and pink sweet peas.*

broadcast | 'brôd,kast |

verb (past and past participle **broadcast**) *[with object]*

1 transmit (a program or some information) by radio or television: *the announcement was broadcast live* | (as noun **broadcasting**) *: the 1920s saw the dawn of broadcasting.*
- *[no object]* take part in a radio or television transmission: *the station broadcasts 24 hours a day.*
- tell (something) to many people; make widely known: *we don't want to broadcast our unhappiness to the world.*

2 scatter (seeds) by hand or machine rather than placing in drills or rows.



~30 million books, 1850-1990, Goog

awful | 'ôfəl |

adjective

1 very bad or unpleasant: *the place smelled awful* | *I look awful in a swimsuit* | *an awful speech.*
- extremely shocking; horrific: *awful, bloody images.*
- (of a person) very unwell, troubled, or unhappy: *I felt awful for being so angry with him* | *you look awful—you should go and lie down.*

2 *[attributive]* used to emphasize the extent of something, especially something unpleasant or negative: *I've made an awful fool of myself.*

3 *archaic* inspiring reverential wonder or fear.

Embeddings and bias

Embeddings reflect cultural bias

Ask “Paris : France :: Tokyo : x”

- x = Japan

Ask “father : doctor :: mother : x”

- x = nurse

Ask “man : computer programmer :: woman : x”

- x = homemaker

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 *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

Measuring cultural bias

Implicit Association test (Greenwald et al 1998): How associated are

- concepts (*flowers*, *insects*) & attributes (*pleasantness*, *unpleasantness*)?
- Studied by measuring timing latencies for categorization.

Psychological findings on US participants:

- African-American names are associated with unpleasant words (more than European-American names)
- Male names associated more with math, female names with arts
- Old people's names with unpleasant words, young people with pleasant words.

Embeddings reflect cultural bias

Caliskan et al. replication with embeddings:

- African-American names (*Leroy, Shaniqua*) had a higher GloVe cosine with unpleasant words (*abuse, stink, ugly*)
- European American names (*Brad, Greg, Courtney*) had a higher cosine with pleasant words (*love, peace, miracle*)

Embeddings reflect and replicate all sorts of pernicious biases.

Aylin Caliskan, Joanna J. Bruson and Arvind Narayanan. 2017. Semantics derived automatically from language corpora contain human-like biases. Science 356:6334, 183-186.

Directions

Debiasing algorithms for embeddings

Use embeddings as a tool to study historical bias

Embeddings as a window onto history

Use the Hamilton historical embeddings

The cosine similarity of embeddings for decade X for occupations (like teacher) to male vs female names

- Is correlated with the actual percentage of women teachers in decade X

Nikhil Garg, Londa Schiebinger, Dan Jurafsky, and James Zou, (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

History of biased framings of women

Embeddings for competence adjectives are biased toward men

- *Smart, wise, brilliant, intelligent, resourceful, thoughtful, logical, etc.*

This bias is slowly decreasing

Nikhil Garg, Londa Schiebinger, Dan Jurafsky, and James Zou, (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

Princeton Trilogy experiments

Study 1: Katz and Braley (1933)

Investigated whether traditional social stereotypes had a cultural basis

Ask 100 male students from Princeton University to choose five traits that characterized different ethnic groups (for example Americans, Jews, Japanese, Negroes) from a list of 84 word

84% of the students said that Negroes were superstitious and 79% said that Jews were shrewd. They were positive towards their own group.

Study 2: Gilbert (1951)
Less uniformity of agreement about unfavorable traits than in 1933.

Study 3: Karlins et al. (1969)

Many students objected to the task but this time there was greater agreement on the stereotypes assigned to the different groups compared with the 1951 study. Interpreted as a re-emergence of social stereotyping but in the direction more favorable stereotypical images.

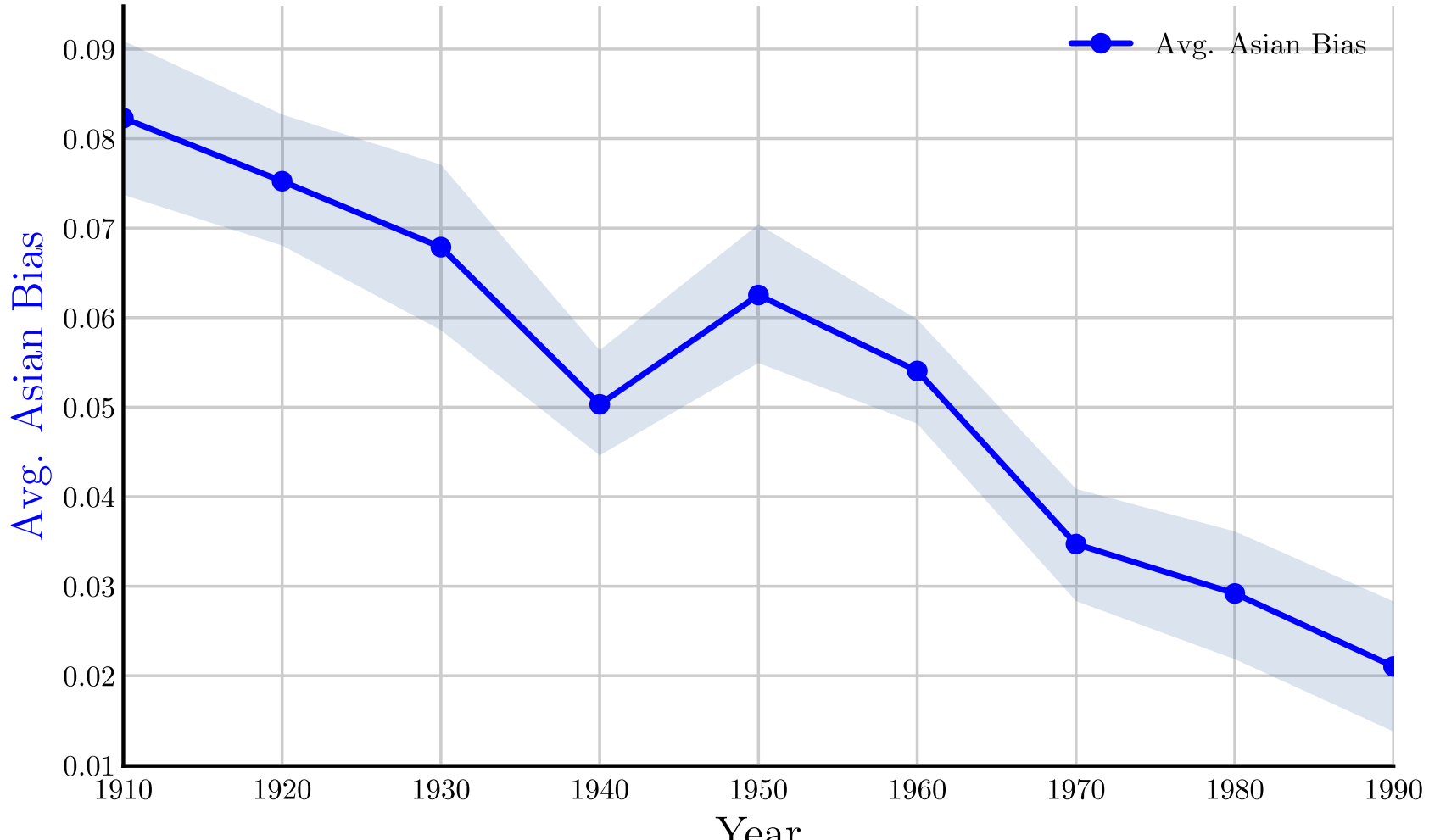
Embeddings reflect ethnic stereotypes over time

- Princeton trilogy experiments
- Attitudes toward ethnic groups (1933, 1951, 1969) scores for adjectives
  - *industrious, superstitious, nationalistic*, etc
- Cosine of Chinese name embeddings with those adjective embeddings correlates with human ratings.

Nikhil Garg, Londa Schiebinger, Dan Jurafsky, and James Zou, (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

Change in linguistic framing 1910-1990

Change in association of Chinese names with adjectives framed as "othering" (*barbaric*, *monstrous*, *bizarre*)



Changes in framing: adjectives associated with Chinese

1910	1950	1990
Irresponsible	Disorganized	Inhibited
Envious	Outrageous	Passive
Barbaric	Pompous	Dissolute
Aggressive	Unstable	Haughty
Transparent	Effeminate	Complacent
Monstrous	Unprincipled	Forceful
Hateful	Venomous	Fixed
Cruel	Disobedient	Active
Greedy	Predatory	Sensitive
Bizarre	Boisterous	Hearty

Nikhil Garg, Londa Schiebinger, Dan Jurafsky, and James Zou, (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, *115*(16), E3635–E3644

Conclusion

Embeddings = vector models of meaning

- More fine-grained than just a string or index
- Especially good at modeling similarity/analogy
  - Just download them and use cosines!!
- Can use sparse models (tf-idf) or dense models (word2vec, GLoVE)

- **Useful in practice but know they encode cultural stereotypes**