

# Introduction to **Information Retrieval**

CS276: Information Retrieval and Web Search  
Christopher Manning and Pandu Nayak

Lecture 15: Distributed Word Representations  
for Information Retrieval

# How can we more robustly match a user's search intent?

We want to **understand** the query, not just do `String equals()`

- If user searches for [Dell notebook battery size], we would like to match documents discussing “Dell laptop battery capacity”
- If user searches for [Seattle motel], we would like to match documents containing “Seattle hotel”

A naïve information retrieval system does nothing to help  
Simple facilities that we have already discussed do a bit to help

- Spelling correction
- Stemming / case folding

But we'd like to better **understand** when query/document match

# How can we more robustly match a user's search intent?

---

- Use of **anchor text** may solve this by providing human authored synonyms, but not for new or less popular web pages, or non-hyperlinked collections
- **Relevance feedback** could allow us to capture this if we get near enough to matching documents with these words
- We can also fix this with information on **word similarities**:
  - A manual **thesaurus** of synonyms
  - A **measure of word similarity**
    - Calculated from a big document collection
    - Calculated by query log mining (common on the web)

# Example of manual thesaurus

The screenshot shows the PubMed search interface. On the left, there's a sidebar with links like 'About Entrez', 'Text Version', 'Entrez PubMed Overview', 'Help | FAQ', 'Tutorial', 'New/Noteworthy', 'E-Utilities', 'PubMed Services', 'Journals Database', 'MeSH Browser', 'Single Citation Mobox'. The main area has the PubMed logo and the National Library of Medicine NLM logo. A search bar contains the text 'Search PubMed for cancer'. Below the search bar are buttons for 'Limits', 'Preview/Index', 'History', 'Clipboard', and 'Details'. A large blue box labeled 'PubMed Query:' contains the search term: ("neoplasms"[MeSH Terms] OR cancer[Text Word]). At the bottom are 'Search' and 'URL' buttons.

PubMed

Nucleotide

Protein

Genome

Structure

PopSet

Taxonomy

Search PubMed for cancer

Go Clear

Limits Preview/Index History Clipboard Details

PubMed Query:

("neoplasms"[MeSH Terms] OR cancer[Text Word])

Search URL

# Thesaurus-based query expansion

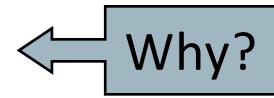
---

- For each term  $t$  in a query, expand the query with synonyms and related words of  $t$  from the thesaurus
  - feline → feline cat
- May weight added terms less than original query terms.
- Generally increases recall
- Widely used in many science/engineering fields
- May significantly decrease precision, particularly with ambiguous terms.
  - “interest rate” ↗ “interest rate fascinate evaluate”
- There is a high cost of manually producing a thesaurus
  - And for updating it for scientific changes

# Search log query expansion

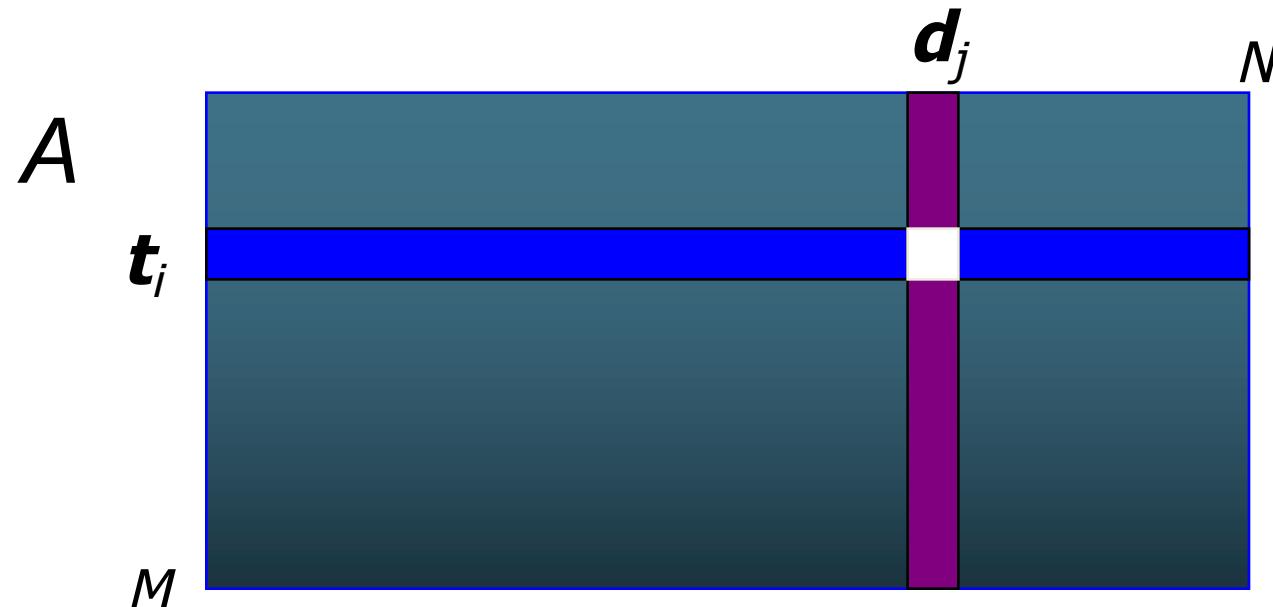
- Context-free query expansion ends up problematic
  - [light hair]  $\approx$  [fair hair] At least in U.K./Australia?  $\approx$  blonde
    - So expand [light]  $\Rightarrow$  [light fair]
  - But [bed light price]  $\neq$  [bed fair price]
- You can learn query context-specific rewritings from search logs by attempting to identify the same user making a second attempt at the same user need
  - [Hinton word vector]
  - [Hinton word embedding]
- In this context, [vector]  $\approx$  [embedding]
  - But not when talking about a *disease vector* or C++!

# Automatic Thesaurus Generation

- Attempt to generate a thesaurus automatically by analyzing a collection of documents
- Fundamental notion: similarity between two words
- Definition 1: Two words are similar if they co-occur with similar words.
- Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.
- You can harvest, peel, eat, prepare, etc. apples and pears, so apples and pears must be similar.
- Co-occurrence based is more robust, grammatical relations are more accurate. 

# Co-occurrence Thesaurus

- Simplest way to compute one is based on term-term similarities in  $C = AA^T$  where  $A$  is term-document matrix.
- $w_{i,j}$  = (normalized) weight for  $(t_i, d_j)$



- For each  $t_i$ , pick terms with high values in  $C$

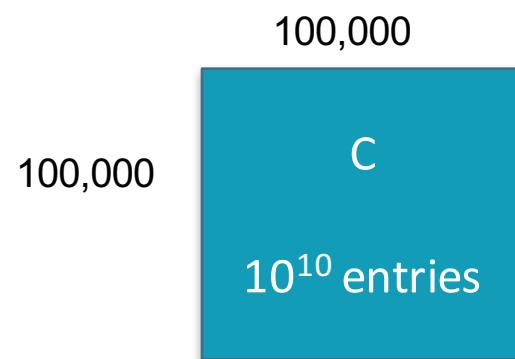
What does  $C$  contain if  $A$  is a term-doc incidence (0/1) matrix?

# Automatic thesaurus generation example

Word	Nearest neighbors
absolutely	absurd, whatsoever, totally, exactly, nothing
bottomed	dip, copper, drops, topped, slide, trimmed
captivating	shimmer, stunningly, superbly, plucky, witty
doghouse	dog, porch, crawling, beside, downstairs
makeup	repellent, lotion, glossy, sunscreen, skin, gel
mediating	reconciliation, negotiate, cease, conciliation
keeping	hoping, bring, wiping, could, some, would
lithographs	drawings, Picasso, Dali, sculptures, Gauguin
pathogens	toxins, bacteria, organisms, bacterial, parasites
senses	grasp, psyche, truly, clumsy, naïve, innate

# Automatic Thesaurus Generation Issues

- Quality of associations is usually a problem
- Sparsity



- Term ambiguity may introduce irrelevant statistically correlated terms.
  - “planet earth facts” ☰ “planet earth soil ground facts”
- Since terms are highly correlated anyway, expansion may not retrieve many additional documents.

# Can you directly learn term relations?

---

- Basic IR is scoring on  $q^T d$
- No treatment of synonyms; no machine learning
- Can we learn parameters  $W$  to rank via  $q^T W d$
- Problem is again sparsity –  $W$  is huge  $> 10^{10}$

# How can we represent term relations?

- With the standard symbolic encoding of terms, each term is a dimension
- Different terms have no inherent similarity
- $\text{motel} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$   
 $\text{hotel} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = 0$
- If query on *hotel* and document has *motel*, then our query and document vectors are **orthogonal**

# Is there a better way?

---

- Idea:
  - Can we learn a low dimensional representation of a word in  $\mathbb{R}^d$  such that dot products  $u^T v$  express word similarity?
  - We could still if we want to include a “translation” matrix between vocabularies (e.g., cross-language):  $u^T W v$ 
    - But now  $W$  is small!
  - Supervised Semantic Indexing (Bai et al. *Journal of Information Retrieval* 2009) shows successful use of learning  $W$  for information retrieval
- But we'll develop direct similarity in this class

# Distributional similarity based representations

- You can get a lot of value by representing a word by means of its neighbors
- “You shall know a word by the company it keeps”
  - (J. R. Firth 1957: 11)
- One of the most successful ideas of modern statistical NLP

government debt problems turning into banking crises as has happened in saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

# Solution: Low dimensional vectors

---

- The number of topics that people talk about is small (in some sense)
  - Clothes, movies, politics, ...
- Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25 – 1000 dimensions
- How to reduce the dimensionality?
  - Go from big, sparse co-occurrence count vector to low dimensional “word embedding”

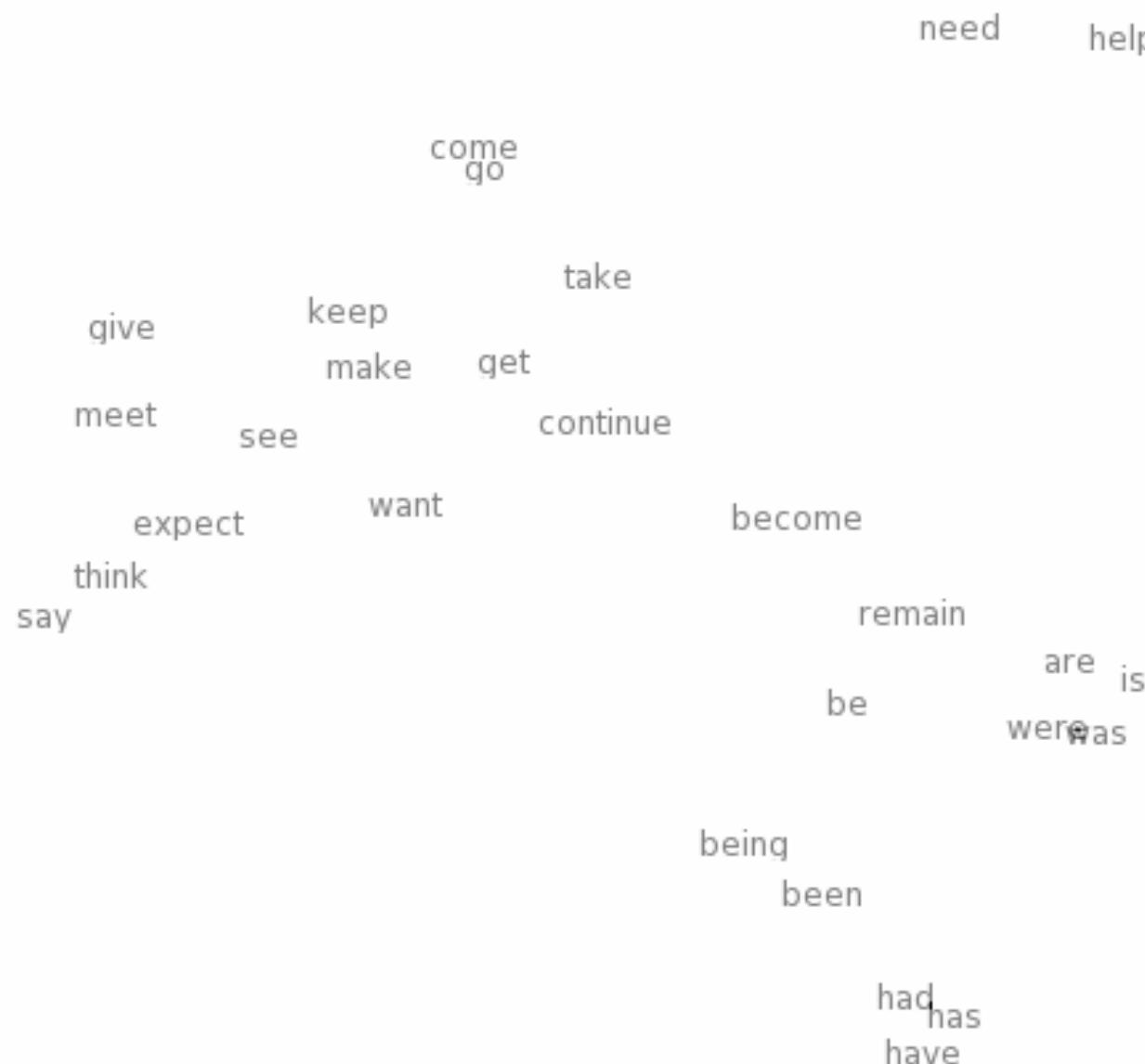
# Traditional Way: Latent Semantic Indexing/Analysis

---

- Use Singular Value Decomposition (SVD) – kind of like Principal Components Analysis (PCA) for an arbitrary rectangular matrix – or just random projection to find a low-dimensional basis
- Theory is that similarity is preserved as much as possible
- Weakly, you can actually gain in IR by doing LSA as “noise” of term variation gets replaced by semantic “concepts”
- Popular in the 1990s [Deerwester et al. 1990, etc.]
  - Results were always somewhat iffy (... it worked sometimes)
  - Harder to implement efficiently in an IR system (dense vectors!)
- Discussed in *IIR* chapter 18, but not discussed further here
  - And not on the exam (!)

# “NEURAL EMBEDDINGS”

# Neural word embeddings - visualization



# Idea: Directly learn low-dimensional word vectors

---

- Old idea. Relevant for this lecture & deep learning:
  - Learning representations by back-propagating errors. (Rumelhart et al., 1986)
  - A neural probabilistic language model (Bengio et al., 2003)
  - NLP (almost) from Scratch (Collobert & Weston, 2008)
  - A recent, even simpler and faster model: word2vec (Mikolov et al. 2013) → intro now
- Initial models were quite non-linear and slow; recent models have used fast, bilinear models

# Main Idea of word2vec

---

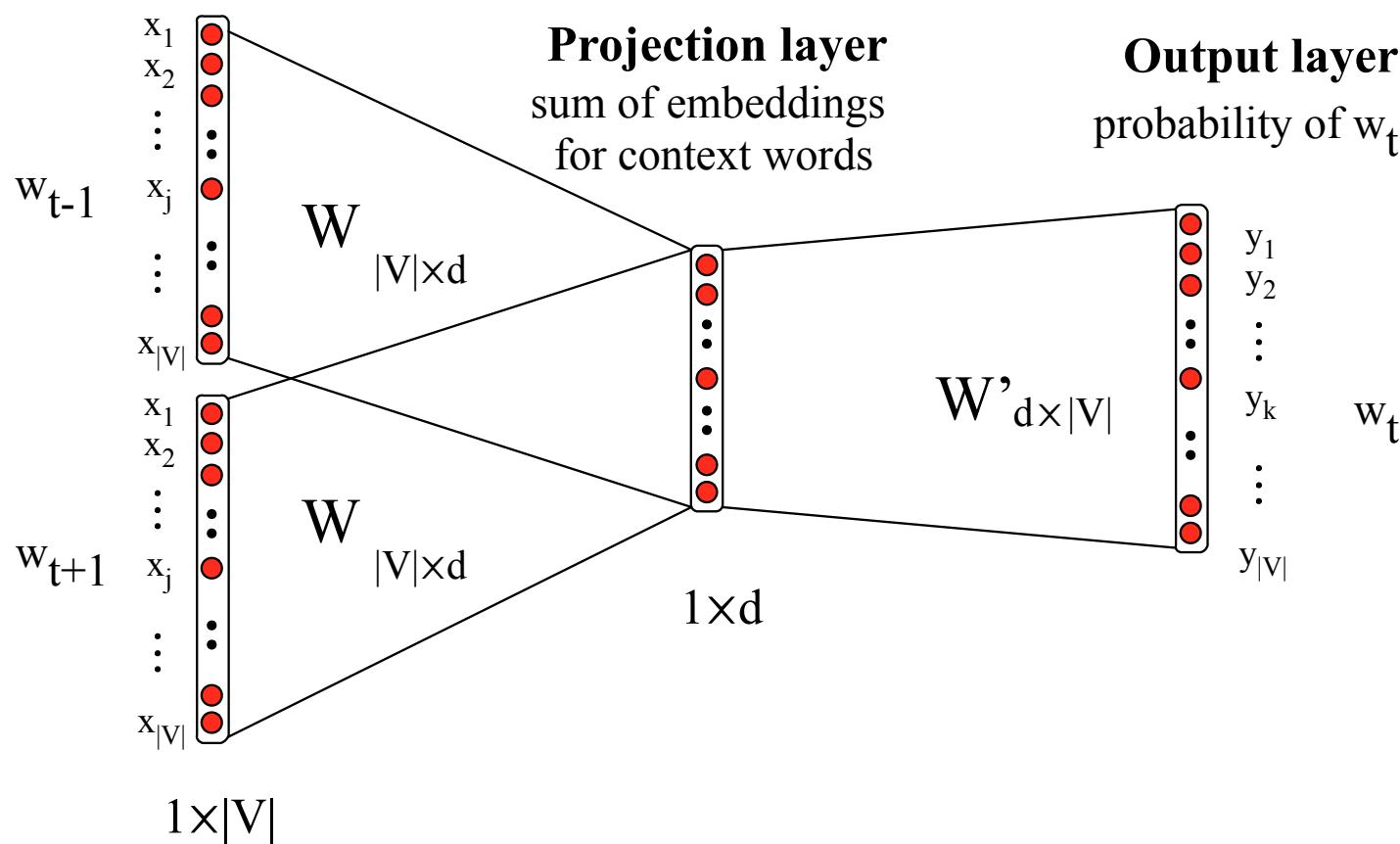
- Instead of capturing co-occurrence counts directly, predict surrounding words of every word
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary
- Two variants:
  - CBOW: Predict target from bag of words context
  - Skipgram: Predict context words from target (position-independent)
- In general SGNS (Skipgram, Negative Sampling) has ended up preferred but we'll look at CBOW, as our IR paper uses it....

# CBOW (Continuous Bag of Words)

## Input layer

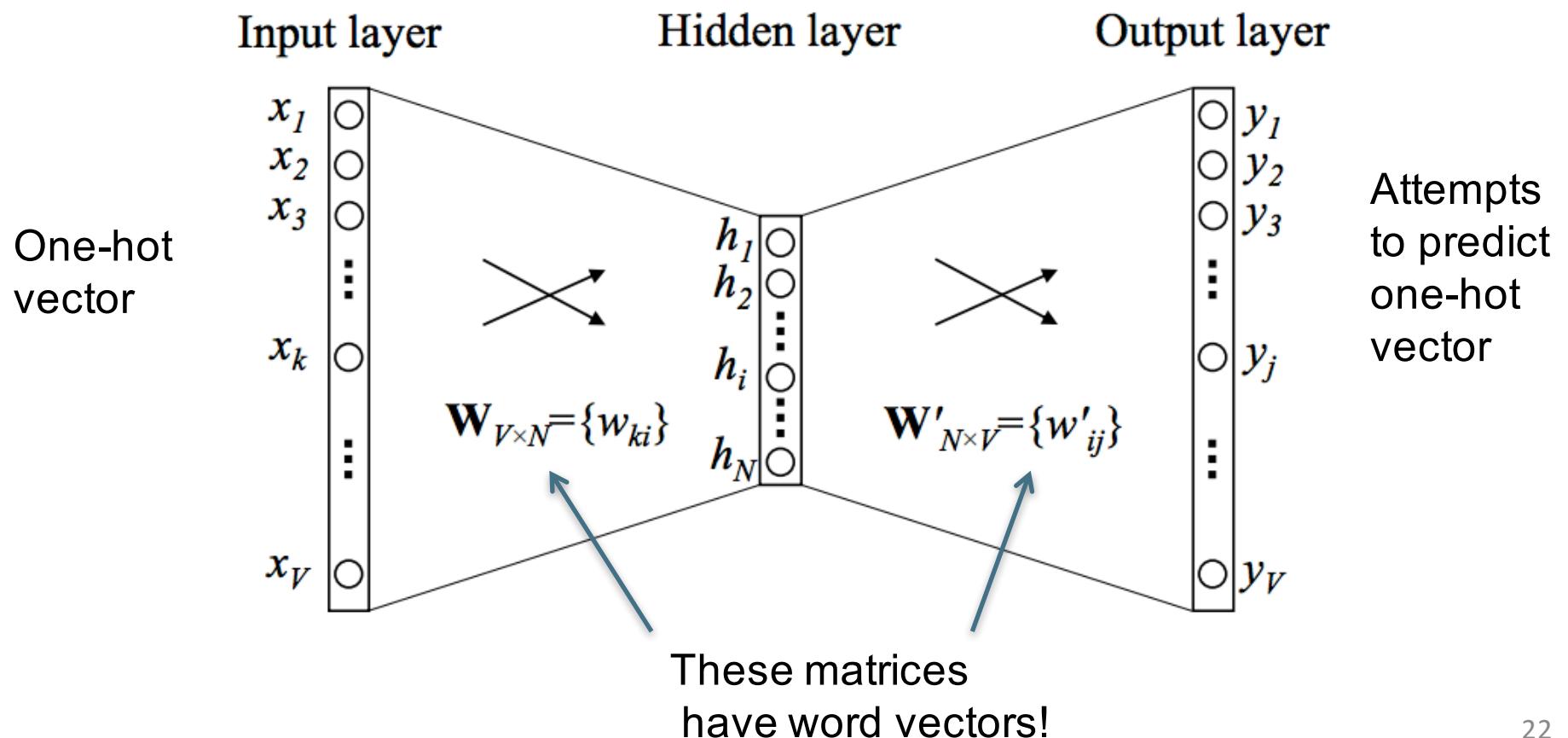
1-hot input vectors  
for each context word

CBOW learns a word embedding by maximizing the log conditional probability of a word given the bag of context words occurring within a fixed-sized window around that word.

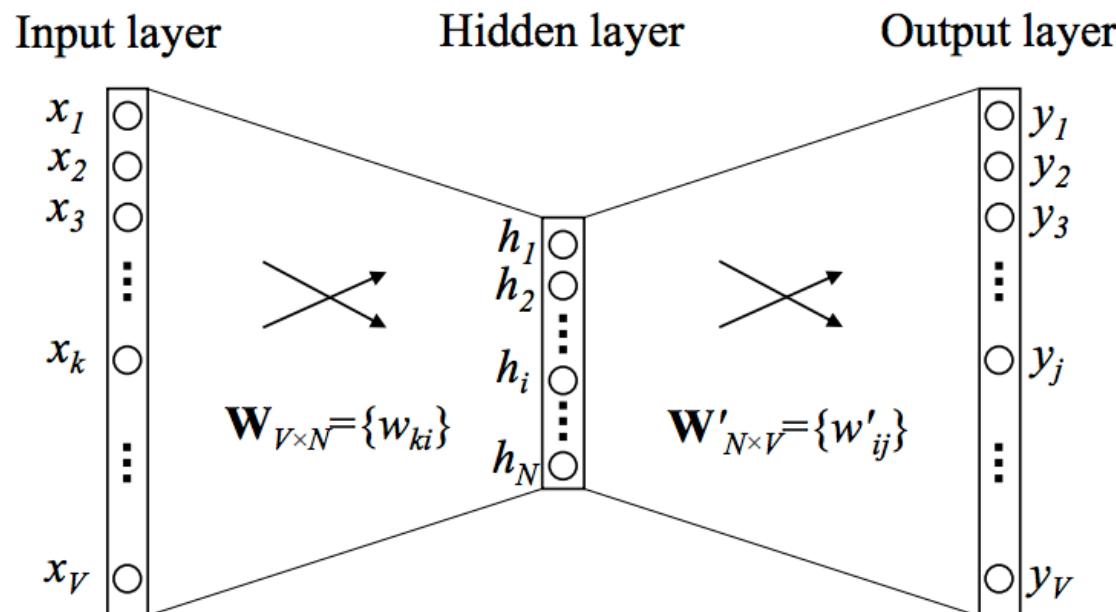


# Details of 1 word context CBOW

- Objective function: Maximize the log probability of a target word given a context word



# CBOW model (one context word)



$$\mathbf{h} = \mathbf{x}^T \mathbf{W} = \mathbf{W}_{(k, \cdot)} := \mathbf{v}_{w_I}, \quad \text{Word score} \quad u_j = \mathbf{v}'_{w_j}^T \cdot \mathbf{h}$$

$$p(w_j | w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^V \exp(u_{j'})} = \frac{\exp(\mathbf{v}'_{w_O}^T \mathbf{v}_{w_I})}{\sum_{j'=1}^V \exp(\mathbf{v}'_{w'_j}^T \mathbf{v}_{w_I})}$$

# CBOW model

---

Want to  
maximize

$$p(w_O | w_I) = \max y_{j^*}$$

$$= \max \log y_{j^*}$$

$$= u_{j^*} - \log \sum_{j'=1}^V \exp(u_{j'}) := -E$$

- Do this by differentiating wrt each variable and walking downhill to minimize  $E$ . Remember:

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

- Chain rule: If  $y = f(u)$  and  $u = g(x)$ , i.e.  $y = f(g(x))$ , then:

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$

# CBOW model

---

Want to  
maximize

$$\begin{aligned}
 p(w_O | w_I) &= \max y_{j^*} \\
 &= \max \log y_{j^*} \\
 &= u_{j^*} - \log \sum_{j'=1}^V \exp(u_{j'}) := -E
 \end{aligned}$$

$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j \quad \text{where } t_j = \mathbb{1}(j = j^*)$$

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

# CBOW model:

## Stochastic gradient descent (SGD) updates

---

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

$$w'_{ij}^{(\text{new})} = w'_{ij}^{(\text{old})} - \eta \cdot e_j \cdot h_i$$

where  $\eta > 0$  is the learning rate

$$\mathbf{v}'_{w_j}^{(\text{new})} = \mathbf{v}'_{w_j}^{(\text{old})} - \eta \cdot e_j \cdot \mathbf{h}$$

These are the standard form of SGD updates. You nudge each parameter a little in the negative direction of gradient to go downhill towards the minimum

# CBOW model: W matrix

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^V \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^V e_j \cdot w'_{ij} := \text{EH}_i$$

$$h_i = \sum_{k=1}^V x_k \cdot w_{ki}$$

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \text{EH}_i \cdot x_k$$

$$\frac{\partial E}{\partial \mathbf{W}} = \mathbf{x} \cdot \text{EH} \quad \mathbf{v}_{w_I}^{(\text{new})} = \mathbf{v}_{w_I}^{(\text{old})} - \eta \cdot \text{EH}$$

# Training regime

- Start with small, random vectors for words
- Iteratively go through millions of words in contexts
  - Work out prediction, work out error
  - Backpropagate error to update word vectors
  - Repeat
- Result is dense vectors for all words

$$\text{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

# Linear Relationships in word2vec

These representations are *very good* at encoding **similarity** and **dimensions of similarity**!

- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

Syntactically

- $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
- Similarly for verb and adjective morphological forms

Semantically (Semeval 2012 task 2)

- $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
- $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

# Word Analogies

Test for linear relationships, examined by Mikolov et al.

a:b :: c:?



$$d = \arg \max_x \frac{(w_b - w_a + w_c)^T w_x}{\|w_b - w_a + w_c\|}$$

man:woman :: king:?

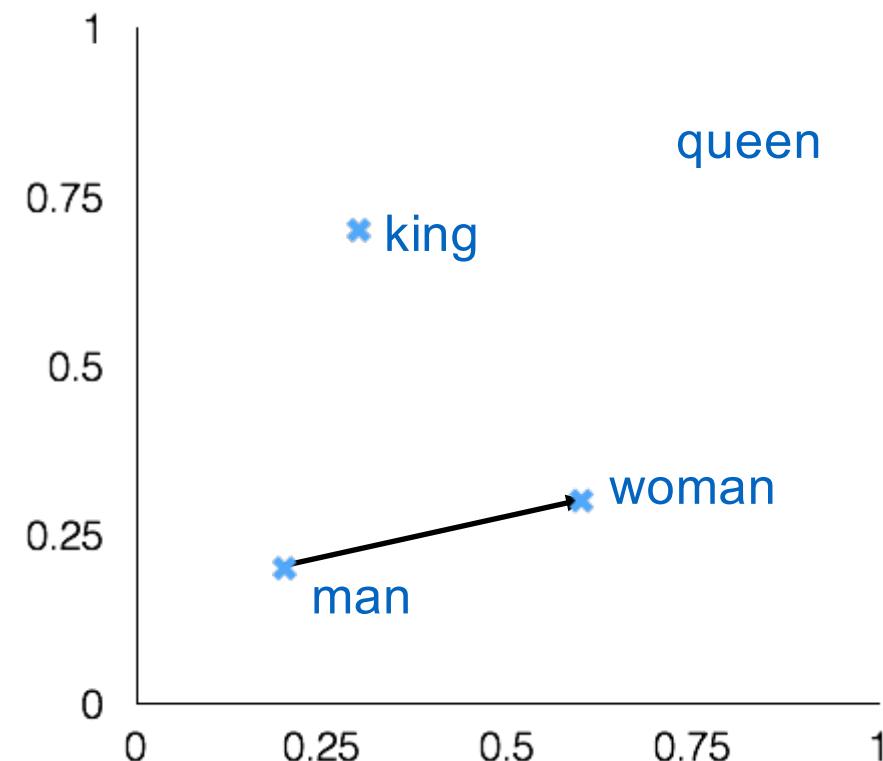
+ king [ 0.30 0.70 ]

- man [ 0.20 0.20 ]

+ woman [ 0.60 0.30 ]

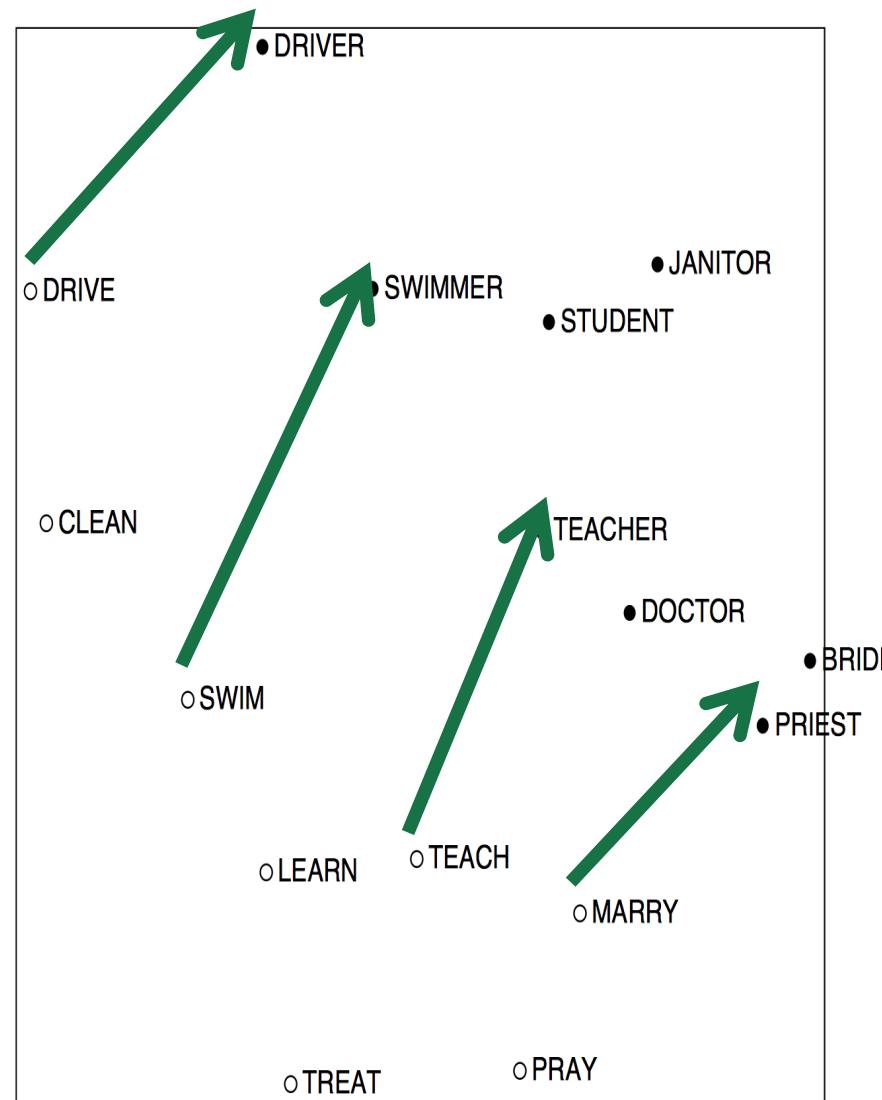
---

queen [ 0.70 0.80 ]



# COALS model (count-modified LSA)

[Rohde, Gonnerman & Plaut, ms., 2005]



# Count based vs. direct prediction

LSA, HAL (Lund & Burgess),  
COALS (Rohde et al),  
**Hellinger-PCA** (Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to small counts

• NNLM, HLBL, RNN, word2vec  
Skip-gram/CBOW, (Bengio et al;  
Collobert & Weston; Huang et al; Mnih &  
Hinton; Mikolov et al; Mnih & Kavukcuoglu)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

# Encoding meaning in vector differences

[Pennington, Socher, and Manning, EMNLP 2014]

**Crucial insight:** Ratios of co-occurrence probabilities can encode meaning components

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{random}$
$P(x \text{ice})$	large	small	large	small
$P(x \text{steam})$	small	large	large	small
$\frac{P(x \text{ice})}{P(x \text{steam})}$	large	small	~1	~1

# Encoding meaning in vector differences

[Pennington, Socher, and Manning, EMNLP 2014]

Crucial insight:

Ratios of co-occurrence probabilities can encode meaning components

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{fashion}$
$P(x \text{ice})$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(x \text{steam})$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$\frac{P(x \text{ice})}{P(x \text{steam})}$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

# GloVe: A new model for learning word representations

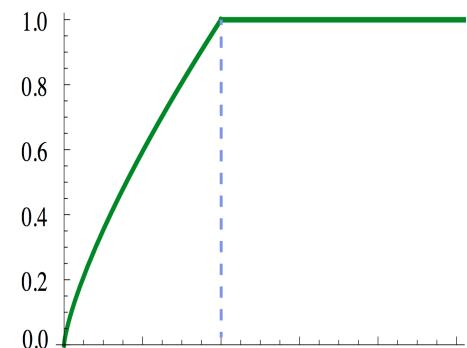
[Pennington, Socher, and Manning, EMNLP 2014]

$$w_i \cdot w_j = \log P(i|j)$$



$$w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$$

$$J = \sum_{i,j=1}^V f(X_{ij}) \left( w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2 \quad f \sim$$



# Word similarities

Nearest words to **frog**:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus



litoria



leptodactylidae



rana

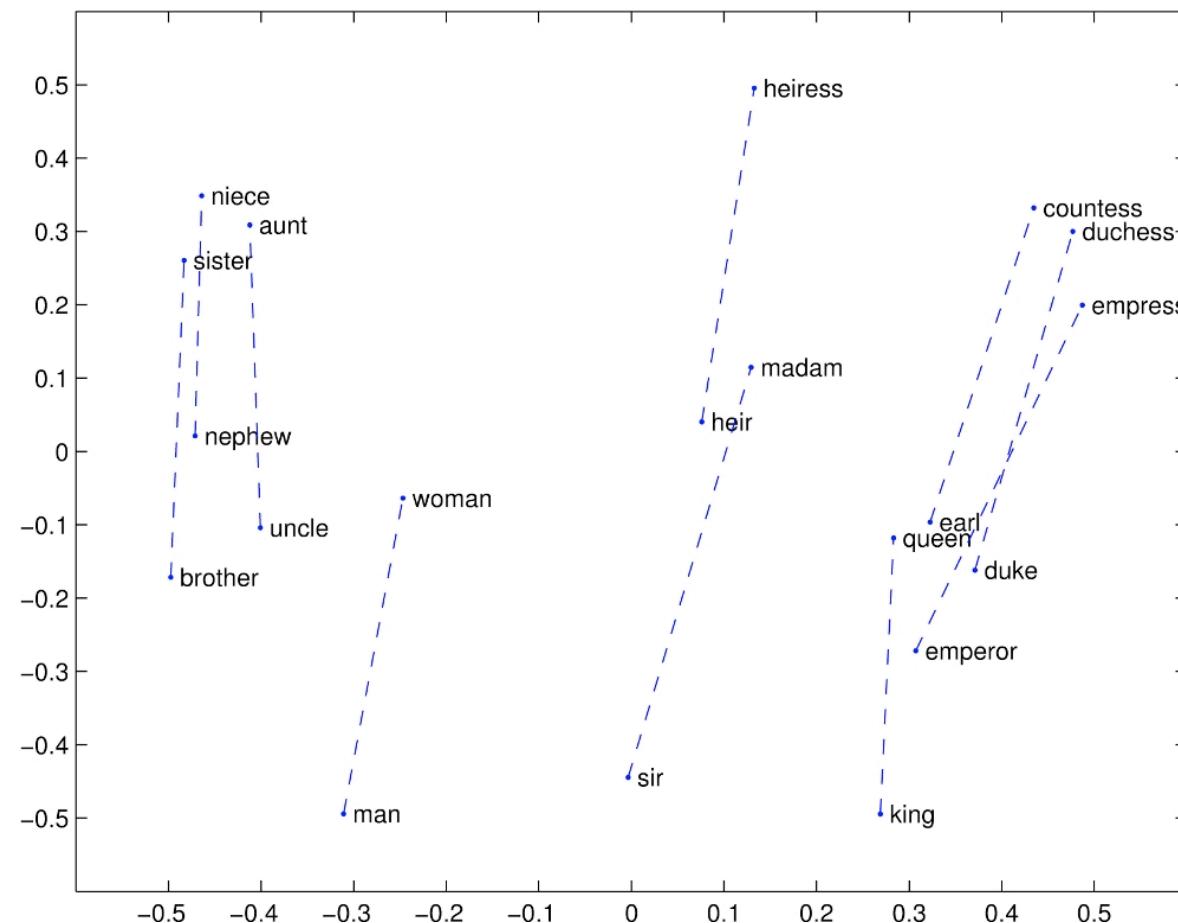


eleutherodactylus

# Word analogy task [Mikolov, Yih & Zweig 2013a]

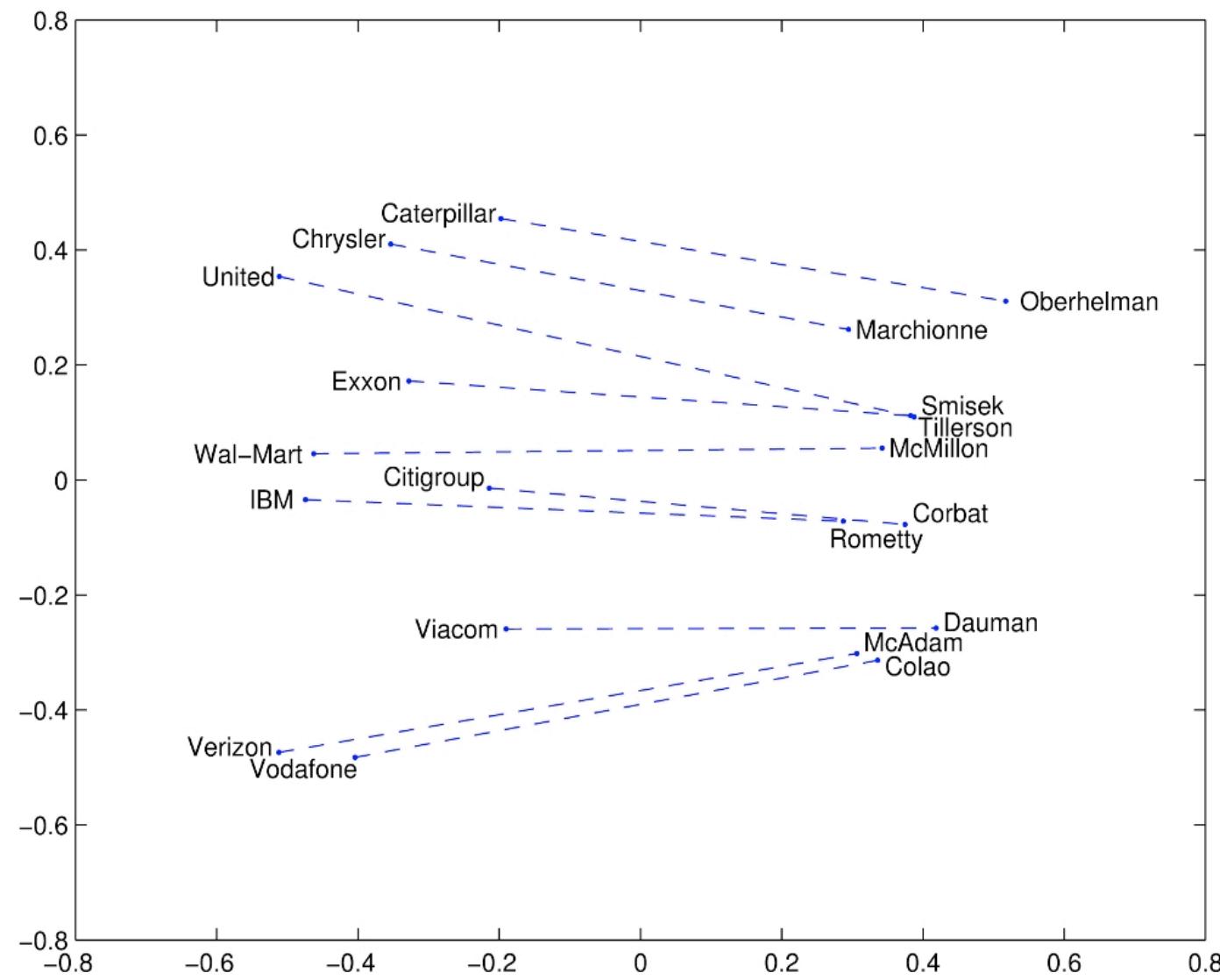
Model	Dimensions	Corpus size	Performance (Syn + Sem)
CBOW (Mikolov et al. 2013b)	300	1.6 billion	36.1

# GloVe Visualizations

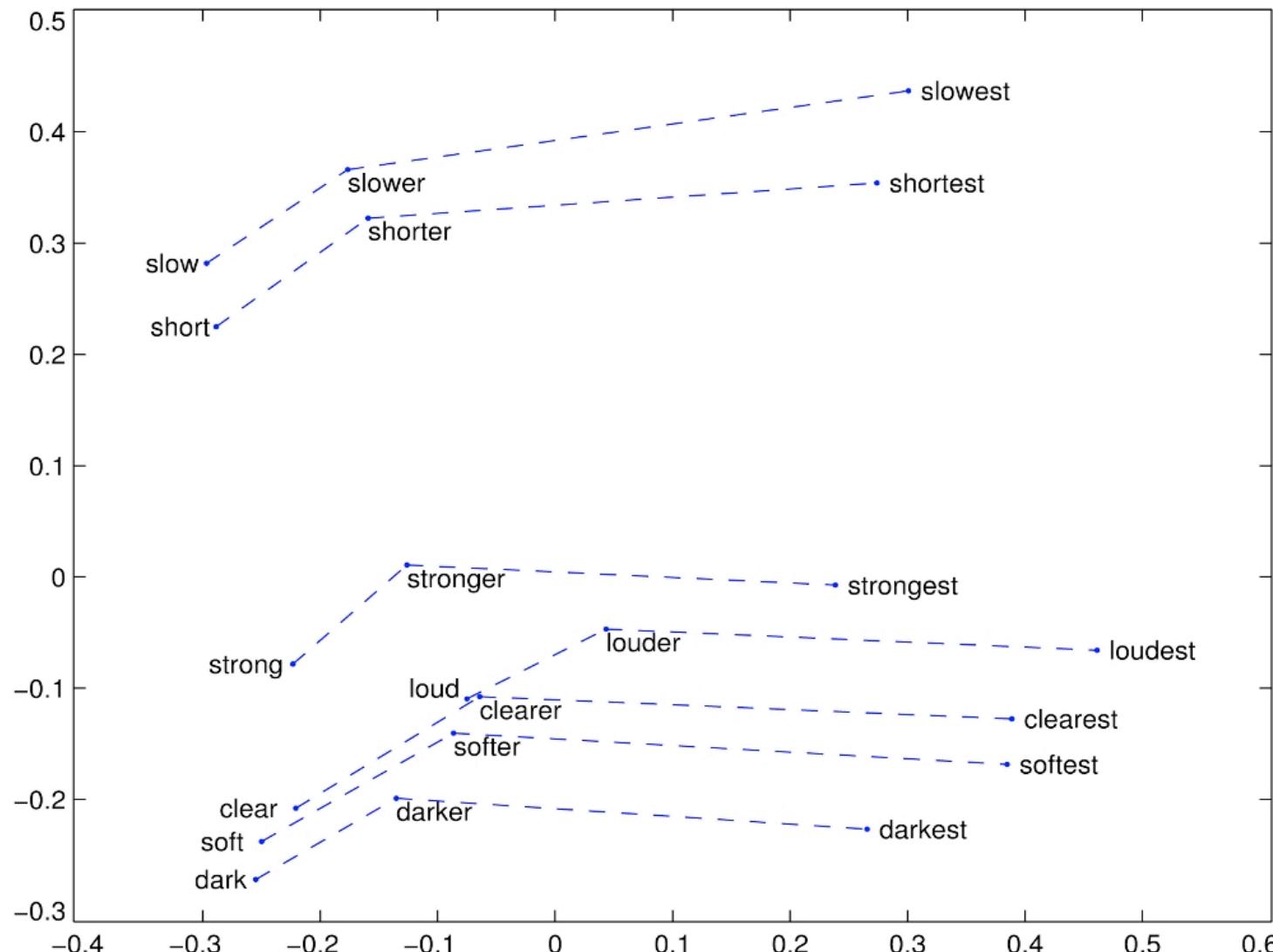


<http://nlp.stanford.edu/projects/glove/>

# Glove Visualizations: Company - CEO



# Glove Visualizations: Superlatives



# Application to Information Retrieval

Application is just beginning – there's almost nothing to go on

- Google's RankBrain – almost nothing is publicly known
  - Bloomberg article by Jack Clark (Oct 26, 2015):  
<http://www.bloomberg.com/news/articles/2015-10-26/google-turning-its-lucrative-web-search-over-to-ai-machines>
- SIGIR workshop this summer!



- Early paper from Bing seems most interesting for now

# An application to information retrieval

---

Nalisnick, Mitra, Craswell & Caruana. 2016. Improving Document Ranking with Dual Word Embeddings. *WWW 2016 Companion*.

<http://research.microsoft.com/pubs/260867/pp1291-Nalisnick.pdf>

Mitra, Nalisnick, Craswell & Caruana. 2016. A Dual Embedding Space Model for Document Ranking. [arXiv:1602.01137 \[cs.IR\]](https://arxiv.org/abs/1602.01137)

Builds on BM25 model idea of “aboutness”

- Not just term repetition indicating aboutness
- Relationship between query terms and *all* terms in the document indicates aboutness (*BM25 uses only query terms*)

Makes clever argument for different use of word and context vectors in word2vec’s CBOW/SGNS or GloVe

# Modeling document aboutness: Results from a search for Albuquerque

$d_1$

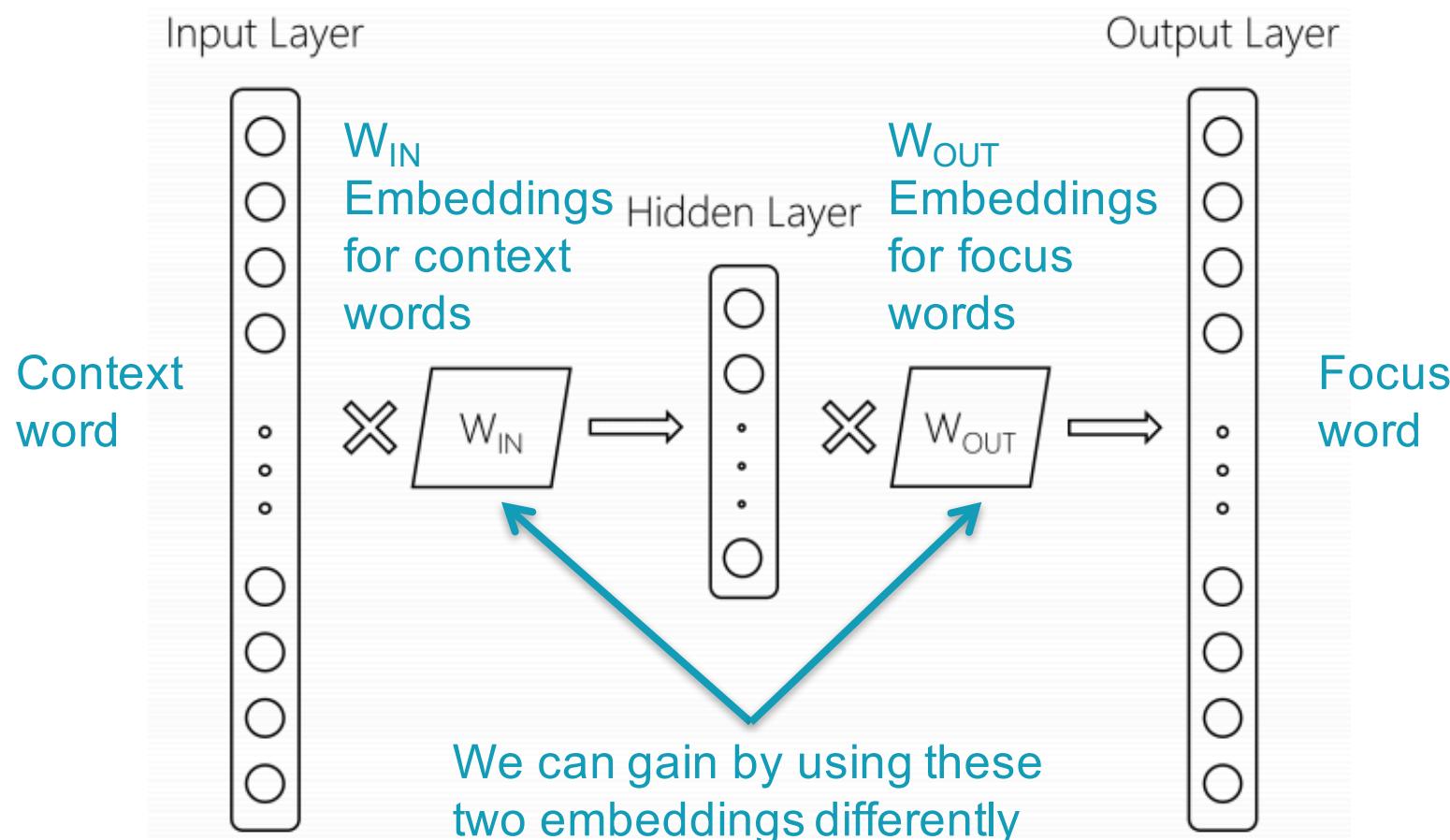
*Allen suggested that they could program a BASIC interpreter for the device; after a call from Gates claiming to have a working interpreter, MITS requested a demonstration. Since they didn't actually have one, Allen worked on a simulator for the Altair while Gates developed the interpreter. Although they developed the interpreter on a simulator and not the actual device, the interpreter worked flawlessly when they demonstrated the interpreter to MITS in Albuquerque, New Mexico in March 1975; MITS agreed to distribute it, marketing it as Altair BASIC.*

$d_2$

*Albuquerque is the most populous city in the U.S. state of New Mexico. The high-altitude city serves as the county seat of Bernalillo County, and it is situated in the central part of the state, straddling the Rio Grande. The city population is 557,169 as of the July 1, 2014, population estimate from the United States Census Bureau, and ranks as the 32nd-largest city in the U.S. The Metropolitan Statistical Area (or MSA) has a population of 902,797 according to the United States Census Bureau's most recently available estimate for July 1, 2013.*

# Using 2 word embeddings

CBOW model with 1 word of context



# Using 2 word embeddings

yale		seahawks	
IN-IN	IN-OUT	IN-IN	IN-OUT
yale	yale	seahawks	seahawks
harvard	faculty	49ers	highlights
nyu	alumni	broncos	jerseys
cornell	orientation	packers	tshirts
tulane	haven	nfl	seattle
tufts	graduate	steelers	hats

# Dual Embedding Space Model (DESM)

- Simple model
- A document is represented by the centroid of its word vectors

$$\overline{\mathbf{D}} = \frac{1}{|D|} \sum_{\mathbf{d}_j \in D} \frac{\mathbf{d}_j}{\|\mathbf{d}_j\|}$$

- Query-document similarity is average over query words of cosine similarity

$$DESM(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{\mathbf{q}_i^T \overline{\mathbf{D}}}{\|\mathbf{q}_i\| \|\overline{\mathbf{D}}\|}$$

# Dual Embedding Space Model (DESM)

- What works best is to use the OUT vectors for the document and the IN vectors for the query

$$DESM_{IN-OUT}(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{q_{IN,i}^T \overline{D_{OUT}}}{\|q_{IN,i}\| \|D_{OUT}\|}$$

- This way similarity measures *aboutness* – words that appear with this word – which is more useful in this context than (*distributional*) *semantic similarity*

# Experiments

---

- Train CBOW from either
  - 600 million Bing queries
  - 342 million web document sentences
- Test on 7,741 randomly sampled Bing queries
  - 5 level eval (Perfect, Excellent, Good, Fair, Bad)
- Two approaches
  1. Use DESM model to rerank top results from BM25
  2. Use DESM alone or a mixture model of it and BM25

$$MM(Q, D) = \alpha DESM(Q, D) + (1 - \alpha) BM25(Q, D)$$
$$\alpha \in \mathbb{R}, 0 \leq \alpha \leq 1$$

# Results – reranking $k$ -best list

	Explicitly Judged Test Set		
	NDCG@1	NDCG@3	NDCG@10
BM25	23.69	29.14	44.77
LSA	22.41*	28.25*	44.24*
DESM (IN-IN, trained on body text)	23.59	29.59	45.51*
DESM (IN-IN, trained on queries)	23.75	29.72	46.36*
DESM (IN-OUT, trained on body text)	24.06	30.32*	46.57*
DESM (IN-OUT, trained on queries)	<b>25.02*</b>	<b>31.14*</b>	<b>47.89*</b>

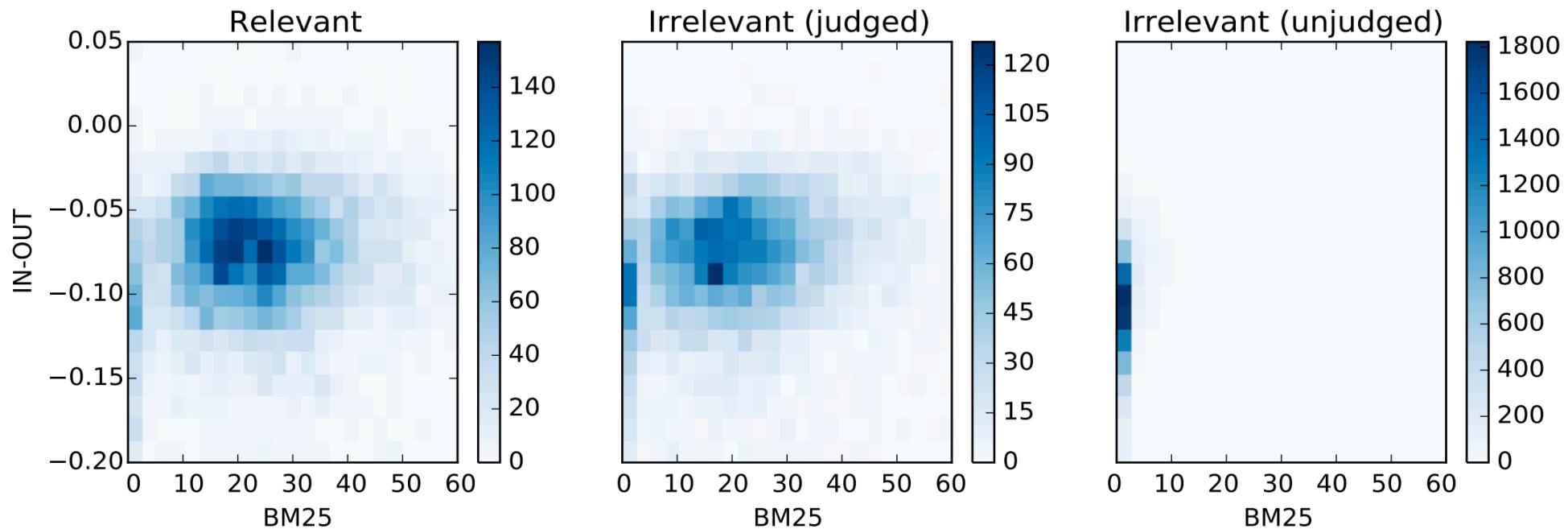
Pretty decent gains – e.g., 2% for NDCG@3

Gains are bigger for model trained on queries than docs

# Results – whole ranking system

	Explicitly Judged Test Set		
	NDCG@1	NDCG@3	NDCG@10
BM25	21.44	26.09	37.53
LSA	04.61*	04.63*	04.83*
DESM (IN-IN, trained on body text)	06.69*	06.80*	07.39*
DESM (IN-IN, trained on queries)	05.56*	05.59*	06.03*
DESM (IN-OUT, trained on body text)	01.01*	01.16*	01.58*
DESM (IN-OUT, trained on queries)	00.62*	00.58*	00.81*
BM25 + DESM (IN-IN, trained on body text)	21.53	26.16	37.48
BM25 + DESM (IN-IN, trained on queries)	<b>21.58</b>	26.20	37.62
BM25 + DESM (IN-OUT, trained on body text)	21.47	26.18	37.55
BM25 + DESM (IN-OUT, trained on queries)	21.54	<b>26.42*</b>	<b>37.86*</b>

# A possible explanation

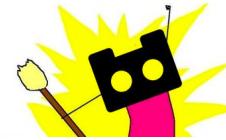


IN-OUT has some ability to prefer Relevant to close-by (judged) non-relevant, but its scores induce too much noise vs. BM25 to be usable alone

# DESM conclusions

---

- DESM is a weak ranker but effective at finding subtler similarities/aboutness
- It is effective at, but only at, ranking at least somewhat relevant documents



# Summary: Embed all the things!

Word embeddings are the hot new technology (again!)

Lots of applications whenever knowing word context or similarity helps prediction:

- Synonym handling in search
- Document aboutness
- Ad serving
- Language models: from spelling correction to email response
- Machine translation
- Sentiment analysis
- ...