Introduction to

Information Retrieval

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

Lecture 13: 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

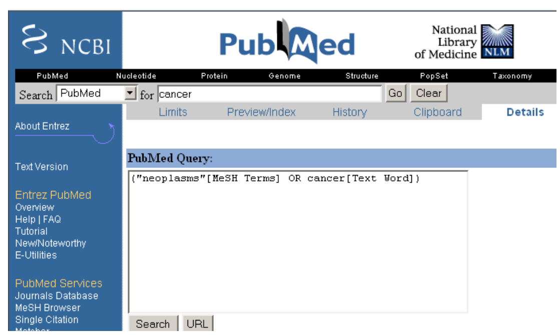
But we'd like to better **understand** when query/document match

---

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



Search log query expansion

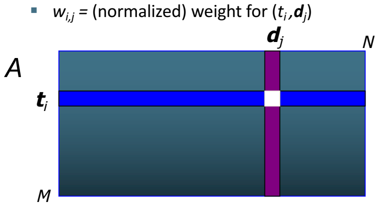
- Context-free query expansion ends up problematic
  - [light hair] ≈ [fair hair] At least in U.K./Australia? = blonde
    - So expand [light] ⇒ [light fair]
  - But [outdoor light price] ≠ [outdoor fair price]
- You can learn query context-specific rewritings from search logs by attempting to identify the same user making a second attempt at the same user need
  - [Hinton word vector]
  - [Hinton word embedding]
- In this context, [vector] ≈ [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. Why?

Simple 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 $(t_i, \boldsymbol{d}_j)$



- 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 … sort of works

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

But data is too sparse in this form 100,000 words = 10^{10} entries in C.

How can we represent term relations?

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

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 $q^T W d$?

"search ranking" "information retrieval ranking"

$$\underset{\text{se in re ra or}}{\overset{q^T}{\begin{pmatrix} 1 & 0 & 0 & 1 & 0 \end{pmatrix}}} \overset{W}{\begin{pmatrix} 1 & 0.7 & 0.5 & 0 & 0 \\ 0.3 & 1 & 0.2 & 0 & 0 \\ 0.5 & 0.2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0.7 \\ 0 & 0 & 0 & 0.7 & 1 \end{pmatrix}} \overset{d}{\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}} \begin{array}{l} \text{se} \\ \text{in} \\ \text{re} \\ \text{ra} \\ \text{or(dering)} \end{array} = 2.2$$

- Problem is again sparsity – W is huge > 10^{10}

Is there a better way?

- Idea:
  - Can we learn a dense 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 or orthogonal vectors
- Theory is that similarity is preserved as much as possible
- You can actually gain in IR (slightly) 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)
  - Hard 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"

Word meaning is defined in terms of vectors

- We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

  … those other words also being represented by vectors … it all gets a bit recursive

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

Neural word embeddings - visualization

Basic idea of learning neural network word embeddings

We define a model that aims to predict between a center word w_t and context words in terms of word vectors

$p(\text{context} \mid w_t) = \dots$

which has a loss function, e.g.,

$J = 1 - p(w_{-t} \mid w_t)$

We look at many positions *t* in a big language corpus

We keep adjusting the vector representations of words to minimize this loss

Idea: Directly learn low-dimensional word vectors based on ability to predict

- 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
  - The GloVe model from Stanford (Pennington, Socher, and Manning 2014) connects back to matrix factorization
- Initial models were quite non-linear and slow; recent work has used fast, bilinear models

Word2vec is a family of algorithms

[Mikolov et al. 2013]

Predict between every word and its context words!

Two algorithms

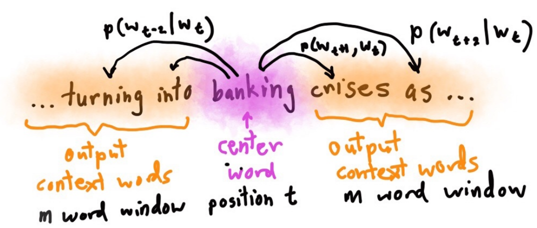
1. **Skip-grams (SG)**
   Predict context words given target (position independent)
2. Continuous Bag of Words (CBOW)
   Predict target word from bag-of-words context

Two (moderately efficient) training methods

1. Hierarchical softmax
2. Negative sampling

Naïve softmax

Skip-gram prediction



Details of word2vec

For each word $t = 1 \ldots T$, predict surrounding words in a window of "radius" m of every word.

Objective function: Maximize the probability of any context word given the current center word:

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} p(w_{t+j} | w_t ; \theta)$$

Negative Log Likelihood $$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log p(w_{t+j} | w_t)$$

Where θ represents all variables we will optimize

Details of Word2Vec

Predict surrounding words in a window of radius m of every word

For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

where o is the outside (or output) word index, c is the center word index, v_c and u_o are "center" and "outside" vectors of indices c and o

Softmax using word c to obtain probability of word o

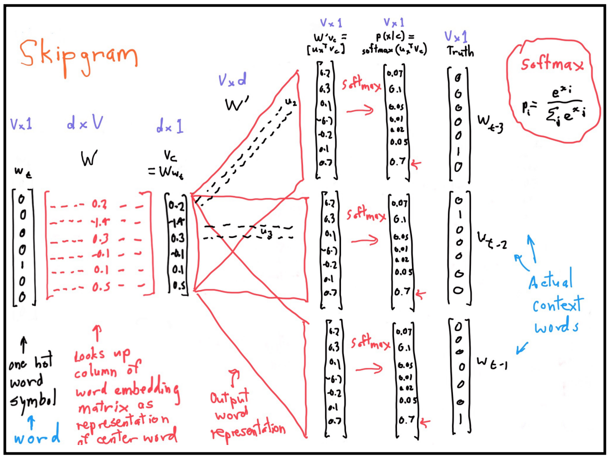
Softmax function: Standard map from $\mathbb{R}^V$ to a probability distribution

Softmax

Exponentiate to make positive

$$p_i = \frac{e^{u_i}}{\sum_j e^{u_j}}$$

Normalize to give probability



To learn good word vectors: Compute all vector gradients!

- We often define the set of **all** parameters in a model in terms of one long vector θ
- In our case with *d*-dimensional vectors and *V* many words:

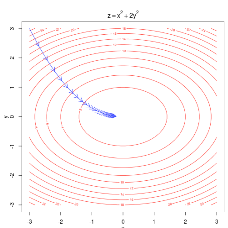
$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

- We then optimize these parameters

Note: Every word has two vectors! Makes it simpler!

Intuition of how to minimize loss for a simple function over two parameters

We start at a random point and walk in the steepest direction, which is given by the derivative of the function

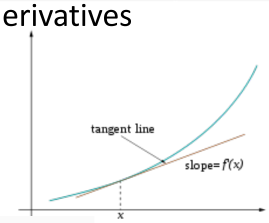


Contour lines show points of equal value of objective function

Descending by using derivatives

We will minimize a cost function by gradient descent

Trivial example: (from Wikipedia)
Find a local minimum of the function $f(x) = x^4 - 3x^3 + 2$,
with derivative $f'(x) = 4x^3 - 9x^2$




```
x_old = 0
x_new = 6 # The algorithm starts at x=6
eps = 0.01 # step size
precision = 0.00001

def f_derivative(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_derivative(x_old)

print("Local minimum occurs at", x_new)
```

Subtracting a fraction of the gradient moves you towards the minimum!

Vanilla Gradient Descent Code

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```


Stochastic Gradient Descent

- But Corpus may have 40B tokens and windows
- You would wait a very long time before making a single update!
- **Very** bad idea for pretty much all neural nets!
- Instead: We will update parameters after each window t
  → Stochastic gradient descent (SGD)

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$$

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

Introduction to Information Retrieval

Working out how to optimize a neural network is really all the chain rule!

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

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = \frac{df(u)}{du}\frac{dg(x)}{dx}$$

Simple example: $\frac{dy}{dx} = \frac{d}{dx}5(x^3+7)^4$

$y = f(u) = 5u^4$ $\qquad$ $u = g(x) = x^3 + 7$

$\frac{dy}{du} = 20u^3$ $\qquad$ $\frac{du}{dx} = 3x^2$

$$\frac{dy}{dx} = 20(x^3+7)^3 . 3x^2$$

Objective Function

Maximize $J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} p(w'_{t+j} | w_t; \theta)$

Or minimize neg. log likelihood $J(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log p(w'_{t+j} | w_t)$

[negate to minimize; log is monotone]

Text length

window size

where $p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$

word IDs

Each word type (vocab entry) has two word representations: as center word and context word

We now take derivatives to work out minimum

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w v_c)}$$

$$= \underbrace{\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c)}_{①} - \underbrace{\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c)}_{②}$$

① $\frac{\partial}{\partial v_c} \underbrace{\log \exp}_{\text{inverses}}(u_o^T v_c) = \frac{\partial}{\partial v_c} u_o^T v_c = u_o$

Vector! Not high school single variable calculus

You can do things one variable at a time, and this may be helpful when things get gnarly.

$\forall j \quad \frac{\partial}{\partial (v_c)_j} u_o^T v_c = \frac{\partial}{\partial (v_c)_j} \sum_{i=1}^{d} (u_o)_i (v_c)_i$

$= (u_o)_j$

Each term is zero except when $i = j$

---

② $\frac{\partial}{\partial v_c} \underbrace{\log}_{f} \underbrace{\sum_{w=1}^{V} \exp(u_w^T v_c)}_{z = g(v_c)}$

$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \frac{\partial}{\partial v_c} \sum_{x=1}^{V} \exp(u_x^T v_c)$ ← Important to change index

$\frac{\partial}{\partial v_c} f(g(v_c)) = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial v_c}$ $\qquad$ Use chain rule

$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \frac{\partial}{\partial v_c} \underbrace{\exp}_{f} \underbrace{(u_x^T v_c)}_{z = g(v_c)} \right)$ $\qquad$ Move deriv inside sum

$\left(\sum_{x=1}^{V} \exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c \right)$ $\qquad$ Chain rule

$\left(\sum_{x=1}^{V} \exp(u_x^T v_c) u_x \right)$

$$\frac{\partial}{\partial v_c} \log(p(o|c)) = u_o - \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \exp(u_x^T v_c) u_x \right)$$

$$= u_o - \sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} u_x$$ Distribute term across sum

$$= u_o - \underbrace{\sum_{x=1}^{V} p(x|c) u_x}$$

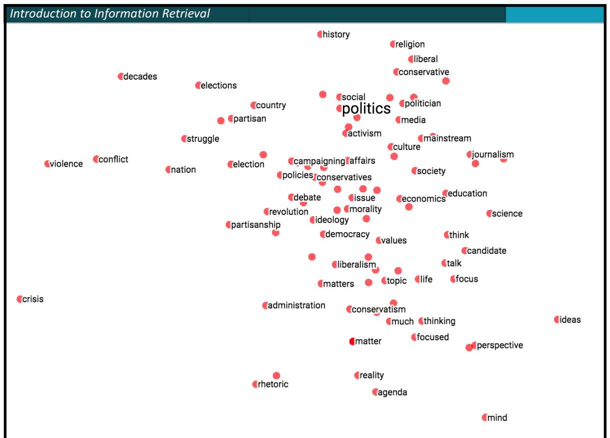
This is an expectation: average over all context vectors weighted by their probability

= observed − expected

This is just the derivatives for the center vector parameters
Also need derivatives for output vector parameters (they're similar)
Then we have derivative w.r.t. all parameters and can minimize

---

Introduction to Information Retrieval



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

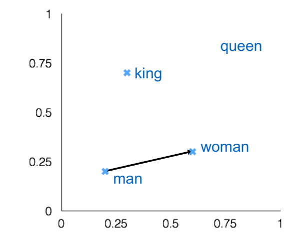
Word Analogies

Test for linear relationships, examined by Mikolov et al.

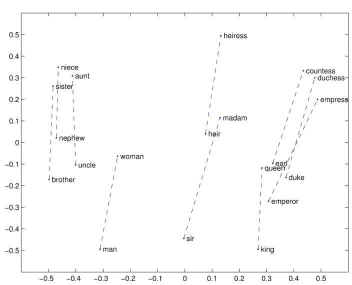
a:b :: c:? → $d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$

man:woman :: king:?

+	king	[0.30 0.70]
–	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]

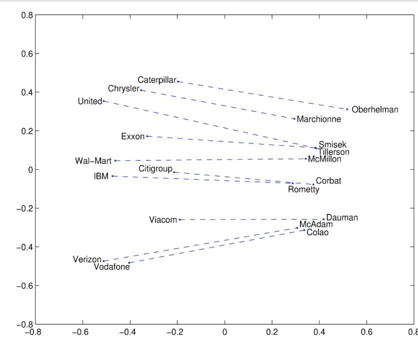


GloVe Visualizations

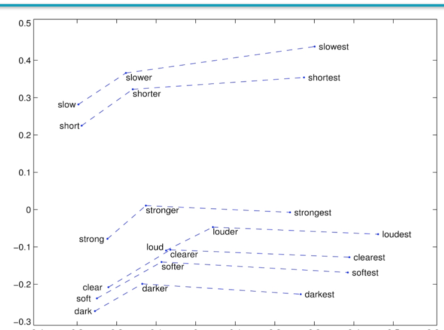


http://nlp.stanford.edu/projects/glove/

Glove Visualizations: Company - CEO



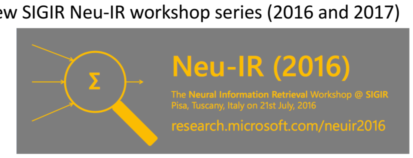
Glove Visualizations: Superlatives



Application to Information Retrieval

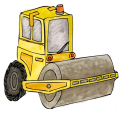
Application is just beginning – there's little to go on

- 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
  - A result reranking system
  - Even though more of the value is in the tail?
- New SIGIR Neu-IR workshop series (2016 and 2017)

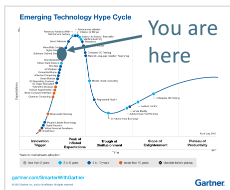


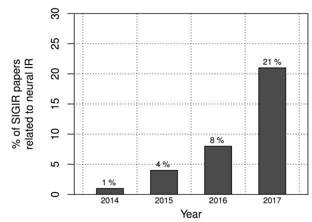
Final Thoughts

from Chris Manning SIGIR 2016 keynote



2011	2013	2015	2017
speech	vision	NLP	IR





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]**

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)

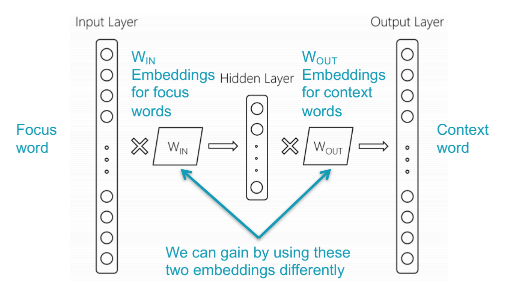
Makes clever argument for different use of word and context vectors in word2vec's CBOW/SGNS or GloVe

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



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

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

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

Experiments

- Train word2vec 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)	**25.02***	**31.14***	**47.89***

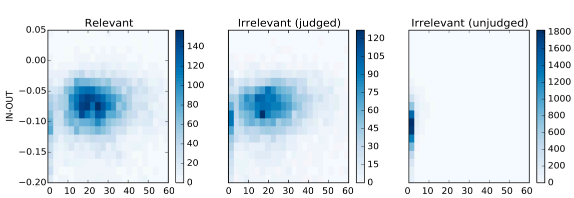
Pretty decent gains – e.g., 2% for NDCG@3

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)	**21.58**	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	**26.42***	**37.86***

A possible explanation



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

DESM conclusions

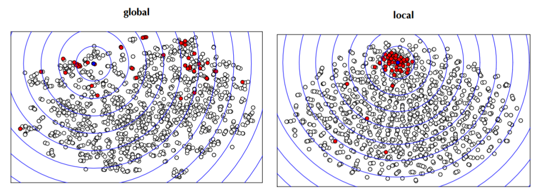
- DESM is a weak ranker but effective at finding subtler similarities/aboutness
- It is effective at, but only at, ranking at least somewhat relevant documents
  - For example, DESM can confuse Oxford and Cambridge
  - Bing rarely makes the Oxford-Cambridge mistake

Global vs. local embedding [Diaz 2016]

global	local
cutting	tax
squeeze	deficit
reduce	vote
slash	budget
reduction	reduction
spend	house
lower	bill
halve	plan
soften	spend
freeze	billion

Figure 3: Terms similar to 'cut' for a word2vec model trained on a general news corpus and another trained only on documents related to 'gasoline tax'.

Global vs. local embedding [Diaz 2016]



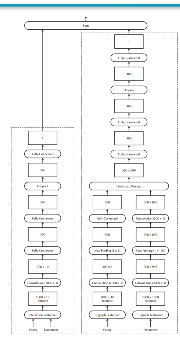
Train w2v on documents from first round of retrieval

Fine-grained word sense disambiguation

Figure 5: Global versus local embedding of highly relevant terms. Each point represents a candidate expansion term. Red points have high frequency in the relevant set of documents. White points have low or no frequency in the relevant set of documents. The blue point represents the query. Contours indicate distance from the query.

Ad-hoc retrieval using local and distributed representation [Mitra et al. 2017]

- Argues both "lexical" and "semantic" matching is important for document ranking
- Duet model is a linear combination of two DNNs using local and distributed representations of query/document as inputs, and jointly trained on labelled data



Summary: Embed all the things!

Word embeddings are the hot new technology (again!)

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

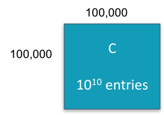
- Synonym handling in search
- Document aboutness
- Ad serving
- Language models: from spelling correction to email response
- Machine translation
- Sentiment analysis
- …

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

Automatic Thesaurus Generation Issues

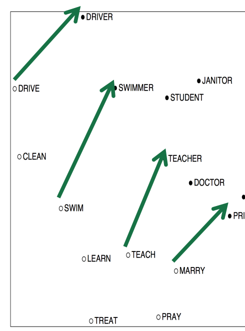
- Quality of associations is usually a problem
- Sparsity



- 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.

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

Encoding meaning in vector differences

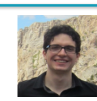
[Pennington, Socher, and Manning, EMNLP 2014]

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

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

GloVe: A new model for learning word representations

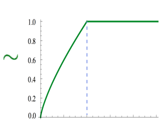
[Pennington, Socher, and Manning, EMNLP 2014]



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

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

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



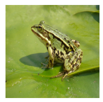
Word similarities

Nearest words to frog:

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

litoria

leptodactylidae



rana

eleutherodactylus

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

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