Introduction to Information Retrieval

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

Lecture 14: Distributed Word Representations for Information Retrieval
How can we more robustly match a user’s search intent?

We want to understand a 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 pure keyword-matching IR 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?

Query expansion:

- **Relevance feedback** could allow us to capture this if we get near enough to matching documents with these words.
- We can also use information on **word similarities**:
  - A manual **thesaurus** of synonyms for query expansion
  - A **measure of word similarity**
    - Calculated from a big document collection
    - Calculated by query log mining (common on the web)

Document expansion:

- 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
Example of manual thesaurus
Search log query expansion

- Context-free query expansion ends up problematic
  - [wet ground] ≈ [wet earth]
  - So expand [ground] ⇒ [ground earth]
  - But [ground coffee] ≠ [earth coffee]

- 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.
- $w_{i,j} = (\text{normalized}) \text{ weight for } (t_i, d_j)$

For each $t_i$, pick terms with high values in $C$.
Automatic thesaurus generation example ... sort of works

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

Too little data (10s of millions of words) treated by too sparse method. 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
- \[
  \text{motel} \begin{bmatrix}
    0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
  \end{bmatrix}^T
  \text{hotel} \begin{bmatrix}
    0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
  \end{bmatrix} = 0
\]
- If query on \textit{hotel} and document has \textit{motel}, then our query and document vectors are \textit{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 $q^T W d$?


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
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

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”
- Somewhat popular in the 1990s [Deerwester et al. 1990, etc.]
  - But 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
  - 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

\[
\text{banking} = \begin{pmatrix}
0.286 \\
0.792 \\
-0.177 \\
-0.107 \\
0.109 \\
-0.542 \\
0.349 \\
0.271
\end{pmatrix}
\]
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.
  
  \[ p(\text{context} | w_t) = \ldots \]

- which has a loss function, e.g.,

  \[ J = 1 - p(w_{-t} | w_t) \]

- 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: 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) \(\rightarrow\) intro now
- The GloVe model from Stanford (Pennington, Socher, and Manning 2014) connects back to matrix factorization
- Per-token representations: Deep contextual word representations: ELMo, ULMfit, BERT
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
3. **Naïve softmax**
Word2Vec Skip-gram Overview

- Example windows and process for computing $P(w_{t+j} | w_t)$

... problems turning into banking crises as ...

- $P(w_{t-2} | w_t)$
- $P(w_{t-1} | w_t)$
- $P(w_{t+1} | w_t)$
- $P(w_{t+2} | w_t)$

outside context words in window of size 2
center word at position $t$
outside context words in window of size 2
Word2vec: objective function

For each position \( t = 1, \ldots, T \), predict context words within a window of fixed size \( m \), given center word \( w_j \).

 Likelihood =

\[
L(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} | w_t; \theta)
\]

\( \theta \) is all variables to be optimized

Sometimes called cost or loss function

The objective function \( J(\theta) \) is the (average) negative log likelihood:

\[
J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} | w_t; \theta)
\]

Minimizing objective function \( \iff \) Maximizing predictive accuracy
Word2vec: objective function

- We want to minimize the objective function:

\[
J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} | w_t; \theta)
\]

- **Question**: How to calculate \( P(w_{t+j} | w_t; \theta) \) ?

- **Answer**: We will *use two* vectors per word \( w \):
  - \( v_w \) when \( w \) is a center word
  - \( u_w \) when \( w \) is a context word

- Then for a center word \( c \) and a context word \( o \):

\[
P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}
\]
**Word2vec: prediction function**

Exponentiation makes anything positive

\[
P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in \mathcal{V}} \exp(u_w^T v_c)}
\]

Dot product compares similarity of \(o\) and \(c\).

\[
u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i
\]

Larger dot product = larger probability

Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** \(\mathbb{R}^n \to (0,1)^n\)

\[
\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i
\]

- The softmax function maps arbitrary values \(x_i\) to a probability distribution \(p_i\)
  - “max” because amplifies probability of largest \(x_i\)
  - “soft” because still assigns some probability to smaller \(x_i\)
  - Frequently used in neural networks/Deep Learning

Open region
Word2vec: 2 matrices of parameters

Center word embeddings as rows

Context word embeddings as columns

$t_i \times W_{in} \rightarrow \ldots \times W_{out} \rightarrow t_{i+j}$

(Transposed!)
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 $\theta$
- In our case with $d$-dimensional vectors and $V$ many words:
- 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 \( f'(x) = 4x^3 - 9x^2 \)

```python
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

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

```python
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 update parameters after each window \( t \)
  \( \rightarrow \) Stochastic gradient descent (SGD)

\[
\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla_{\theta} J_t(\theta)
\]

**while** True:

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

\[
\frac{dy}{dx} = \frac{d}{dx} 5(x^3 + 7)^4
\]

\[
y = f(u) = 5u^4 \quad \text{and} \quad u = g(x) = x^3 + 7
\]

\[
\frac{dy}{du} = 20u^3 \quad \text{and} \quad \frac{du}{dx} = 3x^2
\]

\[
\frac{dy}{dx} = 20(x^3 + 7)^3 \cdot 3x^2
\]
Objective Function
Maximize $J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \quad j \neq 0 \quad w_t \neq w_t}} p(w_t, \theta)$

Or minimize
neg. log likelihood
$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \quad j \neq 0 \quad w_t \neq w_t}} \log p(w_t^j | w_t)$

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

We now take derivatives to work out minimum.

Each word type (vocabulary entry) has two word representations: as center word and context word.
\[
\frac{\partial}{\partial v_c} \log \frac{\exp (u_0^T v_c)}{\sum_{w=1}^{V} \exp (u_w^T v_c)} \\
= \frac{\partial}{\partial v_c} \log \exp (u_0^T v_c) - \frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp (u_w^T v_c)
\]

1. \[\frac{\partial}{\partial v_c} \log \exp (u_0^T v_c) = \frac{\partial}{\partial v_c} u_0^T v_c = u_0\]

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

\[\forall j \frac{\partial}{\partial (v_c)_j} u_0^T v_c = \frac{\partial}{\partial (v_c)_j} \sum_{i=1}^{V} (u_0)_i (v_c)_i \]
\[= (u_0)_j \]

Each term is zero except when \(i = j\)
\[ \frac{\partial}{\partial v_c} \log \frac{\sum_{w=1}^{v} \exp (u_w^T v_c)}{\sum_{w=1}^{v} \exp (u_w^T v_c)} \]

\[ = \frac{1}{\sum_{w=1}^{v} \exp (u_w^T v_c)} \sum_{w=1}^{v} \exp (u_w^T v_c) \cdot \frac{\partial}{\partial v_c} \sum_{x=1}^{v} \exp (u_x^T v_c) \]

\[ \frac{\partial}{\partial v_c} f(g(v_c)) = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial v_c} \]

\[ = \frac{1}{\sum_{w=1}^{v} \exp (u_w^T v_c)} \sum_{w=1}^{v} \exp (u_w^T v_c) \cdot \frac{\partial}{\partial v_c} \underbrace{\sum_{x=1}^{v} \exp (u_x^T v_c)}_{f(z=g(v_c))} \]

\[ \left( \sum_{x=1}^{v} \frac{\partial}{\partial v_c} \exp (u_x^T v_c) \right) \frac{\partial}{\partial v_c} u_x^T v_c \]

\[ \left( \sum_{x=1}^{v} \exp (u_x^T v_c) \right) \frac{\partial}{\partial v_c} \exp (u_x^T v_c) u_x \]
\[ \frac{d}{dv_c} \log(p(o|c)) = u_0 - \sum_{x=1}^{V} \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \exp(u_x^T v_c) u_x \]

\[ = u_0 - \sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} u_x \]

\[ = u_0 - \sum_{x=1}^{V} p(x|c) u_x \]

\[ = \text{observed} - \text{expected} \]

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.
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.

\[ a:b :: c:? \]

\[ \text{man:woman :: king:?} \]

\[ + \quad \text{king} \quad [0.30 \ 0.70] \]
\[ - \quad \text{man} \quad [0.20 \ 0.20] \]
\[ + \quad \text{woman} \quad [0.60 \ 0.30] \]

\[ \text{queen} \quad [0.70 \ 0.80] \]

\[ d = \arg \max_x \frac{(w_b - w_a + w_c)^T w_x}{\|w_b - w_a + w_c\|} \]
GloVe Visualizations

http://nlp.stanford.edu/projects/glove/
Glove Visualizations: Company - CEO
Glove Visualizations: Superlatives
Application to Information Retrieval

Application is just beginning – we’re “at the end of the early years”

- Google’s RankBrain – little is publicly known
  - Bloomberg article by Jack Clark (Oct 26, 2015):
  - A result reranking system. “3rd most valuable ranking signal”
  - But note: more of the potential value is in the tail?
- New SIGIR Neu-IR workshop series (2016 on)
An application to information retrieval


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

word2vec model with 1 word of context

We can gain by using these two embeddings differently.
## Using 2 word embeddings

<table>
<thead>
<tr>
<th></th>
<th><strong>yale</strong></th>
<th></th>
<th><strong>seahawks</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IN-IN</strong></td>
<td>yale</td>
<td><strong>IN-IN</strong></td>
<td>seahawks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>harvard</td>
<td>faculty</td>
<td>49ers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nyu</td>
<td>alumni</td>
<td>broncos</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cornell</td>
<td>orientation</td>
<td>packers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tulane</td>
<td>haven</td>
<td>nfl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tufts</td>
<td>graduate</td>
<td>steelers</td>
<td></td>
</tr>
<tr>
<td><strong>IN-OUT</strong></td>
<td>yale</td>
<td><strong>IN-OUT</strong></td>
<td>seahawks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>faculty</td>
<td>highlights</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>alumni</td>
<td>jerseys</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>orientation</td>
<td>tshirts</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>haven</td>
<td>seattle</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>graduate</td>
<td>hats</td>
<td></td>
</tr>
</tbody>
</table>
Dual Embedding Space Model (DESM)

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

\[ \overline{D} = \frac{1}{|D|} \sum_{d_j \in D} \frac{d_j}{\|d_j\|} \]

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

\[ DESM(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{q_i^T \overline{D}}{\|q_i\| \|\overline{D}\|} \]
Dual Embedding Space Model (DESM)

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

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

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

- Train 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

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

Pretty decent gains – e.g., 2% for NDCG@$3$
Gains are bigger for model trained on queries than docs
# Results – whole ranking system

<table>
<thead>
<tr>
<th></th>
<th>Explicitly Judged Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NDCG@1</td>
</tr>
<tr>
<td>BM25</td>
<td>21.44</td>
</tr>
<tr>
<td>LSA</td>
<td>04.61*</td>
</tr>
<tr>
<td>DESM (IN-IN, trained on body text)</td>
<td>06.69*</td>
</tr>
<tr>
<td>DESM (IN-IN, trained on queries)</td>
<td>05.56*</td>
</tr>
<tr>
<td>DESM (IN-OUT, trained on body text)</td>
<td>01.01*</td>
</tr>
<tr>
<td>DESM (IN-OUT, trained on queries)</td>
<td>00.62*</td>
</tr>
<tr>
<td>BM25 + DESM (IN-IN, trained on body text)</td>
<td>21.53</td>
</tr>
<tr>
<td>BM25 + DESM (IN-IN, trained on queries)</td>
<td><strong>21.58</strong></td>
</tr>
<tr>
<td>BM25 + DESM (IN-OUT, trained on body text)</td>
<td>21.47</td>
</tr>
<tr>
<td>BM25 + DESM (IN-OUT, trained on queries)</td>
<td>21.54</td>
</tr>
</tbody>
</table>
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, reranking at least somewhat relevant documents
  - For example, DESM can confuse Oxford and Cambridge
  - Bing rarely makes an Oxford/Cambridge mistake!
What else can neural nets do in IR?

- Use a neural network as a supervised reranker
- Assume a query and document embedding network (as we have discussed)
- Assume you have \((q,d,rel)\) relevance data
- Learn a neural network (with supervised learning) to predict relevance of \((q,d)\) pair
- An example of “machine-learned relevance”, which we’ll talk about more next lecture
What else can neural nets do in IR?

- A deep transformer-based neural network
- Builds per-token (in context) representations
- Produces a query/document representation as well
- Or jointly embed query and document and ask for a retrieval score
- Incredibly effective!

https://arxiv.org/abs/1810.04805
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
- ...

<EMBED> ALL THE THINGS