Outline Today

• Distributed representations
• “Global” word embeddings, word2vec
  • Particularly the skip-gram model with negative sampling
• “Direct” applications of word embeddings in IR
• “Local” (query-biased) word embeddings
Word representation

Local representation* (only considering the word itself)
- Each word is a unique and orthogonal dimension
- Techniques are mainly counting term frequencies …
- Example: n-gram language models

Problems
- Difficult to incorporate synonyms, word relatedness, etc.
  - An example: retrieve, search, find, seek …
  - Using taxonomy such as WordNet is often not a good solution
    - Manual cost for creating taxonomies
    - Some human-created ones are subjective in nature
    - Inaccurate (e.g., word relations are mostly binary)
    - Inappropriate for computation

Word representation

• Distributed representation
  • Each word is represented using multiple dimensions
  • Each dimension can be used in different words’ representations

• A word is a $k$-dimensional vector
  • Makes it possible to compare words
    • e.g., semantic matching
  • Higher complexity
    • Depends on the value of $k$

• Local representation is a special case
  • Each word is a $1 \times |V|$ vector; only one entry is 1, others are 0

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

• How to create appropriate distributed representations?
• Word-document frequency matrix?
  • Each word is a vector of its TF-IDF scores in different documents
  • Too many dimensions! Depends on the size of the corpus.
  • Complexity (computational) & sparse.

<table>
<thead>
<tr>
<th></th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>retrieval</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>search</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>model</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>evaluation</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>relevance</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Dimensions (cont.)

- **Word co-occurrence matrix?**
  - Each word is a vector of its co-occurrence frequencies (can also be weighted using IDF) with each word in the vocabulary
  - Diagonal values are usually 0 in NLP applications
  - Still too many dimensions!
  - Depends on vocabulary size (and thus corpus size; heaps’ law)

<table>
<thead>
<tr>
<th></th>
<th>index</th>
<th>retrieval</th>
<th>search</th>
<th>model</th>
<th>evaluation</th>
<th>relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>retrieval</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>search</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>model</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>evaluation</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>relevance</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Dimensions (cont.)

• Many possible options for distributed representations
  • Usually problem dependent
  • Word co-occurrence information is the most popular heuristics
  • But directly using a co-occurrence matrix is not ideal
  • Because dimensionality $\propto$ cost

• So …
  • Much fewer dimensions than N and $|V|$ (e.g., a few hundred)
  • Informative (can approximate the original data)
  • Dense (fewer zeros)
Old approaches: Using LSI

- LSI recap
  - First step: SVD
  - $m$ is the rank of the original occurrence matrix

\[ C_{k \times m} = U_{k \times m} S_{m \times m} V_{m \times m}^T \]
Old approaches: Using LSI (cont.)

- LSI recap
  - LSI: use the most important $n$ dimensions ($n \ll \text{rank}(C)$)

\[ C_{k \times m} \approx U_{k \times n} S_{n \times n} V_{m \times n}^T \]
This is a word vector!
Today: Using word embeddings

• **Word2vec**
  • Unsupervised (like most other similar approaches)
  • Train distributed word representations to predict observed words
  • Using word co-occurrence within a window size
  • Readily implemented (almost everywhere today)
  • Two basic models

• **Continuous bag-of-words (CBOW)**
  • Using context words to predict each word
  • A 2c window: \( P( w_i | w_{i-c}, w_{i-c+1}, \ldots, w_{i+c-1}, w_{i+c} ) \)

• **Skip-gram**
  • Using each word to predict its context words
  • A 2c window: \( P( w_{i-c}, w_{i-c+1}, \ldots, w_{i+c-1}, w_{i+c} | w_i ) \)

The Skip-gram model

A “pseudo” task

- Given a word \( t \), to predict its context word \( c \)

\[
P(+|t,c) \quad P(-|t,c) = 1 - P(+|t,c)
\]

- Both \( t \) and \( c \) use vector representations
- Modeling prediction using a logistic regression model

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

- We don’t really care about this task, but hope to optimize the vector representations to through optimizing the prediction
- Optimize the logistic regression model; reduce prediction errors
Skip-gram with negative sampling

Negative sampling

- The corpus only provide positive examples (the actual context words)
- For each positive example, we sample \( k \) negative examples
- An example from Jurafsky et al. (2018)

... lemons, a [tablespoon of apricot preserves, or] pinch ...

\[
\begin{array}{ccc}
| c | c | t | c | c |
\end{array}
\]

positive examples +

<table>
<thead>
<tr>
<th>t</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>tablespoon</td>
</tr>
<tr>
<td>apricot</td>
<td>of</td>
</tr>
<tr>
<td>apricot</td>
<td>preserves</td>
</tr>
<tr>
<td>apricot</td>
<td>or</td>
</tr>
</tbody>
</table>

negative examples -

<table>
<thead>
<tr>
<th>t</th>
<th>c</th>
<th>t</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>aardvark</td>
<td>apricot</td>
<td>twelve</td>
</tr>
<tr>
<td>apricot</td>
<td>puddle</td>
<td>apricot</td>
<td>hello</td>
</tr>
<tr>
<td>apricot</td>
<td>where</td>
<td>apricot</td>
<td>dear</td>
</tr>
<tr>
<td>apricot</td>
<td>coaxial</td>
<td>apricot</td>
<td>forever</td>
</tr>
</tbody>
</table>

Skip-gram with negative sampling

Negative sampling
• The corpus only provide positive examples (the actual context words)
• For each positive example, we sample $k$ negative examples
• Sampling based on word counts in the corpus; setting $\alpha = 0.75$
  • Boost rare words; penalize frequent words; empirically works well
  \[ P_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha} \]
• Objective function: maximize the probabilities of real context words; minimize the probabilities of sampled negative words
• Derive gradients and train using stochastic gradient descent.
  \[ L(\theta) = \sum_{(t,c) \in +} \log P(+|t,c) + \sum_{(t,c) \in -} \log P(-|t,c) \]
The skip-gram model (as a neural-net)

- The skip-gram model learns word embeddings by using words to predict its context words.

\[
h = tW \quad \quad \quad \quad \quad \quad c = hW'
\]

**Input (t):**
the target word
1 x |V|

**Hidden Layer (h):**
1 x k

**Output (c):**
context words
1 x |V|

\[p(c|t): 1 \times |V|\]

\[W: \text{input vectors} \quad |V| \times k\]

\[W': \text{output vectors} \quad k \times |V|\]
The CBOW model (as a neural-net)

- The CBOW (Continuous Bag-Of-Words) model learns embeddings by predicting words using its context words.

\[
h = cW \\
t = hW'
\]

**Input (c):** context words  
1 x |V|

**Hidden Layer (h):**  
1 x k

**Output (t):** the target word  
1 x |V|

**p(t|c):**  
1 x |V|

W: input vectors  
|V| x k

W': output vectors  
k x |V|
Word2vec (cont.)

• Two models (CBOW and skip-gram)
• The most popular choice of distributed word representations today.
• Word embeddings are used in almost all deep learning applications: e.g., enriching input by combining local (occurrence) and global factors (global word co-occurrence)
• Faster than SVD!
• Works better than SVD!
• Directly trained to optimize word prediction
• Optimization function can be customized in customized problems
• Many open implementations
Applications of word embeddings in IR

Two approaches

• Directly apply to refine query & document representations
  • The purpose is to do “semantic” matching
  • Global embeddings + document expansion (e.g., Ganguly et al., 2015)
  • Global embeddings + query expansion (e.g., Zamani et al., 2016)
  • Local word embeddings + query expansion (e.g., Diaz et al., 2016)

• Modeling the input layer in neural networks for ranking
Ganguly, Roy, Mitra, and Jones (2015)

Key idea

- Use word embeddings to refine document representation.
- Using the refined document language model and QL for ranking.
- Very similar to cluster-based and LDA-based language model for ranking, just replaces the K-means & LDA with word embeddings ...

\[
P(t|t', d) = \frac{\text{sim}(t, t')}{\sum_{t'' \in d} \text{sim}(t, t'')}
\]

\[
P(t|d) = \lambda P(t|d) + \alpha \sum_{t' \in d} P(t, t'|d)P(t') + t'f(t', d) \frac{|d|}{df(t', d)}
\]

\[
\beta \sum_{t' \in N_t} P(t, t'|C)P(t') + (1 - \lambda - \alpha - \beta)P(t|C)
\]

Ganguly, Roy, Mitra, and Jones (2015)

Key idea

- Use word embeddings to refine document representation.
- Using the refined document language model and QL for ranking
- Very similar to cluster-based and LDA-based language model for ranking, just replaces the K-means & LDA with word embeddings …

\[ P(t|d) = \lambda P(t|d) + \alpha \sum_{t' \in d} P(t, t'|d) P(t') + \beta \sum_{t' \in N_t} P(t, t'|C) P(t') + (1 - \lambda - \alpha - \beta) P(t|C) \]

- Only select the top 3 most similar terms

\[ P(t|t', C) = \frac{\text{sim}(t, t')}{\sum_{t'' \in N_t} \text{sim}(t, t'')} \frac{cf(t')}{cs} \]

Ganguly, Roy, Mitra, and Jones (2015)

Does it work?

- The method seems to outperform LDA-based LM in different datasets ...

<table>
<thead>
<tr>
<th>Topic Set</th>
<th>Method</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MAP</td>
</tr>
<tr>
<td>TREC-6</td>
<td>LM</td>
<td>0.2148</td>
</tr>
<tr>
<td></td>
<td>LDA-LM</td>
<td>0.2192</td>
</tr>
<tr>
<td></td>
<td>GLM</td>
<td>0.2287</td>
</tr>
<tr>
<td>TREC-7</td>
<td>LM</td>
<td>0.1771</td>
</tr>
<tr>
<td></td>
<td>LDA-LM</td>
<td>0.1631</td>
</tr>
<tr>
<td></td>
<td>GLM</td>
<td>0.1958</td>
</tr>
<tr>
<td>TREC-8</td>
<td>LM</td>
<td>0.2357</td>
</tr>
<tr>
<td></td>
<td>LDA-LM</td>
<td>0.2428</td>
</tr>
<tr>
<td></td>
<td>GLM</td>
<td>0.2503</td>
</tr>
<tr>
<td>Robust</td>
<td>LM</td>
<td>0.2555</td>
</tr>
<tr>
<td></td>
<td>LDA-LM</td>
<td>0.2623</td>
</tr>
<tr>
<td></td>
<td>GLM</td>
<td>0.2864</td>
</tr>
</tbody>
</table>

Recall LDA-based LM for retrieval …

Does it work?

• The results reported by Ganguly et al. (2015) regarding the effectiveness of LDA-based language model seems inconsistent with those reported by Wei & Croft (2006) in their original paper (LBDM) …

<table>
<thead>
<tr>
<th>Collection</th>
<th>QL</th>
<th>CBDM</th>
<th>LBDM</th>
<th>%chg over QL</th>
<th>%chg over CBDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>0.2179</td>
<td>0.2326</td>
<td>0.2651</td>
<td>+21.64*</td>
<td>+13.97*</td>
</tr>
<tr>
<td>FT</td>
<td>0.2589</td>
<td>0.2713</td>
<td>0.2807</td>
<td>+7.54*</td>
<td>+3.46*</td>
</tr>
<tr>
<td>SJMN</td>
<td>0.2032</td>
<td>0.2171</td>
<td>0.2307</td>
<td>+13.57*</td>
<td>+6.26*</td>
</tr>
<tr>
<td>LA</td>
<td>0.2468</td>
<td>0.2590</td>
<td>0.2666</td>
<td>+8.02²</td>
<td>+2.93</td>
</tr>
<tr>
<td>WSJ</td>
<td>0.2958</td>
<td>0.2984</td>
<td>0.3253</td>
<td>+9.97*</td>
<td>+9.01*</td>
</tr>
</tbody>
</table>

Zamani and Croft (2016)

Key idea

- Measuring word similarities using word embeddings
- Apply word similarities to improve query expansion
- Remember the idea of RM1 (RM3 is simply RM1 + the original query)
  - A list of words are selected and weighted from top-ranked results by QL
  - The weight for a weight depends on two part: $P(w|D)$ and $P(q|D)$
  - Words from a higher ranked document $D$ are more important.
  - Popular words in $D$ get higher probabilities.

\[
P(w|q) \propto P(q, w) \\
\propto \sum_{D \in FB} \left[ P(w|D) P(q|D) \right] = \sum_{D \in FB} \left[ P(w|D) \prod_{q_i \in q} P(q_i|D) \right]
\]

Zamani and Croft (2016)

EQE1 (EQE: Embedding-based Query Expansion)

- Expand words based on how similar they are to query terms \( P(q_i|w) \)

\[
p(w|\theta_Q) = \frac{p(w)p(\theta_Q|w)}{p(Q)}
\approx p(w)p(\theta_Q|w)
\approx p(w) \prod_{i=1}^{k} p(q_i|w)
\]

\[
p(q_i|w) = \frac{\delta(q_i,\overline{w})}{\sum_{w' \in V} \delta(\overline{w},\overline{w'})}, \quad p(w) = \sum_{w' \in V} p(w,w') \propto \delta(\overline{w},\overline{w'})
\]

Zamani and Croft (2016)

**EQE1 (EQE: Embedding-based Query Expansion)**

- Expand words based on how similar they are to query terms $P(q_i|w)$

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>EQE1</th>
<th>EQE2</th>
<th>ERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>Cosine</td>
<td>0.2293</td>
<td>0.2366</td>
<td>0.3038</td>
</tr>
<tr>
<td></td>
<td>Sigmoid</td>
<td>0.2388*</td>
<td>0.2391</td>
<td>0.3102*</td>
</tr>
<tr>
<td>Robust</td>
<td>Cosine</td>
<td>0.2247</td>
<td>0.2233</td>
<td>0.2677</td>
</tr>
<tr>
<td></td>
<td>Sigmoid</td>
<td>0.2292*</td>
<td>0.2257</td>
<td>0.2711*</td>
</tr>
<tr>
<td>GOV2</td>
<td>Cosine</td>
<td>0.2709</td>
<td>0.2654</td>
<td>0.2971</td>
</tr>
<tr>
<td></td>
<td>Sigmoid</td>
<td>0.2745*</td>
<td>0.2727*</td>
<td>0.3005</td>
</tr>
</tbody>
</table>

The claimed that transforming the original Cosine similarity using a sigmoid function is more effective.

\[
p(q_i|w) = \frac{\delta(q_i, \bar{w})}{\sum_{w' \in V} \delta(\bar{w}, \bar{w'})}, \quad p(w) = \sum_{w' \in V} p(w, w') \propto \delta(\bar{w}, \bar{w'})
\]

Zamani and Croft (2016)

EQE2 (EQE: Embedding-based Query Expansion)

- Expand words based on how similar they are to query terms $P(w|w')$
- The generation process is a little bit different

\[
p(w|\theta_Q) = \sum_{w' \in V} p(w, w'|\theta_Q)
= \sum_{w' \in V} p(w|w', \theta_Q) p(w'|\theta_Q)
\approx \sum_{w' \in V} p(w|w') p(w'|\theta_Q)
\]

$\delta(\overrightarrow{w}, \overrightarrow{w'}) = \frac{\text{count}(w', Q)}{|Q|}$

Zamani and Croft (2016)

ERM (Embedding-based Relevance Model)

- Boost the original query-document association in RM1 by word embedding-based semantic match

\[
p(w|\theta_F) = \sum_{D \in F} p(w, Q, D) \quad \text{Relevance Model (RM1)}
\]

\[
= \sum_{w' \in V} p(Q|w, D)p(w|D)P(D)
\]

\[
p(Q|w, D) = \beta p_{tm}(Q|w, D) + (1 - \beta)p_{sem}(Q|w, D)
\]

\[
p_{tm}(Q|w, D) = \prod_{i=1}^{k} p(q_i|D) \quad \text{If } \beta = 1 \\
\text{then ERM=RM3}
\]

\[
p_{sem}(Q|w, D) = \prod_{i=1}^{k} p_{sem}(q_i|w, D) = \prod_{i=1}^{k} \frac{\delta(q_i, w)c(q_i, D)}{Z}
\]

Zamani and Croft (2016)

Does it work?

- EQE1 and EQE2 are both effective; they claimed EQE1 is better than EQE2

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>MLE</th>
<th>GLM</th>
<th>VEXP</th>
<th>AWE</th>
<th>EQE1</th>
<th>EQE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>MAP</td>
<td>0.2236</td>
<td>0.2254</td>
<td>0.2338</td>
<td>0.2304</td>
<td>0.2388&lt;sup&gt;1,2,3&lt;/sup&gt;</td>
<td>0.239&lt;sup&gt;1,2,3&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>P@5</td>
<td>0.4260</td>
<td>0.4369</td>
<td>0.4412</td>
<td>0.4356</td>
<td>0.4397</td>
<td>0.4466</td>
</tr>
<tr>
<td></td>
<td>P@10</td>
<td>0.4014</td>
<td>0.4051</td>
<td>0.4038</td>
<td>0.4058</td>
<td>0.4075</td>
<td>0.4014</td>
</tr>
<tr>
<td></td>
<td>RI</td>
<td>–</td>
<td>0.10</td>
<td>0.18</td>
<td>0.14</td>
<td>0.32</td>
<td>0.32</td>
</tr>
<tr>
<td>Robust</td>
<td>MAP</td>
<td>0.2190</td>
<td>0.2244</td>
<td>0.2253</td>
<td>0.2224</td>
<td>0.2292&lt;sup&gt;1,2,3&lt;/sup&gt;</td>
<td>0.2257&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>P@5</td>
<td>0.4606</td>
<td>0.4523</td>
<td>0.4722</td>
<td>0.4680</td>
<td>0.4739</td>
<td>0.4622</td>
</tr>
<tr>
<td></td>
<td>P@10</td>
<td>0.3979</td>
<td>0.3929</td>
<td>0.4133</td>
<td>0.4066</td>
<td>0.4162</td>
<td>0.4183</td>
</tr>
<tr>
<td></td>
<td>RI</td>
<td>–</td>
<td>0.22</td>
<td>0.17</td>
<td>0.14</td>
<td>0.30</td>
<td>0.22</td>
</tr>
<tr>
<td>GOV2</td>
<td>MAP</td>
<td>0.2696</td>
<td>0.2684</td>
<td>0.2687</td>
<td>0.2657</td>
<td>0.2745&lt;sup&gt;1,2,3&lt;/sup&gt;</td>
<td>0.2727&lt;sup&gt;4&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>P@5</td>
<td>0.5592</td>
<td>0.5537</td>
<td>0.5932</td>
<td>0.5537</td>
<td>0.5959</td>
<td>0.5810</td>
</tr>
<tr>
<td></td>
<td>P@10</td>
<td>0.5531</td>
<td>0.5483</td>
<td>0.5537</td>
<td>0.5503</td>
<td>0.5660</td>
<td>0.5517</td>
</tr>
<tr>
<td></td>
<td>RI</td>
<td>–</td>
<td>-0.14</td>
<td>0.10</td>
<td>-0.18</td>
<td>0.20</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Zamani and Croft (2016)

Does it work?

- Combining EQE1 & EQE2 with RM1/ERM can further improve search effectiveness
- EQE1 & EQE2 (global expansion); ERM (local expansion)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>MLE</th>
<th>MLE+RM1 (RM3)</th>
<th>EQE1+RM1</th>
<th>EQE2+RM1</th>
<th>MLE+ERM</th>
<th>EQE1+ERM</th>
<th>EQE2+ERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>MAP</td>
<td>0.2236</td>
<td>0.3051</td>
<td>0.3118&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.3115&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.3102&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.3178&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.3140&lt;sup&gt;12&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>P@5</td>
<td>0.4260</td>
<td>0.4644</td>
<td>0.4808</td>
<td>0.4795</td>
<td>0.4699</td>
<td>0.4822</td>
<td>0.4644</td>
</tr>
<tr>
<td></td>
<td>P@10</td>
<td>0.4014</td>
<td>0.4500</td>
<td>0.4500</td>
<td>0.4452</td>
<td>0.4521</td>
<td>0.4568</td>
<td>0.4479</td>
</tr>
<tr>
<td></td>
<td>RI</td>
<td>–</td>
<td>0.47</td>
<td>0.45</td>
<td>0.41</td>
<td>0.47</td>
<td>0.52</td>
<td>0.52</td>
</tr>
<tr>
<td>Robust</td>
<td>MAP</td>
<td>0.2190</td>
<td>0.2677</td>
<td>0.2712&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.2710&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.2711&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.2731&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.2750&lt;sup&gt;12&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>P@5</td>
<td>0.4606</td>
<td>0.4581</td>
<td>0.4747</td>
<td>0.4722</td>
<td>0.4639</td>
<td>0.4797</td>
<td>0.4730</td>
</tr>
<tr>
<td></td>
<td>P@10</td>
<td>0.3979</td>
<td>0.4191</td>
<td>0.4241</td>
<td>0.4295</td>
<td>0.4241</td>
<td>0.4307</td>
<td>0.4369</td>
</tr>
<tr>
<td></td>
<td>RI</td>
<td>–</td>
<td>0.31</td>
<td>0.35</td>
<td>0.31</td>
<td>0.32</td>
<td>0.36</td>
<td>0.36</td>
</tr>
<tr>
<td>GOV2</td>
<td>MAP</td>
<td>0.2696</td>
<td>0.2938</td>
<td>0.2987&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.2922&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.3005&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.3012&lt;sup&gt;12&lt;/sup&gt;</td>
<td>0.2957&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td>P@5</td>
<td>0.5592</td>
<td>0.5592</td>
<td>0.5687</td>
<td>0.5673</td>
<td>0.5823</td>
<td>0.5850</td>
<td>0.5782</td>
</tr>
<tr>
<td></td>
<td>P@10</td>
<td>0.5531</td>
<td>0.5599</td>
<td>0.5816</td>
<td>0.5714</td>
<td>0.5830</td>
<td>0.5844</td>
<td>0.5782</td>
</tr>
<tr>
<td></td>
<td>RI</td>
<td>–</td>
<td>0.15</td>
<td>0.14</td>
<td>0.14</td>
<td>0.20</td>
<td>0.20</td>
<td>0.20</td>
</tr>
</tbody>
</table>

“Local” Word Embeddings

Embeddings trained on a whole corpus seems intriguing, but …

• In IR, we are almost always concerned with a query 😊
• Learning word representations: old vs. new

<table>
<thead>
<tr>
<th>“old”</th>
<th>“new”</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSI, PLSI, LDA</td>
<td>Word2vec, Glove</td>
</tr>
<tr>
<td>Pseudo-relevance feedback</td>
<td>?</td>
</tr>
</tbody>
</table>

An example: words similar to “apple”?

• Global (both the tech company & the fruit):
  • Samsung, banana, Google, pear, grape …
• Local, query = “iPhone XS” (the tech company)
  • Samsung, Google, …
Diaz, Mitra, & Craswell (2016)

Key idea

• Train a query-biased word embedding, such that documents relevant/similar to the query have higher weights in training.
• Apply the trained “local” word embeddings to expand queries.

“Local” (query-biased) embeddings

• Start from the skip-gram model with negative sampling.
• Sample documents by $P(d)$, which is determined based on the KL-Divergence score between $q$ and $d$ (normalized using softmax).
  • Recall KLD: QL is a special case of KLD where the query model uses MLE

\[
D(p_q || p_d) = \sum_{w \in V} p_q(w) \log \frac{p_q(w)}{p_d(w)} \quad p(d) = \frac{\exp(-D(p_q || p_d))}{\sum_{d'} \exp(-D(p_q || p_{d'}))}
\]

Diaz, Mitra, & Craswell (2016)

Query expansion using word embeddings

- $q' = UU^T q$
- Pick the highest weighted terms (cells with highest values) in $q'$
- Expanded query is interpolated with the original query (similar to RM3)

$U$: $|V| \times k$

$U^T$: $k \times |V|$

$q$: $|V| \times 1$

$q'$: $|V| \times 1$

$UU^T$: a $|V| \times |V|$ word similarity matrix

The original query (only a few cells are 1; others are 0)

Expanded query

Diaz, Mitra, & Craswell (2016)

Training sets for embeddings

• External datasets: Wikipedia; Google News; Gigaword
  • Background: query expansion based on a high-quality, clean corpus is usually more effective than those using the target test corpus (particularly when the corpus is a web corpus)

• Target: the test corpus used for retrieval

Test datasets

• Two newswire datasets (trec12 and robust); a web dataset (clueweb09)

• Runs:
  • QL (no query expansion)
  • Query expansion using global embeddings trained on different corpora
  • Query expansion using local embeddings trained on different corpora

Diaz, Mitra, & Craswell (2016)

Does it work?

- Evaluation metric: nDCG@10 (will explain next week)
- Local > global; external (sometimes) > target
- But no comparison with pseudo-relevance feedback (e.g., RM3)

<table>
<thead>
<tr>
<th></th>
<th>QL</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>gnews</th>
<th>target</th>
<th>target</th>
<th>giga</th>
<th>wiki</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trec12</td>
<td>0.514</td>
<td>0.518</td>
<td>0.518</td>
<td>0.530</td>
<td>0.531</td>
<td>0.530</td>
<td>0.545</td>
<td>0.535</td>
<td>0.563</td>
<td>0.523</td>
</tr>
<tr>
<td>robust</td>
<td>0.467</td>
<td>0.470</td>
<td>0.463</td>
<td>0.469</td>
<td>0.468</td>
<td>0.472</td>
<td>0.465</td>
<td>0.475</td>
<td>0.517</td>
<td>0.476</td>
</tr>
<tr>
<td>web</td>
<td>0.216</td>
<td>0.227</td>
<td>0.229</td>
<td>0.230</td>
<td><strong>0.232</strong></td>
<td>0.218</td>
<td>0.216</td>
<td>0.234</td>
<td>0.236</td>
<td><strong>0.258</strong></td>
</tr>
</tbody>
</table>

Test datasets

Recall the typical improvements of PRF ...

- +0%~10% by precision-biased metrics (e.g., P@10, nDCG@10)
- +10%~20% by average precision (more recall-oriented)
- RM3 is typically considered as a hard baseline for query expansion

<table>
<thead>
<tr>
<th>S.w.</th>
<th>Metric</th>
<th>MLE</th>
<th>RM3</th>
<th>RM4</th>
<th>DMM</th>
<th>SMM</th>
<th>RMM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trained on AP1 and Tested on AP2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/</td>
<td>AvgPr</td>
<td>0.220</td>
<td>0.295</td>
<td>0.301</td>
<td>0.290</td>
<td>0.304</td>
<td>0.299</td>
</tr>
<tr>
<td></td>
<td>Pr@10</td>
<td>0.386</td>
<td>0.408</td>
<td>0.418</td>
<td>0.422</td>
<td>0.400</td>
<td>0.398</td>
</tr>
<tr>
<td></td>
<td>Recall</td>
<td>3074</td>
<td>3810</td>
<td>3892</td>
<td>3681</td>
<td>3933</td>
<td>3859</td>
</tr>
<tr>
<td>w/o</td>
<td>AvgPr</td>
<td>0.231</td>
<td>0.312</td>
<td>0.321</td>
<td>0.289</td>
<td>0.324</td>
<td>0.323</td>
</tr>
<tr>
<td></td>
<td>Pr@10</td>
<td>0.398</td>
<td>0.436</td>
<td>0.448</td>
<td>0.424</td>
<td>0.432</td>
<td>0.446</td>
</tr>
<tr>
<td></td>
<td>Recall</td>
<td>3154</td>
<td>3913</td>
<td>3908</td>
<td>3674</td>
<td>3921</td>
<td>3927</td>
</tr>
<tr>
<td></td>
<td>Trained on TREC6 and Tested on TREC78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/</td>
<td>AvgPr</td>
<td>0.217</td>
<td>0.249</td>
<td>0.242</td>
<td>0.235</td>
<td>0.251</td>
<td>0.243</td>
</tr>
<tr>
<td></td>
<td>Pr@10</td>
<td>0.437</td>
<td>0.438</td>
<td>0.426</td>
<td>0.443</td>
<td>0.443</td>
<td>0.451</td>
</tr>
<tr>
<td></td>
<td>Recall</td>
<td>5114</td>
<td>5805</td>
<td>5739</td>
<td>5476</td>
<td>5821</td>
<td>5625</td>
</tr>
<tr>
<td>w/o</td>
<td>AvgPr</td>
<td>0.217</td>
<td>0.251</td>
<td>0.243</td>
<td>0.235</td>
<td>0.252</td>
<td>0.249</td>
</tr>
<tr>
<td></td>
<td>Pr@10</td>
<td>0.434</td>
<td>0.454</td>
<td>0.446</td>
<td>0.433</td>
<td>0.441</td>
<td>0.443</td>
</tr>
<tr>
<td></td>
<td>Recall</td>
<td>5107</td>
<td>5799</td>
<td>5776</td>
<td>5500</td>
<td>5896</td>
<td>5833</td>
</tr>
<tr>
<td></td>
<td>Well-Tuned on WT2G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/</td>
<td>AvgPr</td>
<td>0.293</td>
<td>0.338</td>
<td>0.319</td>
<td>0.327</td>
<td>0.330</td>
<td>0.309</td>
</tr>
<tr>
<td></td>
<td>Pr@10</td>
<td>0.450</td>
<td>0.500</td>
<td>0.470</td>
<td>0.494</td>
<td>0.496</td>
<td>0.458</td>
</tr>
<tr>
<td></td>
<td>Recall</td>
<td>1830</td>
<td>1822</td>
<td>1806</td>
<td>1843</td>
<td>1856</td>
<td>1811</td>
</tr>
<tr>
<td>w/o</td>
<td>AvgPr</td>
<td>0.306</td>
<td>0.344</td>
<td>0.328</td>
<td>0.326</td>
<td>0.331</td>
<td>0.319</td>
</tr>
<tr>
<td></td>
<td>Pr@10</td>
<td>0.456</td>
<td>0.490</td>
<td>0.490</td>
<td>0.476</td>
<td>0.476</td>
<td>0.482</td>
</tr>
<tr>
<td></td>
<td>Recall</td>
<td>1870</td>
<td>1862</td>
<td>1879</td>
<td>1873</td>
<td>1889</td>
<td>1863</td>
</tr>
</tbody>
</table>

Summary so far ...

Word embeddings are great (and hot)
• Model word semantics & relatedness using distributed representations

Direct applications of word embeddings to IR
• General idea: computing word similarity based on embeddings; apply word similarity to boost query & document representations for retrieval
• Appealing (mostly because word embeddings becomes hot since 2013; similar ideas were proved effective using LSI/LDA etc.)
• A lot of work around 2014~2016 …
• But how much do they outperform previous (strong) baselines?

Next Lecture (Thursday)
• Deep neural nets for ranking