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]  
    - 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.
Introduction to Information Retrieval

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.
- $w_{ij} =$ (normalized) 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>repellant, 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, naive, innate</td>
</tr>
</tbody>
</table>

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
- $motel \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$
- $hotel \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = 0$

If query on hotel and document has motel, then our query and document vectors are orthogonal

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

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”
- 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”
- Popular in the 1990s [Deerwester et al., 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 (!!!)

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

<table>
<thead>
<tr>
<th>linguistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.286</td>
</tr>
<tr>
<td>0.792</td>
</tr>
<tr>
<td>−0.177</td>
</tr>
<tr>
<td>−0.107</td>
</tr>
<tr>
<td>0.109</td>
</tr>
<tr>
<td>−0.542</td>
</tr>
<tr>
<td>0.349</td>
</tr>
<tr>
<td>0.271</td>
</tr>
</tbody>
</table>

Basic idea of learning neural network word embeddings

We define a model that aims to predict between a center word \( w_c \) and context words in terms of word vectors

\[
p(\text{context} | w_c) = ...\]

which has a loss function, e.g.,

\[
J = 1 - p(w_c | \text{context})
\]

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
   - Naive softmax

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:

\[
J'(\theta) = \prod_{t=1}^{T} \prod_{j \in S(t)} p(w_i|w_j; \theta)
\]

Normalized Likelihood
\[
J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{j \in S(t)} \log p(w_i|w_j)
\]

Where \( \theta \) represents all variables we will optimize

Details of Word2Vec
Predict surrounding words in a window of radius \( m \) of every word.

For \( p(w_{i+j}|w_i) \) the simplest first formulation is

\[
p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_v \exp(u_v^T v_c)}
\]

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 \( o \)
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:
  $g = \begin{bmatrix} w_{\text{board game}} \\ w_0 \\ \vdots \\ w_{\text{zebra}} \end{bmatrix} \in \mathbb{R}^{2d/V}$

- 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.

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

Vanilla Gradient Descent Code

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

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 $f'(x) = 4x^3 - 9x^2$.

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

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

Vanilla Gradient Descent Code

```python
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: \( \frac{dy}{dx} = \frac{d}{dx} 5(x^3 + 7) \)

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

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

\[
\frac{dy}{dx} = 20(x^3 + 7)^3 \cdot 3x^2
\]

Objective Function

Maximize \( J'(\theta) = \frac{1}{m} \sum_{i=1}^{m} \log p(y_i|x_i;\theta) \)

Or minimize negative log likelihood

\[
J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \log p(y_i|x_i;\theta)
\]

where

\[
p(y|x) = \sum_{w} \exp(w_i^T v_x)
\]

We now take derivatives to work at minimum

Each word type (each \( w \)) has two word representations: an latent word and context word

\[
\frac{2}{\theta_{ij}} \sum_{x=1}^{n} \exp(w_i^T v_x) - \frac{2}{\theta_{ij}} \sum_{x=1}^{n} \exp(w_j^T v_x)
\]

Important to change index

\[
\frac{2}{\theta_{ij}} \sum_{x=1}^{n} \exp(w_i^T v_x) - \frac{2}{\theta_{ij}} \sum_{x=1}^{n} \exp(w_j^T v_x)
\]

More deriv inside sum

\[
\frac{2}{\theta_{ij}} \sum_{x=1}^{n} \exp(w_i^T v_x) - \frac{2}{\theta_{ij}} \sum_{x=1}^{n} \exp(w_j^T v_x)
\]

Chain rule

\[
\frac{2}{\theta_{ij}} \sum_{x=1}^{n} \exp(w_i^T v_x) - \frac{2}{\theta_{ij}} \sum_{x=1}^{n} \exp(w_j^T v_x)
\]

This is just the derivatives for the vector vector parameters

Also need derivatives for output vector parameters

(they're similar)

Thus 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**
  
  \[ x_{\text{apple}} - x_{\text{apples}} = x_{\text{car}} - x_{\text{cars}} = x_{\text{family}} - x_{\text{families}} \]

- Similarly for verb and adjective morphological forms

  **Semantically (Semeval 2012 task 2)**
  
  \[ x_{\text{shirt}} - x_{\text{clothing}} = x_{\text{chair}} - x_{\text{furniture}} \]

\[ x_{\text{king}} - x_{\text{man}} = x_{\text{queen}} - x_{\text{woman}} \]

Word Analogies

Test for linear relationships, examined by Mikolov et al.

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

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

\[ \begin{align*}
  + \text{king} & \approx [0.30, 0.70] \\
  - \text{man} & \approx [0.20, 0.20] \\
  + \text{woman} & \approx [0.60, 0.30] \\
  \text{queen} & \approx [0.70, 0.80]
\end{align*} \]

GloVe Visualizations

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

Using 2 word embeddings

Using 2 word embeddings

<table>
<thead>
<tr>
<th>yale</th>
<th>seahawks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN-IN</td>
<td>IN-OUT</td>
</tr>
<tr>
<td>yale</td>
<td>yale</td>
</tr>
<tr>
<td>harvard</td>
<td>faculty</td>
</tr>
<tr>
<td>nyu</td>
<td>alumni</td>
</tr>
<tr>
<td>cornell</td>
<td>orientation</td>
</tr>
<tr>
<td>tulane</td>
<td>haven</td>
</tr>
<tr>
<td>tufts</td>
<td>graduate</td>
</tr>
</tbody>
</table>

Dual Embedding Space Model (DESM)

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

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

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

\[
DESM(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{q_i^T \mathbf{D}}{\|q_i\| \|\mathbf{D}\|}
\]
Introduction to Information Retrieval

Dual Embedding Space Model (DESM)

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

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

- 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>Explicitly Judged Test Set</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25</td>
<td>23.69</td>
<td>29.14</td>
<td>44.77</td>
</tr>
<tr>
<td>LSA</td>
<td>22.41*</td>
<td>28.25*</td>
<td>44.24*</td>
</tr>
<tr>
<td>DESM (IN-IN, trained on body text)</td>
<td>23.59</td>
<td>29.59</td>
<td>45.51*</td>
</tr>
<tr>
<td>DESM (IN-IN, trained on queries)</td>
<td>23.75</td>
<td>29.72</td>
<td>46.36*</td>
</tr>
<tr>
<td>DESM (IN-OUT, trained on body text)</td>
<td>24.06</td>
<td>30.32*</td>
<td>46.57*</td>
</tr>
<tr>
<td>DESM (IN-OUT, trained on queries)</td>
<td>25.02*</td>
<td>31.14*</td>
<td>47.39*</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>Explicitly Judged Test Set</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25</td>
<td>21.44</td>
<td>26.09</td>
<td>37.53</td>
</tr>
<tr>
<td>LSA</td>
<td>04.61*</td>
<td>04.63*</td>
<td>04.81*</td>
</tr>
<tr>
<td>DESM (IN-IN, trained on body text)</td>
<td>06.09*</td>
<td>06.80*</td>
<td>07.39*</td>
</tr>
<tr>
<td>DESM (IN-IN, trained on queries)</td>
<td>05.56*</td>
<td>05.59*</td>
<td>06.03*</td>
</tr>
<tr>
<td>DESM (IN-OUT, trained on body text)</td>
<td>01.01*</td>
<td>01.16*</td>
<td>01.58*</td>
</tr>
<tr>
<td>DESM (IN-OUT, trained on queries)</td>
<td>00.62*</td>
<td>00.38*</td>
<td>00.81*</td>
</tr>
<tr>
<td>BM25 + DESM (IN-IN, trained on body text)</td>
<td>21.53</td>
<td>26.16</td>
<td>37.48</td>
</tr>
<tr>
<td>BM25 + DESM (IN-IN, trained on queries)</td>
<td>21.58</td>
<td>26.20</td>
<td>37.62</td>
</tr>
<tr>
<td>BM25 + DESM (IN-OUT, trained on body text)</td>
<td>21.47</td>
<td>26.18</td>
<td>37.55</td>
</tr>
<tr>
<td>BM25 + DESM (IN-OUT, trained on queries)</td>
<td>21.54</td>
<td>26.42*</td>
<td>37.86*</td>
</tr>
</tbody>
</table>

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

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

- For example, DESM can confuse Oxford and Cambridge
- Bing rarely makes the Oxford-Cambridge mistake
Global vs. local embedding [Diaz 2016]

<table>
<thead>
<tr>
<th>global</th>
<th>local</th>
</tr>
</thead>
<tbody>
<tr>
<td>cutting</td>
<td>tax</td>
</tr>
<tr>
<td>squeeze</td>
<td>deficit</td>
</tr>
<tr>
<td>reduce</td>
<td>vote</td>
</tr>
<tr>
<td>slash</td>
<td>budget</td>
</tr>
<tr>
<td>reduction</td>
<td>reduction</td>
</tr>
<tr>
<td>spend</td>
<td>house</td>
</tr>
<tr>
<td>lower</td>
<td>bill</td>
</tr>
<tr>
<td>halve</td>
<td>plan</td>
</tr>
<tr>
<td>soften</td>
<td>spend</td>
</tr>
<tr>
<td>freeze</td>
<td>billion</td>
</tr>
</tbody>
</table>

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.”

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 $\rightarrow$ 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” $\rightarrow$ “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.

Count based vs. direct prediction

LSA, HAL (Sund & Burgess), COALS (Rohde et al), Hellinger-PCA (Lavrik & Corrington)

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

- 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

<table>
<thead>
<tr>
<th>x = solid</th>
<th>x = gas</th>
<th>x = water</th>
<th>x = random</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(x</td>
<td>ice) large</td>
<td>P(x</td>
<td>steam) small</td>
</tr>
<tr>
<td>P(x</td>
<td>steam) small</td>
<td>P(x</td>
<td>steam) large</td>
</tr>
<tr>
<td>P(x</td>
<td>steam) large</td>
<td>P(x</td>
<td>steam) small</td>
</tr>
</tbody>
</table>

GloVe: A new model for learning word representations

[Pennington, Socher, and Manning, EMNLP 2014]

\[ w_i \cdot w_j = \log P(i|j), \]

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

\[ J = \sum_{i,j=1}^{V} f(X_{ij}) (w_i^T X_{ij} + b_i + b_j - \log X_{ij})^2 \quad f \sim \]
Introduction to Information Retrieval

Word similarities

Nearest words to frog:
1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

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

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Dimensions</th>
<th>Corpus size</th>
<th>Performance (Syn + Sem)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBOW (Mikolov et al. 2013b)</td>
<td>300</td>
<td>1.6 billion</td>
<td>36.1</td>
</tr>
<tr>
<td>CBOW (Mikolov et al. 2013b, by us)</td>
<td>1000</td>
<td>6 billion</td>
<td>63.7</td>
</tr>
<tr>
<td>GloVe (this work)</td>
<td>300</td>
<td>42 billion</td>
<td>75.0</td>
</tr>
</tbody>
</table>