Chapter 6:
Vector Semantics, continued

Tf-idf and PPMI are sparse representations

tf-idf and PPMI vectors are
- **long** (length $|V|= 20,000$ to $50,000$)
- **sparse** (most elements are zero)
Alternative: dense vectors

vectors which are
- short (length 50-1000)
- dense (most elements are non-zero)

Sparse versus dense vectors

Why dense vectors?
- Short vectors may be easier to use as features in machine learning (less weights to tune)
- Dense vectors may generalize better than storing explicit counts
- They may do better at capturing synonymy:
  - car and automobile are synonyms; but are distinct dimensions
  - a word with car as a neighbor and a word with automobile as a neighbor should be similar, but aren’t
- In practice, they work better
Dense embeddings you can download!

**Word2vec** (Mikolov et al.)
[https://code.google.com/archive/p/word2vec/](https://code.google.com/archive/p/word2vec/)

**Fasttext** [http://www.fasttext.cc/](http://www.fasttext.cc/)

**Glove** (Pennington, Socher, Manning)

Word2vec

Popular embedding method

Very fast to train

Code available on the web

Idea: **predict** rather than **count**
Word2vec

- Instead of counting how often each word $w$ occurs near "apricot"
- Train a classifier on a binary prediction task:
  - Is $w$ likely to show up near "apricot"?
- We don’t actually care about this task
  - But we'll take the learned classifier weights as the word embeddings

Insight: Use running text as implicitly supervised training data!

- A word $s$ near apricot
- Acts as gold ‘correct answer’ to the question
  - “Is word $w$ likely to show up near apricot?”
- No need for hand-labeled supervision
- The idea comes from neural language modeling
Word2Vec: **Skip-Gram** Task

Word2vec provides a variety of options. Let's do

- "skip-gram with negative sampling" (SGNS)

**Skip-gram algorithm**

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples.
3. Use logistic regression to train a classifier to distinguish those two cases.
4. Use the weights as the embeddings.
Skip-Gram Training Data

Training sentence:

... lemon, a **tablespoon of apricot jam** a pinch ...

\[ c_1 \quad c_2 \quad \text{target} \quad c_3 \quad c_4 \]

Assume context words are those in +/- 2 word window

Skip-Gram Goal

Given a tuple \((t,c) = \text{target, context}\)

\(\text{(apricot, jam)}\)
\(\text{(apricot, aardvark)}\)

Return probability that \(c\) is a real context word:

\[ P(+/t,c) \]
\[ P(-/t,c) = 1 - P(+/t,c) \]
How to compute $p(\ + \mid t,c)$?

**Intuition:**
- Words are likely to appear near similar words
- Model similarity with dot-product!
- $\text{Similarity}(t,c) \propto t \cdot c$

**Problem:**
- *Dot product is not a probability!*
- *(Neither is cosine)*

---

Turning dot product into a probability

The sigmoid lies between 0 and 1:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

![Graph of $y = \frac{1}{1 + e^{-x}}$]
Turning dot product into a probability

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

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

For all the context words:
Assume all context words are independent

\[ P(+) | t, c_{1:k} ) = \prod_{i=1}^{k} \frac{1}{1 + e^{-t \cdot c_i}} \]

\[ \log P(+) | t, c_{1:k} ) = \sum_{i=1}^{k} \log \left( \frac{1}{1 + e^{-t \cdot c_i}} \right) \]
Skip-Gram Training Data

Training sentence:
... lemon, a \textit{tablespoon of apricot jam} a pinch ...

\begin{center}
c1 \quad c2 \quad t \quad c3 \quad c4
\end{center}

Training data: input/output pairs centering on \textit{apricot}
Asssume a +/- 2 word window

Skip-Gram Training

Training sentence:
... lemon, a \textit{tablespoon of apricot jam} a pinch ...

\begin{center}
c1 \quad c2 \quad t \quad c3 \quad c4
\end{center}

\begin{itemize}
  \item For each positive example, we'll create $k$ negative examples.
  \item Using \textit{noise} words
  \item Any random word that isn't $t$
\end{itemize}
Skip-Gram Training

Training sentence:

... lemon, a **tablespoon of apricot jam** a pinch ...

c1       c2       t       c3       c4

<table>
<thead>
<tr>
<th>positive examples</th>
<th>negative examples - k=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>apricot</td>
</tr>
<tr>
<td>tablespoon</td>
<td>aardvark</td>
</tr>
<tr>
<td>apricot of</td>
<td>apricot puddle</td>
</tr>
<tr>
<td>apricot preserves</td>
<td>apricot where</td>
</tr>
<tr>
<td>apricot or</td>
<td>apricot coaxial</td>
</tr>
</tbody>
</table>

Choosing noise words

Could pick w according to their unigram frequency P(w)

More common to chosen then according to pα(w)

\[
P_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_w \text{count}(w)^\alpha}
\]

α= ¾ works well because it gives rare noise words slightly higher probability

To show this, imagine two events p(a)=.99 and p(b) = .01:

\[
P_\alpha(a) = \frac{.99^{\frac{75}{0}}}{.99^{\frac{75}{0}} + .01^{\frac{75}{0}}} = .97
\]

\[
P_\alpha(b) = \frac{.01^{\frac{75}{0}}}{.99^{\frac{75}{0}} + .01^{\frac{75}{0}}} = .03
\]
Setup

Let's represent words as vectors of some length (say 300), randomly initialized.

So we start with 300 * V random parameters

Over the entire training set, we'd like to adjust those word vectors such that we
  ◦ Maximize the similarity of the target word, context word pairs (t,c) drawn from the positive data
  ◦ Minimize the similarity of the (t,c) pairs drawn from the negative data.

Learning the classifier

Iterative process.

We’ll start with 0 or random weights

Then adjust the word weights to
  ◦ make the positive pairs more likely
  ◦ and the negative pairs less likely

over the entire training set:
Objective Criteria

We want to maximize...

\[ \sum_{(t,c) \in +} \log P(+|t,c) + \sum_{(t,c) \in -} \log P(-|t,c) \]

Maximize the + label for the pairs from the positive training data, and the – label for the pairs sample from the negative data.

Focusing on one target word t:

\[
L(\theta) = \log P(+|t,c) + \sum_{i=1}^{k} \log P(-|t,n_i)
\]

\[
= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t)
\]

\[
= \log \frac{1}{1 + e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1 + e^{-n_i \cdot t}}
\]
Train using gradient descent

Actually learns two separate embedding matrices $W$ and $C$

Can use $W$ and throw away $C$, or merge them somehow
Summary: How to learn word2vec (skip-gram) embeddings

Start with $V$ random 300-dimensional vectors as initial embeddings

Use logistic regression, the second most basic classifier used in machine learning after naïve bayes

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

Other approaches

See end of chapter in text for pointers
Evaluating embeddings

Compare to human scores on word similarity-type tasks:

- WordSim-353 (Finkelstein et al., 2002)
- SimLex-999 (Hill et al., 2015)
- Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
- TOEFL dataset: Levied is closest in meaning to: imposed, believed, requested, correlated

Properties of embeddings

Similarity depends on window size C

C = ±2 The nearest words to Hogwarts:
- Sunnydale
- Evernight

C = ±5 The nearest words to Hogwarts:
- Dumbledore
- Malfoy
- halfblood
Analogy: Embeddings capture relational meaning!

vector('king') - vector('man') + vector('woman') = vector('queen')

vector('Paris') - vector('France') + vector('Italy') ≈ vector('Rome')

[Diagram of word vectors and relationships]
Embeddings can help study word history!

Train embeddings on old books to study changes in word meaning!!
Word embeddings for studying language change!

Visualizing changes

Project 300 dimensions down into 2

~30 million books, 1850-1990, Google Books data
Embeddings and bias

Embeddings reflect cultural bias


Ask “Paris : France :: Tokyo : x”
- x = Japan

Ask “father : doctor :: mother : x”
- x = nurse

Ask “man : computer programmer :: woman : x”
- x = homemaker
Embeddings reflect cultural bias


Implicit Association test (Greenwald et al 1998): How associated are
- concepts (flowers, insects) & attributes (pleasantness, unpleasantness)?
- Studied by measuring timing latencies for categorization.

Psychological findings on US participants:
- African-American names are associated with unpleasant words (more than European-American names)
- Male names associated more with math, female names with arts
- Old people's names with unpleasant words, young people with pleasant words.

Caliskan et al. replication with embeddings:
- African-American names (Leroy, Shaniqua) had a higher GloVe cosine with unpleasant words (abuse, stink, ugly)
- European American names (Brad, Greg, Courtney) had a higher cosine with pleasant words (love, peace, miracle)

Embeddings reflect and replicate all sorts of pernicious biases.

Directions

Debiasing algorithms for embeddings

Use embeddings as a historical tool to study bias
Embeddings as a window onto history

The cosine similarity of embeddings for decade X for occupations (like teacher) to male vs female names

- Is correlated with the actual percentage of women teachers in decade X

History of biased framings of women

Embeddings for competence adjectives are biased toward men

- *Smart, wise, brilliant, intelligent, resourceful, thoughtful, logical, etc.*

This bias is slowly decreasing
Embeddings reflect ethnic stereotypes over time

- Princeton trilogy experiments
- Attitudes toward ethnic groups (1933, 1951, 1969) scores for adjectives
  - *industrious, superstitious, nationalistic*, etc
- Cosine of Chinese name embeddings with those adjective embeddings correlates with human ratings.


Change in linguistic framing 1910-1990

Change in association of Chinese names with adjectives framed as "othering" *(barbaric, monstrous, bizarre)*

Changes in framing: adjectives associated with Chinese


<table>
<thead>
<tr>
<th></th>
<th>1910</th>
<th>1950</th>
<th>1990</th>
</tr>
</thead>
<tbody>
<tr>
<td>Irresponsible</td>
<td>Disorganized</td>
<td>Inhibited</td>
<td></td>
</tr>
<tr>
<td>Envious</td>
<td>Outrageous</td>
<td>Passive</td>
<td></td>
</tr>
<tr>
<td>Barbaric</td>
<td>Pompous</td>
<td>Dissolute</td>
<td></td>
</tr>
<tr>
<td>Aggressive</td>
<td>Unstable</td>
<td>Haughty</td>
<td></td>
</tr>
<tr>
<td>Transparent</td>
<td>Effeminate</td>
<td>Complacent</td>
<td></td>
</tr>
<tr>
<td>Monstrous</td>
<td>Unprincipled</td>
<td></td>
<td>Forceful</td>
</tr>
<tr>
<td>Hateful</td>
<td>Venomous</td>
<td>Fixed</td>
<td></td>
</tr>
<tr>
<td>Cruel</td>
<td>Disobedient</td>
<td>Active</td>
<td></td>
</tr>
<tr>
<td>Greedy</td>
<td>Predatory</td>
<td>Sensitive</td>
<td></td>
</tr>
<tr>
<td>Bizarre</td>
<td>Boisterous</td>
<td>Hearty</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

**Concepts** or word senses
- Have a complex many-to-many association with **words** (homonymy, multiple senses)
- Have relations with each other
  - Synonymy, Antonymy, Superordinate
- But are hard to define formally (necessary & sufficient conditions)

**Embeddings** = vector models of meaning
- More fine-grained than just a string or index
- Especially good at modeling similarity/analogy
  - Just download them and use cosines!!
- Can use sparse models (tf-idf) or dense models (word2vec, GLoVE)
- Useful in practice but know they encode cultural stereotypes