Dense Word Embeddings

CMSC 470
Marine Carpuat

Slides credit: Jurasky & Martin
How to generate vector embeddings?
One approach: feedforward neural language models

Training a neural language model just to get word embeddings is expensive!

Is there a faster/cheaper way to get word embeddings if we don’t need the language model?
Roadmap

• Dense vs. sparse word embeddings

• Generating word embeddings with Word2vec
  • Skip-gram model
  • Training

• Evaluating word embeddings
  • Word similarity
  • Word relations
  • Analysis of biases
Word embedding methods we’ve seen so far yield 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)
Why short dense vectors?

• Short vectors may be easier to use as features in machine learning (fewer 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**  
https://code.google.com/archive/p/word2vec/

**Fasttext**  
http://www.fasttext.cc/

**Glove**  
http://nlp.stanford.edu/projects/glove/
Word2vec

- Popular embedding method
- Very fast to train
- Code available on the web
- Key idea: predict rather than count
Word2vec

Approach:
• 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"?

Note: we don’t actually care about this task!
But we'll take the learned classifier weights as the word embeddings
Insight: running text provides 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**
  - Bengio et al. (2003)
  - Collobert et al. (2011)
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 Task

• Given a tuple \((t, c)\) = target, context
  
  \((\text{apricot}, \text{jam})\)
  
  \((\text{apricot}, \text{aardvark})\)

• Return probability that \(c\) is a real context word:
  
  • \(P( + | t, c)\)
  
  • \(P(– | t, c) = 1–P( + | t, c)\)
Skip-Gram Training Data

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

• Training sentence:

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

  \( c_1 \quad c_2 \quad \text{target} \quad c_3 \quad c_4 \)
How to compute $p(+| t, c)$?

• **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:

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]
Turning dot product into a probability

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

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

This is a logistic regression model!
For all the context words:

• Assume all context words are independent

\[
P(+|t,c_1:k) = \prod_{i=1}^{\kappa} \frac{1}{1 + e^{-t \cdot c_i}}
\]

\[
\log P(+|t,c_1:k) = \sum_{i=1}^{k} \log \frac{1}{1 + e^{-t \cdot c_i}}
\]
Skip-Gram Training Data

• Training sentence:
  ... lemon, a tablespoon of apricot jam a pinch ... 
  \[ c_1 \quad c_2 \quad t \quad c_3 \quad c_4 \]

• Training data: input/output pairs centering on apricot
• Assume a +/- 2 word window
Skip-Gram Training

• Training sentence:
... lemon, a tablespoon of apricot jam a pinch ...

\[ \text{c1} \quad \text{c2} \quad \text{t} \quad \text{c3} \quad \text{c4} \]

positive examples +
\[
\begin{array}{cc}
t & c \\
\hline
\text{apricot} & \text{tablespoon} \\
\text{apricot} & \text{of} \\
\text{apricot} & \text{preserves} \\
\text{apricot} & \text{or} \\
\end{array}
\]

• For each positive example, we'll create \( k \) negative examples.
• Using \textit{noise} words
• Any random word that isn't \( t \)
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 -</th>
</tr>
</thead>
<tbody>
<tr>
<td>t       c</td>
<td>t       c</td>
</tr>
<tr>
<td>apricot  tablespoon</td>
<td>apricot  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>

k=2
Choosing noise words

• Could pick $w$ according to their unigram frequency $P(w)$
• More common to chosen then according to $p_\alpha(w)$

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

• $\alpha = \frac{3}{4}$ works well because it gives rare noise words slightly higher probability
  • imagine two events $p(a) = .99$ and $p(b) = .01$:
    $$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$
    $$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$
Skip-gram: training set-up

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

• So we start with $300 \times V$ random parameters and use gradient descent to update these parameters.

• We need to define a loss function / training objective.
Skip-gram: training objective

- Motivation: Over the entire training set, we’d like to adjust those word vectors such that we
  - Maximize the similarity of the positive target word, context word pairs (t,c)
  - Minimize the similarity of the negative (t,c) pairs

- Objective: 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.
Skip-gram: training objective

• Focusing on one target word \( t \)

\[
L(\theta) = \log P(+|t, c) + \sum_{i=1}^{k} \log P(-|t, n_i) \\
= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t) \\
= \log \frac{1}{1 + e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1 + e^{n_i \cdot t}}
\]
Skip-gram illustrated

“...apricot jam...”

increase similarity(apricot, jam) $w_j \cdot c_k$

decrease similarity(apricot, aardvark) $w_j \cdot c_n$
Summary: How to learn word2vec (skip-gram) embeddings

- Choose the embedding dimension, e.g., d=300
- Start with V random 300-dimensional vectors as initial embeddings
- Take a corpus and take pairs of words that co-occur as positive examples
- Construct negative examples
- Train a logistic regression classifier to distinguish positive from negative examples
- Throw away the classifier and keep the embeddings!
Evaluating embeddings

• We can use the same evaluations as for other distributional semantic models (see lecture 2)

• 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*
Analogy: Embeddings capture relational meaning!

\[ \text{vector(‘king’) - vector(‘man’) + vector(‘woman’) } \approx \text{ vector(‘queen’) } \]
\[ \text{vector(‘Paris’) - vector(‘France’) + vector(‘Italy’) } \approx \text{ vector(‘Rome’) } \]
Word embeddings are a very useful tool

- Can be used as features in classifiers
  - Capture generalizations across word types

- Can be used to analyze language usage patterns in large corpora
  - E.g., to study change in word meaning
“dog” 1920 word vector vs. “dog” 1990 word vector
Yet word embeddings are not perfect models of word meaning

• Limitations include
  • One vector per word (even if the word has multiple senses)
  • Cosine similarity not sufficient to distinguish antonyms from synonyms
  • Embeddings reflect cultural bias implicit in training text
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

- 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 had a higher cosine with unpleasant words
  - European American names had a higher cosine with pleasant words
- Embeddings reflect and replicate all sorts of pernicious biases.

So what can we do about bias?

• Attempt to remove or decrease bias by “debiasing” for embeddings

• Use embeddings as a historical tool to study bias
  • Garg, Nikhil, Schiebinger, Londa, Jurafsky, Dan, and Zou, James (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 115(16), E3635–E3644
Roadmap

• Dense vs. sparse word embeddings

• Generating word embeddings with Word2vec
  • Skip-gram model
  • Training

• Evaluating word embeddings
  • Word similarity
  • Word relations
  • Analysis of biases