Lecture 10: Capturing semantics: Word Embeddings

Sanjeev Arora       Elad Hazan

(Borrows from slides of D. Jurafsky Stanford U.)
Last time

N-gram language models. $\Pr[w_2 \mid w_1]$

(Assign a probability to each sentence; trained using an unlabeled corpus)

Unsupervised learning

Today: Unsupervised learning for semantics (meaning)
Semantics

Study of meaning of linguistic expressions.

Mastering and fluently operating with it seems a precursor to AI (Turing test, etc.)

But every bit of partial progress can immediately be used to improve technology:

• Improving web search.
• Siri, etc.
• Machine translation
• Information retrieval,..
Let’s talk about meaning

What does the sentence “You like green cream” mean?
“Come up and see me some time.”
How did words arrive at their meanings?

Your answers seem to involve a mix of

• Grammar/syntax
• How words map to physical experience
• Physical sensations “sweet”, “breathing”, “pain” etc. and mental states “happy,” “regretful” appear to be experienced roughly the same way by everybody, which help anchor some meanings)
•
•
• (at some level, becomes philosophy).
What are simple tests for understanding meaning?

Which of the following means same as pulchritude: 
(a) Truth (b) Disgust (c) Anger (d) Beauty?

Analogy questions: 
Man : Woman :: King : ??

“How can we teach a computer the notion of word similarity?”
Test: Think of a word that co-occurs with: 
*Cow, drink, babies, calcium*...

**Distributional hypothesis of meaning, [Harris’54], [Firth’57]**

Meaning of a word is determined by words it co-occurs with.

“Tiger” and “Lion” are similar because they cooccur with similar words (“jungle”, “hunt”, “predator”, “claws”, etc.)

A computer could learn similarity by simply reading a text corpus; no need to wait for full AI!
How can we quantify “distribution of nearby words”?

A bottle of *tesgüíno* is on the table
Everybody likes *tesgüíno*
*Tesgüíno* makes you drunk
We make *tesgüíno* out of corn.

Recall bigram frequency $P(\text{beer, drunk}) = \frac{\text{count(beer, drunk)}}{\text{Corpus size}}$

Redefine

$\text{count(beer, drunk)} = \# \text{ of times beer and drunk appear within 5 words of each other.}$
Matrix of all such bigram counts (# columns = V = 10^6)

<table>
<thead>
<tr>
<th></th>
<th>study</th>
<th>alcohol</th>
<th>drink</th>
<th>bottle</th>
<th>lion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beer</td>
<td>35</td>
<td>552</td>
<td>872</td>
<td>677</td>
<td>2</td>
</tr>
<tr>
<td>Milk</td>
<td>45</td>
<td>20</td>
<td>935</td>
<td>250</td>
<td>0</td>
</tr>
<tr>
<td>Jungle</td>
<td>12</td>
<td>35</td>
<td>110</td>
<td>28</td>
<td>931</td>
</tr>
<tr>
<td>Tesgüino</td>
<td>2</td>
<td>67</td>
<td>75</td>
<td>57</td>
<td>0</td>
</tr>
</tbody>
</table>

Which words look most similar to each other?

"Distributional hypothesis" → Word similarity boils down to vector similarity.

Vector representation of "beer."

Semantic Embedding [Deerwester et al 1990]

"Ignore bigrams with frequent words like "and", "for"
Cosine similarity

“Rescale vectors to unit length, compute dot product.”

\[ |\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2} \]

\[ \text{dot-product}(\vec{v}, \vec{w}) = \vec{v} \cdot \vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N \]

- High when two vectors have large values in same dimensions.
- Low (in fact 0) for orthogonal vectors with zeros in complementary distribution

Cosine Similarity \( \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|} \)

\[ \vec{a} \cdot \vec{b} = |\vec{a}| |\vec{b}| \cos \theta \]

\[ \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|} = \cos \theta \]

Other measures of similarity have been defined....
Reminder about properties of cosine

- -1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal
Nearest neighbors (cosine similarity)

<table>
<thead>
<tr>
<th>Apple:</th>
<th>Ball:</th>
</tr>
</thead>
<tbody>
<tr>
<td>'macintosh'</td>
<td>'balls'</td>
</tr>
<tr>
<td>'microsoft'</td>
<td>'game'</td>
</tr>
<tr>
<td>'iphone'</td>
<td>'play'</td>
</tr>
<tr>
<td>'ibm'</td>
<td>'thrown'</td>
</tr>
<tr>
<td>'software'</td>
<td>'back'</td>
</tr>
<tr>
<td>'ios'</td>
<td>'hit'</td>
</tr>
<tr>
<td>'ipad'</td>
<td>'throwing'</td>
</tr>
<tr>
<td>'ipod'</td>
<td>'player'</td>
</tr>
<tr>
<td>'apples'</td>
<td>'throw'</td>
</tr>
<tr>
<td>'fruit'</td>
<td>'catch'</td>
</tr>
</tbody>
</table>

Important use of embeddings: allow language processing systems to make a guess when labeled data is insufficient. (borrow data/trained features of nearby words)

(Semisupervised learning:
Learn embeddings from large unlabeled corpus; use as help for supervised learning)
Anthropology field trip: visualization of word vectors obtained from dating profiles

[Loren Shure, matworks.com]
“Evolution of new word meanings”
Kulkarni, Al-Rfou, Perozzi, Skiena 2015
Problems with above naive embedding method

- Raw word frequency is not a great measure of association between words
  - It’s very skewed
    - “the” and “of” are very frequent, but maybe not the most discriminative
- We’d rather have a measure that asks whether a context word is particularly informative about the target word.
  - Positive Pointwise Mutual Information (PPMI)
Pointwise Mutual Information (PMI)

**Pointwise mutual information:**
Do events $x$ and $y$ co-occur more than if they were independent?

**PMI between two words:** (Church & Hanks 1989)
Do words $x$ and $y$ co-occur more than if they were independent?

$$\text{PMI}(\text{word}_1, \text{word}_2) = \log_2 \frac{P(\text{word}_1, \text{word}_2)}{P(\text{word}_1)P(\text{word}_2)}$$

**Positive PMI:** $\max\{0, \text{PMI}\}$ (if PMI is -ve, make it 0).

High for (lion, hunt) but $\approx 0$ for (lion, cake). (Why?)
Improved embedding

<table>
<thead>
<tr>
<th></th>
<th>study</th>
<th>alcohol</th>
<th>drink</th>
<th>bottle</th>
<th>lion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beer</td>
<td>35</td>
<td>552</td>
<td>872</td>
<td>677</td>
<td>2</td>
</tr>
<tr>
<td>Milk</td>
<td>45</td>
<td>20</td>
<td>935</td>
<td>250</td>
<td>0</td>
</tr>
<tr>
<td>Jungle</td>
<td>12</td>
<td>35</td>
<td>110</td>
<td>28</td>
<td>931</td>
</tr>
<tr>
<td>Tesgüino</td>
<td>2</td>
<td>67</td>
<td>75</td>
<td>57</td>
<td>0</td>
</tr>
</tbody>
</table>

Replace each entry by corresponding PPMI(word1, word2)

Better: Before computing PPMI( , ), do Add-1 smoothing from last time. (reduces crazy effects due to rare words)
Implementation issues

• V = vocabulary size (say 100,000)

• Then word embeddings defined above are V-dimensional. Very clunky to compute with! (Will improve soon.)

• We defined bigrams using “windows of size 5”
  
  • The shorter the windows, the more **syntactic** the representation
    ± 1-3 very syntacticaly
  
  • The longer the windows, the more **semantic** the representation
    ± 4-10 more semanticly
Dense word embeddings.

- PPMI vectors are
  - **long** (dimension $|V| = 20,000$ to $50,000$)
  - **sparse** (most elements are zero)
- Alternative: learn vectors which are
  - **short** (dimension 200-1000)
  - **dense** (most elements are non-zero)

Each dimension in dense version represents many old dimensions (but it isn’t some trivial consolidation)
Why dense embeddings

- Short vectors may be easier to use as features in downstream machine learning (fewer dimensions – fewer parameters to tune)
- Dense vectors may generalize better than storing explicit counts (perform better at many tasks)

- Sparse long vectors ignore synonymy:
  - *car* and *automobile* are synonyms; but are represented as distinct coordinates; this fails to capture similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor
How to compute dense embeddings.

For each word, seeking a vector $v_w$ in $\mathbb{R}^{300}$ (nothing special about 300; could be 400)

For every pair of words $w, w'$ desire

$$v_w \cdot v_{w'} \approx \text{PPMI}(w, w')$$

Training with $l_2$ loss: Minimize

$$\sum_{w, w'} (v_w \cdot v_{w'} - \text{PPMI}(w, w'))^2$$

Aside: this optimization is called “Rank-300 SVD” in linear algebra. (aka Principle Component Analysis)
Even better embeddings?

\[
\text{Minimize } \sum_{w, w'} \left( v_w \cdot v_{w'} - \text{PPMI}(w, w') \right)^2
\]

Problem with this compression: gives equal importance to every matrix entry.

Recall: PPMI( ) estimated using cooccurrence counts. Are some estimates more reliable than others?

Better: Minimize \[
\sum_{w, w'} \text{Count}(w, w') \left( v_w \cdot v_{w'} - \text{PPMI}(w, w') \right)^2
\]

Aside 1: Theoretical justification for this objective in [A, Li, Liang, Ma, Risteski TACL 2016].
Aside 2: Many other related methods for compressed embeddings: word2vec, Glove, neural nets,..)
How to solve training objective

Minimize $\sum_{w,w'} \text{Count}(w, w')(v_w \cdot v_{w'} - \text{PPMI}(w, w'))^2$

Suggestions?

Gradient descent works. Number of variables is 300 V. Objective is a quadratic function of these variables.

dot-product($\vec{v}, \vec{w}$) = $\vec{v} \cdot \vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$
Cool property of PMI-based embeddings: Analogy solving

Man : Woman :: King : ??

Find \( w \) to minimize

\[
\|v_{\text{man}} - v_{\text{woman}} + v_{\text{king}} - v_{w}\|_2
\]
Some other cool applications
(don’t expect you to fully understand; details won’t be on exam)
Extending knowledge bases

Given the list Kyoto, Shanghai, Chongqing, Chengdu, Busan, Incheon,... generate more examples.

(Use a large corpus like WSJ or wikipedia)

Solution: Estimate the “best line capturing their embeddings” (rank 1 SVD); find other words whose embeddings are close to this line.

<table>
<thead>
<tr>
<th>word</th>
<th>projection</th>
<th>word</th>
<th>projection</th>
<th>word</th>
<th>projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>taipei</td>
<td>0.837</td>
<td>horses</td>
<td>0.806</td>
<td>dwyane_wade</td>
<td>0.788</td>
</tr>
<tr>
<td>taichung</td>
<td>0.819</td>
<td>moose</td>
<td>0.784</td>
<td>aren</td>
<td>0.721</td>
</tr>
<tr>
<td>kaohsiung</td>
<td>0.818</td>
<td>elk</td>
<td>0.783</td>
<td>kobe_bryant</td>
<td>0.715</td>
</tr>
<tr>
<td>osaka</td>
<td>0.806</td>
<td>raccoon</td>
<td>0.763</td>
<td>chris_bosh</td>
<td>0.712</td>
</tr>
<tr>
<td>tianjin</td>
<td>0.765</td>
<td>goats</td>
<td>0.762</td>
<td>tim_duncan</td>
<td>0.708</td>
</tr>
</tbody>
</table>
Resolving meanings of polysemous words

[Li, Liang, Ma, Risteski’16]

Polysemous: Has multiple meanings.

Example: How many meanings does “tie” have? “Spring?”

<table>
<thead>
<tr>
<th></th>
<th>tie</th>
<th>spring</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>trousers blouse</td>
<td>beginning dampers</td>
</tr>
<tr>
<td></td>
<td>waistcoat skirt</td>
<td>until brakes</td>
</tr>
<tr>
<td></td>
<td>sleeved pants</td>
<td>months suspension</td>
</tr>
<tr>
<td></td>
<td>season teams</td>
<td>earlier absorbers</td>
</tr>
<tr>
<td></td>
<td>winning league</td>
<td>year wheels</td>
</tr>
<tr>
<td></td>
<td>finished</td>
<td>last damper</td>
</tr>
<tr>
<td></td>
<td>championship</td>
<td></td>
</tr>
<tr>
<td></td>
<td>scoreline</td>
<td>flower flowers</td>
</tr>
<tr>
<td></td>
<td>goalless</td>
<td>flowering fragrant</td>
</tr>
<tr>
<td></td>
<td>equaliser</td>
<td>lilies flowered</td>
</tr>
<tr>
<td></td>
<td>clinching</td>
<td></td>
</tr>
<tr>
<td></td>
<td>scoreless</td>
<td></td>
</tr>
<tr>
<td></td>
<td>replay</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wires</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wiring</td>
<td></td>
</tr>
<tr>
<td></td>
<td>electrical</td>
<td></td>
</tr>
<tr>
<td></td>
<td>wire</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>operatic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>soprano</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mezzo</td>
<td></td>
</tr>
<tr>
<td></td>
<td>contralto</td>
<td></td>
</tr>
<tr>
<td></td>
<td>baritone</td>
<td></td>
</tr>
<tr>
<td></td>
<td>coloratura</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>creep brook</td>
</tr>
<tr>
<td></td>
<td></td>
<td>humid winters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>summers ppen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>warm temperatures</td>
</tr>
</tbody>
</table>

Meanings extracted using a linear-algebraic procedure called sparse coding.
Allow programs like machine translation to deal with unknown words

Translator trained using transcripts of Canadian parliament  
(large corpus, but not huge)

Encounters a word that doesn’t occur in training corpus (eg, “dude”)

Can use word embeddings (trained on a large English-only corpus like wikipedia) 
to learn a lot about “dude,” and neighboring words \( \rightarrow \) can guess approximate translation of “dude”.
Sentence embeddings

Similar

a man with a jersey is dunking the ball at a basketball game
the ball is being dunked by a man with a jersey at a basketball game

Dissimilar

people wearing costumes are gathering in a forest and are looking in the same direction
a little girl in costume looks like a woman

Simplest sentence embedding = Average of word embeddings.
(Lots of research to find better embeddings, including using neural nets.)
Next lecture: Collaborative filtering

(eg movie or music recommender systems)

Homework is out today; due next Tues

Midterm: A week from Thurs.