Vector Semantics

Natural Language Processing
Lecture 16

Adapted from Jurafsky and Martin, 3rd ed.
Why vector models of meaning? computing the similarity between words

“fast” is similar to “rapid”
“tall” is similar to “height”

Question answering:

Q: “How tall is Mt. Everest?”
Candidate A: “The official height of Mount Everest is 29029 feet”
Word similarity for plagiarism detection

**MAINFRAMES**

Mainframes are primarily referred to large computers with rapid, advanced processing capabilities that can execute and perform tasks equivalent to many Personal Computers (PCs) machines networked together. It is characterized with high quantity Random Access Memory (RAM), very large secondary storage devices, and high-speed processors to cater for the needs of the computers under its service.

Consisting of advanced components, mainframes have the capability of running multiple large applications required by many and most enterprises and organizations. This is one of its advantages. Mainframes are also suitable to cater for those applications (programs) or files that are of very high demand by its users (clients).

Examples of such organizations and enterprises using mainframes are online shopping websites such as Ebay, Amazon, and computing-giant.

**MAINFRAMES**

Mainframes usually are referred those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater for the needs of those computers under its service.

Due to the advanced components mainframes have, these computers have the capability of running multiple large applications required by most enterprises, which is one of its advantage. Mainframes are also suitable to cater for those applications or files that are of very large demand by its users (clients). Examples of these include the large online shopping websites -i.e. : Ebay, Amazon, Microsoft, etc.
Word similarity for historical linguistics: semantic change over time

Sagi, Kaufmann Clark 2013

Kulkarni, Al-Rfou, Perozzi, Skiena 2015

![Semantic Broadening Chart]

- dog
- deer
- hound
Problems with thesaurus-based meaning

• We don’t have a thesaurus for every language
• We can’t have a thesaurus for every year
  • For historical linguistics, we need to compare word meanings in year $t$ to year $t+1$
• Thesauruses have problems with recall
  • Many words and phrases are missing
  • Thesauri work less well for verbs, adjectives
Distributional models of meaning
= vector-space models of meaning
= vector semantics

Intuitions:
• Zellig Harris (1954):
  o “oculist and eye-doctor ... occur in almost the same environments”
  o “If A and B have almost identical environments we say that they are synonyms.”
• Firth (1957):
  o “You shall know a word by the company it keeps!”
Intuition of distributional word similarity

• Nida example: Suppose I asked you “what is tesgüino?”

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

• From context words humans can guess *tesgüino* means
  - an alcoholic beverage like beer

• Intuition for algorithm:
  - Two words are similar if they have similar word contexts.
Several kinds of vector models

Sparse vector representations

1. Mutual-information weighted word co-occurrence matrices

Dense vector representations:

2. Singular value decomposition (and Latent Semantic Analysis)
3. Neural-network-inspired models (skip-grams, CBOW)
4. ELMo and BERT
5. Brown clusters
Shared intuition

• Model the meaning of a word by “embedding” in a vector space.
• The meaning of a word is a vector of numbers
  • Vector models are also called “embeddings”.
• **Contrast:** word meaning is represented in many computational linguistic applications by a vocabulary index (“word number 545”)
• Old philosophy joke:
  Q: What’s the meaning of life?
  A: LIFE’
Vector Semantics

Words and co-occurrence vectors
Co-occurrence Matrices

• We represent how often a word occurs in a document
  • Term-document matrix
• Or how often a word occurs with another
  • Term-term matrix
    (or word-word co-occurrence matrix
     or word-context matrix)
Term-document matrix

• Each cell: count of word $w$ in a document $d$:
  • Each document is a count vector in $\mathbb{N}^v$: a column below

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>soldier</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>fool</td>
<td>37</td>
<td>58</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>clown</td>
<td>6</td>
<td>117</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Similarity in term-document matrices

Two documents are similar if their vectors are similar

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>soldier</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>fool</td>
<td>37</td>
<td>58</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>clown</td>
<td>6</td>
<td>117</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The words in a term-document matrix

• Each word is a count vector in $\mathbb{N}^D$: a row below

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>soldier</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>fool</td>
<td>37</td>
<td>58</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>clown</td>
<td>6</td>
<td>117</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The words in a term-document matrix

- **Two words** are similar if their vectors are similar

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>soldier</td>
<td>2</td>
<td>2</td>
<td>12</td>
<td>36</td>
</tr>
<tr>
<td>fool</td>
<td>37</td>
<td>58</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>clown</td>
<td>6</td>
<td>117</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The word-word or word-context matrix

• Instead of entire documents, use smaller contexts
  • Paragraph
  • Window of ± 4 words
• A word is now defined by a vector over counts of context words
  • Instead of each vector being of length D, each vector is now of length |V|
• The word-word matrix is |V| x |V|
Word-Word matrix
Sample contexts ± 7 words

sugar, a sliced lemon, a tablespoonful of their enjoyment. Cautiously she sampled her first well suited to programming on the digital for the purpose of gathering data and apricot pineapple computer. information preserve or jam, a pinch each of, and another fruit whose taste she likened In finding the optimal R-stage policy from necessary for the study authorized in the

<table>
<thead>
<tr>
<th></th>
<th>aardvark</th>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>pineapple</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>digital</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>information</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 19.2 Co-occurrence vectors for four words, computed from the Brown corpus, showing only six of the dimensions (hand-picked for pedagogical purposes). Note that a real vector would be vastly more sparse.
Word-word matrix

• We showed only 4x6, but the real matrix is 50,000 x 50,000
  • So it’s very **sparse**: Most values are 0.
  • That’s OK, since there are lots of efficient algorithms for sparse matrices.
• The size of windows depends on your goals
  • The shorter the windows, the more **syntactic** the representation
    ± 1-3 very syntaxy
  • The longer the windows, the more **semantic** the representation
    ± 4-10 more semanticky
2 kinds of co-occurrence between 2 words
(Schütze and Pedersen, 1993)

• First-order co-occurrence (syntagmatic association):
  • They are typically nearby each other.
  • wrote is a first-order associate of book or poem.

• Second-order co-occurrence (paradigmatic association):
  • They have similar neighbors.
  • wrote is a second-order associate of words like said or remarked.
Vector Semantics

Positive Pointwise Mutual Information (PPMI)
Problem with raw counts

• 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

**Pointwise mutual information:**

Do events $x$ and $y$ co-occur more than if they were independent?

$$\text{PMI}(X,Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

**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 Pointwise Mutual Information

• PMI ranges from $-\infty$ to $+\infty$
• But the negative values are problematic
  • Things are co-occurring less than we expect by chance
  • Unreliable without enormous corpora
    • Imagine $w_1$ and $w_2$ whose probability is each $10^{-6}$
    • Hard to be sure $p(w_1, w_2)$ is significantly different than $10^{-12}$
• Plus it’s not clear people are good at “unrelatedness”
• So we just replace negative PMI values by 0
• Positive PMI (PPMI) between word1 and word2:
  \[
  \text{PPMI}(\text{word}_1, \text{word}_2) = \max \left( \log_2 \frac{P(\text{word}_1, \text{word}_2)}{P(\text{word}_1)P(\text{word}_2)}, 0 \right)
  \]
Computing PPMI on a term-context matrix

- Matrix $F$ with $W$ rows (words) and $C$ columns (contexts)
- $f_{ij}$ is # of times $w_i$ occurs in context $c_j$

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij}}$$

$$p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij}}$$

$$p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij}}$$

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

$$ppmi_{ij} = \begin{cases} 
  pmi_{ij} & \text{if } pmi_{ij} > 0 \\
  0 & \text{otherwise}
\end{cases}$$
\[ p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{w} \sum_{j=1}^{c} f_{ij}} \]

\[
p(w=\text{information}, c=\text{data}) = \frac{6}{19} = 0.32
\]
\[
p(w=\text{information}) = \frac{11}{19} = 0.58
\]
\[
p(c=\text{data}) = \frac{7}{19} = 0.37
\]

<table>
<thead>
<tr>
<th>Count(w,context)</th>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>pineapple</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>digital</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>information</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
p(w_i) = \frac{\sum_{j=1}^{c} f_{ij}}{N}
\]

\[
p(c_j) = \frac{\sum_{i=1}^{w} f_{ij}}{N}
\]
\[ pmi_{ij} = \log_2 \frac{p_{ij}}{p_i \cdot p_j} \]

- \[ pmi(\text{information}, \text{data}) = \log_2 \left( \frac{0.05}{0.00 \cdot 0.57} \right) = 0.58 \]

<table>
<thead>
<tr>
<th></th>
<th>PPMI(w,context)</th>
<th>p(w,context)</th>
<th>p(w)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>computer</td>
<td>data</td>
<td>pinch</td>
</tr>
<tr>
<td>apricot</td>
<td>0.00</td>
<td>0.00</td>
<td>0.05</td>
</tr>
<tr>
<td>pineapple</td>
<td>0.00</td>
<td>0.00</td>
<td>0.05</td>
</tr>
<tr>
<td>digital</td>
<td>0.11</td>
<td>0.05</td>
<td>0.00</td>
</tr>
<tr>
<td>information</td>
<td>0.05</td>
<td>0.32</td>
<td>0.00</td>
</tr>
<tr>
<td>[ p(\text{context}) ]</td>
<td>0.16</td>
<td>0.37</td>
<td>0.11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PPMI</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>PPMI(w,context)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apricot</td>
<td></td>
<td>2.25</td>
</tr>
<tr>
<td>pineapple</td>
<td></td>
<td>2.25</td>
</tr>
<tr>
<td>digital</td>
<td>1.66</td>
<td>0.00</td>
</tr>
<tr>
<td>information</td>
<td>0.00</td>
<td>0.57</td>
</tr>
</tbody>
</table>
Weighting PMI

• PMI is biased toward infrequent events
  • Very rare words have very high PMI values

• Two solutions:
  • Give rare words slightly higher probabilities
  • Use add-one smoothing (which has a similar effect)
Weighting PMI: Giving rare context words slightly higher probability

• Raise the context probabilities to $\alpha = 0.75$:

$$\text{PPMI}_\alpha(w, c) = \max(\log_2 \frac{P(w, c)}{P(w)P_\alpha(c)}, 0)$$

$$P_\alpha(c) = \frac{\text{count}(c)^\alpha}{\sum_c \text{count}(c)^\alpha}$$

• This helps because $P_\alpha(c) \gg P(c)$ for rare $c$

• Consider two events, $P(a) = .99$ and $P(b) = .01$

  - $P_\alpha(a) = \frac{.99^{.75}}{.99^{.75}+.01^{.75}} = .97$
  - $P_\alpha(b) = \frac{.01^{.75}}{.01^{.75}+.01^{.75}} = .03$
Use Laplace (add-\(k\)) smoothing

<table>
<thead>
<tr>
<th>Add-2 Smoothed Count(w,context)</th>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>pineapple</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>digital</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>information</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p(w,context) [add-2]</th>
<th>p(w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td></td>
</tr>
<tr>
<td>apricot</td>
<td>0.03</td>
</tr>
<tr>
<td>pineapple</td>
<td>0.03</td>
</tr>
<tr>
<td>digital</td>
<td>0.07</td>
</tr>
<tr>
<td>information</td>
<td>0.05</td>
</tr>
<tr>
<td>p(context)</td>
<td>0.19</td>
</tr>
</tbody>
</table>
PPMI versus add-2 smoothed PPMI

<table>
<thead>
<tr>
<th></th>
<th>PPMI(w,context)</th>
<th>PPMI(w,context) [add-2]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>computer</td>
<td>data</td>
</tr>
<tr>
<td>apricot</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>pineapple</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>digital</td>
<td>1.66</td>
<td>0.00</td>
</tr>
<tr>
<td>information</td>
<td>0.00</td>
<td>0.57</td>
</tr>
</tbody>
</table>
Vector Semantics

Measuring similarity: the cosine
Measuring similarity

- Given 2 target words $v$ and $w$
- We’ll need a way to measure their similarity.
- Most measure of vectors similarity are based on the:
  - Dot product or inner product from linear algebra

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

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

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

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

• Dot product is larger if the vector is longer. Vector length:

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

• Vectors are longer if they have higher values in each dimension
  • That means more frequent words will have higher dot products
  • That’s bad: we don’t want a similarity metric to be sensitive to word frequency
Solution: cosine

• Just divide the dot product by the length of the two vectors!

\[
\frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|}
\]

• This turns out to be the cosine of the angle between them!

\[
\frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|} = \frac{||\vec{a}|||\vec{b}| \cos \theta}{|\vec{a}| |\vec{b}|} = \cos \theta
\]
Cosine for computing similarity

\[
\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \|\vec{w}\|} = \frac{\vec{v}}{\|\vec{v}\|} \cdot \frac{\vec{w}}{\|\vec{w}\|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}
\]

- \(v_i\) is the PPMI value for word \(v\) in context \(i\)
- \(w_i\) is the PPMI value for word \(w\) in context \(i\).

\(\text{Cos}(\vec{v}, \vec{w})\) is the cosine similarity of \(\vec{v}\) and \(\vec{w}\)
Cosine as a similarity metric

• -1: vectors point in opposite directions
• +1: vectors point in same directions
• 0: vectors are orthogonal

• Raw frequency or PPMI are non-negative, so cosine range 0-1
\[
\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\vec{v} \cdot \vec{w}}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}
\]

Which pair of words is more similar?
\[
\cos(\text{apricot, information}) = \frac{2 + 0 + 0}{\sqrt{2 + 0 + 0} \sqrt{1 + 36 + 1}} = \frac{2}{\sqrt{2} \sqrt{38}} = .23
\]
\[
\cos(\text{digital, information}) = \frac{0 + 6 + 2}{\sqrt{0 + 1 + 4} \sqrt{1 + 36 + 1}} = \frac{8}{\sqrt{38} \sqrt{5}} = .58
\]
\[
\cos(\text{apricot, digital}) = \frac{0 + 0 + 0}{\sqrt{1 + 0 + 0} \sqrt{0 + 1 + 4}} = 0
\]

<table>
<thead>
<tr>
<th></th>
<th>large</th>
<th>data</th>
<th>computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>digital</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>information</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
Visualizing vectors and angles

<table>
<thead>
<tr>
<th></th>
<th>large</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>digital</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>information</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Dimension 1: ‘large’

Dimension 2: ‘data’
Clustering vectors to visualize similarity in co-occurrence matrices

Rohde, Gonnerman, Plaut Modeling Word Meaning Using Lexical Co-occurrence

Rohde et al. (2006)
Other possible similarity measures

\[
\text{sim}_{\text{cosine}}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}
\]

\[
\text{sim}_{\text{Jaccard}}(\vec{v}, \vec{w}) = \frac{\sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} \max(v_i, w_i)}
\]

\[
\text{sim}_{\text{Dice}}(\vec{v}, \vec{w}) = \frac{2 \times \sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} (v_i + w_i)}
\]

\[
\text{sim}_{\text{JS}}(\vec{v}||\vec{w}) = D\left(\vec{v} \mid \frac{\vec{v} + \vec{w}}{2}\right) + D\left(\vec{w} \mid \frac{\vec{v} + \vec{w}}{2}\right)
\]
Vector Semantics

Adding syntax
Using syntax to define a word’s context

• Zellig Harris (1968)
  “The meaning of entities, and the meaning of grammatical relations among them, is related to the restriction of combinations of these entities relative to other entities”

• Two words are similar if they have similar syntactic contexts

Duty and responsibility have similar syntactic distribution:

<table>
<thead>
<tr>
<th>Modified by adjectives</th>
<th>additional, administrative, assumed, collective, congressional, constitutional ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects of verbs</td>
<td>assert, assign, assume, attend to, avoid, become, breach.</td>
</tr>
</tbody>
</table>
Co-occurrence vectors based on syntactic dependencies

Dekang Lin, 1998 “Automatic Retrieval and Clustering of Similar Words”

- Each dimension: a context word in one of R grammatical relations
  - Subject-of- “absorb”
- Instead of a vector of /V/ features, a vector of R/V/
- Example: counts for the word *cell*:

<table>
<thead>
<tr>
<th>subj-of, absorb</th>
<th>subj-of, adapt</th>
<th>subj-of, behave</th>
<th>obj-of, inside</th>
<th>obj-of, into</th>
<th>nmod-of, abnormality</th>
<th>nmod-of, anemia</th>
<th>nmod-of, architecture</th>
<th>obj-of, attack</th>
<th>obj-of, call</th>
<th>obj-of, come from</th>
<th>obj-of, decorate</th>
<th>nmod, bacteria</th>
<th>nmod, body</th>
<th>nmod, bone marrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>16</td>
<td>30</td>
<td>3</td>
<td>8</td>
<td>1</td>
<td>6</td>
<td>11</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Syntactic dependencies for dimensions

• Alternative (Padó and Lapata 2007):
  • Instead of having a $|V| \times R|V|$ matrix
  • Have a $|V| \times |V|$ matrix
  • But the co-occurrence counts aren’t just counts of words in a window
  • But counts of words that occur in one of R dependencies (subject, object, etc).
  • So $M(“cell”,“absorb”) = \text{count}(\text{subj(cell,absorb)}) + \text{count}(\text{obj(cell,absorb)}) + \text{count}(\text{pobj(cell,absorb)}), \text{ etc.}$
PMI applied to dependency relations

Hindle, Don. 1990. Noun Classification from Predicate-Argument Structure. ACL

<table>
<thead>
<tr>
<th>Object of “drink”</th>
<th>Count</th>
<th>PMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>tea</td>
<td>2</td>
<td>11.8</td>
</tr>
<tr>
<td>liquid</td>
<td>2</td>
<td>10.5</td>
</tr>
<tr>
<td>wine</td>
<td>2</td>
<td>9.3</td>
</tr>
<tr>
<td>anything</td>
<td>3</td>
<td>5.2</td>
</tr>
<tr>
<td>it</td>
<td>3</td>
<td>1.3</td>
</tr>
</tbody>
</table>

• “Drink it”
• But “wine” is a
Vector Semantics

Dense Vectors
Sparse versus dense vectors

• PPMI vectors are
  • **long** (length |V| = 20,000 to 50,000)
  • **sparse** (most elements are zero)

• Alternative: learn vectors which are
  • **short** (length 200-1000)
  • **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 represented as distinct dimensions; this fails to capture similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor
Three methods for getting short dense vectors

• Singular Value Decomposition (SVD)
  • A special case of this is called LSA – Latent Semantic Analysis

• “Neural Language Model”-inspired predictive models
  • skip-grams
  • CBOW

• Brown clustering
Vector Semantics

Embeddings inspired by neural language models: skip-grams and CBOW
Prediction-based models:
An alternative way to get dense vectors

• **Skip-gram** (Mikolov et al. 2013a) **CBoW** (Mikolov et al. 2013b)
• Learn embeddings as part of the process of word prediction.
• Train a neural network to predict neighboring words
  • Inspired by **neural language models**.
  • In so doing, learn dense embeddings for the words in the training corpus.
• Advantages:
  • Fast, easy to train
  • Available online in the **word2vec** package
  • Including sets of pretrained embeddings!
Skip-Gram versus CBOW

• We will talk about Skip-Gram and Continuous Bag of Words in greater detail below
• Here is a high level introduction
• Both algorithms learn embeddings by training classifiers
  • **Skip-Gram**: predict the context given the target word
  • **CBOW**: predict the target word given the context
• We will now give an extended introduction to Skip-Gram and a shorter introduction to CBOW
Skip-grams

• Predict each neighboring word
  • in a context window of 2\(C\) words
  • from the current word.

• So for \(C=2\), we are given word \(w_t\) and predicting these 4 words:

\[
[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]
\]
The Intuition behind Skip-Gram with Negative Sampling

• Treat a target word and a neighboring context word as positive examples
• Randomly sample other words in the lexicon to get negative examples (the NS in SGNS)
• Use a logistic regression to train a classifier to distinguish those two cases (positive and negative examples)
• Use the regression weights as embeddings
Skip-grams learn two embeddings for each $w$

**input embedding** $v$, in the input matrix $W$

- Column $i$ of the input matrix $W$ is the $1 \times d$ embedding $v_i$ for word $i$ in the vocabulary.

**output embedding** $v'$, in output matrix $W'$

- Row $i$ of the output matrix $W'$ is a $d \times 1$ vector embedding $v'_i$ for word $i$ in the vocabulary.
Setup

• Walking through corpus pointing at word $w(t)$, whose index in the vocabulary is $j$, so we’ll call it $w_j$ ($1 < j < |V|$).

• Let’s predict $w(t+1)$, whose index in the vocabulary is $k$ ($1 < k < |V|$). Hence our task is to compute $P(w_k | w_j)$. 
One-hot vectors

• A vector of length $|V|$
• 1 for the target word and 0 for other words
• So if automaton is vocabulary word 5
• The one-hot vector is
• $[0,0,0,0,1,0,0,0,0,\ldots,0]$
Skip-gram

Input layer
1-hot input vector

Projection layer
embedding for $w_t$

Output layer
probabilities of context words

$W_{d \times |V|}$

$W'_{d \times |V|}$

$W_{|V| \times d}$

$W'_{|V| \times d}$

$W_t$

$W_{t-1}$

$W_{t+1}$
Skip-gram

Input layer

1-hot input vector

$w_t$

$x_1$

$x_2$

$\vdots$

$x_j$

$\vdots$

$x_{|V|}$

$1 \times |V|$  

Projection layer

embedding for $w_t$

$W$

$|V| \times d$

$1 \times d$

$W'_{d \times |V|}$

Output layer

probabilities of context words

$y_k$

$y_{|V|}$

$y_1$

$y_2$

$\vdots$

$W'_{d \times |V|}$

$W'_{d \times |V|}$

$h = v_j$

$o = W'h$

$w_{t-1}$

$w_{t+1}$

$1$-hot input vector

$W$

$|V| \times d$

$1 \times |V|$  

$y_1$

$y_2$

$\vdots$

$y_{|V|}$

$W'$

$|V| \times d$

$1 \times |V|$
**Skip-gram**

\[
h = v_j
\]

**Input layer**
- 1-hot input vector

\[
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_j \\
  \vdots \\
  x_{|V|}
\end{bmatrix}
\]

\[
1 \times |V|
\]

**Projection layer**
- embedding for \( w_t \)

\[
W
\]

\[
|V| \times d
\]

**Output layer**
- probabilities of context words

\[
o = W'h
\]

\[
w_{t-1} \quad o_k = v'_k h
\]

\[
w_{t+1} \quad o_k = v'_k \cdot v_j
\]
Turning outputs into probabilities

• $o_k = v_k' \cdot v_j$

• We use softmax to turn into probabilities

\[
p(w_k | w_j) = \frac{\exp(v_k' \cdot v_j)}{\sum_{w' \in |V|} \exp(v_{w'}' \cdot v_j)}
\]
Embeddings from W and W’

• Since we have two embeddings, \( v_j \) and \( v'_j \) for each word \( w_j \)

• We can either:
  • Just use \( v_j \)
  • Sum them
  • Concatenate them to make a double-length embedding
But wait; how do we learn the embeddings?

$$\arg\max_{\theta} \log p(\text{Text})$$

$$\arg\max_{\theta} \log \prod_{t=1}^{T} p(w^{(t-C)}, \ldots, w^{(t-1)}, w^{(t+1)}, \ldots, w^{(t+C)})$$

$$\arg\max_{\theta} \sum_{-c \leq j \leq c, j \neq 0} \log p(w^{(t+j)}|w^{(t)})$$

$$= \arg\max_{\theta} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log \frac{\exp(v'^{(t+j)} \cdot v^{(t)})}{\sum_{w \in |V|} \exp(v'_w \cdot v^{(t)})}$$

$$= \arg\max_{\theta} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \left[ v'^{(t+j)} \cdot v^{(t)} - \log \sum_{w \in |V|} \exp(v'_w \cdot v^{(t)}) \right]$$
Relation between skipgrams and PMI!

• If we multiply $WW'^T$

• We get a $|V|\times|V|$ matrix $M$, each entry $m_{ij}$ corresponding to some association between input word $i$ and output word $j$

• Levy and Goldberg (2014b) show that skip-gram reaches its optimum just when this matrix is a shifted version of PMI:

$$WW'^T = M^{\text{PMI}} - \log k$$

• So skip-gram is implicitly factoring a shifted version of the PMI matrix into the two embedding matrices.
CBOW (Continuous Bag of Words)

**Input layer**
- 1-hot input vectors for each context word

**Projection layer**
- sum of embeddings for context words

**Output layer**
- probability of $w_t$
Properties of embeddings

• Nearest words to some embeddings (Mikolov et al. 2013)

<table>
<thead>
<tr>
<th>target:</th>
<th>Redmond</th>
<th>Havel</th>
<th>ninjutsu</th>
<th>graffiti</th>
<th>capitulate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redmond Wash.</td>
<td>Vaclav Havel</td>
<td>ninja</td>
<td>spray paint</td>
<td>capitulation</td>
<td></td>
</tr>
<tr>
<td>Redmond Washington</td>
<td>president Vaclav Havel</td>
<td>martial arts</td>
<td>grafitti</td>
<td>capitulated</td>
<td></td>
</tr>
<tr>
<td>Microsoft</td>
<td>Velvet Revolution</td>
<td>swordsmanship</td>
<td>taggers</td>
<td>capitulating</td>
<td></td>
</tr>
</tbody>
</table>
Embeddings capture relational meaning, they said!

\[
\text{vector}(\text{‘king’}) - \text{vector}(\text{‘man’}) + \text{vector}(\text{‘woman’}) \approx \text{vector}(\text{‘queen’})
\]

\[
\text{vector}(\text{‘Paris’}) - \text{vector}(\text{‘France’}) + \text{vector}(\text{‘Italy’}) \approx \text{vector}(\text{‘Rome’})
\]
Or do they?

• Levy, Goldberg, and Ido (2015) showed that it is problematic if you treat embeddings as compositional

• It is true that $\text{vector('king')} - \text{vector('man')} + \text{vector('woman')} \approx \text{vector('queen')}$

• It is also true that $\text{vector('king')} + \text{vector('woman')} \approx \text{vector('queen')}$

• This is because the relationship that is encoded in word-embeddings is **similarity**, not a collection of semantic components.
Vector Semantics

BERT
Context, context, context

• In Word2Vec (SkipGram and CBOW), each word—each type—has exactly one embedding
  • *bank* as a financial institution has the same embedding as *bank* as the earth at the edge of a river
  • *bass* the fish has the same embedding as the *bass* about which all of it is
• This is inherit in the architectures of these models
• Wouldn’t it be nice if you could have context-sensitive embeddings of words?

**ENTER BERT**

THE HOTTEST BLOCKBUSTER IN NLP THIS YEAR
Pay Attention to the Transformers

• In the lecture on deep learning, you will learn about fundamentals of neural architectures—including notions like attention—as well as state-of-the-art architectures like transformers

• These are essential to having a deep understanding of BERT

• I don’t have the time or resources to explain these to you here, so I’m going to abstract over them

• When you understand (self-)attention and transformers, do yourself a favor and read the BERT paper:

Before BERT

• BERT is actually not the first way of doing distributed contextual word representations
• **ELMo** uses the concatenation of independently-trained left-to-right and right-to-left LSTMs
  • Can use left and right context
  • Less powerful architecture
• **OpenAI GPT** uses a left-to-right transformer
  • More powerful architecture
  • Can only use left context
• **BERT** uses a bidirectional transformer
  • How, though? This would mean that words could—indirectly—”see” themselves, which would foul everything up
  • The answer: **cloze task**
BERT and Cloze

• Cloze tasks are task in which one or more words in a text are masked and a person/machine is required to fill in an appropriate word
• “Fill-in-the-blank”
• BERT is a _________ architecture.
  • neural (high probability)
  • impressive (high probability)
  • purple (medium probability)
  • the (low probability)
• In training BERT, around 15% of the words are masked, as in a cloze task
• The model is trained to “guess” these words from context
• We take the model that results and make embeddings out of it
Use the output of the masked word’s position to predict the masked word

Randomly mask 15% of tokens

Input

BERT in Practice

• You feed BERT a passage of text (like a sentence)
• BERT returns a tensor
  • One column for each token in the passage
  • One row for each layer in the network
• Usually, you want to get a useful vector out of the tensor
• You can do this in various ways:
  • Take the top-most layer
  • Take the mean of the topmost-few layers
  • Take the concatenation of the top couple of layers
  • Etc.