Distributed Semantics & Embeddings

Yejin Choi - University of Washington

[Slides adapted from Dan Jurafsky]
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”
Similar words in 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
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):
- “oculist and eye-doctor … occur in almost the same environments”
- “If A and B have almost identical environments we say that they are synonyms.”

Firth (1957):
- “You shall know a word by the company it keeps!”
Intuition of distributional word similarity

- 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.
Four kinds of vector models

Sparse vector representations
  1. Word co-occurrence matrices
     -- weighted by mutual-information

Dense vector representations:
  2. Singular value decomposition (and Latent Semantic Analysis)
  3. Neural-network inspired models (skip-grams, CBOW)
  4. Brown clusters
Skip-gram

Input layer
1-hot input vector

Projection layer
embedding for $w_t$

Output layer
probabilities of context words

$W_{|V| \times d}$

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

$w_t$ $x_1$ $x_2$ $x_j$ $x_{|V|}$

$1 \times |V|$ $1 \times |V|$ $1 \times d$
Shared intuition

- Model the meaning of a word by “embedding” it in a vector space.
- The meaning of a word is a vector of numbers
  - Vector models are also called “embeddings”.

§ Model the meaning of a word by “embedding” it in a vector space.
§ The meaning of a word is a vector of numbers
  § Vector models are also called “embeddings”.

§
Thought vector?

- *You can't cram the meaning of a whole %&!*$# sentence into a single $&!*#* vector!*

Raymond Mooney
Vector Semantics

I. 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|
words and vectors

The term-document matrix $X$ has $V$ rows (one for each word type in the vocabulary) and $D$ columns (one for each document in the collection). Each column represents a document. A query is also represented by a vector $q$ of length $|V|$. We go about finding the most relevant document to query by finding the document whose vector is most similar to the query; later in the chapter we'll introduce some of the components of this process: the tf-idf term weighting, and the cosine similarity metric.

But now let's turn to the insight of vector semantics for representing the meaning of words. The idea is that we can also represent each word by a vector, now a row vector representing the counts of the word's occurrence in each document. Thus the vectors for fool $[37, 58, 1, 5]$ and clown $[5, 117, 0, 0]$ are more similar to each other (occurring more in the comedies) while battle $[1, 1, 8, 15]$ and soldier $[2, 2, 12, 36]$ are more similar to each other (occurring less in the comedies).

More commonly used for vector semantics than this term-document matrix is an alternative formulation, the term-term matrix, more commonly called the word-term term matrix, word matrix or the term-context matrix, in which the columns are labeled by words rather than documents. This matrix is thus of dimensionality $|V| \times |V|$ and each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus. The context could be the document, in which case the cell represents the number of times the two words appear in the same document. It is most common, however, to use smaller contexts, such as a window around the word, for example of 4 words to the left and 4 words to the right, in which case the cell represents the number of times (in some training corpus) the column word occurs in such a $\pm 4$ word window around the row word.

For example here are 7-word windows surrounding four sample words from the Brown corpus (just one example of each word):

sugar, a sliced lemon, a tablespoonful of apricot preserve or jam, a pinch each of, pineapple their enjoyment. Cautiously she sampled her first and another fruit whose taste she likened well suited to programming on the digital information In finding the optimal R-stage policy from for the purpose of gathering data and computer data pinch result sugar...
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 words)
  - The longer the windows…
    - the more **semantic** the representation (± 4-10 words)
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)
Informativeness of a context word $X$ for a target word $Y$

- $\text{Freq(} \text{the, beer)} \quad \text{VS} \quad \text{freq(} \text{drink, beer)}$ ?
- How about joint probability?
- $P(\text{the, beer}) \quad \text{VS} \quad (\text{drink, beer})$ ?
- Frequent words like “the” and “of” are not quite informative
- Normalize by the individual word frequencies!
  - ➔ Pointwise Mutual Information (PMI)
Pointwise Mutual Information

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

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

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

$$PMI(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$
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 $c_j$

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

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

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

\[
PMI_{ij} = \log \frac{p_{ij}}{p_{i*}p_{*j}}
\]

\[
PPMI_{ij} = \max(0, PMI_{ij})
\]
\[ p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij}} \]

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

<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>

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

<table>
<thead>
<tr>
<th>( p(w, \text{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>0.00</td>
<td>0.00</td>
<td>0.05</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>
<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>
<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>
<td>0.21</td>
<td>0.00</td>
</tr>
</tbody>
</table>

\[ p(\text{context}) \]

<table>
<thead>
<tr>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.16</td>
<td>0.37</td>
<td>0.11</td>
<td>0.26</td>
<td>0.11</td>
</tr>
</tbody>
</table>

\[ p(w) \]

<table>
<thead>
<tr>
<th>apricot</th>
<th>pineapple</th>
<th>digital</th>
<th>information</th>
<th>0.11</th>
<th>0.11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.11</td>
<td>0.11</td>
<td>0.21</td>
<td>0.58</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PMI_{ij} = \log \frac{p_{ij}}{p_i \times p_j}

<table>
<thead>
<tr>
<th></th>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
<th>p(w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>0.00</td>
<td>0.00</td>
<td>0.05</td>
<td>0.00</td>
<td>0.05</td>
<td>0.11</td>
</tr>
<tr>
<td>pineapple</td>
<td>0.00</td>
<td>0.00</td>
<td>0.05</td>
<td>0.00</td>
<td>0.05</td>
<td>0.11</td>
</tr>
<tr>
<td>digital</td>
<td>0.11</td>
<td>0.05</td>
<td>0.00</td>
<td>0.05</td>
<td>0.00</td>
<td>0.21</td>
</tr>
<tr>
<td>information</td>
<td>0.05</td>
<td>0.32</td>
<td>0.00</td>
<td>0.21</td>
<td>0.00</td>
<td>0.58</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>p(context)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.16</td>
<td>0.37</td>
</tr>
</tbody>
</table>

\[
\text{pmi(}\text{information, data)} = \log_2 \left( \frac{0.32}{(0.37 \times 0.58)} \right) = 0.58
\]

(.57 using full precision)

<table>
<thead>
<tr>
<th></th>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td></td>
<td>-</td>
<td>2.25</td>
<td>-</td>
<td>2.25</td>
</tr>
<tr>
<td>pineapple</td>
<td></td>
<td>-</td>
<td>2.25</td>
<td>-</td>
<td>2.25</td>
</tr>
<tr>
<td>digital</td>
<td>1.66</td>
<td>0.00</td>
<td>-</td>
<td>0.00</td>
<td>-</td>
</tr>
<tr>
<td>information</td>
<td>0.00</td>
<td>0.57</td>
<td>-</td>
<td>0.47</td>
<td>-</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 $\alpha = 0.75$:

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

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

- This helps because $P_\alpha(c) > P(c)$ for rare $c$
- Consider two events, $P(a) = .99$ and $P(b) = .01$
- $P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$  $P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$
TF-IDF: Alternative to PPMI for measuring association

- **tf-idf** (that’s a hyphen not a minus sign)
- The combination of two factors
  - **Term frequency** (Luhn 1957): frequency of the word
  - **Inverse document frequency** (IDF) (Sparck Jones 1972)
    - $N$ is the total number of documents
    - $df_i = \text{“document frequency of word } i\text{”}$
      - $= \# \text{ of documents with word } i$
    
    $$\text{idf}_i = \log \left( \frac{N}{df_i} \right)$$

- $w_{ij} = tf_{ij} \times idf_i = \text{weight of word } i \text{ in document } j$
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

**Dot product or inner product** from linear algebra

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

- 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 longer 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!

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

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

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

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

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

Dot product

Unit vectors

\(v_i\) 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
Visualizing vectors and angles

Dimension 1: ‘large’

Dimension 2: ‘data’

<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>
Vector Semantics

Evaluating similarity
Evaluating similarity

- **Extrinsic (task-based, end-to-end) Evaluation:**
  - Question Answering
  - Spell Checking
  - Essay grading

- **Intrinsic Evaluation:**
  - Correlation between algorithm and human word similarity ratings
    - Wordsim353: 353 noun pairs rated 0-10. \( \text{sim}(\text{plane}, \text{car}) = 5.77 \)
  - Taking TOEFL multiple-choice vocabulary tests
    - **Levied** is closest in meaning to: imposed, believed, requested, correlated
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 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 and CBOW
- **Brown clustering**
Vector Semantics

Dense Vectors via SVD
Intuition

- Approximate an N-dimensional dataset using fewer dimensions
- By first rotating the axes into a new space
- In which the highest order dimension captures the most variance in the original dataset
- And the next dimension captures the next most variance, etc.
- Many such (related) methods:
  - PCA – principle components analysis
  - Factor Analysis
  - SVD
Dimensionality reduction

PCA dimension 1

PCA dimension 2
Singular Value Decomposition

Any \((w \times c)\) matrix \(X\) equals the product of 3 matrices:

\[
\text{Words} = X = W \times S \times C
\]

Where:
- \(X\) is \((w \times c)\) matrix
- \(W\) is \((w \times m)\) matrix
- \(S\) is \((m \times m)\) matrix
- \(C\) is \((m \times c)\) matrix
Singular Value Decomposition

Any \((w \times c)\) matrix \(X\) equals the product of 3 matrices:

\[
X = W S C
\]

\(W\): \((w \times m)\) matrix: rows corresponding to original but \(m\) columns represents a dimension in a new latent space, such that

- \(m\) column vectors are orthogonal to each other
- \(m = \text{“Rank” of } X\).

\(S\): \((m \times m)\) matrix: diagonal matrix of singular values expressing the importance of each dimension.

\(C\): \((m \times c)\) matrix: columns corresponding to original but \(m\) rows corresponding to singular values.
Singular Value Decomposition

\[ X = W S C \]

\[ w \times c \quad w \times m \]

Landauer and Dumais 1997
SVD applied to term-document matrix: Latent Semantic Analysis (LSA)

- Often $m$ is not small enough!
- If instead of keeping all $m$ dimensions, we just keep the top $k$ singular values. Let’s say 300.
- The result is a least-squares approximation to the original $X$.
- But instead of multiplying, we’ll just make use of $W$.
- Each row of $W$:
  - A $k$-dimensional vector
  - Representing word $W$
LSA more details

- 300 dimensions are commonly used
- The cells are commonly weighted by a product of two weights
  - Local weight: Log term frequency
  - Global weight: either idf or an entropy measure
Let’s return to PPMI word-word matrices

- Can we apply SVD to them?
SVD applied to term-term matrix

\[
\begin{bmatrix}
X \\
|V| \times |V|
\end{bmatrix}
= 
\begin{bmatrix}
W \\
|V| \times |V|
\end{bmatrix}
\begin{bmatrix}
\sigma_1 & 0 & 0 & \ldots & 0 \\
0 & \sigma_2 & 0 & \ldots & 0 \\
0 & 0 & \sigma_3 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \ldots & \sigma_V \\
|V| \times |V|
\end{bmatrix}
\begin{bmatrix}
C \\
|V| \times |V|
\end{bmatrix}
\]

(simplifying assumption: the matrix has rank |V|)
Truncated SVD on term-term matrix

\[
X_{|V| \times |V|} = W_{|V| \times k} \sigma_k \begin{bmatrix}
\sigma_1 & 0 & 0 & \ldots & 0 \\
0 & \sigma_2 & 0 & \ldots & 0 \\
0 & 0 & \sigma_3 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \ldots & \sigma_k \\
\end{bmatrix} \begin{bmatrix}
C \\
k \times |V|
\end{bmatrix}
\]
Truncated SVD produces embeddings

- Each row of \( W \) matrix is a \( k \)-dimensional representation of each word \( w \)
- \( K \) might range from 50 to 1000
- Generally we keep the top \( k \) dimensions, but some experiments suggest that getting rid of the top 1 dimension or even the top 50 dimensions is helpful (Lapesa and Evert 2014).
Embeddings versus sparse vectors

Dense SVD embeddings sometimes work better than sparse PPMI matrices at tasks like word similarity

- Denoising: low-order dimensions may represent unimportant information
- Truncation may help the models generalize better to unseen data.
- Having a smaller number of dimensions may make it easier for classifiers to properly weight the dimensions for the task.
- Dense models may do better at capturing higher order co-occurrence.
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 net language models**.
  - In so doing, learn dense embeddings for the words in the training corpus.
- **Advantages:**
  - Fast, easy to train (much faster than SVD)
  - Available online in the `word2vec` package
  - Including sets of pretrained embeddings!
Skip-grams

- Predict each neighboring word
  - in a context window of 2C words
  - from the current word.
- So for C=2, we are given word \( w_t \) and predicting these 4 words:
  \[
  [w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]
  \]
Skip-grams learn 2 embeddings for each word $w$.

**Input embedding** $v$, in the input matrix $W$
- Embedding of the target word
- 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 the output matrix $W'$
- Embedding of the context word
- Row $i$ of the output matrix $W'$ is a $d \times 1$ 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 “popsicle” is vocabulary word 5 
- The **one-hot vector** is 
- $[0,0,0,0,1,0,0,0,0.......0]$
Skip-gram

Input layer
1-hot input vector

Projection layer
embedding for $w_t$

Output layer
probabilities of context words

$W_{|V| \times d}$

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

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

$y_1$

$y_2$

$\ldots$

$y_k$

$y_{|V|}$

$w_{t-1}$

$w_t$

$w_{t+1}$
Skip-gram

\[ W^T w_t = v_j \]

Output layer
probabilities of context words

\[ W'^T v_j = w_{t-1} \]

\[ y_k = v'^T_k v_j \]
Turning outputs into probabilities

\[ y_k = v'_k^T v_j = v'_k \cdot v_j \]

- 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
Training embeddings

\[
\arg\max_{\theta} \log p(\text{Text}) \\
\arg\max_{\theta} \log \prod_{t=1}^{T} p(w^{(t-C)}, \ldots, w^{(t-1)}, w^{(t+1)}, \ldots, w^{(t+C)}|w^{(t)}) \\
= \arg\max_{\theta} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w^{(t+j)}|w^{(t)}) \\
= \arg\max_{\theta} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log \frac{\exp(v'(t+j) \cdot v(t))}{\sum_{w \in |V|} \exp(v'_w \cdot v(t))} \\
= \arg\max_{\theta} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \left[ v'(t+j) \cdot v(t) - \log \sum_{w \in |V|} \exp(v'_w \cdot v(t)) \right]
\]
Training: Noise Contrastive Estimation (NCE)

\[
\operatorname{arg\,max}_\theta \log p(\text{Text}) \quad = \operatorname{arg\,max}_\theta \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \left[ v'(t+j) \cdot v(t) - \log \sum_{w \in |V|} \exp(v'_w \cdot v(t)) \right]
\]

- the normalization factor is too expensive to compute exactly (why?)
- **Negative sampling**: sample only a handful of negative examples to compute the normalization factor
- (some engineering detail) the actual skip-gram training also converts the problem into binary classification (logistic regression) of predicting whether a given word is a context word or not
Relation between skipgrams and PMI!

- If we multiply $WW'\mathbf{T}$
- 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:
  \[ WW'^\mathbf{T} = M^{PMI} - \log k \]
- 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 $w_t$

$w_t \rightarrow W_{1\times|V|} \rightarrow 1\times|V| \rightarrow W_{|V|\times d} \rightarrow 1\times d \rightarrow W'_{d\times|V|} \rightarrow y_1, y_2, \ldots, y_{|V|}, w_t$
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>capitation</td>
<td></td>
</tr>
<tr>
<td>Redmond Washington</td>
<td>president Vaclav Havel</td>
<td>martial arts</td>
<td>graffiti</td>
<td>capitated</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!

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

\[
\text{vector('Paris')} - \text{vector('France')} + \text{vector('Italy')} \approx \text{vector('Rome')}
\]
Vector Semantics

Brown clustering
Brown clustering

- An agglomerative clustering algorithm that clusters words based on which words precede or follow them
- These word clusters can be turned into a kind of vector
- We’ll give a very brief sketch here.
Class-based language model

- Suppose each word was in some class $c_i$:

$$P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$$

$$P(\text{corpus}|C) = \prod_{i=1}^{n} P(c_i|c_{i-1})P(w_i|c_i)$$
Brown clustering algorithm

- Each word is initially assigned to its own cluster.
- We now consider merging each pair of clusters. Highest quality merge is chosen.
  - Quality = merges two words that have similar probabilities of preceding and following words
  - (More technically quality = smallest decrease in the likelihood of the corpus according to a class-based language model)
- Clustering proceeds until all words are in one big cluster.
Brown Clusters as vectors

- By tracing the order in which clusters are merged, the model builds a binary tree from bottom to top.
- Each word represented by binary string = path from root to leaf
- Each intermediate node is a cluster
- Chairman is 0010, “months” = 01, and verbs = 1
Brown cluster examples

Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays June March July April January December October November September August pressure temperature permeability density porosity stress velocity viscosity gravity tension anyone someone anybody somebody had hadn’t hath would’ve could’ve should’ve must’ve might’ve asking telling wondering instructing informing kidding reminding bothering thanking deposing mother wife father son husband brother daughter sister boss uncle great big vast sudden mere sheer gigantic lifelong scant colossal down backwards ashore sideways southward northward overboard aloft downwards adrift
Brown Clustering on Twitter!
http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html

1110101010010 (52)
sorry gutted sry srry soz #thankful sory sorrry sowwy sori thankgod soreyy sowi sorri sorryyy sorrrrry luckyyy sowwie paiseh sowie soory sorry- sorrrrry soweer -sorry sorryyyyy #dintwannatellyou sorreh sorrr sowy soorry sorrryy apols sawry #iforgiveyou sryy sorrie sowwwy offski s0rry sorrryy soryy sorrrrrry sawwy sorryyyyy sozz nitm sowry thankgoodness sowwi

00101110 (79)
really rly raly really genuinley rlly realllly realllly really rele realli relly reallliilly
reli reali sholl rily reallyyy reeeeaaly realllllllly reaally reeeally rili reaaally
reaaaally reallyyyyy rilly reallllllly reeeeeeaally reallyy shol reallllyyy reely r
really2 reallyyyyy _really_ realllllllllly reaaly reallllyy realllt genuinly relli reallllyyy reeeeeeaaally weally reaaalllly realllllyyy

1110101110 (582)
:
( / - - - - - - - - ( :'( d: | :s - ___ - (= / > - ___ - - :/ < /3 :| - ___ - - ;( /: ️
:(( >_< = [ [ #fml 😒 - ______ - =\>: ( 😒 - - - , > > :o ; / 😞 d; -. -
_________ - >_ > : ((( - _ " = s __; #ugh :-\ = . = - _______ -
Summary

- Distributional (vector) models of meaning
  - **Sparse** (PPMI-weighted word-word co-occurrence matrices)
  - **Dense**:  
    - Word-word SVD 50-2000 dimensions  
    - Skip-grams and CBOW  
    - Brown clusters 5-20 binary dimensions.