Vector Semantics

16 March 2021
What do words mean?
There are many approaches to word meaning.
Dictionaries define words for people (using other words):
We can try to encode definitional meaning using logical rules like

$$\forall x \cdot \text{Dog}(x) \Rightarrow \text{Mammal}(x)$$
Or we can arrange word senses into hierarchies of sets of synonyms (synsets), as WordNet does:

- **S:** (n) **dog**, **domestic dog**, **Canis familiaris** (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds) "the dog barked all night"
  - direct hyponym / full hyponym
  - part meronym
  - member holonym
- **S:** (n) **carnivore** (a terrestrial or aquatic flesh-eating mammal)
  - terrestrial carnivores have four or five clawed digits on each limb
  - direct hypernym / inherited hypernym / sister term
- **S:** (n) **placental, placental mammal, eutherian, eutherian mammal** (mammals having a placenta; all mammals except monotremes and marsupials)
  - S: (n) **mammal, mammalian** (any warm-blooded vertebrate having the skin more or less covered with hair; young are born alive except for the small subclass of monotremes and nourished with milk)
  - S: (n) **vertebrate, craniate** (animals having a bony or cartilaginous skeleton with a segmented spinal column and a large brain enclosed in a skull or cranium)
  - S: (n) **chordate** (any animal of the phylum Chordata having a notochord or spinal column)
  - S: (n) **animal, animate being, beast, brute, creature, fauna** (a living organism characterized by voluntary movement)
  - S: (n) **organism, being** (a living thing that has (or can develop) the ability to act or function independently)
  - S: (n) **living thing, animate thing** (a living (or once living) entity)
  - S: (n) **whole, unit** (an assemblage of parts that is regarded as a
There’s a vast landscape of work in lexical semantics, and we’ll return to some of these approaches later.

For the moment, let’s think about one of the most basic questions we can ask about words: How similar are they?
Relation: Similarity

Words with similar meanings – not synonyms, but sharing some elements of meaning, e.g.,

- *car* and *bicycle*
- *cow* and *horse*
Ask humans how similar two words are

<table>
<thead>
<tr>
<th>Word 1</th>
<th>Word 2</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>vanish</td>
<td>disappear</td>
<td>9.8</td>
</tr>
<tr>
<td>behave</td>
<td>obey</td>
<td>7.3</td>
</tr>
<tr>
<td>belief</td>
<td>impression</td>
<td>5.95</td>
</tr>
<tr>
<td>muscle</td>
<td>bone</td>
<td>3.65</td>
</tr>
<tr>
<td>modest</td>
<td>flexible</td>
<td>0.98</td>
</tr>
<tr>
<td>hole</td>
<td>agreement</td>
<td>0.3</td>
</tr>
</tbody>
</table>

SimLex-999 data set (Hill et al, 2015)

But how to capture word similarity through data?
The notion of similarity gives us a basis for talking about word meaning.
What does *ongchoi* mean?

Suppose you see these sentences:

*Ong choi is delicious sautéed with garlic.*

*Ong choi is superb over rice*

*Ong choi leaves with salty sauces*
What does ongchoi mean?

Suppose you see these sentences:

- *Ong choi is delicious sautéed with garlic.*
- *Ong choi is superb over rice*
- *Ong choi leaves with salty sauces*

And you’ve also seen these:

- *…spinach sautéed with garlic over rice*
- *Chard stems and leaves are delicious*
- *Collard greens and other salty leafy greens*
What does *ongchoi* mean?

Suppose you see these sentences:

- Ong choi is delicious *sautéed with garlic*.
- Ong choi is superb *over rice*
- Ong choi *leaves* with salty sauces

And you’ve also seen these:

- …spinach *sautéed with garlic over rice*
- Chard stems and *leaves* are *delicious*
- Collard greens and other *salty* leafy greens

Conclusion:

*Ongchoi* is a leafy green like spinach, chard, or collard greens
Ongchoi: *Ipomoea aquatica* or “water spinach”
“The meaning of a word is its use in language”

“If $A$ and $B$ have almost identical environments we say that they are synonyms.”
Zellig Harris, 1954

“You shall know a word by the company it keeps.”
John Firth, 1957

These form the philosophical foundation of *distributional semantics* – words are defined by their environments, i.e., the words around them.
Each word is a vector, and similar words are “nearby in space”
We define a word as a vector called an embedding because it’s embedded into a space.

Word embeddings have become the de facto standard way to represent meaning in NLP.
For NLP tasks like sentiment analysis,

using words, we require the *same* word to be in training and test;

using embeddings, it’s ok if *similar* words occurred!
We’ll look at two kinds of embeddings:

**TF–IDF**

“Term frequency–inverse document frequency”

A common baseline model

**Sparse** vectors

Words are represented by a simple function of the counts of words in the same document

**Word2vec**

**Dense** vectors

Representation is created by training a classifier to predict whether a word is likely to appear nearby
Words and vectors
## Term–document matrix

Each document is represented by a vector of word counts:

<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>0</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>good</td>
<td>114</td>
<td>80</td>
<td>62</td>
<td>89</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>58</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>wit</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Visualizing document vectors

Henry V [4,13]
Julius Caesar [1,7]
As You Like It [36,1]
Twelfth Night [58,0]
Vectors are the basis of information retrieval

<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>0</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>good</td>
<td>114</td>
<td>80</td>
<td>62</td>
<td>89</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>58</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>wit</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Vectors are similar for the two comedies, but different than the history.

Comedies have more fools and wit and fewer battles.
Words can be vectors too

<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>0</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>good</td>
<td>114</td>
<td>80</td>
<td>62</td>
<td>89</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>58</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>wit</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

*battle* is “the kind of word that occurs in *Julius Caesar* and *Henry V*”

*fool* is “the kind of word that occurs in comedies, especially *Twelfth Night*”
More common: word–word matrix (or “term–context matrix”)

Two *words* are similar in meaning if their context vectors are similar.

<table>
<thead>
<tr>
<th>aardvark</th>
<th>computer</th>
<th>data</th>
<th>pinch</th>
<th>result</th>
<th>sugar</th>
<th>...</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>
</tbody>
</table>

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 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
But how to measure similarity between digital and information?
Cosine as a similarity metric

Word frequency counts are non-negative, so the cosine here ranges from 0–1.

-1: vectors point in opposite directions
+1: vectors point in the same direction
0: vectors are orthogonal
TF–IDF
But raw frequency is a bad representation

Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that’s useful information.

But overly frequent words like *the*, *it*, or *they* are not very informative about the context.

Need a function that resolves this frequency paradox!
The **document frequency** $\text{df}_t$ of term $t$ is the number of documents $t$ occurs in.

A “document” can be anything; we can treat each paragraph as a document.

The **inverse document frequency** $\text{idf}_t$ of term $t$ is

$$\text{idf}_i = \log_{10} \left( \frac{N}{\text{df}_i} \right)$$

<table>
<thead>
<tr>
<th>Word</th>
<th>$DF$</th>
<th>$IDF$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Romeo</td>
<td>1</td>
<td>1.57</td>
</tr>
<tr>
<td>salad</td>
<td>2</td>
<td>1.27</td>
</tr>
<tr>
<td>Falstaff</td>
<td>4</td>
<td>0.967</td>
</tr>
<tr>
<td>forest</td>
<td>12</td>
<td>0.489</td>
</tr>
<tr>
<td>battle</td>
<td>21</td>
<td>0.246</td>
</tr>
<tr>
<td>wit</td>
<td>34</td>
<td>0.037</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>0.012</td>
</tr>
<tr>
<td>good</td>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td>sweet</td>
<td>37</td>
<td>0</td>
</tr>
</tbody>
</table>
TF–IDF combines the inverse document frequency (which discounts words that appear in many documents) with the *term frequency* (a squashed count of the term in the document):

\[
w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t
\]

where

\[
\text{tf}_{t,d} = \log_{10} (\text{count}(t, d) + 1)
\]
### Raw counts:

<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>0</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>good</td>
<td>114</td>
<td>80</td>
<td>62</td>
<td>89</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>58</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>wit</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

### TF–IDF:

<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>0.074</td>
<td>0</td>
<td>0.22</td>
<td>0.28</td>
</tr>
<tr>
<td>good</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fool</td>
<td>0.019</td>
<td>0.021</td>
<td>0.0036</td>
<td>0.0083</td>
</tr>
<tr>
<td>wit</td>
<td>0.049</td>
<td>0.044</td>
<td>0.018</td>
<td>0.022</td>
</tr>
</tbody>
</table>
Dense vectors
TF–IDF vectors are

- **long** (length \( |V| = 20,000 \) to 50,000)
- **sparse** (most elements are zero)

Alternative: learn vectors which are

- **short** (length 50–1000)
- **dense** (most elements are non-zero)
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 represented as distinct dimensions in sparse vectors

This fails to capture similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor
Word2vec
We can learn embeddings as part of the process of word prediction

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 itself, but we’ll take the learned classifier weights as the word embeddings.

Brilliant insight: Use running text as implicitly supervised training data!

A word $c$ that occurs near apricot acts as gold “correct answer” to the question “Is word $w$ likely to show up near apricot?”
Advantages

No need for hand-labeled supervision!

Fast, easy to train

Available online in the word2vec package (and others)

Including sets of pretrained embeddings!

Word2vec (Mikolov et al.) code.google.com/archive/p/word2vec
FastText fasttext.cc
GloVe (Pennington, Socher, Manning) nlp.stanford.edu/projects/glove
What is `word2vec`?

`word2vec` is not a single algorithm; it’s a software package for representing words as vectors, containing:

- Two distinct models
  - Continuous bag-of-words (CBoW)
  - **Skip-gram (SG)**
- Various training methods
  - **Negative sampling (NS)**
  - Hierarchical softmax
- A rich preprocessing pipeline
  - Dynamic context windows
  - Subsampling
  - Deleting rare words
Skip-gram classifier
Approach: predict if candidate word $c$ is a “neighbor”

1. Treat the target word $t$ and a neighboring context word $c$ as positive examples.

2. Randomly sample other words in the lexicon to get negative examples.

3. Use logistic regression (maximum entropy) to train a classifier to distinguish those two cases.

4. Use the learned weights as the embeddings.
Skip-gram training data

Assume a ±2 word window, given training sentence:

…lemon, a [tablespoon of *apricot* jam, a] pinch…
Skip-gram classifier

Assume a ±2 word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

Goal: train a classifier that is given a candidate (word, context) pair
(apricot, tablespoon),
(apricot, aardvark), …

And assigns each pair a probability: P(+ | w, c)
Similarity is computed from dot product

Two vectors are similar if they have a high dot product

- Cosine is just a normalized dot product

So, similarity\((w, c)\) \(\propto w \cdot c\)

We’ll need to normalize to get a probability.
Turning dot products into probabilities

\[
\text{Sim}(w, c) \approx w \cdot c
\]

To turn this into a probability, we can use the sigmoid function (from logistic regression):

\[
P( + \mid w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}
\]

\[
P( - \mid w, c) = 1 - P( + \mid w, c)
\]

\[
= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)}
\]
How skip-gram classifier computes $P( + \mid w, c)$

$$P( + \mid w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words. We’ll assume independence and just multiply them:

$$P( + \mid w, c_1:L) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P( + \mid w, c_1:L) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$
Skip-gram classifier: summary

A probabilistic classifier that, given

- a test target word \( w \) and
- its context window of \( L \) words \( c_{1:L} \),

assigns a probability that \( w \) occurs in this window.

To compute this, we just need embeddings for all the words.
These embeddings we’ll need: a set for $w$, a set for $c$

$$\theta = \begin{align*}
W & \text{ target words} \\
C & \text{ context & noise words}
\end{align*}$$
Learning the embeddings
Skip-gram training data

Assume a ±2 word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

positive examples +
t  c
apricot tablespoon
apricot of
apricot jam
apricot a
Skip-gram training data

Assume a ±2 word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

For each positive example, we’ll grab k negative examples, sampled by frequency
Skip-gram training data

Assume a ±2 word window, given training sentence:

...lemon, a [tablespoon of apricot jam, a] pinch...
Skip-gram with negative sampling

In the training phase, the algorithm walks through the corpus

Chooses the surrounding context words as positive examples
For each positive example also chooses k noise samples or negative samples (“non-neighbor words”)

Goal: move the embeddings toward the neighbor words and away from the noise words
Example

**Apricot** has context words $c_1$–$c_4$:

lemon, a [tablespoon of apricot preserves or] jam

$c_1$ $c_2$ $w$ $c_3$ $c_4$

**Goal:** learn an embedding whose dot product with each context word is high

Skip-gram uses a sigmoid function $\sigma$ of the dot product, where $\sigma(x) = \frac{1}{1 + e^{-x}}$

Want $\sigma(c_1 \cdot w) + \sigma(c_2 \cdot w) + \sigma(c_3 \cdot w) + \sigma(c_4 \cdot w)$ to be high

For each **context word** the algorithm chooses $k$ random noise words according to their **unigram frequency**

If $k = 2$, for each target/context pair there will be 2 noise words for each of the 4 context words

[cement metaphysical dear coaxial apricot attendant whence forever puddle]

$n_1$ $n_2$ $n_3$ $n_4$ $n_5$ $n_6$ $n_7$ $n_8$

Want $\sigma(n_1 \cdot w) + \sigma(n_2 \cdot w) + \cdots + \sigma(n_8 \cdot w)$ to be low
Continuous bag of words (CBOW)

Roughly the mirror image of the skip-gram model

Like skip-grams, based on a predictive model

But predicts current word $w_t$ from the context window of $2L$ words around it

  e.g., for $L = 2$ the context is $[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$

CBOW and Skip-gram

Similar algorithms, produce similar embeddings

  Have slightly different behavior

  Often one will be the better choice for any particular task
Properties of embeddings
Evaluating embeddings

Compare to human scores on word-similarity 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 $C$

The nearest words to $Hogwarts$, $C = \pm 2$:

- Sunnydale
- Evernight

The nearest words to $Hogwarts$, $C = \pm 5$:

- Dumbledore
- Malfoy
- halfblood
Embeddings capture relational meaning!

The offsets between vector embeddings can capture relations between words.

E.g., $\text{vector}(\text{king}) - \text{vector}(\text{man}) + \text{vector}(\text{woman}) \approx \text{vector}(\text{queen})$

Left panel visualizes this by projecting a representation down to two dimensions

Right panel shows how embeddings can capture grammatical number
Simple vector addition can often produce meaningful results. For example, \( \text{vec}(\text{Russia}) + \text{vec}(\text{river}) \) is close to \( \text{vec}(\text{Volga River}) \), and \( \text{vec}(\text{Germany}) + \text{vec}(\text{capital}) \) is close to \( \text{vec}(\text{Berlin}) \).

Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.
Another example
A bit of magic

```
>>> from gensim.models import KeyedVectors
>>> vectors = KeyedVectors.load_word2vec_format("GoogleNews-vectors-negative300.bin", binary=True)
>>> vectors.most_similar(positive=['aunt', 'man'], negative=['woman'])[0]
('uncle', 0.7947131991386414)
>>> vectors.most_similar(positive=['yellow', 'apple'], negative=['banana'])[0]
('red', 0.5296208262443542)
```

Output from the gensim package using word2vec vectors pretrained on Google News

This is not a fancy language model

No external knowledge base

Just vector addition and subtraction with cosine similarity
[Word2vec Python demo]
Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

Embeddings reflect cultural bias!

Ask “Paris : France :: Tokyo : x”

\[ x = \text{Japan} \]

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring
Embeddings reflect cultural bias!

Ask “Paris : France :: Tokyo : x”

x = Japan

Ask “father : doctor :: mother : x”

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring
Embeddings reflect cultural bias!

Ask “Paris : France :: Tokyo : x”

x = Japan

Ask “father : doctor :: mother : x”

x = nurse

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring
Embeddings reflect cultural bias!

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

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

Ask “man : computer programmer :: woman : x”

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring
Embeddings reflect cultural bias!

Ask “Paris : France :: Tokyo : x”
   \( x = \text{Japan} \)

Ask “father : doctor :: mother : x”
   \( x = \text{nurse} \)

Ask “man : computer programmer :: woman : x”
   \( x = \text{homemaker} \)

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring
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

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. “Man is to computer programmer as woman is to homemaker? debiasing word embeddings”. In NeurIPS, pp. 4349–4357. 2016.
Historical embedding as a tool to study cultural biases

Compute a gender or ethnic bias for each adjective: e.g., how much closer the adjective is to woman synonyms than man synonyms, or names of particular ethnicities

Embeddings for competence adjective (smart, wise, brilliant, resourceful, thoughtful, logical) are biased toward men, a bias slowly decreasing 1960–1990

Embeddings for dehumanizing adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.

These match the results of old surveys done in the 1930s

Playing with embeddings

WebVectors

Video on Word Embedding Explained and Visualized and Accompanying slides
Acknowledgments

The lecture incorporates material from:

Nae-Rae Han, University of Pittsburgh
Nancy Ide, Vassar College
Daniel Jurafsky and James Martin, *Speech and Language Processing*