word representations

CS 585, Fall 2018
Introduction to Natural Language Processing
http://people.cs.umass.edu/~miyyer/cs585/

Mohit Iyyer
College of Information and Computer Sciences
University of Massachusetts Amherst

many slides from Brendan O’Connor and Jurafsky & Martin Ed. 3
Questions from last time

• What is regularization? We’ll discuss it next class.
• Exercise solutions: on Piazza
• HW1? Coming out tonight or tomorrow, due 10 days from its release
What do words mean?

First thought: look in a dictionary

http://www.oed.com/
pepper, *n.*

**Pronunciation:** BrE. /ˈpeər/ US. /ˈpeər/

**Forms:** OE *peopor* (rare), OE *piper* (transmission error), OE *piper*, OE *piper* (rare).

**Frequency (in current use):**

*in taste and in some cases are used as a substitute for it.

**Etymology:** A borrowing from Latin. *Etymon:* Latin *piper*.

- classical Latin *piper*, a loanword < Indo-Aryan (as is ancient Greek *πίπερι*); compare Sanskrit *πिपर*.

**Usu. with distinguishing word:** any of numerous plants of other families having hot pungent fruits or leaves which resemble pepper (1a) in taste and in some cases are used as a substitute for it.

1. The spice or the plant.

a. A hot pungent spice derived from the prepared fruits (peppercorns) of the pepper plant, *Piper nigrum* (see sense 2a), used from early times to season food, either whole or ground to powder (often in association with salt). Also (locally, chiefly with distinguishing word): a similar spice derived from the fruits of certain other species of the genus *Piper*; the fruits themselves.

   - The ground spice from *Piper nigrum* comes in two forms, the more pungent *black pepper*, produced from black peppercorns, and the milder *white pepper*, produced from white peppercorns: see *black* *adj.* and *n.* Special uses 5a, *peppercorn* *n.* 1a, and *white* *adj.* and *n.* Special uses 7b(a).

b. The plant *Piper nigrum* (family *Piperaceae*), a climbing shrub indigenous to South Asia and also cultivated elsewhere in the tropics, which has alternate stalked entire leaves, with pendulous spikes of small green flowers, opposite the leaves, succeeded by small berries turning red when ripe. Also more widely: any plant of the genus *Piper* or the family *Piperaceae*.

c. *U.S. The California pepper tree, Schinus molle.* Cf. *pepper tree* *n.*

2. Any of various forms of capsicum, esp. *Capsicum annuum* var. *annuum*. Originally (chiefly with distinguishing word): any variety of the *C. annuum* Longum group, with elongated fruits having a hot, pungent taste, the source of cayenne, chilli powder, paprika, etc., or of the perennial *C. frutescens*, the source of Tabasco sauce. Now frequently (more fully *sweet pepper*): any variety of the *C. annuum* Grossum group, with large, bell-shaped or apple-shaped, mild-flavoured fruits, usually ripening to red, orange, or yellow and eaten raw in salads or cooked as a vegetable. Also: the fruit of any of these capsicums.

   - Sweet peppers are often used in their green immature state (more fully *green pepper*), but some new varieties remain green when ripe.
Relation: Synonymity

Synonyms have the same meaning in some or all contexts.

- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- Water / H₂O
Relation: Antonymy

Senses that are opposites with respect to one feature of meaning

Otherwise, they are very similar!

dark/light  short/long  fast/slow  rise/fall
hot/cold  up/down  in/out
Relation: Similarity

Words with similar meanings. Not synonyms, but sharing some element of meaning

car, bicycle

cow, horse
Ask humans how similar two words are on scale of 1-10

<table>
<thead>
<tr>
<th>word1</th>
<th>word2</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 dataset (Hill et al., 2015)
in NLP, we commonly represent word types with vectors!
why use vectors to encode meaning?

• computing the similarity between two words (or phrases, or documents) is extremely useful for many NLP tasks

• Q: how tall is Mount Everest?
  A: The official height of Mount Everest is 29029 ft
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

**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
visualizing semantic word change over time

~30 million books, 1850-1990, Google Books data
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

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...
Intuition of distributional word similarity

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.
one-hot vectors

- we’ve already seen these before in bag-of-words models (e.g., naive Bayes)!
- represent each word as a vector of zeros with a single 1 identifying the index of the word

<table>
<thead>
<tr>
<th>vocabulary</th>
<th>movie</th>
<th>film</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>&lt;0, 0, 0, 0, 1, 0&gt;</td>
<td>&lt;0, 0, 0, 0, 0, 1&gt;</td>
</tr>
<tr>
<td>hate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>love</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the</td>
<td></td>
<td></td>
</tr>
<tr>
<td>movie</td>
<td></td>
<td></td>
</tr>
<tr>
<td>film</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

what are the issues of representing a word this way?
all words are equally (dis)similar!

movie = <0, 0, 0, 0, 1, 0>
film = <0, 0, 0, 0, 0, 1>

dot product is zero!
these vectors are orthogonal

how can we compute a vector representation such that the dot product correlates with word similarity?
We'll introduce 2 kinds of embeddings

**Tf-idf**
- A common baseline model
- Sparse vectors
- Words are represented by a simple function of the counts of nearby words

**Word2vec**
- Dense vectors
- Representation is created by training a classifier to distinguish nearby and far-away words
Two words are similar in meaning if their context vectors are similar

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** jam, a pinch each of, and another fruit whose taste she likened

**pineapple** In finding the optimal R-stage policy from

**digital** necessary for the study authorized in the

**information**

<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>
</tbody>
</table>

Figure 6.5 Co-occurrence vectors for four words, computed from the Brown corpus, showing only six of the dimensions (hand-picked for pedagogical purposes). The vector for the word digital is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.
cosine similarity of two vectors

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

\( v_i \) is the count for word \( v \) in context \( i \)

\( w_i \) is the count for word \( w \) in context \( i \).

\[ \text{Cos}(v, w) \] is the cosine similarity of \( v \) and \( w \)

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

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

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

Frequency is non-negative, so cosine range 0-1
$$\cos(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}| |\vec{w}|} = \frac{\vec{v} \cdot \vec{w}}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}} = \sum_{i=1}^{N} v_i w_i$$

Which pair of words is more similar?

<table>
<thead>
<tr>
<th>Word</th>
<th>large</th>
<th>data</th>
<th>computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>1</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>

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

Which pair of words is more similar?

\[
\text{cosine}(\text{apricot}, \text{information}) = \frac{1+0+0}{\sqrt{1+0+0} \sqrt{1+36+1}} = \frac{1}{\sqrt{38}} = .16
\]

\[
\text{cosine}(\text{digital}, \text{information}) = \frac{0+6+2}{\sqrt{0+1+4} \sqrt{1+36+1}} = \frac{8}{\sqrt{38 \sqrt{5}}} = .58
\]

\[
\text{cosine}(\text{apricot}, \text{digital}) = \frac{0+0+0}{\sqrt{1+0+0} \sqrt{0+1+4}} = 0
\]
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!
tf-idf: combine two factors

**tf: term frequency.** frequency count (usually log-transformed):

\[
tf_{t,d} = \begin{cases} 
1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\
0 & \text{otherwise}
\end{cases}
\]

**Idf: inverse document frequency:** tf-

\[
idf_i = \log \left( \frac{N}{df_i} \right)
\]

Words like "the" or "good" have very low idf

**tf-idf value for word t in document d:**

\[
w_{t,d} = tf_{t,d} \times idf_t
\]
An alternative to tf-idf

Ask 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?

$$\text{PMI}(\text{word}_1, \text{word}_2) = \log_2 \frac{P(\text{word}_1, \text{word}_2)}{P(\text{word}_1)P(\text{word}_2)}$$
what is the range of values $\text{PMI}(w_1, w_2)$ can take?

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

$$(-\infty, \infty)$$

Positive $\text{PMI}(w_1, w_2)$:

$$\text{PPMI}(\text{word}_1, \text{word}_2) = \max \left( \log_2 \frac{P(\text{word}_1, \text{word}_2)}{P(\text{word}_1)P(\text{word}_2)}, 0 \right)$$
\[
\text{PMI}(\text{information}, \text{data}) = ???
\]
\[
\log_2 \left( \frac{0.32}{(0.37 \times 0.58)} \right) = 0.57
\]

PMI(information, data) = ???
\[
\begin{array}{llllll}
\text{computer} & \text{data} & \text{pinch} & \text{result} & \text{sugar} \\
\hline
\text{apricot} & 0.00 & 0.00 & 0.05 & 0.00 & 0.05 \\
\text{pineapple} & 0.00 & 0.00 & 0.05 & 0.00 & 0.05 \\
\text{digital} & 0.11 & 0.05 & 0.00 & 0.05 & 0.00 \\
\text{information} & 0.05 & 0.32 & 0.00 & 0.21 & 0.00 \\
\hline
\text{p(context)} & 0.16 & 0.37 & 0.11 & 0.26 & 0.11 \\
\end{array}
\]

\[
\begin{array}{llllll}
\text{computer} & \text{data} & \text{pinch} & \text{result} & \text{sugar} \\
\hline
\text{apricot} & - & - & 2.25 & - & 2.25 \\
\text{pineapple} & - & - & 2.25 & - & 2.25 \\
\text{digital} & 1.66 & 0.00 & - & 0.00 & - \\
\text{information} & 0.00 & 0.57 & - & 0.47 & - \\
\end{array}
\]
Tf-idf and PPMI are sparse representations

tf-idf and PPMI vectors are
  • long
  • sparse
Tf-idf and PPMI are sparse representations

tf-idf and PPMI vectors are

- long (length $|V| = 20,000$ to $50,000$)
- sparse (most elements are zero)
dense word vectors

• model the meaning of a word as an embedding in a vector space
  • this vector space is commonly low dimensional (e.g., 100-500d).
  • what is the dimensionality of a one-hot word representation?

• embeddings are real-valued vectors (not binary or counts)
how can we learn embeddings?

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. Brown clusters
Word2vec  (Mikolov et al., 2013)

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

- 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
  - But we'll take the learned classifier weights as the word embeddings
Brilliant insight: Use running text as implicitly supervised training data!

• A word $s$ near *apricot*
  • Acts as gold ‘correct answer’ to the question
  • “Is word $w$ likely to show up near *apricot*?”

• No need for hand-labeled supervision

• The idea comes from **neural language modeling**
  • Bengio et al. (2003)
  • Collobert et al. (2011)
Setup

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

So we start with $300 \times V$ random parameters.

Over the entire training set, we'd like to adjust those word vectors such that we

- Maximize the similarity of the target word, context word pairs $(t,c)$ drawn from the positive data
- Minimize the similarity of the $(t,c)$ pairs drawn from the negative data.
<table>
<thead>
<tr>
<th>word</th>
<th>dim0</th>
<th>dim1</th>
<th>dim2</th>
<th>dim3</th>
<th>...</th>
<th>dim300</th>
</tr>
</thead>
<tbody>
<tr>
<td>today</td>
<td>0.35</td>
<td>-1.3</td>
<td>2.2</td>
<td>0.003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>-3.1</td>
<td>-1.7</td>
<td>1.1</td>
<td>-0.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sleep</td>
<td>0.55</td>
<td>3.0</td>
<td>2.4</td>
<td>-1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>watch</td>
<td>-0.09</td>
<td>0.8</td>
<td>-1.8</td>
<td>2.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bird</td>
<td>2.0</td>
<td>0.16</td>
<td>-1.9</td>
<td>2.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Skip-gram with negative sampling (SGNS)

1. From a large source of text (e.g., Wikipedia), generate positive examples by pairing a target word with a word in its neighboring context.

2. Create negative examples for that target word by randomly sampling other words in the vocabulary.

3. Train a logistic regression model to identify whether a given pair of words is a positive or negative example.

4. Use the weights of this model as the embeddings.
Skip-Gram Training Data

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

c1 c2 target c3 c4

Asssume context words are those in +/- 2 word window
Skip-Gram Goal

Given a tuple \((t, c)\) = target, context

- \((\text{apricot, jam})\)
- \((\text{apricot, aardvark})\)

Return probability that \(c\) is a real context word:

\[
P(+) \mid t, c\)
\]

\[
P(-) \mid t, c = 1 - P(+) \mid t, c\)
\]
How to compute $p(+ | t, c)$?

Intuition:
- Words are likely to appear near similar words
- Model similarity with dot-product!
- $\text{Similarity}(t, c) \propto t \cdot c$

Problem:
- *Dot product is not a probability!*
  - *(Neither is cosine)*

$t$ and $c$ here are vectors for target and context!
Turning dot product into a probability

The sigmoid lies between 0 and 1:

\[ P(+) \mid t, c) \] (6.15)

The probability that word \( c \) is not a real context word for \( t \) is just 1 minus Eq. 6.15:

\[ P(\mid t, c) = 1 - P(+) \mid t, c) \] (6.16)

How does the classifier compute the probability \( P(+) \mid t, c) \)? The intuition of the skip-gram model is to base this probability on similarity: a word is likely to occur near the target if its embedding is similar to the target embedding. How can we compute similarity between embeddings? Recall that two vectors are similar if they have a high dot product (cosine, the most popular similarity metric, is just a normalized dot product). In other words:

\[ \text{Similarity}(t, c) \approx t \cdot c \] (6.17)

Of course, the dot product \( t \cdot c \) is not a probability, it's just a number ranging from 0 to 1. (Recall, for that matter, that cosine isn't a probability either). To turn the dot product into a probability, we'll use the logistic or sigmoid function \( s(x) \), the fundamental core of logistic regression:

\[ s(x) = \frac{1}{1 + e^{-x}} \] (6.18)

The probability that word \( c \) is a real context word for target word \( t \) is thus computed as:

\[ P(+) \mid t, c) = \frac{1}{1 + e^{-t \cdot c}} \] (6.19)

The sigmoid function just returns a number between 0 and 1, so to make it a probability we'll need to make sure that the total probability of the two possible events (\( c \) being a context word, and \( c \) not being a context word) sum to 1. The probability that word \( c \) is not a real context word for \( t \) is thus:

\[ P(\mid t, c) = \frac{e^{t \cdot c}}{1 + e^{t \cdot c}} \] (6.20)
Turning dot product into a probability

The sigmoid lies between 0 and 1:

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

The classifier

Let's start by thinking about the classification task, and then turn to how to train. Imagine a sentence like the following, with a target word apricot and assume we're using a window of ±2 context words:...

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

c1 c2 t c3 c4

Our goal is to train a classifier such that, given a tuple \((t, c)\) of a target word \(t\) paired with a candidate context word (for example \((\text{apricot, jam})\), or perhaps \((\text{apricot, aardvark})\) it will return the probability that \(c\) is a real context word (true for jam, false for aardvark):

\[ P(+) | t, c) \]

The probability that word \(c\) is not a real context word for \(t\) is just 1 minus Eq. 6.15:

\[ P(-) | t, c) = 1 - P(+) | t, c) \]

How does the classifier compute the probability \(P\)? The intuition of the skip-gram model is to base this probability on similarity: a word is likely to occur near the target if its embedding is similar to the target embedding. How can we compute similarity between embeddings? Recall that two vectors are similar if they have a high dot product (cosine, the most popular similarity metric, is just a normalized dot product). In other words:

\[ \text{Similarity}(t, c) \approx t \cdot c \]

Of course, the dot product \(t \cdot c\) is not a probability, it's just a number ranging from 0 to 1. (Recall, for that matter, that cosine isn't a probability either). To turn the dot product into a probability, we'll use the logistic or sigmoid function \(s(x)\), the fundamental core of logistic regression:

\[ s(x) = \frac{1}{1 + e^{-x}} \]

The probability that word \(c\) is a real context word for target word \(t\) is thus computed as:

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

The sigmoid function just returns a number between 0 and 1, so to make it a probability we'll need to make sure that the total probability of the two possible events (\(c\) being a context word, and \(c\) not being a context word) sum to 1. The probability that word \(c\) is not a real context word for \(t\) is thus:

\[ P(-) | t, c) = \frac{1}{1 + e^{-t \cdot c}} \]
Turning dot product into a probability

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

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

think back to last class… what are our features and weights here???

both target and context vectors are learned, so we have no explicit featurization!
Learning the classifier

Iterative process.

We’ll start with 0 or random weights

Then adjust the word weights to
  ◦ make the positive pairs more likely
  ◦ and the negative pairs less likely

over the entire training set

guess what algorithm we’ll use to make this happen?
gradient descent!!!!!!!
Objective Criteria

We want to maximize...

\[ \sum_{(t,c) \in +} logP(+) | t, c) + \sum_{(t,c) \in -} logP(- | t, c) \]

Maximize the + label for the pairs from the positive training data, and the – label for the pairs sample from the negative data.
Focusing on one target word $t$:

$$L(\theta) = \log P(+|t, c) + \sum_{i=1}^{k} \log P(-|t, n_i)$$

$$= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t)$$

$$= \log \frac{1}{1 + e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1 + e^{n_i \cdot t}}$$

$n_i$ is the vector for the negative sample
in practice, we learn two different sets of embeddings ($W$ for target words, $C$ for context words), but throw away $C$
Summary: How to learn word2vec (skip-gram) embeddings

Start with V random 300-dimensional vectors as initial embeddings

Use logistic regression, the second most basic classifier used in machine learning after naïve bayes

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance

- Throw away the classifier code and keep the embeddings.
Evaluating embeddings

Compare to human scores on word similarity-type tasks:

- WordSim-353 (Finkelstein et al., 2002)
- SimLex-999 (Hill et al., 2015)
- Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
- TOEFL dataset: *Levied is closest in meaning to:* imposed, believed, requested, correlated
Properties of embeddings

Similarity depends on window size $C$

$C = \pm 2$ The nearest words to *Hogwarts*:
- *Sunnydale*
- *Evernight*

$C = \pm 5$ The nearest words to *Hogwarts*:
- *Dumbledore*
- *Malfoy*
- *halfblood*
Analogy: Embeddings capture relational meaning!

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

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

huge concern for NLP systems deployed in the real world that use embeddings!
<table>
<thead>
<tr>
<th>Occupations</th>
<th>Adjectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Man</td>
<td>Woman</td>
</tr>
<tr>
<td>carpenter</td>
<td>nurse</td>
</tr>
<tr>
<td>mechanic</td>
<td>midwife</td>
</tr>
<tr>
<td>mason</td>
<td>librarian</td>
</tr>
<tr>
<td>blacksmith</td>
<td>housekeeper</td>
</tr>
<tr>
<td>retired</td>
<td>dancer</td>
</tr>
<tr>
<td>architect</td>
<td>teacher</td>
</tr>
<tr>
<td>engineer</td>
<td>cashier</td>
</tr>
<tr>
<td>mathematician</td>
<td>student</td>
</tr>
<tr>
<td>shoemaker</td>
<td>designer</td>
</tr>
<tr>
<td>physicist</td>
<td>weaver</td>
</tr>
</tbody>
</table>

Table 7: Top occupations and adjectives by gender in the Google News embedding.
Changes in framing: adjectives associated with Chinese

<table>
<thead>
<tr>
<th>1910</th>
<th>1950</th>
<th>1990</th>
</tr>
</thead>
<tbody>
<tr>
<td>Irresponsible</td>
<td>Disorganized</td>
<td>Inhibited</td>
</tr>
<tr>
<td>Envious</td>
<td>Outrageous</td>
<td>Passive</td>
</tr>
<tr>
<td>Barbaric</td>
<td>Pompous</td>
<td>Dissolute</td>
</tr>
<tr>
<td>Aggressive</td>
<td>Unstable</td>
<td>Haughty</td>
</tr>
<tr>
<td>Transparent</td>
<td>Effeminate</td>
<td>Complacent</td>
</tr>
<tr>
<td>Monstrous</td>
<td>Unprincipled</td>
<td>Forceful</td>
</tr>
<tr>
<td>Hateful</td>
<td>Venomous</td>
<td>Fixed</td>
</tr>
<tr>
<td>Cruel</td>
<td>Disobedient</td>
<td>Active</td>
</tr>
<tr>
<td>Greedy</td>
<td>Predatory</td>
<td>Sensitive</td>
</tr>
<tr>
<td>Bizarre</td>
<td>Boisterous</td>
<td>Hearty</td>
</tr>
</tbody>
</table>

I had tried building an algorithm for sentiment analysis based on word embeddings — evaluating how much people like certain things based on what they say about them. When I applied it to restaurant reviews, I found it was ranking Mexican restaurants lower. The reason was not reflected in the star ratings or actual text of the reviews. It’s not that people don’t like Mexican food. The reason was that the system had learned the word “Mexican” from reading the Web.
Directions

Debiasing algorithms for embeddings


Use embeddings as a historical tool to study bias
exercise!