Word Embeddings

Adapted by Diana Inkpen, 2021 for csi4107 at the University of Ottawa

From Chapter 6 of Speech and Language Processing (3rd ed.), by Dan Jurafsky and James H. Martin.
What do words mean?

Introductory logic classes:
- The meaning of "dog" is DOG; cat is CAT
  \[ \forall x \text{ DOG}(x) \rightarrow \text{MAMMAL}(x) \]

Word senses: look in a dictionary
http://www.oed.com/
pepper, n.

Pronunciation: Brit. /ˈpɛpə/, U.S. /ˈpɛpər/

Forms: OE peopor (rare), OE pipcer (transmission error), OE pipor, OE pipur (rare)

Frequency (in current use):

Etymology: A borrowing from Latin. Etymology: Latin pipor. < classical Latin pipor, a loanword < Indo-Aryan (as is ancient Greek πόρνα); compare Sanskrit पिर "pepper."

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 7a(b).

b. Used 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.

c. U.S. The California pepper tree, Schinus molle. Cf. PEPPER TREE n. 3.

3. 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 capiscums.

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

Sense 1: spice from pepper plant
Sense 2: the pepper plant itself
Sense 3: another similar plant (Jamaican pepper)
Sense 4: another plant with peppercorns (California pepper)
Sense 5: *capsicum* (i.e. chili, paprika, bell pepper, etc)

A *sense* or “concept” is the meaning component of a word
Relations between senses: Synonymy

Synonyms have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- water / H₂O
Relation: Synonymy

Note that there are probably no examples of perfect synonymy.

- Even if many aspects of meaning are identical
- Still may not preserve the acceptability based on notions of politeness, slang, register, genre, etc.
Relation: Synonymy?

water/H$_2$O
big/large
brave/courageous
The Linguistic Principle of Contrast

Difference in form $\rightarrow$ difference in meaning
Re: "exact" synonyms

"je ne crois pas qu’il y ait de mot synonime dans aucune Langue."

[I do not believe that there is a synonymous word in any language]
Relation: Similarity

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

car, bicycle
cow, horse
Ask humans how similar 2 words are

<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)
Relation: Word relatedness

Also called "word association"

Words can be related in any way, perhaps via a semantic frame or field

- car, bicycle: similar
- car, gasoline: related, not similar
Semantic field

Words that

- cover a particular semantic domain
- bear structured relations with each other.

**hospitals**

- *surgeon*, *scalpel*, *nurse*, *anaesthetic*, *hospital*

**restaurants**

- *waiter*, *menu*, *plate*, *food*, *menu*, *chef*

**houses**

- *door*, *roof*, *kitchen*, *family*, *bed*
Relation: Antonymy

Senses that are opposites with respect to only one feature of meaning

Otherwise, they are very similar!

- dark/light
- short/long
- fast/slow
- rise/fall
- hot/cold
- up/down
- in/out

More formally: antonyms can
- define a binary opposition or be at opposite ends of a scale
- long/short, fast/slow
- Be *reversives*:
  - rise/fall, up/down
Relation: Superordinate/ subordinate

One sense is a **subordinate** of another if the first sense is more specific, denoting a subclass of the other

- *car* is a subordinate of *vehicle*
- *mango* is a subordinate of *fruit*

Conversely **superordinate**

- *vehicle* is a superordinate of *car*
- *fruit* is a superordinate of *mango*

<table>
<thead>
<tr>
<th>Superordinate</th>
<th>vehicle</th>
<th>fruit</th>
<th>furniture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subordinate</td>
<td>car</td>
<td>mango</td>
<td>chair</td>
</tr>
</tbody>
</table>
So far

**Concepts** or word senses
- Have a complex many-to-many association with *words* (homonymy, multiple senses)

Have relations with each other
- Synonymy
- Antonymy
- Similarity
- Relatedness
- Superordinate/subordinate, basic level
- Connotation
Distributional Semantics
Let's define words by their usages

One way to define "usage": words are defined by their environments (the words around them)

Zellig Harris (1954):
If A and B have almost identical environments we say that they are synonyms.
What does recent English borrowing *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
A new model of meaning focusing on distributional similarity

Each word = a vector
- Not just "word" or word45.

Similar words are "nearby in space"
We define a word as a vector

Called an "embedding" because it's embedded into a space

The standard way to represent meaning in NLP

**Every modern NLP algorithm uses embeddings as the representation of word meaning**

Fine-grained model of meaning for similarity
Intuition: why vectors?

Consider sentiment analysis:

- With **words**, a feature is a word identity
  - Feature 5: 'The previous word was "terrible"
  - requires **exact same word** to be in training and test

- With **embeddings**:
  - Feature is a word vector
  - 'The previous word was vector [35,22,17...]
  - Now in the test set we might see a similar vector [34,21,14]
  - We can generalize to **similar but unseen** words!!!
Two kinds of embeddings

**tf-idf**
- Information Retrieval workhorse!
- 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 *predict* whether a word is likely to appear nearby
- In later chapters we'll discuss extensions called *contextual embeddings*
Word vectors: Term-document matrix

Each document is represented by a vector of words

<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>14</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
Different than the history

Comedies have more *fools* and *wit* and fewer *battles*.
Idea for word meaning: 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>...</th>
<th>computer</th>
<th>data</th>
<th>result</th>
<th>pie</th>
<th>sugar</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>cherry</td>
<td>0</td>
<td>...</td>
<td>2</td>
<td>8</td>
<td>9</td>
<td>442</td>
<td>25</td>
</tr>
<tr>
<td>strawberry</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>60</td>
<td>19</td>
</tr>
<tr>
<td>digital</td>
<td>0</td>
<td>...</td>
<td>1670</td>
<td>1683</td>
<td>85</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>information</td>
<td>0</td>
<td>...</td>
<td>3325</td>
<td>3982</td>
<td>378</td>
<td>5</td>
<td>13</td>
</tr>
</tbody>
</table>

- **cherry** pie, a traditional dessert
- **strawberry** rhubarb pie. Apple pie assistants. These devices usually available on the internet
- **digital** information available on the internet
Dot product and cosine

The dot product between two vectors is a scalar:

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

The dot product tends to be high when the two vectors have large values in the same dimensions. Dot product can be a similarity metric between vectors.
Problem with raw dot-product

Dot product favors long vectors

Dot product is higher if a vector is longer (has higher values in many dimension)

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

Frequent words (of, the, you) have long vectors (since they occur many times with other words).

So dot product overly favors frequent words
Alternative: cosine for computing word similarity

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

The same as using normalized vectors.
Cosine as a similarity metric

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

But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1
Cosine examples

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

\[
\cos(\text{cherry}, \text{information}) = \frac{442 \times 5 + 8 \times 3982 + 2 \times 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017
\]

\[
\cos(\text{digital}, \text{information}) = \frac{5 \times 5 + 1683 \times 3982 + 1670 \times 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996
\]
Visualizing cosines (well, angles)

Dimension 1: ‘pie’

500 cherry

500 1000 1500 2000 2500 3000

Dimension 2: ‘computer’
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!
Two common solutions for word weighting

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

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

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

**PMI:** (Pointwise mutual information)

- $\text{PMI}(w_1, w_2) = \log \frac{p(w_1, w_2)}{p(w_1)p(w_2)}$

See if words like "good" appear more often with "great" than we would expect by chance
Term frequency (tf)

\[ tf_{t,d} = \text{count}(t,d) \]

Instead of using raw count, we squash a bit:

\[ tf_{t,d} = \log_{10}(\text{count}(t,d)+1) \]
Document frequency (df)

df\_t is the number of documents t occurs in.

(note this is not collection frequency: total count across all documents)

"Romeo" is very distinctive for one Shakespeare play:

<table>
<thead>
<tr>
<th></th>
<th>Collection Frequency</th>
<th>Document Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Romeo</td>
<td>113</td>
<td>1</td>
</tr>
<tr>
<td>action</td>
<td>113</td>
<td>31</td>
</tr>
</tbody>
</table>

Important: documents can be anything; we can call each paragraph a document
Inverse document frequency (idf)

\[ \text{idf}_t = \log_{10} \left( \frac{N}{\text{df}_t} \right) \]

N is the total number of documents in the collection

<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>
Final tf-idf weighted value for a word

$$w_{t,d} = tf_{t,d} \times idf_t$$

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>
Sparse versus 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)
Sparse versus dense vectors

Why dense vectors?

- Short vectors may be easier to use as features in machine learning (fewer weights to tune)
- Dense vectors may generalize better than explicit counts
- They may do better at capturing synonymy:
  - car and automobile are synonyms; but are distinct dimensions
    - a word with car as a neighbor and a word with automobile as a neighbor should be similar, but aren't
- In practice, they work better
Common methods for getting short dense vectors

“Neural Language Model”-inspired models

◦ Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)

◦ A special case of this is called LSA – Latent Semantic Analysis

Alternative to these "static embeddings":

• Contextual Embeddings (ELMo, BERT)
• Compute distinct embeddings for a word in its context
• Separate embeddings for parts of words (FastText)
Embeddings you can download!

Word2vec (Mikolov et al)
https://code.google.com/archive/p/word2vec/

Glove (Pennington, Socher, Manning)
http://nlp.stanford.edu/projects/glove/

A lot more
Word2vec

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

Big idea: self-supervision:

- A word $c$ that occurs near apricot in the corpus asks as the gold "correct answer" for supervised learning
- No need for human labels
- Bengio et al. (2003); Collobert et al. (2011)
Word2Vec: Skip-grams vs. CBOW (Mikolov, 2014 Tutorial)

Skip-grams

CBOCW
Word2Vec: Skip-Grams

Word2vec provides a variety of options. We'll do:

skip-grams with negative sampling (SGNS)
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 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

(assuming a +/- 2 word window)

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

Goal: train a classifier that is given a candidate \((w, c)\) pair

(apricot, tablespoon)

(apricot, aardvark)

... And assigns each pair a probability:

\[ P(+|w, c) \]
Similarity is computed from dot product

Remember: two vectors are similar if they have a high dot product
  ◦ Cosine is just a normalized dot product

So:
  ◦ $\text{Similarity}(w,c) \propto w \cdot c$

We’ll need to normalize to get a probability
  ◦ (cosine isn't a probability either)
Turning dot products into probabilities

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

To turn this into a probability

We'll use the sigmoid from logistic regression:

\[
\begin{align*}
P(+|w,c) & = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)} \\
P(-|w,c) & = 1 - P(+|w,c) \\
& = \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)}
\end{align*}
\]
How Skip-Gram Classifier computes $P(+|w, c)$

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

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

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

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

A probabilistic classifier that, given a test target word $w$, 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 $c$
Skip-Gram Training data

...lemon, a [tablespoon of apricot jam, a] pinch... c1 c2 [target] c3 c4

positive examples +

t     c

_____________________
apricot  tablespoon
apricot  of
apricot  jam
apricot  a
Skip-Gram Training data

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

c1        c2 [target]        c3        c4

positive examples +

t c
apricot tablespoon
apricot of
apricot jam
apricot a

For each positive example we'll grab $k$ negative examples, sampling by frequency
Skip-Gram Training data

...lemon, a **tablespoon of apricot jam, a** pinch...

<table>
<thead>
<tr>
<th>positive examples +</th>
<th>negative examples -</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>tablespoon</td>
</tr>
<tr>
<td>apricot</td>
<td>of</td>
</tr>
<tr>
<td>apricot</td>
<td>jam</td>
</tr>
<tr>
<td>apricot</td>
<td>a</td>
</tr>
</tbody>
</table>

\[c_1 \quad c_2 [\text{target}] \quad c_3 \quad c_4\]
Word2vec: how to learn vectors

Given the set of positive and negative training instances, and an initial set of embedding vectors

The goal of learning is to adjust those word vectors such that we:

- **Maximize** the similarity of the **target word**, **context word** pairs \((w , c_{pos})\) drawn from the positive data
- **Minimize** the similarity of the \((w , c_{neg})\) pairs drawn from the negative data.
Loss function for one $w$ with $c_{pos}$, $c_{neg1} \ldots c_{negk}$

Maximize the dot product of the word with the actual context words, and minimize the dot products of the word with the $k$ negative sampled non-neighbor words.

$$L_{CE} = -\log \left[ P(+|w,c_{pos}) \prod_{i=1}^{k} P(-|w,c_{negi}) \right]$$

$$= - \left[ \log P(+|w,c_{pos}) + \sum_{i=1}^{k} \log P(-|w,c_{negi}) \right]$$

$$= - \left[ \log P(+|w,c_{pos}) + \sum_{i=1}^{k} \log (1 - P(+|w,c_{negi})) \right]$$

$$= - \left[ \log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{negi} \cdot w) \right]$$
Learning the classifier

How to learn?
  ◦ Stochastic gradient descent!

We’ll adjust the word weights to
  ◦ make the positive pairs more likely
  ◦ and the negative pairs less likely,
  ◦ over the entire training set.
Intuition of one step of gradient descent

move *apricot* and *jam* closer, increasing $c_{pos} \cdot w$

“...apricot jam...”

move *apricot* and *matrix* apart, decreasing $c_{neg1} \cdot w$

move *apricot* and *Tolstoy* apart, decreasing $c_{neg2} \cdot w$
The derivatives of the loss function

\[
L_{CE} = - \left[ \log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w) \right]
\]

\[
\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w
\]

\[
\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w
\]

\[
\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w)]c_{neg_i}
\]
Update equation in SGD

Start with randomly initiatized $C$ and $W$ matrices, then incrementally do updates

$$
c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w) - 1]w
$$

$$
c_{neg}^{t+1} = c_{neg}^t - \eta [\sigma(c_{neg}^t \cdot w)]w
$$

$$
w^{t+1} = w^t - \eta [\sigma(c_{pos} \cdot w^t) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w^t)]c_{neg_i}
$$
Two sets of embeddings

SGNS learns two sets of embeddings
  Target embeddings matrix $W$
  Context embedding matrix $C$

It's common to just add them together, representing word $i$ as the vector $w_i + c_i$
Summary: How to learn word2vec (skip-gram) embeddings

Start with $V$ random $d$-dimensional vectors as initial embeddings

Train a classifier based on embedding similarity
  ◦ 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.
Properties of Embeddings

Large windows (C = +/- 5): nearest words are related words in same semantic field
  • *Hogwarts* nearest neighbors are Harry Potter world:
    • *Dumbledore, Half-blood, Malfoy*

Small windows (C = +/- 2): nearest words are similar nouns, words in same taxonomy
  • *Hogwarts* nearest neighbors are other fictional schools
    • *Sunnydale, Evernight, Blandings*
Word2vec: CBOW

• The ‘continuous bag-of-words model’ (CBOW) adds inputs from words within short window to predict the current word.
• The weights for different positions are shared.
• The hidden layer is linear.

(Mikolov, 2014 Tutorial)
Skip-grams vs. CBOW (Mikolov, 2014 Tutorial)

Skip-gram: work well with a small amount of the training data, represent well even rare words or phrases.

CBOW: faster to train, slightly better accuracy for the frequent words.
FastText Embeddings

• FastText is another word embedding method that is an extension of the word2vec model.
• Instead of learning vectors for words directly, fastText represents each word as an n-gram of characters.
• This helps capture the meaning of shorter words and allows the embeddings to understand suffixes and prefixes.
GloVe Embeddings (Pennington et al. 2014)

- Count-based method not neural network.
- Very large corpus.
- Log-bilinear model with a weighted least-squares objective.
- Focus on encoding vector differences.
Sentence and document embeddings

Can use word vectors and average over all words in the text, or over selected words (by idf) or build text vectors directly.

- Sent2vec
- Doc2vec

For IR:
- Produce a dense vector for the query.
- Produce dense vectors for documents.
- Compute cosine similarities (too many, still need a classical IR system to add neural retrieval on top).