How do we represent the meaning of a word?

Definition: **Meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.
How to represent meaning in a computer?

Common answer: Use a taxonomy like WordNet that has hypernyms (is-a) relationships and synonym sets (good):

```python
from nltk.corpus import wordnet as wn
panda = wn.synset('panda.n.01')
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
 Synset('carnivore.n.01'),
 Synset('placental.n.01'),
 Synset('mammal.n.01'),
 Synset('vertebrate.n.01'),
 Synset('chordate.n.01'),
 Synset('animal.n.01'),
 Synset('organism.n.01'),
 Synset('living_thing.n.01'),
 Synset('whole.n.02'),
 Synset('object.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]  
S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced, proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
...
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good
```
Problems with this discrete representation

• Great as resource but missing nuances, e.g. *synonyms*: adept, expert, good, practiced, proficient, skillful?

• Missing new words (impossible to keep up to date): wicked, badass, nifty, crack, ace, wizard, genius, ninja

• Subjective

• Requires human labor to create and adapt

• Hard to compute accurate word similarity →
Problems with this discrete representation

The vast majority of rule-based and statistical NLP work regards words as atomic symbols: hotel, conference, walk

In vector space terms, this is a vector with one 1 and a lot of zeroes

\[
[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ ]
\]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

We call this a “one-hot” representation. Its problem:

\[
\text{motel } [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ ] \ \text{AND} \ \text{hotel } [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] = 0
\]
Distribu9ional similarity based representations

You can get a lot of value by representing a word by means of its neighbors

“You shall know a word by the company it keeps”

(J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

```
government debt problems turning into banking crises as has happened in
saying that Europe needs unified banking regulation to replace the hodgepodge

⇒ These words will represent banking ⇒
```
How to make neighbors represent words?

Answer: With a cooccurrence matrix X

• 2 options: full document vs windows

• Word - document cooccurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”

• Window allows us to capture both syntactic (POS) and semantic information
Window based cooccurrence matrix

- Window length 1 (more common: 5 - 10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.
Window based cooccurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

<table>
<thead>
<tr>
<th>counts</th>
<th>I</th>
<th>like</th>
<th>enjoy</th>
<th>deep</th>
<th>learning</th>
<th>NLP</th>
<th>flying</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>like</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>enjoy</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>deep</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>learning</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NLP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>flying</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Problems with simple cooccurrence vectors

Increase in size with vocabulary

Very high dimensional: require a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust
Solution: Low dimensional vectors

• Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector

• Usually around 25 – 1000 dimensions

• How to reduce the dimensionality?
Method 1: Dimensionality Reduction on $X$

Singular Value Decomposition of cooccurrence matrix $X$.

$X = \hat{X}$

$X = n \times m$

$\hat{X} = k \times m$

$U = n \times r$

$S = r \times r$

$V^T = m \times r$

$\hat{U} = n \times k$

$\hat{S} = k \times k$

$\hat{V}^T = m \times k$

$\hat{X}$ is the best rank $k$ approximation to $X$, in terms of least squares.

Richard Socher

4/1/15
Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

```python
import numpy as np
la = np.linalg

corpus = [
    "I", "like", "enjoy",
    "deep", "learning", "NLP", "flying", "."
]

words = [corpus]

X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```
Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values

```python
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```
Word meaning is defined in terms of vectors

- In all subsequent models, including deep learning models, a word is represented as a dense vector

\[
\text{linguistics} = \begin{pmatrix}
0.286 \\
0.792 \\
-0.177 \\
-0.107 \\
0.109 \\
-0.542 \\
0.349 \\
0.271
\end{pmatrix}
\]
Hacks to X

• Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
  • \( \min(X, t) \), with \( t \sim 100 \)
  • Ignore them all
• Ramped windows that count closer words more
• Use Pearson correlations instead of counts, then set negative values to 0
• +++
Interesting semantic patterns emerge in the vectors

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005
Interesting semantic patterns emerge in the vectors

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005
Interesting semantic patterns emerge in the vectors

An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005
Problems with SVD

Computational cost scales quadratically for $n \times m$ matrix:

$O(mn^2)$ flops (when $n<m$)

→ Bad for millions of words or documents

Hard to incorporate new words or documents

Different learning regime than other DL models
Idea: Directly learn low-dimensional word vectors

• Old idea. Relevant for this lecture & deep learning:
  • Learning representations by back-propagating errors. (Rumelhart et al., 1986)
  • A neural probabilistic language model (Bengio et al., 2003)
  • NLP from Scratch (Collobert & Weston, 2008)
  • A recent and even simpler model: word2vec (Mikolov et al. 2013) → intro now
Main Idea of word2vec

• Instead of capturing cooccurrence counts directly,
• Predict surrounding words of every word
• Both are quite similar, see “Glove: Global Vectors for Word Representation” by Pennington et al. (2014)

• Faster and can easily incorporate a new sentence/document or add a word to the vocabulary
Details

- Predict surrounding words in a window of length c of every word.
- Objective function: Maximize the log probability of any context word given the current center word:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$
Details

- Predict surrounding words in a window of length $c$ of every word
- For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(w_O|w_I) = \frac{\exp \left( v'_w^\top v_{wI} \right)}{\sum_{w=1}^W \exp \left( v'_w^\top v_{wI} \right)}$$

- where $v$ and $v'$ are “input” and “output” vector representations of $w$ (so every word has two vectors!)
- This is essentially “dynamic” logistic regression
Cost/Objective functions

We will optimize (maximize or minimize) our objective/cost functions

For now: minimize $\rightarrow$ gradient descent

Refresher with trivial example: (from Wikipedia)
Find a local minimum of the function $f(x)=x^4-3x^3+2$, with derivative $f'(x)=4x^3-9x^2$.

```python
x_old = 0
x_new = 6  # The algorithm starts at x=6
eps = 0.01  # step size
precision = 0.00001

def f_derivative(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_derivative(x_old)

print("Local minimum occurs at", x_new)
```
Derivations of gradient

• Whiteboard (see video if you’re not in class ;)

• Most basic Lego piece, speed will depend on participation

• Useful basics: \[
\frac{\partial x^T a}{\partial x} = \frac{\partial a^T x}{\partial x} = a
\]

• Chain rule! If \( y = f(u) \) and \( u = g(x) \), i.e. \( y=f(g(x)) \), then:

\[
\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}
\]
Whiteboard!
Approximations: PSet 1

• With large vocabularies this objective function is not scalable and would train too slowly! Why?

• Idea: approximate the normalization or

• Define negative prediction that only samples a few words that do not appear in the context

• Similar to focusing on mostly positive correlations

• You will derive and implement this in Pset 1!
Linear Relationships in word2vec

These representations are very good at encoding dimensions of similarity!

• Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

   Syntactically
   • \( \mathbf{x}_{apple} - \mathbf{x}_{apples} \approx \mathbf{x}_{car} - \mathbf{x}_{cars} \approx \mathbf{x}_{family} - \mathbf{x}_{families} \)

   • Similarly for verb and adjective morphological forms

   Semantically (Semeval 2012 task 2)
   • \( \mathbf{x}_{shirt} - \mathbf{x}_{clothing} \approx \mathbf{x}_{chair} - \mathbf{x}_{furniture} \)
   • \( \mathbf{x}_{king} - \mathbf{x}_{man} \approx \mathbf{x}_{queen} - \mathbf{x}_{woman} \)
Count based vs direct prediction

- **LSA, HAL** (Lund & Burgess),
- **COALS** (Rohde et al),
- **Hellinger-PCA** (Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to small counts

- **NNLM, HLBL, RNN, Skip-gram/CBOW**, (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity
Combining the best of both worlds: GloVe

\[ J = \frac{1}{2} \sum_{ij} f(P_{ij})(w_i \cdot \tilde{w}_j - \log P_{ij})^2 \]

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus, and small vectors
Glove results

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus
Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:?  

man:woman :: king:?  

+ king  [ 0.30 0.70 ]  
- man  [ 0.20 0.20 ]  
+ woman  [ 0.60 0.30 ]  
queen  [ 0.70 0.80 ]  

d = \arg \max_x \frac{\langle w_b - w_a + w_c \rangle^T w_x}{\| w_b - w_a + w_c \|}
Glove Visualizations
Glove Visualizations: Company - CEO

Richard Socher

4/1/15
Word embedding matrix

- Initialize most word vectors of future models with our “pre-trained” embedding matrix \( L \in \mathbb{R}^{n \times |V|} \)

\[
L = \begin{bmatrix}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & |V| & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}^n
\]

aardvark  a  at  ... 

- Also called a look-up table
  - Conceptually you get a word’s vector by left multiplying a one-hot vector \( e \) (of length \( |V| \)) by \( L \): \( x = Le \)
Advantages of low dimensional word vectors

What is the major benefit of deep learned word vectors?

Ability to also propagate any information into them via neural networks (next lecture).

\[
P(c \mid d, \lambda) = \frac{e^{\lambda^T f(c,d)}}{\sum_{c'} e^{\lambda^T f(c',d)}}
\]

\[
p(c \mid x) = \frac{\exp(S c \cdot a)}{\sum_{c'} \exp(S c' \cdot a)}
\]
Advantages of low dimensional word vectors

• Word vectors will form the basis for all subsequent lectures.

• All our semantic representations will be vectors!

• We can compute compositional representations for longer phrases or sentences with them and solve lots of different tasks. ➔ Next lecture!