Word Embeddings

Fall 2019

(Slides adapted from Chris Manning, Dan Jurafsky)
How to represent words?

N-gram language models

It is 76 F and ____.

P(w | it is 76 F and)

[0.0001, 0.1, 0, 0, 0.002, ..., 0.3, ..., 0]

red

sunny

Text classification

I like this movie. 👍

I don’t like this movie. 👎

P(y = 1 | x) = σ(θᵀw + b)

w(1) [0, 1, 0, 0, ..., 1, ..., 1]

w(2) [0, 1, 0, 1, 0, ..., 1, ..., 1]
Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols: hotel, conference, motel — a localist representation

Words can be represented by one-hot vectors:

hotel  = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
motel = [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000)

There is no way to encode similarity of words in these vectors!
Representing words by their context

**Distributional hypothesis:** words that occur in similar contexts tend to have similar meanings

J.R. Firth 1957

- “You shall know a word by the company it keeps”
- One of the most successful ideas of modern statistical NLP!

These context words will represent *banking*. 
"tejuino"

C1: A bottle of ____ is on the table.

C2: Everybody likes ____.

C3: Don’t have ____ before you drive.

C4: We make ____ out of corn.
Distributional hypothesis

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>tejuino</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>loud</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>motor-oil</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>tortillas</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>choices</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>wine</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

“words that occur in similar contexts tend to have similar meanings”
Words as vectors

• We’ll build a new model of meaning focusing on similarity
  • Each word is a vector
  • Similar words are “nearby in space”

• A first solution: we can just use context vectors to represent the meaning of words!

• word-word co-occurrence matrix:

<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>
Words as vectors

\[
\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}
\]

\[
\cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{V} u_i v_i}{\sqrt{\sum_{i=1}^{V} u_i^2} \sqrt{\sum_{i=1}^{V} v_i^2}}
\]

What is the range of \(\cos(\cdot)\)?
Words as vectors

Problem: using raw frequency counts is not always very good..

- Solution: let’s weight the counts!
- PPMI = Positive Pointwise Mutual Information

\[
PPMI(w, c) = \max(\log_2 \frac{P(w, c)}{P(w)P(c)}, 0)
\]

<table>
<thead>
<tr>
<th></th>
<th>computer</th>
<th>data</th>
<th>result</th>
<th>pie</th>
<th>sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>cherry</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>0</td>
<td>1</td>
<td>60</td>
<td>19</td>
</tr>
<tr>
<td>digital</td>
<td>1670</td>
<td>1683</td>
<td>85</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>information</td>
<td>3325</td>
<td>3982</td>
<td>378</td>
<td>5</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>computer</th>
<th>data</th>
<th>result</th>
<th>pie</th>
<th>sugar</th>
</tr>
</thead>
<tbody>
<tr>
<td>cherry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.38</td>
<td>3.30</td>
</tr>
<tr>
<td>strawberry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.10</td>
<td>5.51</td>
</tr>
<tr>
<td>digital</td>
<td>0.18</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>information</td>
<td>0.02</td>
<td>0.09</td>
<td>0.28</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Sparse vs dense vectors

- Still, the vectors we get from word-word occurrence matrix are sparse (most are 0’s) & long (vocabulary size)

- Alternative: we want to represent words as **short** (50-300 dimensional) & **dense** (real-valued) vectors
  - The focus of this lecture
  - The basis of all the modern NLP systems
Dense vectors

employees =

\[
\begin{pmatrix}
0.286 \\
0.792 \\
-0.177 \\
-0.107 \\
10.109 \\
-0.542 \\
0.349 \\
0.271 \\
0.487
\end{pmatrix}
\]
Why dense vectors?

- Short vectors are easier to use as features in ML systems
- Dense vectors may generalize better than storing explicit counts
- They do better at capturing synonymy
  - $w_1$ co-occurs with “car”, $w_2$ co-occurs with “automobile”

- Different methods for getting dense vectors:
  - Singular value decomposition (SVD)
  - word2vec and friends: “learn” the vectors!
Word2vec and friends

(Mikolov et al, 2013): Distributed Representations of Words and Phrases and their Compositionality
Word2vec

- **Input:** a large text corpora, $V, d$
  - $V$: a pre-defined vocabulary
  - $d$: dimension of word vectors (e.g. 300)
  - Text corpora:
    - Wikipedia + Gigaword 5: 6B
    - Twitter: 27B
    - Common Crawl: 840B

- **Output:** $f : V \rightarrow \mathbb{R}^d$

\[
\begin{align*}
v_{\text{cat}} &= \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} & v_{\text{dog}} &= \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix} \\
v_{\text{the}} &= \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} & v_{\text{language}} &= \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}
\end{align*}
\]
**Word2vec**

<table>
<thead>
<tr>
<th>Word</th>
<th>Cosine distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>norway</td>
<td>0.760124</td>
</tr>
<tr>
<td>denmark</td>
<td>0.715460</td>
</tr>
<tr>
<td>finland</td>
<td>0.620022</td>
</tr>
<tr>
<td>switzerland</td>
<td>0.588132</td>
</tr>
<tr>
<td>belgium</td>
<td>0.585835</td>
</tr>
<tr>
<td>netherlands</td>
<td>0.574631</td>
</tr>
<tr>
<td>iceland</td>
<td>0.562368</td>
</tr>
<tr>
<td>estonia</td>
<td>0.547621</td>
</tr>
<tr>
<td>slovenia</td>
<td>0.531408</td>
</tr>
</tbody>
</table>

word = “sweden”
Word2vec

Continuous Bag of Words (CBOw)

Skip-grams
Skip-gram

- The idea: we want to use words to predict their context words
- Context: a fixed window of size $2m$
Skip-gram
Skip-gram: objective function

- For each position $t = 1, 2, \ldots, T$, predict context words within context size $m$, given center word $w_j$:

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta)$$

- The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log \mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta)$$
How to define $P(w_{t+j} \mid w_t; \theta)$?

- We have two sets of vectors for each word in the vocabulary
  
  $u_i \in \mathbb{R}^d$: embedding for target word $i$
  
  $v_{i'} \in \mathbb{R}^d$: embedding for context word $i'$

- Use inner product $u_i \cdot v_{i'}$ to measure how likely word $i$ appears with context word $i'$, the larger the better

  “softmax” we learned last time!

  $$P(w_{t+j} \mid w_t) = \frac{\exp(u_{w_t} \cdot v_{w_{t+j}})}{\sum_{k \in V} \exp(u_{w_t} \cdot v_k)}$$

  $\theta = \{\{u_k\}, \{v_k\}\}$ are all the parameters in this model!

  Q: Why two sets of vectors?
How to train the model

Calculating all the gradients together!

$$\theta = \{\{u_k\}, \{v_k\}$$

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} | w_t; \theta) \quad \nabla_\theta J(\theta) = ?$$

Q: How many parameters are in total?

We can apply stochastic gradient descent (SGD)!

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta J(\theta)$$

Let’s walk through the math..
Warm-up

\[ f(x) = \exp(x) \quad \frac{df}{dx} = \exp(x) \]

\[ f(x) = \log(x) \quad \frac{df}{dx} = \frac{1}{x} \]

chain rule:

\[ f(x) = f_1(f_2(x)) \quad \frac{df}{dx} = \frac{df_1(z)}{dz} \frac{df_2(x)}{dx} \quad z = f_2(x) \]

\[ f(x) = x \cdot a \quad \frac{\partial f}{\partial x} = a \]

\[ \frac{\partial f}{\partial x} = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}] \]
Computing the gradients

Consider one pair of target/context words (t, c):

\[ y = - \log \left( \frac{\exp(u_t \cdot v_c)}{\sum_{k \in V} \exp(u_t \cdot v_k)} \right) \]

\[ \frac{\partial y}{\partial u_t} = \frac{\partial}{\partial u_t} \left( -u_t \cdot v_c + \log \left( \sum_{k \in V} \exp(u_t \cdot v_k) \right) \right) \]

\[ = -v_c + \frac{\sum_{k \in V} \frac{\partial \exp(u_t \cdot v_k)}{\partial u_t}}{\sum_{k \in V} \exp(u_t \cdot v_k)} \]

\[ = -v_c + \frac{\sum_{k \in V} \exp(u_t \cdot v_k)v_k}{\sum_{k \in V} \exp(u_t \cdot v_k)} \]

\[ = -v_c + \sum_{k \in V} P(k \mid t)v_k \]

\[ \frac{\partial y}{\partial v_k} = -1(k = c)u_t + P(k \mid t)u_t \]

Make sure you know how to do this!
Putting it together

- Input: text corpus, context size m, embedding size d, V
- Initialize $u_i, v_i$ randomly
- Walk through the training corpus and collect training data (t, c):
  - Update $u_t \leftarrow u_t - \eta \frac{\partial y}{\partial u_t}$
  - Update $v_k \leftarrow v_k - \eta \frac{\partial y}{\partial v_k}, \forall k \in V$

Any issues?
Skip-gram with negative sampling (SGNS)

Problem: every time you get one pair of \((t, c)\), you need to use \(v_k\) with all the words in the vocabulary! It is very computationally expensive.

\[
\frac{\partial y}{\partial u_t} = -v_c + \sum_{k \in V} P(k \mid t) v_k \\
\frac{\partial y}{\partial v_k} = -1(k = c) u_t + P(k \mid t) u_t
\]

**Negative sampling**: instead of considering all the words in \(V\), let’s randomly sample \(K\) (5-20) negative examples.

softmax: \[
y = -\log \left( \frac{\exp(u_t \cdot v_c)}{\sum_{k \in V} \exp(u_t \cdot v_k)} \right)
\]

NS: \[
y = -\log(\sigma(u_t \cdot v_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-u_t \cdot v_j))
\]
Skip-gram with negative sampling (SGNS)

\[ y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j)) \]

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

**positive examples +**

<table>
<thead>
<tr>
<th>t</th>
<th>c</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>

**negative examples -**

<table>
<thead>
<tr>
<th>t</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>apricot</td>
<td>aardvark</td>
</tr>
<tr>
<td>apricot</td>
<td>my</td>
</tr>
<tr>
<td>apricot</td>
<td>where</td>
</tr>
<tr>
<td>apricot</td>
<td>coaxial</td>
</tr>
</tbody>
</table>

Same as training a **logistic regression** for binary classification!

\[ P(D = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c) \]

Compute the gradients: assignment 2!
Continuous Bag of Words (CBOW)

\[
L(\theta) = \prod_{t=1}^{T} P(w_t \mid \{w_{t+j}\}, -m \leq j \leq m, j \neq 0)
\]

\[
\vec{v}_t = \frac{1}{2m} \sum_{-m \leq j \leq m, j \neq 0} v_{t+j}
\]

\[
P(w_t \mid \{w_{t+j}\}) = \frac{\exp(u_{w_t} \cdot \vec{v}_t)}{\sum_{k \in V} \exp(u_k \cdot \vec{v}_t)}
\]
GloVe: Global Vectors

- Let’s take the global co-occurrence statistics: $X_{i,j}$

\[ J = \sum_{i,j=1}^{V} f(X_{ij}) \left( w_i^T \hat{w}_j + b_i + \hat{b}_j - \log X_{ij} \right)^2 \]

- Training faster

- Scalable to very large corpora

(Pennington et al, 2014): GloVe: Global Vectors for Word Representation
GloVe: **Global Vectors**

Nearest words to *frog*:

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

(Pennington et al, 2014): GloVe: Global Vectors for Word Representation
FastText: Sub-Word Embeddings

• Similar as Skip-gram, but break words into n-grams with n = 3 to 6

  where:

  3-grams: <wh, whe, her, ere, re>

  4-grams: <whe, wher, here, ere>

  5-grams: <wher, where, here>

  6-grams: <where, where>

• Replace \( \mathbf{u}_i \cdot \mathbf{v}_j \) by \( \sum_{g \in n\text{-grams}(w_i)} \mathbf{u}_g \cdot \mathbf{v}_j \)

• More to come! Contextualized word embeddings

(Bojanowski et al, 2017): Enriching Word Vectors with Subword Information
Trained word embeddings available

- word2vec: https://code.google.com/archive/p/word2vec/
- GloVe: https://nlp.stanford.edu/projects/glove/
- FastText: https://fasttext.cc/

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 19M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

Differ in algorithms, text corpora, dimensions, cased/uncased...
Evaluating Word Embeddings
Extrinsic vs intrinsic evaluation

Extrinsic evaluation

- Let’s plug these word embeddings into a real NLP system and see whether this improves performance.
- Could take a long time but still the most important evaluation metric.

Intrinsic evaluation

- Evaluate on a specific/intermediate subtask.
- Fast to compute.
- Not clear if it really helps the downstream task.
Intrinsic evaluation

Word similarity
Example dataset: wordsim-353
353 pairs of words with human judgement
http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

<table>
<thead>
<tr>
<th>Word 1</th>
<th>Word 2</th>
<th>Human (mean)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tiger</td>
<td>cat</td>
<td>7.35</td>
</tr>
<tr>
<td>tiger</td>
<td>tiger</td>
<td>10</td>
</tr>
<tr>
<td>book</td>
<td>paper</td>
<td>7.46</td>
</tr>
<tr>
<td>computer</td>
<td>internet</td>
<td>7.58</td>
</tr>
<tr>
<td>plane</td>
<td>car</td>
<td>5.77</td>
</tr>
<tr>
<td>professor</td>
<td>doctor</td>
<td>6.62</td>
</tr>
<tr>
<td>stock</td>
<td>phone</td>
<td>1.62</td>
</tr>
<tr>
<td>stock</td>
<td>CD</td>
<td>1.31</td>
</tr>
<tr>
<td>stock</td>
<td>jaguar</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Cosine similarity:

\[
\cos(u_i, u_j) = \frac{u_i \cdot u_j}{||u_i||_2 \times ||u_j||_2}.
\]

Metric: Spearman rank correlation
## Intrinsic evaluation

### Word Similarity

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>WS353</th>
<th>MC</th>
<th>RG</th>
<th>SCWS</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVD</td>
<td>6B</td>
<td>35.3</td>
<td>35.1</td>
<td>42.5</td>
<td>38.3</td>
<td>25.6</td>
</tr>
<tr>
<td>SVD-S</td>
<td>6B</td>
<td>56.5</td>
<td>71.5</td>
<td>71.0</td>
<td>53.6</td>
<td>34.7</td>
</tr>
<tr>
<td>SVD-L</td>
<td>6B</td>
<td>65.7</td>
<td>72.7</td>
<td>75.1</td>
<td>56.5</td>
<td>37.0</td>
</tr>
<tr>
<td>CBOW$^+$</td>
<td>6B</td>
<td>57.2</td>
<td>65.6</td>
<td>68.2</td>
<td>57.0</td>
<td>32.5</td>
</tr>
<tr>
<td>SG$^+$</td>
<td>6B</td>
<td>62.8</td>
<td>65.2</td>
<td>69.7</td>
<td>58.1</td>
<td>37.2</td>
</tr>
<tr>
<td>GloVe</td>
<td>6B</td>
<td>65.8</td>
<td>72.7</td>
<td>77.8</td>
<td>53.9</td>
<td>38.1</td>
</tr>
<tr>
<td>SVD-L</td>
<td>42B</td>
<td>74.0</td>
<td>76.4</td>
<td>74.1</td>
<td>58.3</td>
<td>39.9</td>
</tr>
<tr>
<td>GloVe</td>
<td>42B</td>
<td><strong>75.9</strong></td>
<td><strong>83.6</strong></td>
<td><strong>82.9</strong></td>
<td><strong>59.6</strong></td>
<td><strong>47.8</strong></td>
</tr>
<tr>
<td>CBOW*</td>
<td>100B</td>
<td>68.4</td>
<td>79.6</td>
<td>75.4</td>
<td>59.4</td>
<td>45.5</td>
</tr>
</tbody>
</table>
Intrinsic evaluation

Word analogy
man: woman ≈ king: ?

\[
\text{arg max}_i (\cos(u_i, u_b - u_a + u_c))
\]

semantic  syntactic


More examples at
http://download.tensorflow.org/data/questions-words.txt