Word Embeddings

Wei Xu

(many slides from Greg Durrett)
This Lecture

- Word representations
- word2vec/GloVe
- Evaluating word embeddings
Word Representations
Word Representations

- Neural networks work very well at continuous data, but words are discrete.
- Continuous model <-> expects continuous semantics from input.
- "You shall know a word by the company it keeps" Firth (1957)

A bottle of *tesgüino* is on the table.
Everybody likes *tesgüino*.
*Tesgüino* makes you drunk.
We make *tesgüino* out of corn.

slide credit: Dan Klein, Dan Jurafsky
Discrete Word Representations

- Brown clusters: hierarchical agglomerative *hard* clustering (each word has one cluster, not some posterior distribution like in mixture models)

- Maximize
  \[
  P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)
  \]

- Useful features for tasks like NER, not suitable for NNs

Brown et al. (1992)
Brown clusters: hierarchical agglomerative *hard* clustering

- We give a very brief sketch of the algorithm here:

- \( k \): a hyper-parameter, sort words by frequency
- Take the top \( k \) most frequent words, put each of them in its own cluster \( c_1, c_2, c_3, \ldots c_k \)
- For \( i = (k+1) \ldots |V| \)
  - Create a new cluster \( c_{k+1} \) (we have \( k + 1 \) clusters)
  - Choose two clusters from \( k + 1 \) clusters based on quality(C) and merge (back to \( k \) clusters)

\[
\text{Quality}(C) = \sum_{i=1}^{n} \log p(w_i | C(w_i)) q(C(w_i) | C(w_{i-1})) = \sum_{c=1}^{k} \sum_{c'=1}^{k} p(c,c') \log \frac{p(c,c')} {p(c)p(c')} + G
\]

- Carry out \( k - 1 \) final merges (full hierarchy)
- Running time \( O\left( |V| k^2 + n \right) \), \( n=\#\text{words in corpus} \)

Learn more: Percy Liang's phd thesis - Semi-Supervised Learning for Natural Language
Discrete Word Representations

- Brown clusters: hierarchical agglomerative *hard* clustering
- Example Clusters from Miller et al. 2004

<table>
<thead>
<tr>
<th>Word</th>
<th>Bit String</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailman</td>
<td>10000011010111</td>
</tr>
<tr>
<td>salesman</td>
<td>10000011010000</td>
</tr>
<tr>
<td>bookkeeper</td>
<td>100000110100010</td>
</tr>
<tr>
<td>troubleshooter</td>
<td>100000110100010</td>
</tr>
<tr>
<td>bouncer</td>
<td>100000110100011</td>
</tr>
<tr>
<td>technician</td>
<td>100000110100100</td>
</tr>
<tr>
<td>janitor</td>
<td>100000110100101</td>
</tr>
<tr>
<td>saleswoman</td>
<td>100000110100110</td>
</tr>
<tr>
<td>Nike</td>
<td>101101110010010011100</td>
</tr>
<tr>
<td>Maytag</td>
<td>101101110010010011101</td>
</tr>
<tr>
<td>Generali</td>
<td>101101110010010011101</td>
</tr>
<tr>
<td>Gap</td>
<td>101101110010010011110</td>
</tr>
<tr>
<td>Harley-Davidson</td>
<td>101101110010010011111</td>
</tr>
<tr>
<td>Enfield</td>
<td>101101110010010011111</td>
</tr>
<tr>
<td>genus</td>
<td>101101110010010011111</td>
</tr>
<tr>
<td>Microsoft</td>
<td>101101110010010011000</td>
</tr>
<tr>
<td>Ventritex</td>
<td>101101110010010011001</td>
</tr>
<tr>
<td>Tractebel</td>
<td>101101110010010011010</td>
</tr>
<tr>
<td>Synopsys</td>
<td>101101110010010011011</td>
</tr>
<tr>
<td>WordPerfect</td>
<td>101101110010010011000</td>
</tr>
<tr>
<td>John</td>
<td>10110100000000000000</td>
</tr>
<tr>
<td>Consuelo</td>
<td>10110100000000000001</td>
</tr>
<tr>
<td>Jeffrey</td>
<td>10110100000000000010</td>
</tr>
<tr>
<td>Kenneth</td>
<td>10110100000000000100</td>
</tr>
<tr>
<td>Phillip</td>
<td>10110100000000000110</td>
</tr>
<tr>
<td>WILLIAM</td>
<td>10110100000000001101</td>
</tr>
<tr>
<td>Timothy</td>
<td>10110100000000001110</td>
</tr>
</tbody>
</table>

**word cluster features** (bit string prefix)
Word Embeddings

- Part-of-speech tagging with FFNNs
  
  Fed raises interest rates in order to ...

- Word embeddings for each word form input

- What properties should these vectors have?

Botha et al. (2017)
Deep Averaging Networks: feedforward neural network on average of word embeddings from input

\[ h_1 = f(W_1 \cdot av + b_1) \]

\[ h_2 = f(W_2 \cdot h_1 + b_2) \]

\[ av = \sum_{i=1}^{4} \frac{c_i}{4} \]

 Predicate: Predator, is, a, masterpiece

Iyyer et al. (2015)
Word Embeddings

- Want a vector space where similar words have similar embeddings

\[
\text{the movie was great} \\
\sim \\
\text{the movie was good}
\]

- Goal: come up with a way to produce these embeddings

- For each word, want “medium” dimensional vector (50-300 dims) representing it.
Word Representations

- Count-based: tf*idf, PPMI, ...
- Class-based: Brown Clusters, ...
- Distributed prediction-based embeddings: Word2vec, GloVe, FastText, ...
- Distributed contextual embeddings: ELMo, BERT, GPT, ...
- + many more variants: multi-sense embeddings, syntactic embeddings, ...
word2vec/GloVe
Neural Probabilistic Language Model

Figure 1: Neural architecture: $f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))$ where $g$ is the neural network and $C(i)$ is the $i$-th word feature vector.
word2vec: Continuous Bag-of-Words

- Predict word from context

**Parameters:**
- $d \times |V|$ (one $d$-length context vector per vocabulary word),
- $|V| \times d$ output parameters ($W$)

$P(w|w_{-1}, w_{+1}) = \text{softmax}(W(c(w_{-1}) + c(w_{+1})))$

Mikolov et al. (2013)
word2vec: Skip-Gram

- Predict one word of context from word
- d-dimensional word embeddings

\[ P(w' \mid w) = \text{softmax}(W e(w)) \]

- Another training example: \textit{bit} -> \textit{the}
- Parameters: \( d \times |V| \) vectors, \( |V| \times d \) output parameters (W) (also usable as vectors!)

Mikolov et al. (2013)
Hierarchical Softmax

\[ P(w|w_{-1}, w_{+1}) = \text{softmax} \left( W(c(w_{-1}) + c(w_{+1})) \right) \quad P(w'|w) = \text{softmax}(W e(w)) \]

- Matmul + softmax over \(|V|\) is very slow to compute for CBOW and SG

- Huffman encode vocabulary, use binary classifiers to decide which branch to take

- \(\log(|V|)\) binary decisions

- Standard softmax: \(O(|V|)\) dot products of size \(d\) - per training instance per context word

- Hierarchical softmax: \(O(\log(|V|))\) dot products of size \(d\), \(|V| \times d\) parameters

Mikolov et al. (2013)

http://building-babylon.net/2017/08/01/hierarchical-softmax/
Skip-Gram with Negative Sampling

- Take (word, context) pairs and classify them as “real” or not. Create random negative examples by sampling from unigram distribution:
  
  \[
  (\text{bit, the}) \Rightarrow +1 \\
  (\text{bit, cat}) \Rightarrow -1 \\
  (\text{bit, a}) \Rightarrow -1 \\
  (\text{bit, fish}) \Rightarrow -1
  \]

- \( d \times |V| \) vectors, \( d \times |V| \) context vectors (same # of params as before)

- Objective = \[
\log P(y = 1|w, c) - \sum_{i=1}^{k} \log P(y = 0|w_i, c)
\]

Mikolov et al. (2013)
Skip-gram model looks at word-word co-occurrences and produces two types of vectors. Two words are “similar” in meaning if their context vectors are similar. Similarity == relatedness.
Skip-gram model looks at word-word co-occurrences and produces two types of vectors.

Looks almost like a matrix factorization…can we interpret it this way?

Levy et al. (2014)
Skip-Gram as Matrix Factorization

\[ M_{ij} = \text{PMI}(w_i, c_j) - \log k \]

\[ \text{PMI}(w_i, c_j) = \frac{P(w_i, c_j)}{P(w_i)P(c_j)} = \frac{\text{count}(w_i, c_j)}{D} \frac{\text{count}(w_i)}{D} \frac{\text{count}(c_j)}{D} \]

Skip-gram objective exactly corresponds to factoring this matrix:

- If we sample negative examples from the uniform distribution over words
- ...and it’s a weighted factorization problem (weighted by word freq)

Levy et al. (2014)
Typical problems in word-word co-occurrences:
- Raw frequency is not the best measure of association between words.
- Frequent words are often more important than rare words that only appear once or twice;
- But, frequent words (e.g., the) that appear in all documents are also not very useful signal.

Solutions — weighing terms in word-word/word-doc co-occurrence matrix
- Tf*idf
- PPMI (Positive PMI)
Co-occurrence Matrix

- **Tf*idf**
- **Tf**: term frequency
  
  \[ tf = \log_{10}(\text{count}(t, d) + 1) \]

- **Idf**: inverse document frequency
  
  \[ idf_i = \log_{10}(\frac{N}{df_i}) \]

  - Total number of docs in collection
  - Number of docs that have word i

<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>17</td>
</tr>
<tr>
<td>soldier</td>
<td>2</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>clown</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
GloVe (Global Vectors)

- Also operates on counts matrix, weighted regression on the log co-occurrence matrix

\[ \text{Loss} = \sum_{i,j} f(\text{count}(w_i, c_j)) \left( w_i^\top c_j + a_i + b_j - \log \text{count}(w_i, c_j) \right)^2 \]

- Constant in the dataset size (just need counts), quadratic in voc size

- By far the most common non-contextual word vectors used today (10000+ citations)

Pennington et al. (2014)
Using Word Embeddings

- Approach 1: learn embeddings as parameters from your data
  - Often works pretty well
- Approach 2: initialize using GloVe/word2vec/ELMo, keep fixed
  - Faster because no need to update these parameters
- Approach 3: initialize using GloVe, fine-tune
  - Works best for some tasks, not used for ELMo, often used for BERT
Evaluation
Visualization

- Male-Female
- Verb tense
- Country-Capital

Source: tensorflow.org
Figure 1: A 2-dimensional projection of the latent semantic space captured by our algorithm. Notice the semantic trajectory of the word gay transitioning meaning in the space.

Kulkarni et al. (2015)
Evaluating Word Embeddings

- What properties of language should word embeddings capture?
- Similarity: similar words are close to each other
- Analogy:
  
  good is to best as smart is to ???
  
  Paris is to France as Tokyo is to ???

![Diagram showing word embeddings in a 2D space with examples of similarity and analogy.]
Word Similarity

- Cosine Similarity:

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

Levy et al. (2015)
Word Similarity

<table>
<thead>
<tr>
<th>Method</th>
<th>WordSim Similarity</th>
<th>WordSim Relatedness</th>
<th>Bruni et al. MEN</th>
<th>Radinsky et al. M. Turk</th>
<th>Luong et al. Rare Words</th>
<th>Hill et al. SimLex</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPMI</td>
<td>.755</td>
<td>.697</td>
<td>.745</td>
<td>.686</td>
<td>.462</td>
<td>.393</td>
</tr>
<tr>
<td>SVD</td>
<td><strong>.793</strong></td>
<td>.691</td>
<td><strong>.778</strong></td>
<td>.666</td>
<td><strong>.514</strong></td>
<td>.432</td>
</tr>
<tr>
<td>SGNS</td>
<td><strong>.793</strong></td>
<td>.685</td>
<td><strong>.774</strong></td>
<td><strong>.693</strong></td>
<td>.470</td>
<td><strong>.438</strong></td>
</tr>
<tr>
<td>GloVe</td>
<td>.725</td>
<td>.604</td>
<td>.729</td>
<td>.632</td>
<td>.403</td>
<td>.398</td>
</tr>
</tbody>
</table>

- SVD = singular value decomposition on PMI matrix
- GloVe does not appear to be the best when experiments are carefully controlled, but it depends on hyperparameters + these distinctions don’t matter in practice

Levy et al. (2015)
Hypernymy Detection

- Hypernyms: detective *is a* person, dog *is a* animal
- Do word vectors encode these relationships?

<table>
<thead>
<tr>
<th>Dataset</th>
<th>TM14</th>
<th>Kotlerman 2010</th>
<th>HypeNet</th>
<th>WordNet</th>
<th>Avg (10 datasets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>52.0</td>
<td>30.8</td>
<td>24.5</td>
<td>55.2</td>
<td>23.2</td>
</tr>
<tr>
<td>Word2Vec + C</td>
<td>52.1</td>
<td><strong>39.5</strong></td>
<td>20.7</td>
<td><strong>63.0</strong></td>
<td>25.3</td>
</tr>
<tr>
<td>GE + C</td>
<td>53.9</td>
<td>36.0</td>
<td>21.6</td>
<td>58.2</td>
<td>26.1</td>
</tr>
<tr>
<td>GE + KL</td>
<td>52.0</td>
<td>39.4</td>
<td>23.7</td>
<td>54.4</td>
<td>25.9</td>
</tr>
<tr>
<td>DIVE + C·ΔS</td>
<td><strong>57.2</strong></td>
<td>36.6</td>
<td><strong>32.0</strong></td>
<td>60.9</td>
<td><strong>32.7</strong></td>
</tr>
</tbody>
</table>

- word2vec (SGNS) works barely better than random guessing here

Table 1: Comparison with other unsupervised embedding methods. The scores are AP@all (%) for the first 10 datasets and Spearman ρ (%) for HyperLex. Avg (10 datasets) shows the micro-average AP of all datasets except HyperLex. Word2Vec+C scores word pairs using cosine similarity on skip-grams. GE+C and GE+KL compute cosine similarity and negative KL divergence on Gaussian embedding, respectively.

Chang et al. (2017)
Analyses

\[(king - man) + woman = queen\]

\[king + (woman - man) = queen\]

- Why would this be?
- woman - man captures the difference in the contexts that these occur in
- Dominant change: more “he” with man and “she” with woman — similar to difference between king and queen
These methods can perform well on analogies on two different datasets using two different methods.

Maximizing for $b$: Add $= \cos(b, a_2 - a_1 + b_1)$ \hspace{1cm} Mul $= \frac{\cos(b_2, a_2) \cos(b_2, b_1)}{\cos(b_2, a_1) + \epsilon}$

Levy et al. (2015)
Using Semantic Knowledge

- Structure derived from a resource like WordNet
- Doesn’t help most problems

Faruqui et al. (2015)
Takeaways

- **Word vectors:** learning word -> context mappings has given way to matrix factorization approaches (constant in dataset size)
- Lots of pretrained embeddings work well in practice, they capture some desirable properties
- Even better: context-sensitive word embeddings (ELMo/BERT/etc.) — will talk later in the semester
- Next time: sequence modeling, HMM, ...