A survey on vector representation for words and its applications

Yanglei Song
Motivation

In many text applications, we usually first create a vocabulary and then treat word as an index in the vocabulary. Or equivalently, “one-hot” representation:

```
motel [0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0] = 0
```

This representation can’t capture the similarity between words. As a result, it can’t generalize well:

\[
P(\text{“The cat is walking in the bedroom”}) \text{ vs. } P(\text{“A dog was running in a room”})
\]
Motivation (ctd)

• This “data sparsity” problem also happens very often in supervised learning: classifier doesn’t perform as well on unseen examples
  
  – Getting label: very expensive.
  – Utilizing unlabeled data (Wikipedia): use as feature in supervised learning

• Given a large corpus, learn a vector representation such that syntactically or semantically related words should have similar representation
Related work – word clustering

• Brown clustering (Brown et al, 1992)

Consider 2-gram:

\[ P(w_i | w_{i-1}) \leftarrow V^2 \text{ parameters} \]

If we have a partition of word, denote \( c(w) \) as the class of the word,

\[
P(w_i | w_{i-1}) = P(w_i | C(w_i))P(C(w_i) | C(w_{i-1}))
\]

\[ K^2 + V \text{ parameters} \]
word clustering (ctd)

- Find a partition of words such that the likelihood of corpus is maximized.
- NP hard. **Heuristics**: hierarchical clustering
  1. Slow. Can only handle vocabulary size of ~10K
  2. We are still using “one-hot” representation. Thus not enough expressive power
Related work – dense embedding

- Latent semantic analysis

1. SVD is used. $V \sim 10K$, $D \sim 1M$ -> this matrix has $\sim 10b$ entries
2. Only capture the semantic similarity
Neural network

\[ h_{w,b}(x) = f(w^T x + b) \]

\[ f(z) = \frac{1}{1 + e^{-z}} \]

In matrix notation

\[ z = Wx + b \]

\[ a = f(z) \]
Recursive neural network

Each computational units share the same weight. As a result, all nodes will be required to have same dimension.
Neural network word embedding

• Neural probabilistic language models (Bengio et al, JMLR06)
  – N-gram: compute the probability of current word given previous (n-1) words
    \[ P(w_i \mid w_{i-1}, \ldots, w_{i-n+1}) \]
  – With the goal of giving the corpus maximal likelihood
  – Use neural network to model the above quantity
Neural probabilistic language models

Input: preview (n-1) words
Output: the prob. of current word

First, each word is mapped to a k-dimensional vector
Second, concatenation of the word embedding serve as the input to the hidden layer.

Find the embedding and parameter in the neural network which maximize the likelihood
Extensions

• With the same philosophy, there are several extensions with the goal of speeding up training and testing. (Mnih&Hinton NIPS07, ICML08; Mikolov NIPS13; …)

• (Mikolov NIPS13) claim to be able to train on 100b words on single-machine within a day. And the source code is available here.
Neural network word embedding (ctd)

- C & W embedding (ICML08): don’t care language model; only focus on meaningful vector representation.

Idea: A word and its context is a positive training sample; a random word in that same context gives a negative training sample:

\[
\text{score(} \text{cat chills on a mat}\text{)} > \text{score(} \text{cat chills Jeju a mat}\text{)}
\]
C & W embedding

\[ s = U^T f(Wx + b) \quad x \in \mathbb{R}^{20 \times 1}, \ W \in \mathbb{R}^{8 \times 20}, \ U \in \mathbb{R}^{8 \times 1} \]

\[ s = U^T a \]

\[ a = f(z) \]

\[ z = WX + b \]

\[ x = [x_{\text{cat}} \ x_{\text{chills}} \ x_{\text{on}} \ x_{\text{a}} \ x_{\text{mat}}] \]

\[ L \in \mathbb{R}^{n \times |V|} \]

\[ s = \text{score(cat chills on a mat)} \]

\[ s_c = \text{score(cat chills Jeju a mat)} \]

Minimize

\[ J = \max(0, 1 - s + s_c) \]
Visualization of vector representation

• One way to visualize the word embedding is to map them into 2-D plane while roughly preserving their distance.

• [Here is one results.](#)
Some experiment – word2vec

• Run on DBLP with minimal processing

• 0.35M abstract, total 70M words -> less than 10 mins (quite scalable)

• It’ll run a naïve algorithm to find phrases, then phrases are treated as single words.
**FP_growth**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>F-Measure (FP_growth)</th>
<th>F-Measure (prefixspan)</th>
<th>F-Measure (eclat)</th>
<th>F-Measure (apriori)</th>
<th>F-Measure (genmax)</th>
<th>F-Measure (gspan)</th>
<th>F-Measure (futi)</th>
<th>F-Measure (intertransaction_frequent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefixspan</td>
<td>0.6449</td>
<td>0.6449</td>
<td>0.6445</td>
<td>0.6445</td>
<td>0.6353</td>
<td>0.6347</td>
<td>0.6449</td>
<td>0.6329</td>
</tr>
<tr>
<td>eclat</td>
<td>0.6953</td>
<td>0.6953</td>
<td>0.6953</td>
<td>0.6953</td>
<td>0.6381</td>
<td>0.6381</td>
<td>0.6445</td>
<td>0.6313</td>
</tr>
<tr>
<td>apriori</td>
<td>0.6817</td>
<td>0.6817</td>
<td>0.6817</td>
<td>0.6817</td>
<td>0.6353</td>
<td>0.6353</td>
<td>0.6456</td>
<td>0.6299</td>
</tr>
<tr>
<td>genmax</td>
<td>0.6710</td>
<td>0.6710</td>
<td>0.6710</td>
<td>0.6710</td>
<td>0.6353</td>
<td>0.6353</td>
<td>0.6456</td>
<td>0.6299</td>
</tr>
<tr>
<td>gspan</td>
<td>0.6544</td>
<td>0.6544</td>
<td>0.6544</td>
<td>0.6544</td>
<td>0.6347</td>
<td>0.6347</td>
<td>0.6456</td>
<td>0.6214</td>
</tr>
</tbody>
</table>

**Database:**

<table>
<thead>
<tr>
<th>Databases</th>
<th>F-Measure (databases)</th>
<th>F-Measure (relational_dbms)</th>
<th>F-Measure (sql)</th>
<th>F-Measure (olap_rolap)</th>
<th>F-Measure (oodbms)</th>
<th>F-Measure (archis)</th>
<th>F-Measure (codasyl)</th>
</tr>
</thead>
<tbody>
<tr>
<td>databases</td>
<td>0.727</td>
<td>0.727</td>
<td>0.538</td>
<td>0.538</td>
<td>0.466</td>
<td>0.466</td>
<td>0.466</td>
</tr>
<tr>
<td>relational</td>
<td>0.587</td>
<td>0.587</td>
<td>0.511</td>
<td>0.511</td>
<td>0.463</td>
<td>0.463</td>
<td>0.463</td>
</tr>
<tr>
<td>dbms</td>
<td>0.574</td>
<td>0.574</td>
<td>0.500</td>
<td>0.500</td>
<td>0.463</td>
<td>0.463</td>
<td>0.463</td>
</tr>
<tr>
<td>relational_databases</td>
<td>0.560</td>
<td>0.560</td>
<td>0.479</td>
<td>0.479</td>
<td>0.458</td>
<td>0.458</td>
<td>0.458</td>
</tr>
<tr>
<td>rdbms</td>
<td>0.548</td>
<td>0.548</td>
<td>0.474</td>
<td>0.474</td>
<td>0.449</td>
<td>0.449</td>
<td>0.449</td>
</tr>
</tbody>
</table>
Applications 1 – NLP task

- (C& W ICML08): A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning.

- Part-Of-Speech Tagging/ Chunking/ Named Entity Recognition/ Semantic Role Labeling/ C&W embedding
  1. Different task share same word embedding
  2. C&W embedding: “virtually” introduce large amount of labelled data
NLP task (ctd)

- **Software Senna**: based on the above architecture; general NLP tool.

- Achieve the-state-of-art: rather remarkably, it doesn’t use any hand-crafted feature. It indicates it’ll generalize well in different domains.

- Later, quite a few NLP tasks is reported to have better performance by incorporating word embedding as feature. (Turian et al, ACL10)
Application 2 – sentiment analysis

- **Live demo** (Socher et al, EMNLP13)

- Architecture

```
... not very good ...

a b c
```

\[ p_2 = g(a, p_1) \]

\[ p_1 = g(b, c) \]
Application 3 – knowledge base completion

- Socher et al, NIPS13
Some thinking

• Instead of treating word as index in vocabulary, we now have vector representation which captures the similarity between words. This may change our view on text.

• Publicly available embedding can be used as feature in your own tasks. Or you can train the word embedding on your own dataset.