Vector Semantics

Representing word on vector using SVD and Skipgram

Jeong Min Lee (jlee@cs.pitt.edu)
3/31/2017
Outline

- Word as vector
- Singular Value Decomposition
- Skipgram
- CBOW (brief)
Word as vector

- Words can be represented as vectors
- The simplest form would be **one-hot** vector
  - Represent every word as an $\mathbb{R}^{d \times 1}$ with all 0s and one 1
  - The 1 is at the index of that word in the corpus
  - $d$: number of all words in corpus
- Known as **bag-of-word** representation
Example of one-hot vector

Let's represent words in a sentence:

```
<table>
<thead>
<tr>
<th>The fox jumped over the lazy dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>• the : [1,0,0,0,0,0,0]</td>
</tr>
<tr>
<td>• fox : [0,1,0,0,0,0,0]</td>
</tr>
<tr>
<td>• jumped : [0,0,1,0,0,0,0]</td>
</tr>
<tr>
<td>• over : [0,0,0,1,0,0,0]</td>
</tr>
<tr>
<td>• the : [0,0,0,0,1,0,0]</td>
</tr>
<tr>
<td>• lazy : [0,0,0,0,0,1,0]</td>
</tr>
<tr>
<td>• dog : [0,0,0,0,0,0,1]</td>
</tr>
</tbody>
</table>
```
- Dimension of the vectors: 7
  - Assumed that we are only dealing with the 7 words
- There are an estimated 13 million tokens in English
  - It will result in very sparse, high-dimensional vectors
- How we represent more efficiently?
• Represent word as dense real-valued vector
  ◦ Better than sparse high-dimensional one-hot vector
  ◦ Find lower-dimensional embedding of the representation
    ▪ Singular Value Decomposition
    ▪ Skipgram and CBOW
Singular Value Decomposition
SVD Based Models

- Key idea is to find embeddings of word (i.e. lower-dimensional representation)
- Two step approach:
  i. Create co-occurrence matrix $X$
  ii. Perform Singular Value Decomposition on $X$ use rows of the left-singular matrix as the embeddings
Create co-occurrence matrix

Two different approaches exist

- Word-Document matrix
- Window based word co-occurrence matrix
Word-Document Matrix

Word-document matrix $X$ is of

- Rows are each word
- Columns are each document
- Update $X_{i,j}$ by adding 1 when observing word $i$ at document $j$, while looping over all documents
Window based word co-occurrence matrix

Word-word co-occurrence matrix $X \in \mathbb{R}^{d \times d}$ consists of each row and column as words in corpus

- $X_{i,j}$ is the number of co-occurrence of word $i$ and $j$ within a particular size window
- Go over all sentences in corpus
- Also called word-context matrix (in literatures of skipgram)
Source Text

The quick brown fox jumps over the lazy dog. ➞

Training Samples

The quick brown fox jumps over the lazy dog. ➞

- (the, quick)
- (the, brown)

The quick brown fox jumps over the lazy dog. ➞

- (quick, the)
- (quick, brown)
- (quick, fox)

The quick brown fox jumps over the lazy dog. ➞

- (brown, the)
- (brown, quick)
- (brown, fox)
- (brown, jumps)

The quick brown fox jumps over the lazy dog. ➞

- (fox, quick)
- (fox, brown)
- (fox, jumps)
- (fox, over)
Singular Value Decomposition

\[ X = U S V^T \]

- Assume \( X \in \mathbb{R}^{N \times d} \), SVD decomposes \( X \) into followings:
  - \( U \), a column-orthogonal \( N \times r \) matrix
  - \( S \), a diagonal matrix with singular values sorted in descending order
  - \( V \), a row-orthogonal \( r \times d \) matrix
  - \( r \) is the rank of the matrix \( X \) (# of linearly independent column/rows)
Implication of SVD

(Example of Word-Document Matrix)

- On a document-word matrix, we have two types of concepts (CS and Medical)
- $\mathbf{U}$ reveals document-concept similarity
- $\mathbf{V}$ reveals term-concept similarity
- We can use columns of $\mathbf{V}$ to represent each word
SVD on co-occurrence matrix

- Perform SVD on co-occurrence matrix
  - Either word-word or word-doc matrix can be used
- Use the rows of $U$ as the word embeddings
- Reduce dimensionality by using only first $k$ columns of $U$
• We have lower-dimensional embedding of word vector
• But performing SVD costs quadratic $O(mn^2)$, when $n < m$
  ○ Challenging for large documents and words
• Hard to incorporate new word to embedding: SVD needs to be conducted again.
Skipgram
Skipgram

- Represent words as dense real-valued vectors
- Learning representation
  \[=\] predicting surrounding words of every word
Objective function

- Maximize log probability of context words given the current target word:

\[
J(\theta) = \frac{1}{d} \sum_{t=1}^{d} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t)
\]
• $p(w_{t+j} | w_t)$ is the Softmax function of inner product of target and its context word:

$$p(w_{t+j} | w_t) = \frac{\exp(u_{t+j}^T \cdot v_t)}{\sum_{w=1}^d \exp(u_w^T \cdot v_t)}$$

• $d$ is the cardinality of all words in corpus and $\theta$ represents parameters of the model

• $u$ is the input→hidden layer lower-dimensional embedding

• $v$ is the hidden→output layer lower-dimensional embedding
Parameters

- $\theta = \{U, V\}$
- $V \in \mathbb{R}^{d \times r}$ is input word matrix (Input layer $\rightarrow$ Hidden layer)
  - $v_i$ is $i$-th row of $V$
- $U \in \mathbb{R}^{r \times d}$ is output word matrix (Hidden layer $\rightarrow$ Output layer)
  - $u_i$ is $i$-th column of $U$
- Every word $i$ has two vector representations $v_i$ and $u_i$
  - Usually use the rows of $V$ for word representation
Overall process

For each word $t$:

1. Get one-hot input vector $x_t$
2. Get embedding of input word vector $v_t = \mathcal{V}x_t$
3. For each context words of $t$, get the output vectors $u_{t-m},...,u_{t-1},u_{t+1},...,u_{t+m}$ using $u_i = \mathcal{U}v_i$
4. Turn each of the $u_i \cdot v_t$s into Softmax probabilities $y_i = \text{softmax}(u_{\text{context word } i} \cdot v_{\text{target word } t})$ for $i \in t - m, ..., t + m$
5. When the model is properly trained, we can expect that the Softmax probability vectors would match the true probabilities, the actual one-hot output vectors
CBOW (Continuous Bag-of-Words)

- Similar idea to Skipgram
- Difference: Given context words, predict the target word
- For each of context words, get the real-value vector representation with $\mathcal{V}$ and average them to one vector
- Then multiply the vector with $\mathcal{U}$ to predict the one-hot representation of the target vector
Application on Clinical Machine Learning

Med2Vec (Choi, et al. 2016)

- Learn lower dimensional representation of medical concepts (medication, diagnosis, procedure, etc)
- Architecture is on multi-layer perceptron to capture code-level and visit-level cooccurrence information
- Make it predict next visit's concepts and some dimensions of the embedding matrix showed clinically meaningful results

..and some more!
Credit

Note that this slide is based on these materials:

- Christopher Moody, A word is worth a thousand vectors
- Edward Choi, Multi-layer Representation Learning for Medical Concepts
- Eric Strobl, Slide on SVD at CS3750
- Radim Rehurek, Making Sense of Word2Vec
- Richard Socher, Simple Word Vector representations at CS224d
- Tomas Mikolov, Distributed Representations of Words and Phrases and their Compositionality
- Xin Rong, Word2vec Parameter Learning Explained
Questions?
Optimizing Computational Efficiency

- With the objective function, the model is trained on gradient descent
- The computational challenge of the training is the term with $d$

$$J(\theta) = \frac{1}{d} \sum_{t=1}^{d} \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(u_{t+j}^T \cdot v_t)}{\sum_{w=1}^{d} \exp(u_w^T \cdot v_t)}$$

- Any gradient update involves $O(d)$
Negative Sampling

Key idea: Instead of looping over entire vocabulary $V$, just sample negative examples

- Decompose the objective function

$$J(\theta) = \frac{1}{d} \sum_{t=1}^{d} \sum_{-m \leq j \leq m, j \neq 0} u_{t+j}^T \cdot v_t - \log \sum_{w=1}^{d} \exp(u_w^T \cdot v_t)$$
Make the second term samples in its process

\[
J(\theta) = \frac{1}{V} \sum_{t=1}^{V} \sum_{-m \leq j \leq m, j \neq 0} u_{t+j}^{T} \cdot v_{t} - \log \sum_{k=1}^{K} \exp(\tilde{u}_{k}^{T} \cdot v_{t})
\]

where \( \tilde{u}_{k} | k = 1 \ldots K \) are sampled from \( P_{n}(w) \)
Sampling Distribution

$P_n(w)$ is Unigram Model raised to the power of $3/4$
(introduced in the original Skipgram paper of Mikolov 2014)

- Seems like the power of $3/4$ has done smoothing for low-frequency words
  - is: $0.9^{3/4} = 0.92$
  - Constitution: $0.09^{3/4} = 0.16$
  - bombastic: $0.01^{3/4} = 0.032$

- Infrequent word 'Bombastic' is 3 times more likely to be sampled with the power of $3/4$

- Frequent word 'is' only went up marginally