Distributional semantics (II)

CS 690N, Spring 2018
Advanced Natural Language Processing
http://people.cs.umass.edu/~brenocon/anlp2018/

Brendan O’Connor
College of Information and Computer Sciences
University of Massachusetts Amherst
Matrix factorization

\[ V \approx E \] (embeddings)

\[ B \approx \text{latent dims} \]

Reconstruct the co-occurrence matrix

\[ V_{i,c} \approx \sum_k E_{i,k} B_{k,c} \]

Singualar Value Decomposition learns E,B
(or other matrix factorization techniques)

Preserve pairwise distances between words i,j

\[ V_i^T V_j \approx E_i^T E_j \]

Eigen Decomposition learns E
Local models

- **CBOW:** faster
- **Skip-grams:** work better (because more like context matrix factorization?)

(a) Continuous bag-of-words (CBOW)

(b) Skipgram

Figure 13.3: The CBOW and skipgram variants of \textsc{Word2Vec}. The parameter $U$ is the matrix of word embeddings, and each $v_m$ is the context embedding for word $w_m$. The inner product $u_i \cdot v_j$ represents the compatibility between word $i$ and context $j$. By incorporating this inner product into an approximation to the log-likelihood of a corpus, it is possible to estimate both parameters by backpropagation.

\cite{Mikolov2013} includes two such approximations: continuous bag-of-words (CBOW) and skipgrams.

13.5.1 Continuous bag-of-words (CBOW)

In recurrent neural network language models, each word $w_m$ is conditioned on a recurrently-updated state vector, which is based on word representations going all the way back to the beginning of the text. The continuous bag-of-words (CBOW) model is a simplification: the local context is computed as an average of embeddings for words in the immediate neighborhood $m, m+1, \ldots, m+h$.

Thus, CBOW is a bag-of-words model, because the order of the context words does not matter; it is continuous, because rather than conditioning on the words themselves, we condition on a continuous vector constructed from the word embeddings. The parameter $h$ determines the neighborhood size, which \cite{Mikolov2013} set to $h=4$.

(c) Jacob Eisenstein 2018. Work in progress.

Thursday, February 22, 18
Skip-gram model

\[ u_\theta(w, c) = \exp \left( a_w^\top b_c \right) \]

\[ J = \frac{1}{M} \sum_m \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{m+j} | w_m) \]

\[ p(w_{m+j} | w_m) = \frac{u_\theta(w_{m+j}, w_m)}{\sum_{w'} u_\theta(w', w_m)} \]

\[ = \frac{u_\theta(w_{m+j}, w_m)}{Z(w_m)} \]

- \textbf{[Mikolov et al. 2013]}
- In word2vec. Learning: SGD under a contrastive sampling approximation of the objective
- Levy and Goldberg: mathematically similar to factorizing a PMI(w,c) matrix; advantage is streaming, etc. (though see Arora et al.’s followups...)
- Practically: very fast open-source implementation
- Variations: enrich contexts
Skip-gram model

\[ \log p(w) \approx \sum_{m=1}^{M} \sum_{n=1}^{h_m} \log p(w_{m-n} | w_m) + \log p(w_{m+n} | w_m) \]

\[ = \sum_{m=1}^{M} \sum_{n=1}^{h_m} \log \frac{\exp(u_{w_{m-n}} \cdot v_{w_m})}{\sum_{j=1}^{V} \exp(u_i \cdot v_{w_m})} + \log \frac{\exp(u_{w_{m+n}} \cdot v_{w_m})}{\sum_{j=1}^{V} \exp(u_i \cdot v_{w_m})} \]

\[ = \sum_{m=1}^{M} \sum_{n=1}^{h_m} u_{w_{m-n}} \cdot v_{w_m} + u_{w_{m+n}} \cdot v_{w_m} - 2 \log \sum_{j=1}^{V} \exp(u_i \cdot v_{w_m}) \]

- [Mikolov et al. 2013]
- In word2vec. Learning: SGD under a contrastive sampling approximation of the objective
- Levy and Goldberg: mathematically similar to factorizing a PMI(w,c) matrix; advantage is streaming, etc. (though see Arora et al.’s followups...)
- Practically: very fast open-source implementation
- Variations: enrich contexts
Dealing with large output vocab

- Hierarchical softmax
  - Helps to have a good hierarchy: for example, Brown clusters work well. Or less expensive alternatives

Figure 13.4: A fragment of a hierarchical softmax tree. The probability of each word is computed as a product of probabilities of local branching decisions in the tree.

\[
\text{Ahab} = \sigma(u_0 \cdot v_c) \\
\text{whale} = \sigma(-u_0 \cdot v_c) \times \sigma(u_2 \cdot v_c) \\
\text{blubber} = \sigma(-u_0 \cdot v_c) \times \sigma(-u_2 \cdot v_c)
\]
\[ \psi(i, j) = \log \sigma(u_i \cdot v_j) + \sum_{i' \in \mathcal{W}_{\text{neg}}} \log(1 - \sigma(u_{i'} \cdot v_j)) \]

- **Negative sampling**
  - Choose negative set somehow -- e.g. a (rescaled) unigram language model (2-5? 5-20? samples)
- **Levy and Goldberg** show equivalence to word-context matrix factorization, where matrix cells are:

\[ M_{ij} = \max(0, \text{PMI}(i, j) - \log k) \]
• “Distributional / Word Embedding” models
  • Typically, they learn embeddings to be good at word-context factorization, which seems to often give useful embeddings

• Pre-trained embeddings resources
  • GLOVE, word2vec, etc.
  • Make sure it’s trained on a corpus sufficiently similar to what you care about!

• How to use?
  • Fixed (or initializations) for word embedding model parameters
  • Similarity lookups
Extensions

- Alternative: Task-specific embeddings (always better...)
- Multilingual embeddings
- Better contexts: direction, syntax, morphology / characters...
- Phrases and meaning composition
  - $\text{vector(hardly awesome)} = g(\text{vector(hardly)}, \text{vector(awesome)})$