Glove

VL Embeddings

Uni Heidelberg

SS 2019

Acknowledgements

- Many of the slides taken from Richard Socher's excellent lecture on word embeddings (Deep Learning for Natural Language Processing (CS224d), Stanford University) http://cs224d.stanford.edu/
- See video lecture: https://www.youtube.com/watch?v=ASn7ExxLZws
- Slides: https://cs224d.stanford.edu/lectures/CS224d-Lecture3.pdf

Recap: SkipGram

Main ideas

- Go through each word in the whole corpus
- Predict surrounding words of each center word (in window of size m)

$$p(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w=1}^{v} exp(u_w^T v_c)} \tag{1}$$

- where o is the outside word id, c is the center word, u and v are center and outside vectors of o and c
- Every word has two vectors

Updating the word vectors with SGD

- In each window, only $2m+1$ words (for window size m)

$\rightarrow$ $\nabla_\theta J_t(\theta)$ is very sparse

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} = \in \mathbb{R}^{2dV} \tag{2}$$

Updating the word vectors with SGD

- Large updates $\rightarrow$ inefficient
- Solution: Stochastic Gradient Descent
  $\Rightarrow$ only update word vectors in context window

- Cost of computing $\nabla log\ p(w_o|w_i)$ is proportional to V
- Solution: Hierarachical softmax
  - Use binary tree to encode words in vocabulary (Huffman tree)

  $\Rightarrow$ instead of evaluating V words, only evaluate $log_2(V)$

Negative sampling

- Normalisation factor is too computationally expensive (iterate over full vocabulary each time!)

$$p(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w=1}^{V} exp(u_w^T v_c)} \tag{3}$$

- Solution: Negative sampling
  - Main idea: train binary logistic regressions for a true pair (center word and context word) versus a couple of noise pairs (center word paired with random word)

Word2vec summary

- Go through each word in the whole corpus and
  a) predict surrounding words of each word (skip-gram)
  b) predict center word, based on surrounding words (CBOW)
- This captures co-occurrence of words one at a time

Word2vec summary

- Go through each word in the whole corpus and
  a) predict surrounding words of each word (skip-gram)
  b) predict center word, based on surrounding words (CBOW)
- This captures co-occurrence of words one at a time

Why not capture co-occurrence counts directly?

Idea: Use co-occurrence counts between words directly

Create co-occurrence matrix X

- Two options:
  1. windows
  2. full documents
- Window:
  - similar to word2vec
  - use window around each word $\rightarrow$ captures both syntactic (POS) and semantic information
- Document:
  - word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to "Latent Semantic Analysis"

Example: Window-based co-occurrence matrix

- Window length 1 (more common: 5 - 10)
- Symmetric (not sensitive to position: left or right context)
- Toy corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	**I**	**like**	**enjoy**	**deep**	**learning**	**NLP**	**flying**	**.**
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

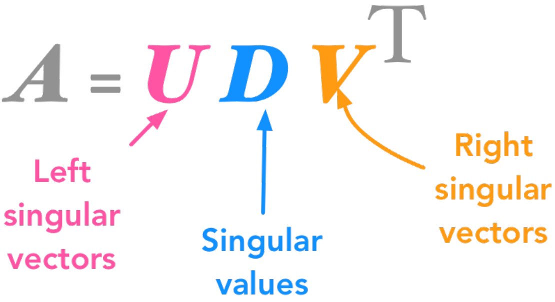
Problems with simple co-occurrence vectors

- Increase in size with vocabulary
- very high dimensional: require a lot of storage
- very sparse (problem for classification models)

$\rightarrow$ models are less robust

Solution: low-dimensional vectors

- Idea: store important information in a dense vector:
  - fixed, smaller number of dimensions
  - usually 25 - 1000 dimensions
- Reduce dimensionality: SVD



Problems with SVD

- Computationally expensive (for n x m matrix: $O(mn^2)$) $\rightarrow$ bad for large corpora
- Hard to incorporate new words or documents

Count or predict – Pros and cons

Count-based	**vs.**	**Prediction**
Fast training (if corpus not too large)		Scales with corpus size
Efficient usage of stats		Inefficient usage of stats
Primarily used to capture word similarity		Can capture complex patterns beyond word similarity
Disproportionate importance given to large counts		Improved performance on extrinsic tasks

GloVe: Objective function

- GloVe: Global Vectors (Pennington, Socher & Manning 2014)

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - logP_{ij})^2 \tag{4}$$

Training objective:

- learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence

GloVe: Objective function

- GloVe: Global Vectors (Pennington, Socher & Manning 2014)

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - logP_{ij})^2 \tag{4}$$

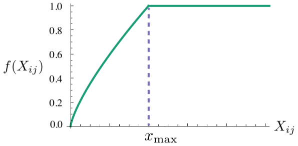
parameters of the model

GloVe: Objective function

- GloVe: Global Vectors (Pennington, Socher & Manning 2014)

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - logP_{ij})^2 \tag{4}$$

weighting function: $f(x) = \begin{cases} (x/x_{max})^{\alpha} & if \;\; x < x_{\max} \\ 1 & otherwise \end{cases}$



GloVe: Objective function

- GloVe: Global Vectors (Pennington, Socher & Manning 2014)

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - logP_{ij})^2 \tag{4}$$

for each pair of words i, j, minimise distance between dot product and overall log count in corpus

GloVe: Objective function

- GloVe: Global Vectors (Pennington, Socher & Manning 2014)

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - log P_{ij})^2 \quad (4)$$

  - weighted least-squares objective:
    assigns more weight to frequent word pairs,
    overall solution minimises the sum of the squares

- Advantages:
  - fast training → **instead of large SVD, optimise one count at a time**
  - scales well to large corpora
  - good performance even with smaller datasets and small vectors

GloVe: the best of two worlds?

- GloVe constructs an explicit word-context matrix.
- The optimisation objective is weighted least-squares loss, assigning more weight to the correct reconstruction of frequent items.
- When using the same word and context vocabularies, the GloVe model represents each word as the sum of its corresponding word and context embedding vectors.

How to get the final embeddings from vectors u, v?

- We end up with U and V from all the vectors u and v (in columns)

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - log P_{ij})^2 \tag{5}$$

- Both capture similar co-occurrence information
- How to get one single word vector? (concatenate, avg., sum?)
- Best solution in practice: simply sum them up

$$X_{final} = U + V \tag{6}$$

Glove – Sum-up

Glove vs Skipgram

- Skip-gram captures co-occurrences one window at a time
- GloVe captures the counts of the overall statistics of how often words appear

- Glove shows connection between count-based and predict-based models:
  $\Rightarrow$ appropriate scaling and objective gives count-based models the properties and performance of predict-based models

- Related work
  - Levy & Goldberg (2014)
  - Arora, Li, Liang, Ma & Risteski (2016)
  - Hashimoto, Alvarez-Melis & Jaakkola (2016)

References

- J. Pennington, R. Socher and C.D. Manning (2014): GloVe: Global Vectors for Word Representation. EMNLP 2014. Doha, Qatar.
- O. Levy, Y. Goldberg and I. Dagan (2015): Improving Distributional Similarity with Lessons Learned from Word Embeddings. TACL.
- S. Arora, Y. Li, Y. Liang, T. Ma and A. Risteski (2018): Linear Algebraic Structure of Word Senses, with Applications to Polysemy. TACL: Transactions of the Association for Computational Linguistics. 2018.
- T.B. Hashimoto, D. Alvarez-Melis and T.S. Jaakkola (2016). Word Embeddings as Metric Recovery in Semantic Spaces. TACL: Transactions of the Association for Computational Linguistics. 2016.

Code and Embeddings

- `https://nlp.stanford.edu/projects/glove/`