Statistical Natural Language Processing

Distributed (word) representations

Çağrı Çöltekin

University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2019

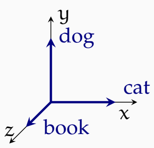
Representations of linguistic units

- Most ML methods we use depend on how we represent the objects of interest, such as
  - words, morphemes
  - sentences, phrases
  - letters, phonemes
  - documents
  - speakers, authors
  - ...
- The way we represent these objects interacts with the ML methods
- We will mostly talk about word representations
  - They are also applicable any of the above

Symbolic (one-hot) representations

A common way to represent words is one-hot vectors

$$\begin{aligned} \text{cat} &= (0, \ldots, 1, 0, 0, \ldots, 0) \\ \text{dog} &= (0, \ldots, 0, 1, 0, \ldots, 0) \\ \text{book} &= (0, \ldots, 0, 0, 1, \ldots, 0) \\ &\ldots \end{aligned}$$

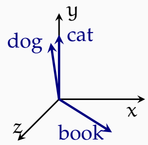


- No notion of similarity
- Large and sparse vectors

More useful vector representations

- The idea is to represent similar words with similar vectors

$$\begin{aligned} \text{cat} &= (0, 3, 1, \ldots, 4) \\ \text{dog} &= (0, 3, 0, \ldots, 3) \\ \text{book} &= (4, 1, 4, \ldots, 5) \\ &\ldots \end{aligned}$$



- The similarity between the vectors may represent similarities based on
  - syntactic
  - semantic
  - topical
  - form
  - ... features useful in a particular task

Where do the vector representations come from?

- The vectors are (almost certainly) learned from the data
- Typically using an unsupervised (or self-supervised) method
- The idea goes back to,
  You shall know a word by the company it keeps. —Firth (1957)
- In practice, we make use of the contexts (company) of the words to determine their representations
- The words that appear in similar contexts are mapped to similar representations

How to calculate word vectors?

count word in context

$$\begin{array}{r c} & \begin{array}{ccccc} c_1 & c_2 & c_3 & \ldots & c_m \end{array} \\ \begin{array}{r} \text{cat} \\ \text{dog} \\ \text{book} \end{array} & \begin{bmatrix} 0 & 3 & 1 & \ldots & 4 \\ 0 & 3 & 0 & \ldots & 3 \\ 4 & 1 & 4 & \ldots & 5 \end{bmatrix} \end{array}$$

+ Now words that appear in the same contexts will have similar vectors
- The frequencies are often normalized (PMI, TF-IDF)
− The data is highly correlated: lots of redundant information
− Still large and sparse

How to calculate word vectors?

count, factorize, truncate

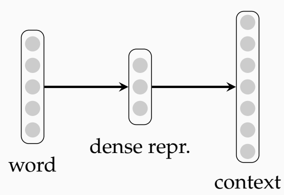
$$\begin{array}{r c} & \begin{array}{ccccc} c_1 & c_2 & c_3 & \ldots & c_m \end{array} \\ \begin{array}{r} w_1 \\ w_2 \\ w_3 \\ \end{array} & \begin{bmatrix} 0 & 3 & 1 & \ldots & 4 \\ 0 & 3 & 0 & \ldots & 3 \\ 4 & 1 & 4 & \ldots & 5 \\ & & \ldots & & \end{bmatrix} \end{array} =$$

$$\begin{array}{r c} & \begin{array}{ccccc} z_1 & z_2 & z_3 & \ldots & z_m \end{array} \\ \begin{array}{r} w_1 \\ w_2 \\ w_3 \\ \end{array} & \begin{bmatrix} 1 & 5 & 9 & \ldots & 4 \\ 1 & 4 & 1 & \ldots & 3 \\ 9 & 1 & 1 & \ldots & 5 \\ & & \ldots & & \end{bmatrix} \end{array} \begin{bmatrix} \sigma_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma_m \end{bmatrix} \begin{array}{c l} \begin{array}{ccccc} c_1 & c_2 & c_3 & \ldots & c_m \end{array} & \\ \begin{bmatrix} 0 & 3 & 1 & \ldots & 4 \\ 0 & 3 & 0 & \ldots & 3 \\ 9 & 1 & 8 & \ldots & 0 \\ & & \ldots & & \end{bmatrix} & \begin{array}{l} u_1 \\ u_2 \\ u_3 \\ \end{array} \end{array}$$

How to calculate word vectors?

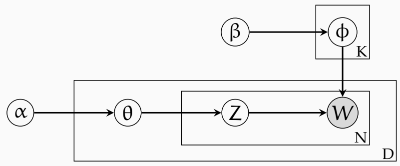
predict the context from the word, or word from the context

- The task is predicting
  - the context of the word from the word itself
  - or the word from its context
- Task itself is not (necessarily) interesting
- We are interested in the hidden layer representations learned



How to calculate word vectors?

latent variable models (e.g., LDA)



- Assume that the each 'document' is generated based on a mixture of latent variables
- Learn the probability distributions
- Typically used for *topic modeling* (θ)
- Can model words too (ϕ)

A toy example

A four-sentence corpus with *bag of words* (BOW) model.

The corpus:

```
S1: She likes cats and
    dogs
S2: He likes dogs and
   cats
S3: She likes books
S4: He reads books
```

Term-document (sentence) matrix

	S1	S2	S3	S4
she	1	0	1	0
he	0	1	0	1
likes	1	1	1	0
reads	0	0	0	1
cats	1	1	0	0
dogs	1	1	0	0
books	0	0	1	1
and	1	1	0	0

A toy example

A four-sentence corpus with *bag of words* (BOW) model.

The corpus:

```
S1: She likes cats and
    dogs
S2: He likes dogs and
   cats
S3: She likes books
S4: He reads books
```

Term-term (left-context) matrix

	#	*she*	*he*	*likes*	*reads*	*cats*	*dogs*	*books*	*and*
she	2	0	0	0	0	0	0	0	0
he	2	0	0	0	0	0	0	0	0
likes	0	2	1	0	0	0	0	0	0
reads	0	0	1	0	0	0	0	0	0
cats	0	0	0	1	0	0	0	0	1
dogs	0	0	0	1	0	0	0	0	1
books	0	0	0	1	1	0	0	0	0
and	0	0	0	0	0	1	1	0	0

Term-document matrices

- The rows are about the terms: similar terms appear in similar contexts
- The columns are about the context: similar contexts contain similar words
- The term-context matrices are typically sparse and large

Term-document (sentence) matrix

	S1	S2	S3	S4
she	1	0	1	0
he	0	1	0	1
likes	1	1	1	0
reads	0	0	0	1
cats	1	1	0	0
dogs	1	1	0	0
books	0	0	1	1
and	1	1	0	0

SVD (again)

- Singular value decomposition is a well-known method in linear algebra
- An $n \times m$ (n terms m documents) term-document matrix $\mathbf{X}$ can be decomposed as

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

$\mathbf{U}$ is a $n \times r$ unitary matrix, where r is the rank of $\mathbf{X}$ ($r \leqslant \min(n, m)$). Columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{X}\mathbf{X}^T$

$\mathbf{\Sigma}$ is a $r \times r$ diagonal matrix of singular values (square root of eigenvalues of $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}^T\mathbf{X}$)

$\mathbf{V}^T$ is a $r \times m$ unitary matrix. Columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{X}^T\mathbf{X}$

- One can consider $\mathbf{U}$ and $\mathbf{V}$ as PCA performed for reducing dimensionality of rows (terms) and columns (documents)

Truncated SVD

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

- Using eigenvectors (from $\mathbf{U}$ and $\mathbf{V}$) that correspond to k largest singular values ($k < r$), allows reducing dimensionality of the data with minimum loss
- The approximation,

$$\hat{\mathbf{X}} = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k$$

results in the best approximation of $\mathbf{X}$, such that $\|\hat{\mathbf{X}} - \mathbf{X}\|_F$ is minimum

Truncated SVD

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

- Using eigenvectors (from $\mathbf{U}$ and $\mathbf{V}$) that correspond to k largest singular values ($k < r$), allows reducing dimensionality of the data with minimum loss
- The approximation,

$$\hat{\mathbf{X}} = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k$$

results in the best approximation of $\mathbf{X}$, such that $\|\hat{\mathbf{X}} - \mathbf{X}\|_F$ is minimum

- Note that r and n may easily be millions (of words or contexts), while we choose k much smaller (a few hundreds)

Truncated SVD (2)

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,m} \\ x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,m} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,m} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,m} \end{bmatrix} =$$

$$\begin{bmatrix} u_{1,1} & \dots & u_{1,k} \\ u_{2,1} & \dots & u_{2,k} \\ u_{3,1} & \dots & u_{3,k} \\ \vdots & \ddots & \vdots \\ u_{n,1} & \dots & u_{n,k} \end{bmatrix} \times \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_k \end{bmatrix} \times \begin{bmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k,1} & v_{k,2} & \dots & v_{n,m} \end{bmatrix}$$

Truncated SVD (2)

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,m} \\ x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,m} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,m} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,m} \end{bmatrix} =$$

$$\begin{bmatrix} u_{1,1} & \dots & u_{1,k} \\ u_{2,1} & \dots & u_{2,k} \\ u_{3,1} & \dots & u_{3,k} \\ \vdots & \ddots & \vdots \\ u_{n,1} & \dots & u_{n,k} \end{bmatrix} \times \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_k \end{bmatrix} \times \begin{bmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k,1} & v_{k,2} & \dots & v_{n,m} \end{bmatrix}$$

The term$_1$ can be represented using the first row of U_k

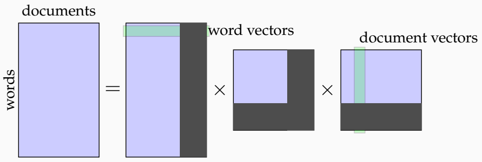
Truncated SVD (2)

$$\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,m} \\ x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,m} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,m} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,m} \end{bmatrix} =$$

$$\begin{bmatrix} u_{1,1} & \dots & u_{1,k} \\ u_{2,1} & \dots & u_{2,k} \\ u_{3,1} & \dots & u_{3,k} \\ \vdots & \ddots & \vdots \\ u_{n,1} & \dots & u_{n,k} \end{bmatrix} \times \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_k \end{bmatrix} \times \begin{bmatrix} v_{1,1} & v_{1,2} & \dots & v_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k,1} & v_{k,2} & \dots & v_{n,m} \end{bmatrix}$$

The document$_1$ can be represented using the first column of V_k^T

Truncated SVD: with a picture



Step 1 Get word-context associations

Step 2 Decompose

Step 3 Truncate

Truncated SVD example

The corpus:

```
(S1) She likes cats and dogs
(S2) He likes dogs and cats
(S3) She likes books
(S4) He reads books
```

	S1	S2	S3	S4
she	1	0	1	0
he	0	1	0	1
likes	1	1	1	0
reads	0	0	0	1
cats	1	1	0	0
dogs	1	1	0	0
books	0	0	1	1
and	1	1	0	0

Truncated SVD ($k = 2$)

$$U = \begin{array}{cl} \begin{bmatrix} -0.30 & 0.28 \\ -0.24 & -0.63 \\ -0.52 & 0.15 \\ -0.03 & -0.49 \\ -0.43 & 0.01 \\ -0.43 & 0.01 \\ -0.03 & -0.49 \\ -0.43 & 0.01 \end{bmatrix} & \begin{array}{l} \text{she} \\ \text{he} \\ \text{likes} \\ \text{reads} \\ \text{cats} \\ \text{dogs} \\ \text{books} \\ \text{and} \end{array} \end{array}$$

$$\Sigma = \begin{bmatrix} 3.11 & 0 \\ 0 & 1.81 \end{bmatrix}$$

$$V^T = \begin{array}{c} \begin{array}{cccc} \text{S1} & \text{S2} & \text{S3} & \text{S4} \end{array} \\ \begin{bmatrix} -0.68 & 0.26 & -0.11 & -0.66 \\ -0.66 & -0.23 & 0.48 & 0.50 \end{bmatrix} \end{array}$$

Truncated SVD (with BOW sentence context)

The corpus:

```
(S1) She likes cats and dogs
(S2) He likes dogs and cats
(S3) She likes books
(S4) He reads books
```


Truncated SVD (with single word context)

The corpus:

```
(S1) She likes cats and dogs
(S2) He likes dogs and cats
(S3) She likes books
(S4) He reads books
```


SVD: LSI/LSA

SVD applied to term-document matrices are called

- *Latent semantic analysis* (LSA) if the aim is constructing *term* vectors
  - Semantically similar words are closer to each other in the vector space
- *Latent semantic indexing* (LSI) if the aim is constructing *document* vectors
  - Topicaly related documents are closer to each other in the vector space

Context matters

In SVD (and other) vector representations, the choice of context matters

- Larger contexts tend to find semantic/topical relationships
- Smaller (also order-sensitive) contexts tend to find syntactic generalizations

SVD based vectors: practical concerns

- In practice, instead of raw counts of terms within contexts, the term-document matrices typically contain
  - pointwise mutual information
  - tf-idf
- If the aim is finding latent (semantic) topics, frequent/syntactic words (*stopwords*) are often removed
- Depending on the measure used, it may also be important to normalize for the document length

SVD-based vectors: applications

- The SVD-based methods are commonly used in information retrieval
  - The system builds document vectors using SVD
  - The search terms are also considered as a 'document'
  - System retrieves the documents whose vectors are similar to the search term
- The well known Google *PageRank* algorithm is a variation of the SVD

> In this context, the results is popularly called "the \$25 000 000 000 eigenvector".

SVD-based vectors: applications

- The SVD-based methods for semantic similarity is also common
- It was shown that the vector space models outperform humans in
  - TOEFL synonym questions

    Receptors for the sense of smell are located at the top of the nasal cavity.
    A. upper end **B.** inner edge **C.** mouth **D.** division
  - SAT analogy questions

    Paltry is to significance as _____ is to _____.
    A. redundant : discussion
    B. austere : landscape
    C. opulent : wealth
    D. oblique : familiarity
    E. banal : originality
- In general the SVD is a very important method in many fields

the song

Predictive models

- Instead of dimensionality reduction through SVD, we try to predict
  - either the target word from the context
  - or the context given the target word
- We assign each word to a fixed-size random vector
- We use a standard ML model and try to reduce the prediction error with a method like gradient descent
- During learning, the algorithm optimizes the vectors as well as the model parameters
- In this context, the word-vectors are called embeddings
- This types of models have become very popular in the last few years

Predictive models

- The idea is the 'locally' predict the context a particular word occurs
- Both the context and the words are represented as low dimensional dense vectors
- Typically, neural networks are used for the prediction
- The hidden layer representations are the vectors we are interested

word2vec

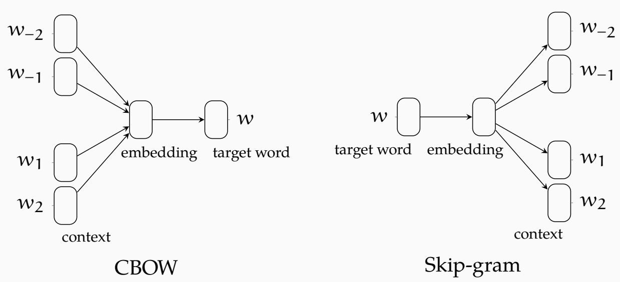
- word2vec is a popular algorithm and open source application for training word vectors (Mikolov et al. 2013)
- It has two modes of operation

CBOW
: or continuous bag of words predict the word using a window around the word

Skip-gram
: does the reverse, it predicts the words in the context of the target word using the target word as the predictor

word2vec

CBOW and skip-gram modes – conceptually



word2vec

a bit more in detail

- For each word w algorithm learns two sets of embeddings

  v_w for words
  c_w for contexts
- Objective of the learning is to maximize (skip-gram)

$$P(c \mid w) = \frac{e^{v_w \cdot c_c}}{\sum_{c' \in C} e^{c_{c'} v_w}}$$

  Note that the above is simply *softmax* – the learning method is equivalent to logistic regression
- Now, we can use gradient-based approaches to find word and context vectors that maximize this objective

Issues with softmax

$$P(c \mid w) = \frac{e^{v_w \cdot c_c}}{\sum_{c' \in c} e^{c_{c'} v_w}}$$

- A particular problem with models with a softmax output is high computational cost:
  - For each instance in the training data denominator need to be calculated over the whole vocabulary (can easily be millions)
- Two workarounds exist:
  - *Negative sampling:* a limited number of negative examples (sampled from the corpus) are used to calculate the denominator
  - *Hierarchical softmax:* turn output layer to a binary tree, where probability of a word equals to the probability of the path followed to find the word
- Both methods are applicable to training, during prediction, we still need to compute the full softmax

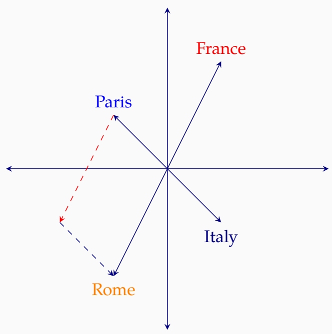
word2vec: some notes

- Note that word2vec is not ‘deep’
- word2vec preforms well, and it is much faster than earlier (more complex) ANN architectures developed for this task
- The resulting vectors used by many (deep) ANN models, but they can also be used by other ‘traditional’ methods
- word2vec treats the context as a BoW, hence vectors capture (mainly) semantic relationships
- There are many alternative formulations

Word vectors and syntactic/semantic relations

Word vectors map some syntactic/semantic relations to vector operations

- Paris - France + Italy = Rome
- king - man + woman = queen
- ducks - duck + mouse = mice



Other methods for building vector representations

- There (quite) a few other popular methods for building vector representations
- *GloVe* tries to combine local information (similar to word2vec) with global information (similar to SVD)
- *FastText* makes use of characters (n-grams) within the word as well as their context
- Recnetly some models of ‘embeding in context’ become popular

Using vector representations

- Dense vector representations are useful for many ML methods
- They are particularly suitable for neural network models
- ‘General purpose’ vectors can be trained on unlabeled data
- They can also be trained for a particular purpose, resulting in ‘task specific’ vectors
- Dense vector representations are not specific to words, they can be obtained and used for any (linguistic) object of interest

Evaluating vector representations

- Like other unsupervised methods, there are no ‘correct’ labels
- Evaluation can be

Intrinsic based on success on finding analogy/synonymy
Extrinsic based on whether they improve a particular task (e.g., parsing, sentiment analysis)
- Correlation with human judgments

Differences of the methods

...or the lack thereof

- It is often claimed, after excitement created by word2vec, that prediction-based models work better
- Careful analyses suggest, however, that word2vec can be seen as an approximation to a special case of SVD
- Performance differences seem to boil down to how well the hyperparameters are optimized
- In practice, the computational requirements are probably the biggest difference

Summary

- Dense vector representations of linguistic units (as opposed to symbolic representations) allow calculating similarity/difference between the units
- They can be either based on counting (SVD), or predicting (word2vec, GloVe)
- They are particularly suitable for ANNs, deep learning architectures

Next:

Mon Text classification
Fri Parsing

Additional reading, references, credits

- Upcoming edition of the textbook (Jurafsky and Martin 2009, ch.15 and ch.16) has two chapters covering the related material.
- See Levy, Goldberg, and Dagan (2015) for a comparison of different ways of obtaining embeddings.

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. second. Pearson Prentice Hall. ISBN: 978-0-13-504196-3.

Levy, Omer, Yoav Goldberg, and Ido Dagan (2015). "Improving distributional similarity with lessons learned from word embeddings". In: *Transactions of the Association for Computational Linguistics* 3, pp. 211–225.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781. URL: `http://arxiv.org/abs/1301.3781`.