Representing linguistic data

Statistical Methods in NLP 2
ISCL-BA-08

Çağrı Çöltekin

ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2025

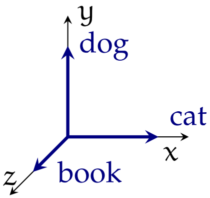
version: none @2025-05-09

Representations of linguistic units

- The success of NLP methods 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 used for the task
- We will mostly talk about word representations
  - They are also applicable any of the above and more

Symbolic (one-hot) representations

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



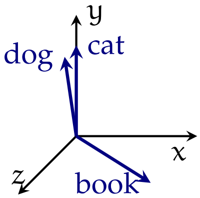
Problems with one-hot representations

- No notion of similarity
- Large and sparse vectors

More useful vector representations: embeddings

- 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
  - ... features useful in a particular task

Where do the vector representations come from?

- The vectors are (almost certainly) learned from 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
- Words that appear in similar contexts are mapped to similar representations

How do we learn word vectors?

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

- Counting/weighting based on context already allows us to learn similarities between words
- But these vectors are large and *sparse*
- *Dense* vectors have a number of desirable properties
  - More efficient to process
  - Removed redundancy also means better generalizations
  - Less sensitive to noise

How to calculate dense word vectors?

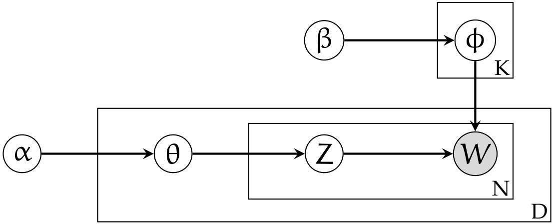
(1) count, factorize, truncate

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

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

How to calculate word vectors?

(2) latent variable models (e.g., LDA)

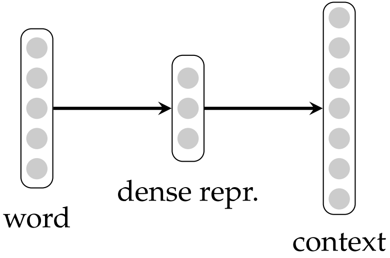


- Assume that the each ‘document’ is generated based on a mixture of latent variables
- Learn the probability distributions
- Typically used for *topic modeling* (θ)
- Can model words too (ϕ)

How to calculate word vectors?

(3) 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



Applications of word vectors: similarity and analogy

- 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.

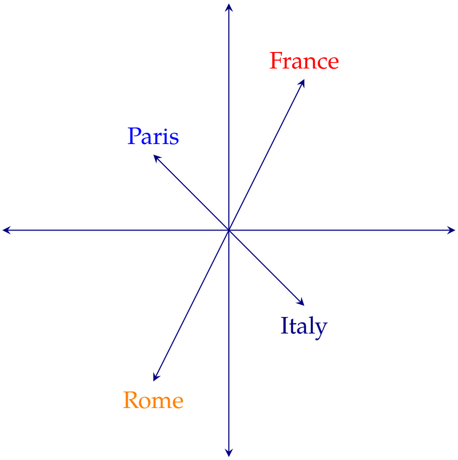
    A. upper end **B.** inner edge **C.** mouth **D.** division
  - SAT analogy questions

    Paltry is to significance as ______ is to ______.

    A. redundant : discussion
    B. austere : landscape
    C. opulent : wealth
    D. oblique : familiarity
    E. banal : originality

Vector arithmetic with embeddings

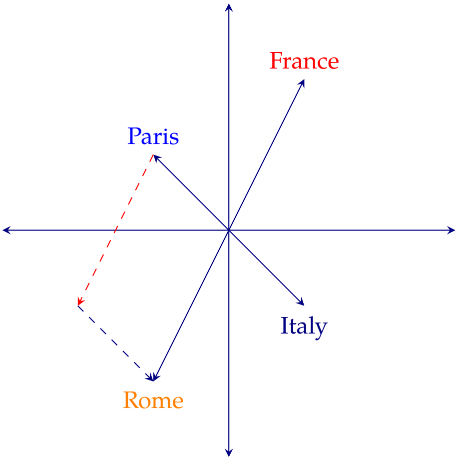
Word vectors map some syntactic/semantic relations to vector operations



Vector arithmetic with embeddings

Word vectors map some syntactic/semantic relations to vector operations

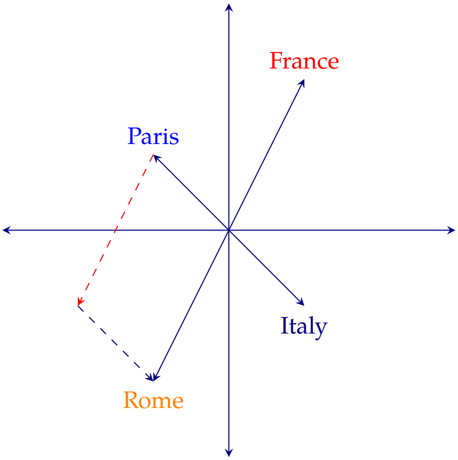
- Paris - France + Italy = Rome



Vector arithmetic with embeddings

Word vectors map some syntactic/semantic relations to vector operations

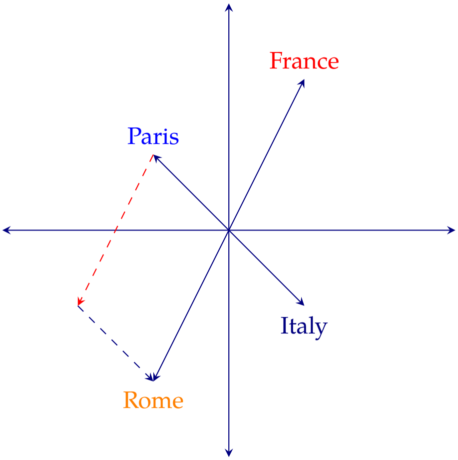
- Paris - France + Italy = Rome
- king - man + woman = queen



Vector arithmetic with embeddings

Word vectors map some syntactic/semantic relations to vector operations

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



Singular Value Decomposition (SVD)

a very short introduction

- 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

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

$\mathbf{U}$ 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 $\mathbf{X}\mathbf{X}^{\mathsf{T}}$

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

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

Truncated SVD

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

- 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,

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

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

Truncated SVD

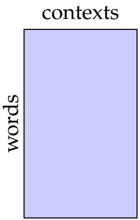
$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

- 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,

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

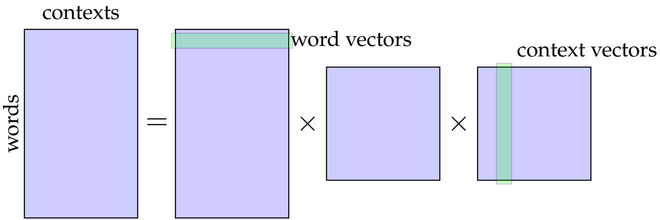
  results in the best approximation of $\mathbf{X}$, such that $\|\hat{\mathbf{X}} - \mathbf{X}\|_F$ is minimum
- Note that r and n may easily be millions (of words or contexts), while we choose k much smaller (a few hundreds)

Truncated SVD: with a picture



Step 1 Get word-context associations

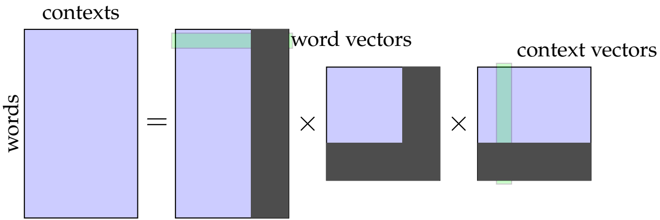
Truncated SVD: with a picture



Step 1 Get word-context associations

Step 2 Decompose

Truncated SVD: with a picture

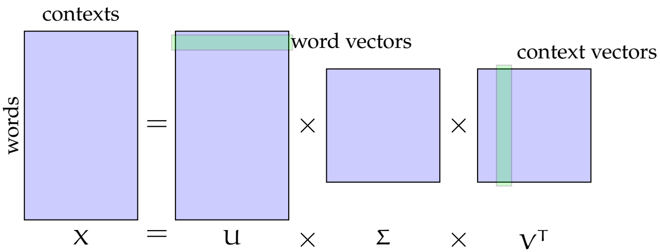


Step 1 Get word-context associations

Step 2 Decompose

Step 3 Truncate

More notes on word vectors from SVD



$$U = XV\Sigma^{-1}$$

- Each component of a 'reduced' word vector is a weighted sum of the original word vector
- SVD removes correlations, resulting in less redundancy

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
  - Topically related documents are closer to each other in the vector space

SVD based vectors: practical concerns

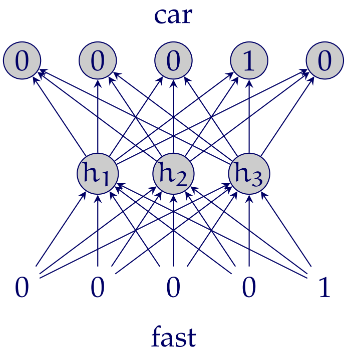
- In practice, instead of raw counts of terms within contexts, the term-document matrices typically contain
  - pointwise mutual information
  - tf-idf
- If the aim is finding latent semantic/topical dimensions, frequent/syntactic words (*stopwords*) are often removed
- Depending on the measure used,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

Matrix factorization: summary

+ Matrix factorization methods were around for a long time: they are well studied and well known
+ These methods are effective: guaranteed optimality / convergence
- The methods do not scale well for large data sets

$\pm$ The mappings are linear

Predictive models



- The idea is the 'locally' predict the context a particular word occurs
- The hidden layer representations are the dense vectors we are interested
- Conceptually, the hidden dimensions encode properties of the word
- Typically we use larger contexts
- Deeper networks may be used for non-linear mappings
- For this lecture, we are interested in *static* embeddings. We will discuss *contextual* representations later

Predictive models

common approaches

- Instead of dimensionality reduction through SVD, we try to predict
  - either the target word from the context
  - or the context given the target word
- In practice ‘shallow’ methods are shown to be effective
- Typically,
  - 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

word2vec

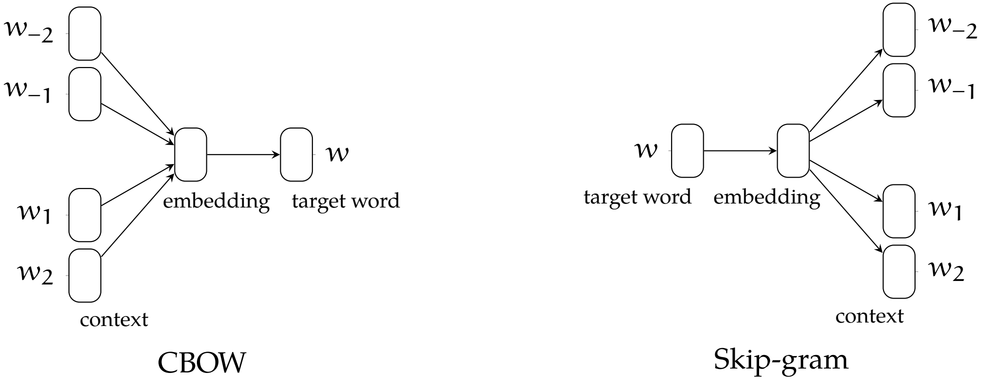
- word2vec is a popular algorithm and open source application for training word vectors
- It has two modes of operation

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, the 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, but we have additional parameters (c) to estimate
- Now, we can use gradient-based approaches to find word and context vectors that maximize this objective

Issues with softmax

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

- A particular problem with models with a softmax output is high computational cost:
  - For each instance in the training data denominator has 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 during 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
- We need to keep the vocabulary (relatively) small, the method does not help with out-of-vocabulary words

Other predictive methods for building vector representations

- There a few other popular methods for building 'general purpose' 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
- One can also train embeddings for a particular task/application, by plugging an 'embedding layer' to any neural network

Using vector representations

- Dense vector representations are useful for many ML methods
- They are particularly suitable as input to neural network models
- The embeddings alone can be used in many applications that require measuring similarities between words
- Dense vector representations are not specific to words, they can be obtained and used for any (linguistic) object of interest

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

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 allow calculating similarity/difference between the units
- General purpose embeddings can be 'trained' using counting (SVD), or predicting (word2vec, GloVe)
- They are particularly suitable for ANNs as low-dimensional inputs
- Although these general purpose embeddings are useful,
  - they typically do not distinguish some important properties (e.g., they assign similar vectors to antonyms)
  - they do not handle polysemy, meaning in context
- Embeddings can also be trained on a particular task
- Also works for other linguistic objects (e.g., letters, sentences)
- Reading: Jurafsky and Martin (2025, Chapter 6)

Summary

- (Dense) vector representations of linguistic units allow calculating similarity/difference between the units
- General purpose embeddings can be 'trained' using counting (SVD), or predicting (word2vec, GloVe)
- They are particularly suitable for ANNs as low-dimensional inputs
- Although these general purpose embeddings are useful,
  - they typically do not distinguish some important properties (e.g., they assign similar vectors to antonyms)
  - they do not handle polysemy, meaning in context
- Embeddings can also be trained on a particular task
- Also works for other linguistic objects (e.g., letters, sentences)
- Reading: Jurafsky and Martin (2025, Chapter 6)

Next:

- Gradient descent, Reading: Jurafsky and Martin (2025, Section 5.6)

Some sources of information

- Jurafsky and Martin (Chapter 6, 2025)

Jurafsky, Daniel and James H. Martin (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. URL: `https://web.stanford.edu/~jurafsky/slp3/`.