

# Deep Learning for Natural Language Processing

## Lecture 4 – Word embeddings

---

Dr. Ivan Habernal

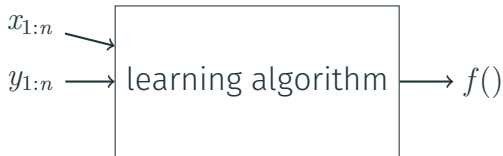
May 3, 2022

Trustworthy Human Language Technologies  
Department of Computer Science  
Technical University of Darmstadt



[www.trusthlt.org](http://www.trusthlt.org)

# What's the problem



Input to neural models  $f()$ : numerical vector but NLP work with texts

How can we represent texts via numerical vectors?

# This lecture

”Classic” features for text representation

Word embeddings

Count-based embeddings

Continuous bag of words and Skip-Gram

Evaluation

# Text Representation

Vector representation of a text should ideally reflect various linguistic properties of the text



In this lecture we focus on feature functions rather than learning algorithms

# Terminology

**Letters:** smallest units in a language → a,b,c,...

**Tokens** and **words**: tokens are outputs of a tokenizer and words are meaning-bearing units

- note: in this course we use the terms “word” and “token” interchangeably

# Terminology

**Lemma:** the dictionary entry of the word → “book” is the lemma of “booking”, “booked”, and “books”

- How to obtain lemmas? Use morphological analyzers
- Available for many languages
- Lemmatization may not work for any sequence of letters, e.g., for mis-spelling

**Stems:** a shorter venison of a word defined based on some language-specific heuristic → “pictur” is the stem of “pictures”, “pictured”, and “picture”

- Stemmer’s output need not be a valid word

# lexical resources

**Lexical resources:** provide information about words and their relations

Dictionaries

WordNet for English

- Semantic knowledge about words
- Each word is associated with one or several synsets
- Synsets are linked to each other according to semantic relations between their words
- Semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, etc.
- Contains nouns, verbs, adjectives, and adverbs
- Manually created

## "Classic" features for text representation

---



# Common Features for Textual Data

What information can we extract directly from a word

- Letters comprising a word and their order
- Length of word
- Is the first letter capitalized?
- Does word include hyphen?

# Bag-of-Words

Bag-of-Words (BoW): histogram of words as a feature vector

## The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

15



|           |     |
|-----------|-----|
| it        | 6   |
| I         | 5   |
| the       | 4   |
| to        | 3   |
| and       | 3   |
| seen      | 2   |
| yet       | 1   |
| would     | 1   |
| whimsical | 1   |
| times     | 1   |
| sweet     | 1   |
| satirical | 1   |
| adventure | 1   |
| genre     | 1   |
| fairy     | 1   |
| humor     | 1   |
| have      | 1   |
| great     | 1   |
| ...       | ... |

BOW does not care about the order of words

TF-IDF: Term Frequency - Inverse Document Frequency

Let  $d$  be a document from a given corpus  $D$

Map word  $w$  of  $d$  to a number as follows:

$$\frac{\#(w, d)}{\sum_{w'} \#(w', d)} \times \log \frac{|D|}{|\{d \in D : w \in d\}|}$$

N-grams: instead of using the frequency of a word, we use the frequency of  $N$  sequence of words

$N=2 \rightarrow$  bi-gram;  $N=3 \rightarrow$  tri-gram

# Common Features for Textual Data

Context: each piece of a text occurs within a larger text which is known as context

How to encode contextual information?

Using position of a word within a sentence or a document

Using the words that appear in an immediate context of a word

- Immediate context: a window of  $k$  words surrounding the target word
- “the cat sat on the mat” ,  $k=2 \rightarrow \{ \text{word-minus-2=the, word-minus-1=cat, word-plus-1=on, word-plus-2=the} \}$

# Common Features for Textual Data

what information can we extract from the relation of text with external source of information?

- is a word in a list of common person names in Germany?
- is a word a female or male person name?
- what is the lemma of the word?
- what is the stem of the word?
- what information do lexical resources give us about the word?

# Different Types of Features

## Numerical features

- Value of a feature is a number, e.g., the frequency of a word in a text

## Categorical features

- Value of a feature is from a set of values
- "What is the POS of a word? "

## How to encode categorical features?

- One-hot encoding
- Dense embedding

# One-hot Encoding

Assume that values of a feature is in categories  $\{v_0, v_1, v_2\}$

Encode the space of feature values via vectors in which

- each entry is either 0 or 1
- each entry is associated with one of the categories
- only one item in a vector can be 1, the rest is 0

Example:

- $v_0 = [1, 0, 0]$
- $v_1 = [0, 1, 0]$
- $v_2 = [0, 0, 1]$

## Example: Categorical Labels

Let  $f()$  take a vector  $x$  which encodes a text and also return a vector  $\hat{y}$  representing the topic of the text which can be from {news, sport, science, politic}

How can we represent topic labels using one-hot encoding?

- news = [1, 0, 0, 0]
- sport = [0, 1, 0, 0]
- science = [0, 0, 1, 0]
- politic = [0, 0, 0, 1]

What is the length of one-hot vectors for a feature with  $k$  categories?



## Example: Word Encodings

Let  $V$  be vocabulary of a language

We take  $V$  as categories a word can take in a text

Let  $V = \{v_0, v_1, v_2, \dots, v_{|V|-1}\}$

- $v_0 = [1, 0, 0, \dots, 0]$
- $v_1 = [0, 1, 0, \dots, 0]$
- $v_2 = [0, 0, 1, \dots, 0]$
- $\dots$
- $v_{|V|-1} = [0, 0, 0, \dots, 1]$

We can easily represent a text via BOW and then represent each word in BOW with its one-hot vector

How can we define vocabulary  $V$  given a corpus  $D$ ?

# One-hot Encodings

- word vectors are very sparse
- semantic relations between words are not encoded in word vectors
- it's better to use one-hot representations for a few distinct features where we expect no correlation between features

# Dense Encodings

- a categorical feature is embedded as a vector in a  $d$  dimensional space
- assuming categories  $C = \{c_0, c_1, \dots, c_{|C|-1}\}$ ,  $d = 4$ 
  - $c_0 = [+0.1, -0.2, +0.3, +0.5]$
  - $c_1 = [-0.2 - 0.1, +0.1, +0.2]$
  - ...
  - $c_{|C|-1} = [+0.2, -0.2, -0.1, +0.3]$

## Example: Word Encodings

- assuming vocabulary  $V = \{v_0, v_1, \dots, v_{|V|-1}\}$ ,  $d = 4$ 
  - $v_0 = [+0.1, -0.2, +0.3, +0.5]$
  - $v_1 = [-0.2 - 0.1, +0.1, +0.2]$
  - ...
  - $v_{|V|-1} = [+0.2, -0.2, -0.1, +0.3]$
- in the one-hot method we represent each word with a sparse vector of size  $|V|$
- in dense encoding method we represent each word with a dense vector with a small size  $d$

# Dense Encodings

- dimensionality of vectors is  $d$
- model training will cause similar features to have similar vectors
- it's mainly useful when we expect some correlations between features  $\rightarrow$  “cat” and “dog” are semantically related
- is also useful when we have a large number of features  $\rightarrow$  for example vocabulary

# Word embeddings

---

# Word Embeddings

- dense representations of words in an embedding space is known as word embeddings
- how to obtain word embeddings?

# Random Initialization

- we initialize the embedding vectors to random values
- values are uniformly sampled from numbers in the range
  - $[-\frac{1}{2d}, +\frac{1}{2d}]$  where  $d$  is the number of dimensions (Mikolov et al., 2013)
  - $[-\frac{\sqrt{6}}{\sqrt{d}}, +\frac{\sqrt{6}}{\sqrt{d}}]$  (xavier initialization)



# Word Embeddings

- dense representations of words in an embedding space is known as word embeddings
- these vectors can be obtained by random initialization and tuned for any NLP task
- however, we as human beings can define semantic relations between words independent of any NLP task
  - e.g., “blue” and “black” are colors
  - e.g., “dog” is more similar to “cat” than to “chair”
  - e.g., “easy” is the opposite of “difficult”
- how can we find word embeddings such that vectors of words with similar meaning be close to each other in the embedding space?

# Distributional Hypothesis

- words that occur in the same contexts tend to have similar meanings (Harris, 1945)
- how can we model the distributional hypothesis?

# Word-Context Matrix

- we count how often a word has occurred with a context in texts from a large corpus
- context is defined by a window over words
- let  $V$  be the set of words in vocabulary and  $C$  be the set of possible contexts
- word-context matrix  $M$  is a two dimensional matrix whose rows are associated with  $V$  and columns with  $C$
- each entry of the matrix indicates how often a word co-occurs with a context in the given corpus
- this matrix is also known as co-occurrence matrix

# Count-based embeddings

---

# Word-Context Matrix

- corpus:
  - I like DL
  - I like NLP
  - I love ML
  - I love NLP
- window size = 1

|      | I | like | love | DL | NLP | ML |
|------|---|------|------|----|-----|----|
| I    | 0 | 2    | 2    | 0  | 0   | 0  |
| like | 2 | 0    | 0    | 1  | 1   | 0  |
| love | 2 | 0    | 0    | 0  | 1   | 1  |
| DL   | 0 | 1    | 0    | 0  | 0   | 0  |
| NLP  | 0 | 1    | 1    | 0  | 0   | 0  |
| ML   | 0 | 0    | 1    | 0  | 0   | 0  |

# Word-Context Matrix

- corpus:
  - I like DL
  - I like NLP
  - I love ML
  - I love NLP
- window size = 1

|      | I | like | love | DL | NLP | ML |
|------|---|------|------|----|-----|----|
| I    | 0 | 2    | 2    | 0  | 0   | 0  |
| like | 2 | 0    | 0    | 1  | 1   | 0  |
| love | 2 | 0    | 0    | 0  | 1   | 1  |
| DL   | 0 | 1    | 0    | 0  | 0   | 0  |
| NLP  | 0 | 1    | 1    | 0  | 0   | 0  |
| ML   | 0 | 0    | 1    | 0  | 0   | 0  |

# Count-based Embeddings

- for a large corpus, the size of matrix is very large
- the matrix could be sparse too
- to encounter this, we may use dimensionality reduction techniques
- however, for adding a new word we need to enlarge the matrix and apply dimensionality reduction again

# CBoW Method

- CBoW stands for Continuous Bag of Words
- task: given a context  $\rightarrow$  predict a missing word from the context

input: "I \_ NLP"  $\rightarrow$  output: "like"

input: ("I \_ NLP", "like")  $\rightarrow$  output: 1 and

input: ("I \_ NLP", "apple")  $\rightarrow$  output: 0



# Continuous bag of words and Skip-Gram

---

# CBoW Method

- given a corpus, we create two sets:
  - positive set ( $D$ ): consisting of pairs  $(c, w)$  where  $c$  is a context and  $w$  is the correct value for the missing word
  - negative set ( $D'$ ): consisting of pairs  $(c, w)$  where  $c$  is a context and  $w$  is a random value for the missing word
- we compute the score estimating similarity between context  $c$  and word  $w$  given context-word pair  $(c, w)$

$$s(c, w) = e(w) \sum_{w_i \in c} e(w_i)$$

where  $e$  is a function that maps each word to its embeddings

# CBoW Method

- we compute the score estimating similarity between context  $c$  and word  $w$  given context-word pair  $(c, w)$

$$s(c, w) = e(w) \sum_{w_i \in c} e(w_i)$$

where  $e$  is a function that maps each word to its embeddings

- Example

$$s(\text{I} \_ \text{NLP}, \text{like}) = e(\text{like})(e(\text{I}) + e(\text{NLP}))$$

# CBoW Method

Use the sigmoid function to map the score to a probability

$$P(y = 1|(c, w)) = \frac{1}{1 + e^{-s(c, w)}}$$

Minimize the following loss

$$\begin{aligned} L(\Theta) = & -\frac{1}{|D|} \sum_{(c, w) \in D} \log \Pr(y = 1|(c, w)) \\ & -\frac{1}{|D'|} \sum_{(c, w) \in D'} \log \Pr(y = 0|(c, w)) \end{aligned}$$

# Skip-Gram Method

Treat words of a context independent from each other

$$P(y = 1|(c, w)) = \prod_{c_i \in c} P(y = 1|(w, c_i)) = \prod_{c_i \in c} \frac{1}{1 + e^{-e(w)e(c_i)}}$$

Loss in Skip-Gram is identical to that in CBoW

We fine-tune parameters (word embeddings) using SGD

# CBoW and Skip-Gram

- CBoW loses the order information between context's elements
- CBoW allows the use of variable-length contexts
- Skip-Gram decouples the dependence between context elements even further than CBoW
- Skip-Gram treats each context's element as an independent context

# Context Matters!

- context of a word translates to its surrounding words in a sentence or paragraph
- window-based context: for a word we consider all words in a window of length  $2m + 1$  centred on the word as context
- the size of window: large windows tend to capture more topical similarities and small windows capture syntactic similarities

# Context Matters!

- contexts can be defined based on text units, for example, words that occur in the same sentence, paragraph, or text
- contexts can also be defined based on syntax of a sentence for example using parse tree or dependency tree of a sentence
- contexts of a word can be foreign words that are aligned to the word in multilingual corpora



# Word2Vec Software Package

- the implementations of CBoW and Skip-Gram methods exist in Word2Vec
- `https://code.google.com/archive/p/word2vec/`
- Skip-Gram is more effective in practice

# GLoVe (Pennington et al., 2014)

- GloVe: Global Vectors for Word Representation
- GloVe is an unsupervised method for obtaining word embeddings
- GloVe aims at reconciling the advantages of corpus-wide co-occurrence counts and local context windows
- `https://nlp.stanford.edu/projects/glove/`

# Evaluating Word Embeddings

- intrinsic
  - word similarity tasks
  - word analogy tasks
- extrinsic
  - on a downstream NLP task, we compare the performance of two models that differ only in the word embeddings they use
  - named entity recognition (NER): accuracy
  - machine translation (MT): BLEU score
  - summarization: ROUGE score
  - information retrieval (IR): precision - recall - F1 score

# Evaluation

---

# Word Similarity Tasks

- similar words should have similar representations
- dataset: <http://alfonseca.org/eng/research/wordsim353.html>
- word-1 and word-2  $\rightarrow$  similarity score  $\in [0, 10]$
- our function  $f()$  should map two words to a similarity score using the distance between the word vectors of the words
- cosine similarity

$$\text{sim}(w_i, w_j) = \frac{e(w_i)e(w_j)}{\|e(w_i)\|^2\|e(w_j)\|^2}$$

# Word Analogy Tasks

- A is to B as C to ?
- Germany is to Berlin as France is to ?
- dataset: <http://download.tensorflow.org/data/questions-words.txt>
- capital-common-countries
  - Athens Greece Baghdad → Iraq
  - Athens Greece Berlin → Germany
- family
  - boy girl brother → sister
  - brother sister dad → mom
- currency, adj-to-adverb, comparative, ...

# Finding the Prototype of a Group of Words

- if we have a group of words  $g = \{w_1, w_2, \dots, w_k\}$
- the prototype of this group can be computed as follows

$$\text{proto}(g) = \frac{1}{k} \sum_{i \in 1..k} e(w_i)$$

where  $e$  maps a word to its embeddings

# Finding Similar Words

- cosine similarity

$$\text{sim}(w_i, w_j) = \frac{e(w_i)e(w_j)}{\|e(w_i)\|^2\|e(w_j)\|^2}$$

- words found by embedding-based similarities can be filtered with other types of word similarities



# Short Text Similarities

$$d_1 = \{w_1^1, w_2^1, \dots, w_m^1\} \text{ and } d_2 = \{w_1^2, w_2^2, \dots, w_n^2\}$$

$$\text{sim}(d_1, d_2) = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n \cos(e(w_i^1), e(w_j^2))$$

# Pre-Trained Embeddings

- Word2Vec
  - trained on Google News (100 billion tokens)
- GloVe
  - trained on Wikipedia (6 billion tokens)
  - trained on CommonCrawl (42 and 840 billion tokens)
  - trained on Twitter (27 million tokens)
- many other pre-trained embeddings for different languages
  - <https://fasttext.cc/docs/en/crawl-vectors.html>

# Practical Hints

- always try out different word embeddings (consider them as a hyperparameter)
- results may vary drastically with different embeddings
- consider source of corpora used to train word embeddings (larger is not always better, a smaller but more domain-focused can be more effective)
- consider what contexts were used to define similarities
- it's better to use the same tokenization and text normalization methods that were used for creating word embeddings

# Limitations

- the algorithms discussed provide very little control over the kind of similarity they include
  - “cat” is more similar to “dog” than to “tiger” as they both are pets
  - “cat” is more similar to “tiger” than to “dog” as they both as felines
- many of the trivial properties of words are ignored because people are less likely to mention known information than they are to mention novel one

# Limitations

- text corpora can easily be biased for better or worse
  - so word embeddings can become biased too
    - gender and racial biases are very common
- these word vectors are context independent
  - in reality, there is no such a thing to have context independent word meaning
  - some words have multiple sense e.g., “bank” may refer to a financial institution or to the side of a river

# Summary

Common features used for converting textual data into numerical vectors

Basics of word embeddings

- How to get them?
- Where to use them?
- How to use them?

Limitations

# License and credits

Licensed under Creative Commons  
Attribution-ShareAlike 4.0 International  
(CC BY-SA 4.0)



## Credits

Ivan Habernal, Mohsen Mesgar, Steffen Eger

Content from ACL Anthology papers licensed under CC-BY  
<https://www.aclweb.org/anthology>

Pictures courtesy of TODO