6 Embedding Methods

- Why Do We Need Embedding Methods?
- word2vec
- doc2vec
- Global Vectors for Word Representation (GloVe)
- fastText
- Lex2Sent
- Applications and Characteristics of Static Word Embeddings

Motivation

Olaf Scholz talks to the media in Berlin.
The chancellor addresses the journalists in the capital.

- no overlap of words - except stopwords
- (calculated) sentence similarity near 0
- meanings are very similar
- aim: build a model which ensures

$$\text{Scholz} \sim \text{chancellor}$$
$$\text{talk} \sim \text{address}$$
$$\text{media} \sim \text{journalist}$$
$$\text{Berlin} \sim \text{capital}$$

Idea

- find vector representation
- of lower dimensionality than vocabulary
- define words by their surrounding words

- short and dense instead of long and sparse vectors
- better ability to generalize (to similar but unseen words)
  - e.g., sentiment analysis
  - no need of exact same word in training and test
  - feature: (discrete) word identity $\rightarrow$ vector with (continuous) values
- easier to use in machine learning approaches

Synonyms and polysemy

- synonyms
  - clustered together in LSA, LDA into topics
  - get similar vectors for similar words
- polysemy
  - e.g., mouse (computer/animal)
  - treated as same word in topic models
  - also not covered by (global) embeddings
  - but can be modeled using contextual embeddings, cf. transformer

word2vec

- not counting co-occurrences
- prediction task: likeliness of co-occurrence
- fit a NN only as helper task $\rightarrow$ throw away the model itself
- take weights as the word embedding
- no need for human labels
- self-supervision: actually co-occurring words as gold standard

Skip-gram with negative sampling (SGNS)

approach:

- treat pairs of target word w and a neighboring context word c as positive examples.
- get negative examples by randomly sample other words from the vocabulary
- train logistic regression to distinguish between positive and negative
- learned weights = embeddings

SGNS training data

... lemon, a [tablespoon of apricot jam, a] pinch

- here: +/- 2 word window
- here: target word w: apricot
- here: $L = 4$ (positive) context words $c_i^{\text{pos}}, i = 1, 2, 3, 4$
- for each positive example k negative examples
- train classifier with positive and negative pairs
- result: embeddings (of size d) for each word in vocabulary

w	c^{pos}	$c^{\text{neg},1}$	$c^{\text{neg},2}$	...	$c^{\text{neg},k}$
apricot	tablespoon	aardvark	seven	...	book
apricot	of	my	forever	...	example
apricot	jam	where	dear	...	on
apricot	a	fish	if	...	while

Probability estimation $P(+ \mid w, c)$

- sigmoid function (for one context word)

$$P(+ \mid w, c) = \sigma(cw) = \frac{1}{1 + \exp(-cw)}$$

$$P(- \mid w, c) = 1 - P(+ \mid w, c)$$

$$= \sigma(-cw) = \frac{1}{1 + \exp(cw)}$$

- assume independence (to take lots of context words into consideration)

$$P(+ \mid w, c_{1,\dots,L}) = \prod_{i=1}^{L} \sigma(c_i w)$$

$$\log P(+ \mid w, c_{1,\dots,L}) = \sum_{i=1}^{L} \log \sigma(c_i w)$$

Skip-gram classifier idea

- estimate probability of w in context window
- by calculating similarities of embedding of w and context words

$$W = (W_{\text{in}}) = \begin{bmatrix} \text{aardvark} & & 1 \\ \text{apricot} & (w_{\text{in},i}^{\text{apricot}})_{i=1,\dots,d} & 2 \\ \dots & & \dots \\ \text{zebra} & & V \end{bmatrix}$$

$$C = (W_{\text{out}}) = \begin{bmatrix} \text{aardvark} & & 1 \\ \text{apricot} & (w_{\text{out},i}^{\text{apricot}})_{i=1,\dots,d} & 2 \\ \dots & & \dots \\ \text{zebra} & & V \end{bmatrix}$$

How to learn vectors

- initialize V d-dimensional embedding vectors randomly
- maximize similarity between target word and (positive) context words
- minimize similarity between target word and sampled (negative) words

$$
\begin{aligned}
L_{\text{SGNS}} &= -\log\left[P(+ \mid w, c^{\text{pos}}) \prod_{i=1}^{k} P(- \mid w, c^{\text{neg},i})\right] \\
&= -\left[\log P(+ \mid w, c^{\text{pos}}) + \sum_{i=1}^{k} \log(1 - P(+ \mid w, c^{\text{neg},i}))\right] \\
&= -\left[\log \sigma(c^{\text{pos}} w) + \sum_{i=1}^{k} \log \sigma(-c^{\text{neg},i} w)\right]
\end{aligned}
$$

- stochastic gradient descent (learning rate: η)
- adjust word weights to make positive/negative pairs more/less likely

Learning step

- derivatives

$$\frac{\partial L_{\text{SGNS}}}{\partial c^{\text{pos}}} = (\sigma(c^{\text{pos}} w) - 1) w$$

$$\frac{\partial L_{\text{SGNS}}}{\partial c^{\text{neg},i}} = \sigma(c^{\text{neg},i} w) w$$

$$\frac{\partial L_{\text{SGNS}}}{\partial w} = (\sigma(c^{\text{pos}} w) - 1) c^{\text{pos}} + \sum_{i=1}^{k} \sigma(c^{\text{neg},i} w) c^{\text{neg},i}$$

- update steps

$$c_{t+1}^{\text{pos}} = c_t^{\text{pos}} - \eta(\sigma(c_t^{\text{pos}} w_t) - 1) w_t$$

$$c_{t+1}^{\text{neg},i} = c_t^{\text{neg},i} - \eta\sigma(c_t^{\text{neg},i} w_t) w_t$$

$$w_{t+1} = w_t - \eta \left[(\sigma(c^{\text{pos}} w) - 1) c^{\text{pos}} + \sum_{i=1}^{k} \sigma(c^{\text{neg},i} w) c^{\text{neg},i} \right]$$

Skip-gram as matrix factorization

- skip-gram can be interpreted as matrix factorization of

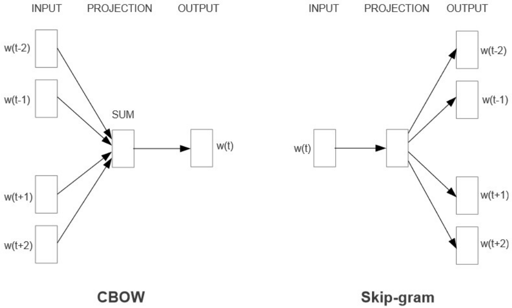
$$W_{\text{in}} W_{\text{out}}^T = WC^T \approx X$$

- where X is the matrix of $\text{PMI}(x, y)$
- word2vec is (implicit) matrix factorization of $\text{PMI}^{0.75} - \log k$

- word2vec output are matrices
  - W target embeddings
  - C context embeddings
- common to use $w^{\text{apricot}} + c^{\text{apricot}}$ as word embedding for apricot

Skip-gram and CBOW as NNs

- idea: predict context from central word → one hidden layer NN
- alternative: CBOW predicts central word from context → one hidden layer NN



Efficient estimation of word representations in vector space (Mikolov et al., 2013)
https://doi.org/10.48550/arXiv.1301.3781

word2vec architecture & training algorithm

architecture

- skip-gram:
  - predicting context words
  - rare words are well represented
  - +/- 10 context window size
- CBOW:
  - predicting central word
  - faster
  - +/- 5 context window size

training algorithm

- negative sampling:
  - sampling of negative examples
  - needs multiple iterations for small amounts of training data
  - rare words are well represented
- hierarchical softmax:
  - Huffman tree
  - better representations for infrequent words
- it is possible to combine negative sampling and hierarchical softmax

see `https://code.google.com/archive/p/word2vec/` and `https://groups.google.com/g/word2vec-toolkit/c/WUWad9fL0jU/m/LdbWy1jQjUIJ`

Context window size

- small windows (+/- 2): syntactically similar words in same taxonomy
- e.g., Hogwarts nearest neighbors are other fictional schools:
  - Sunnydale
  - Evernight
  - Blandings
- large windows (+/- 5): related words in same semantic field
- e.g., Hogwarts nearest neighbors are words from Harry Potter world:
  - Dumbledore
  - half-blood
  - Malfoy
- authors recommend +/- 5 for CBOW and +/- 10 for skip-gram

see `https://code.google.com/archive/p/word2vec/`

How to "doc2vec"?

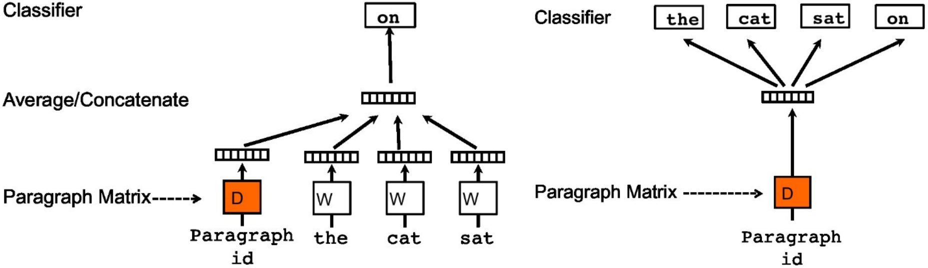
- naive approach?

How to "doc2vec"?

- naive approach? average word embeddings $\rightarrow$ does not work well

How to "doc2vec"?

- naive approach? average word embeddings → does not work well
- distributed memory/bag of words version of paragraph vector (PV-DM → left figure / PV-DBOW → right figure)
- find vector representation for paragraph/document



Distributed representations of sentences and documents (Le & Mikolov, 2014)
https://doi.org/10.48550/arXiv.1405.4053

Global vectors for word representation (GloVe)

- not
  - extracting embeddings from neural network
  - fake task
  - predicting context words (SGNS)
  - predicting target word (CBOW)
- but
  - optimize embeddings directly
  - co-occurrence matrix
  - $w_1 \cdot w_2 = \log \#\text{co-occurrence}(w_1, w_2)$
- https://nlp.stanford.edu/projects/glove/

GloVe: Global Vectors for Word Representation (Pennington et al., 2014)
https://nlp.stanford.edu/pubs/glove.pdf

fastText

- split words into n-grams
- train embeddings for n-grams and complete words
- sum n-gram and complete word embeddings $\rightarrow$ target embedding
- predict context words (not n-grams!) using target embedding (SGNS)

- understands suffixes and prefixes
- understands composed words
- works (often) also with unseen words
- `https://fasttext.cc/`

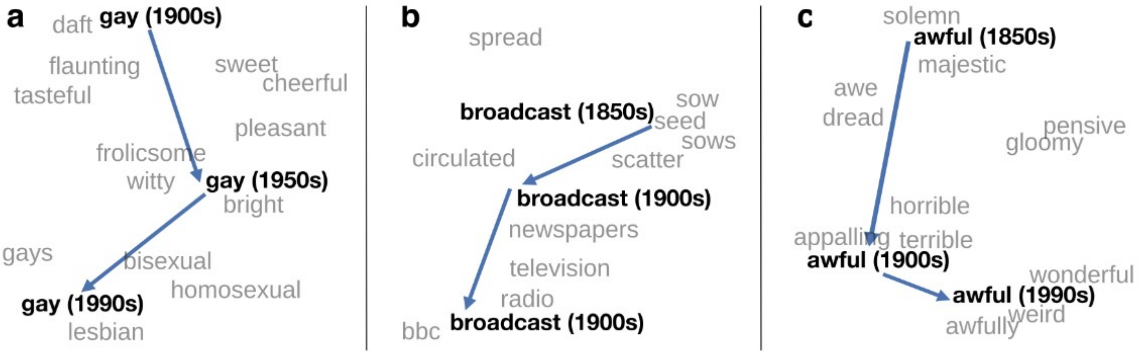
Enriching word vectors with subword information (Bojanowski et al., 2016)
`https://doi.org/10.48550/arXiv.1607.04606`

Lex2Sent

- unsupervised sentiment analysis/prediction (no need of labeled data)
- static embeddings, e.g., Doc2Vec, and sentiment dictionaries, e.g., WKWSCI
- split dictionary in two halves: positive, negative
- not bad -> negbad -> added as positve word
- $\text{diff}_d = \text{neg}_d - \text{pos}_d$ for each document d, where, e.g., neg_d is the cosine similarity of the document embedding to the negative lexicon half
- classification: $\text{diff}_d < 0$ -> text is predicted to be positive
- average diff_d over multiple resamplings of the same document to get rid of noisy/unreliable predictions
- turns out to better, more robust, than common dictionary approaches

Lex2Sent: A bagging approach to unsupervised sentiment analysis (Lange et al., 2024)
`https://aclanthology.org/2024.konvens-main.28/`

Diachronic embeddings



Diachronic word embeddings reveal statistical laws of semantic change (Hamilton et al., 2016)
`http://dx.doi.org/10.18653/v1/P16-1141`

Biases

- king relates to man as queen relates to woman (king−man+woman=queen)
- france−paris+tokyo=

Biases

- king relates to man as queen relates to woman (king$-$man$+$woman$=$queen)
- france$-$paris$+$tokyo$=$japan

Biases

- king relates to man as queen relates to woman (king−man+woman=queen)
- france−paris+tokyo=japan
- doctor−father+mother=

Biases

- king relates to man as queen relates to woman (king−man+woman=queen)
- france−paris+tokyo=japan
- doctor−father+mother=nurse

Biases

- king relates to man as queen relates to woman (king−man+woman=queen)
- france−paris+tokyo=japan
- doctor−father+mother=nurse
- programmer−man+woman=

Biases

- king relates to man as queen relates to woman (king−man+woman=queen)
- france−paris+tokyo=japan
- doctor−father+mother=nurse
- programmer−man+woman=homemaker[1]

- not only retrospectively biased, unfair and sexist
- if used in hiring searches for, e.g., programmers amplifies gender bias[1]

- but can also be used to study cultural biases[2]

[1] Man is to computer programmer as woman is to homemaker? Debiasing word embeddings (Bolukbasi et al., 2016)
https://papers.nips.cc/paper/2016/hash/a486cd07e4ac3d270571622f4f316ec5-Abstract.html
[2] Word embeddings quantify 100 years of gender and ethnic stereotypes (Garg et al., 2016)
https://doi.org/10.1073/pnas.1720347115