MY360/459 Quantitative Text Analysis: Word Embedding Fundamentals

Friedrich Geiecke

March 18, 2024

Course website: lse-my459.github.io

1. Overview and Fundamentals
2. Descriptive Statistical Methods for Text Analysis
3. Automated Dictionary Methods
4. Machine Learning for Texts
5. Supervised Scaling Models for Texts
6. *Reading Week*
7. Unsupervised Models for Scaling Texts
8. Similarity and Clustering Methods
9. Probabilistic Topic Models
10. Word Embedding Fundamentals
11. Neural Network Based Language Models

Demo

- Let us start with a demo of word embeddings
- The lecture will work towards building these functions from scratch

Outline

- Introduction
- word2vec
- GloVe
- Geometry
- Applications and biases
- Coding

Outline

- Introduction
- word2vec
- GloVe
- Geometry
- Applications and biases
- Coding

Word embeddings

- Bag of word approaches only track the frequency of terms, but ignore context, grammar, word order
- Other approaches often use word (or token) embeddings
- Word embeddings represent words as numerical vectors
- The goal is to obtain a measure of a word's "meaning" by its position in space relative to the position of other words
- "You shall know a word by the company it keeps" (John Rupert Firth, 1957)
- Word embeddings can be static (this week) or context-specific (next week)

Word embeddings

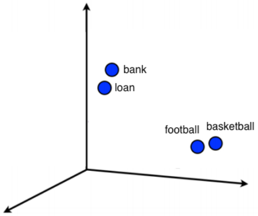
- Learning vector representations of words has been an area of research for long, see e.g. Bengio et al. (2003): "A neural probabilistic language model"
- Yet, *word2vec* (Mikolov, et al. 2013) was able to yield word embeddings of previously unknown quality
- Other static word embeddings are e.g. *GloVe* (Pennington et al., 2014)
- Limitations: Multiple meanings are represented by single vector, e.g. the vector for "interest" is the same in the sentences "The bank charges interest." and "Quantitative text analysis is my main interest."
- In contrast, context specific embeddings often arise naturally from training current neural network architecture as these also require numerical representations of words/tokens internally
- Static embeddings are helpful for studying the fundamentals and obtaining intuition

Word embeddings example

- After training a word2vec or GloVe model e.g. on the corpus of Wikipedia, four exemplary embeddings with 300 dimensions could look like the following

word	D_1	D_2	D_3	...	D_{300}
bank	0.46	0.67	0.05	...	...
loan	0.46	-0.89	-0.08	...	...
football	0.79	0.96	0.02	...	...
basketball	0.80	-0.58	-0.14	...	...

- You can think of each vector as a point in space
- Words that tend to appear together in texts should be close in vector space
- Stylised visualisation in three dimensions:



Outline

- Introduction
- word2vec
- GloVe
- Geometry
- Applications and biases
- Coding

word2vec

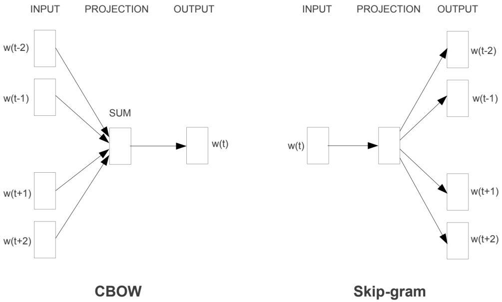
- One popular way to obtain static embeddings is *word2vec* by Mikolov et a. (2013)
- Embeddings are obtained from models that predict context or center words
- Excerpt from some exemplary sentence:

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1          c2    t     c3       c4
```

- Center word at position t: "apricot"
- Context words: "tablespoon", "of", "jam", "a"

word2vec

- The paper actually includes two models that learn word embeddings:



- And two algorithms: Hierarchical softmax and negative sampling
- We will focus on the skip-gram and negative sampling here
- Skip-gram is slower but tends to work better for rarer words

Some preliminaries

- $t = 1, \ldots, T$ is a sequence of training word indices, e.g. going from left to right through all concatenated documents of the corpus
- In the models we discuss here, each unique word i is in fact characterised by two word vectors. One vector as a context word u_i, and one vector as a centre word v_i
- We start with randomly initialised vectors
- These vectors are the parameters of the models and we stack all the u_i's and v_i's into one very large vector θ to simplify notation
- We will furthermore assume a *softmax* structure for probabilities:

$$Pr(x_k) = \frac{exp(x_k)}{\sum_{j=1}^{J} exp(x_j)}$$

Skip-gram model

- Idea: Build a model that predicts context words from centre words
- Can write objective function as[1]:

$$\max_{\theta} J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} log(\ Prob(\ \underbrace{w_{t+j}}_{\text{context word c}}\ |\ \underbrace{w_t}_{\text{word at position } t}\ ;\theta)\ )$$

- Assume for a given context word c and a centre word at position t:

$$Prob(w_{t+j}|w_t;\theta) = \frac{exp(u_{w_{t+j}}^T v_{w_t})}{\sum_{s\in V} exp(u_s^T v_{w_t})}$$

- The u_i's and v_i's are learned with gradient descent such that the relative probabilities for the observed context words are maximised also in the model

1: "For additional details, see CS224N: Natural Language Processing with Deep Learning" http://web.stanford.edu/class/cs224n/, in particular lectures 1 & 2 from 2019 here

Model intuition

- A key question is how a prediction model like this can produce word vectors which are close to each other in vector space when their words also appear in texts together

- For this note that if we had normalised vectors to be of length one (i.e. divided them by their length $||x||$), we could compute cosine similarity simply by the dot product

$$\text{cosine similarity}(x, y) = \frac{x^T y}{||x|| \, ||y||} = \left(\frac{x}{||x||} \right)^T \left(\frac{y}{||y||} \right)$$

- Hence, dot product and similarity measures are closely related

- The predicted probability of a context word is high if the dot product between the vector of the target word and of the content word is high - this is when the similarity of the two vectors tends to be high

Skip-gram model continued

- Say we have a corpus with $10,000$ unique words
- After optimisation, we would have $20,000$ word vectors (two for each word)
- We could e.g. take the average of each u_i and v_i to obtain a final word vector for each word
- Challenge with the so-called *naive softmax* algorithm stated so far: It is very slow because computing the softmax denominator again after each gradient update to u and v is computationally expensive for large vocabularies
- One solution presented in the paper: Use the *negative sampling* algorithm instead to learn good word vectors with an approximation

Negative sampling algorithm

Idea - Create a new supervised learning problem. Run the following loop:

- Sample one word at a position t and one word from its context, i.e. in some window of 5 - 10 words around it
- For a positive case (a true context word) also sample K (e.g. 5) negative words at random from the entire corpus
- Side note: For this sampling of negative words, some slightly modified empirical distribution over the word frequencies of the corpus is used which makes it relatively more likely to sample rarer words
- Then update the vectors by training a logistic regression trying to distinguish the positive $(y = 1)$ case from the negative cases $(y = 0)$ with $Prob(y = 1|c, t) = \sigma(u_c^T v_t)$

After training, use only the v_i vectors as embeddings for each word

Negative sampling: Training data intuition

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Outline

- Introduction
- word2vec
- GloVe
- Geometry
- Applications and biases
- Coding

Example of other vector embeddings: GloVe

- There are many different approaches how one could obtain word embeddings
- "Global Vectors for Word Representation" or GloVe (Pennington et al. 2014) is one other we will discuss briefly
- It tries to combine ideas from word2vec with older approaches of looking at co-occurance matrices such as Rohde et al. (2005)
- Co-ocurrance matrix for an exemplary corpus of three documents "I like deep learning", "I like NLP", "I enjoy flying" with a window length of 1 (more commonly 5-10)

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

From: CS224N - Natural Language Processing with Deep Learning, 2021

GloVe objective function

- Intuition: Solve for all word vectors u_i and v_i with gradient descent such that their inner product is a good predictor for log word co-occurance (b's are intercepts)

$$\min_{\theta} J(\theta) = \sum_{i=1}^{N} \sum_{j=1}^{N} f(X_{i,j})(u_i^T v_j + b_i^u + b_j^v - log(X_{i,j}))^2$$

- $X_{i,j}$: Count in cell i, j of the corpus co-occurrence matrix
- $f(X_{i,j})$: A weighting function which (i) particularly caps the weight of very frequent pairs and (ii) is zero for pairs that do not co-occur (see next point)
- If $X_{i,j} = 0$, then the weighting factor will be $f(X_{i,j})$ and we just assume that "$0 * log(0) = 0$"
- Use $(u_i + v_i)/2$ as final embeddings

Training word vector models

- word2vec and GloVe models have been trained on many different corpora, e.g. Wikipedia, GoogleNews, etc.
- Pretrained GloVe embeddings can e.g. be downloaded here and word2vec embeddings here, or obtained directly through download options in some natural language processing libraries
- You can also train word vector models yourself, e.g. on a recent copy of Wikipedia or on any other set of documents where word similarities might be of interest

Outline

- Introduction
- word2vec
- GloVe
- Geometry
- Applications and biases
- Coding

Visualising word vectors

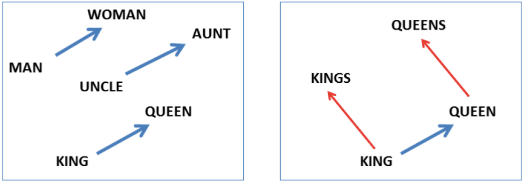
- Word vectors are often visualised in 2 or 3 dimensions. For example with:
- PCA (principal component analysis)
- t-SNE (t-distributed stochastic neighbour embeddings) by van der Maaten et al. (2008)
- t-SNE is a method specifically for visualisation which tries to preserve clusters from high dimensional space also in lower dimensional plots in the commonly shown plots
- A similar method is UMAP (Uniform Manifold Approximation and Projection)
- Yet, a nonlinear mapping from the high to the low dimensional space will distort linear relationships in the low dimensional space
- In general, it is important to keep in mind that the high dimensional space still has much richer structures which we cannot see
- Illustration by Tensorflow

Similarities

- Use original vectors to compute similarities, not the visualisation ones
- Similarity between word vectors is usually computed with the cosine similarity mention earlier: $\frac{x \cdot y}{||x|| \, ||y||}$
- It describes the cosine of the angle between the two vectors and therefore normalised on the interval $[-1, 1]$
- A cosine similarity of 1 implies an angle between the two vectors of 0 degrees, 0 implies 90 degrees, and -1 implies 180 degrees
- If you want to use Euclidian distance instead, normalise all vectors to the same length as otherwise differences in lengths can mechanically drive differences in semantic similarity (particularly relevant to document vectors with different amount of words but the same shares of words)

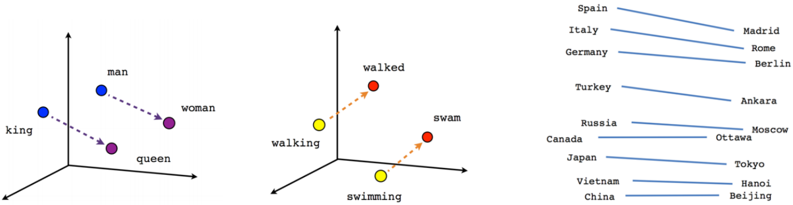
Analogies

- Among the most widely discussed features of word embeddings is their ability to capture analogies via their geometry



- vector('king') - vector('man') + vector('woman') $\approx$ vector('queen')
- How to find 'queen' in detail:
  1. Compute the new vector x = vector('king') - vector('man') + vector('woman')
  2. Find the vector most similar to x via cosine similarity (convention to exclude the vectors 'king', 'man', 'women' individually from outcomes)

Analogies



Male-Female　　Verb tense　　Country-Capital

- Vectors capture general semantic information about words and their relationships to one another
- Analogies work for a surprisingly wide range of examples (see coding session)

Outline

- Introduction
- word2vec
- GloVe
- Geometry
- Applications and biases
- Coding

Studying culture

American Sociological Review
2019, Vol. 84(5) 905–949

DOI: 10.1177/0003122419877135
journals.sagepub.com/home/asr
SAGE

The Geometry of Culture: Analyzing the Meanings of Class through Word Embeddings

Austin C. Kozlowski,[a] Matt Taddy,[b]
and James A. Evans[a,c]

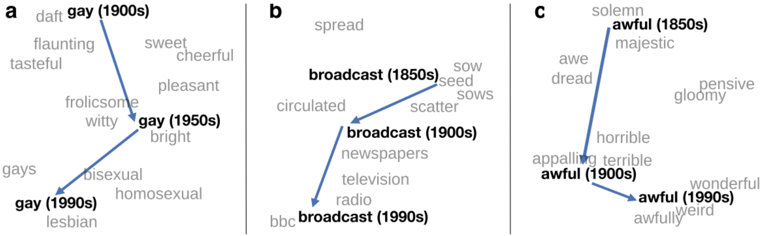
Abstract

We argue word embedding models are a useful tool for the study of culture using a historical analysis of shared understandings of social class as an empirical case. Word embeddings represent semantic relations between words as relationships between vectors in a high-dimensional space, specifying a relational model of meaning consistent with contemporary theories of culture. Dimensions induced by word differences (*rich* – *poor*) in these spaces correspond to dimensions of cultural meaning, and the projection of words onto these dimensions reflects widely shared associations, which we validate with surveys. Analyzing text from millions of books published over 100 years, we show that the markers of class continuously shifted amidst the economic transformations of the twentieth century, yet the basic cultural dimensions of class remained remarkably stable. The notable exception is education, which became tightly linked to affluence independent of its association with cultivated taste.

Kozlowski et al. (2019)

Semantic shifts

Using word embeddings to visualize changes in word meaning:



Source: Hamilton et al. (2016) ACL
https://nlp.stanford.edu/projects/histwords/

Biases in word embeddings

- Important to be aware that semantic relationships in the embedding space have many biases through the data on which they are trained

Gender stereotype *she-he* analogies.

sewing-carpentry	register-nurse-physician	housewife-shopkeeper
nurse-surgeon	interior designer-architect	softball-baseball
blond-burly	feminism-conservatism	cosmetics-pharmaceuticals
giggle-chuckle	vocalist-guitarist	petite-lanky
sassy-snappy	diva-superstar	charming-affable
volleyball-football	cupcakes-pizzas	hairdresser-barber

Gender appropriate *she-he* analogies.

queen-king	sister-brother	mother-father
waitress-waiter	ovarian cancer-prostate cancer	convent-monastery

Source: Bolukbasi et al. (2016) arXiv:1607.06520

- See also e.g. Garg et al. (2018) PNAS and Caliskan et al. (2017) Science

Outline

- Introduction
- word2vec
- GloVe
- Geometry
- Applications and biases
- Coding

Coding

- 01-replicating-key-results.Rmd
- 02-training-word2vec.Rmd